

TMS320C2000™ Digital Signal Controller Power Line Communication

User's Guide

Literature Number: SPRU714
August 2005



| | |
|--|-----------|
| Preface | 7 |
| 1 Single-Frequency Power Line Communication (SFPLC) Introduction | 9 |
| 1.1 System Overview | 10 |
| 1.2 Setting Up the eZdsp Board | 11 |
| 1.2.1 eZdsp Sockets and Jumpers | 11 |
| 1.2.2 Modem Board Jumpers | 11 |
| 1.2.3 Cables | 11 |
| 1.3 Loading the DSP Firmware | 11 |
| 1.4 PC Software | 12 |
| 1.4.1 Software Installation | 12 |
| 1.4.2 Starting the PLC Control Software | 12 |
| 1.5 Status LED Definitions | 12 |
| 2 Single-Frequency Power Line Communication (SFPLC) Hardware Description - Phase Shift Key | 13 |
| 2.1 Introduction | 14 |
| 2.2 System Overview | 14 |
| 2.2.1 Signal Reception | 15 |
| 2.2.2 Signal Transmission | 16 |
| 2.3 Transmit Hardware Description | 16 |
| 2.3.1 PWM Generation of the Transmit Waveform | 16 |
| 2.3.2 Transmitter Lowpass Amplifier Equations | 18 |
| 2.4 Receiver Hardware Description | 20 |
| 2.4.1 Signal Selection Circuits | 20 |
| 2.4.2 Signal Detection Logic | 22 |
| 2.4.3 Analog Bandpass Filter Circuit | 23 |
| 2.5 DSP Processor Utilization | 24 |
| 2.6 Auxiliary Circuits on the PLC Modem Platform | 24 |
| 2.6.1 RS-232 Interface | 24 |
| 2.6.2 Status LEDs | 25 |
| 2.6.3 White LEDs | 25 |
| 2.6.4 Power Supplies | 25 |
| 2.6.5 Stack Headers | 26 |
| 3 Single Frequency Power Line Communication (SFPLC) Hardware Description - Differential PSK | 27 |
| 3.1 Introduction | 28 |
| 3.2 System Overview | 28 |
| 3.2.1 Signal Reception | 29 |
| 3.2.2 Signal Transmission | 30 |
| 3.3 Transmit Hardware Description | 30 |
| 3.4 Receiver Hardware Description | 30 |
| 3.4.1 Signal Selection Circuits | 30 |

| | | |
|----------|--|-----------|
| 3.4.2 | Signal Detection Logic..... | 32 |
| 3.4.3 | Analog Bandpass Filter Circuit | 35 |
| 3.5 | DSP Processor Utilization..... | 36 |
| 3.6 | Auxiliary Circuits on the PLC Modem Platform..... | 37 |
| 3.6.1 | RS-232 Interface | 37 |
| 3.6.2 | Status LEDs..... | 38 |
| 3.6.3 | White LEDs | 38 |
| 3.6.4 | Power Supplies | 38 |
| 3.6.5 | Stack Headers | 38 |
| 4 | Single Frequency Power Line Communication (SFPLC) Software Architecture | 39 |
| 4.1 | System Overview..... | 40 |
| 4.1.1 | PLC Timing and Frequency..... | 40 |
| 4.1.2 | Message Composition | 40 |
| 4.2 | Modem Software (TMS320F2812 Controller) | 41 |
| 4.2.1 | System Initialization..... | 41 |
| 4.2.2 | Main Loop | 41 |
| 4.2.3 | Interrupts | 42 |
| 4.3 | DSP Command Interface..... | 48 |
| 4.4 | Status Reporting..... | 52 |
| 4.4.1 | PLC Statistics | 52 |
| 4.4.2 | Status LED Definitions..... | 53 |
| 4.5 | Lamp Control..... | 53 |
| 4.6 | DSP Processor Utilization..... | 54 |
| 4.6.1 | PSK System Memory Usage | 55 |
| 4.6.2 | Processor MIPS Utilization..... | 56 |
| 4.6.3 | FLASH Memory | 57 |
| 4.7 | DPSK System Software Functions | 57 |
| 4.7.1 | Function Names and Descriptions | 57 |
| 4.8 | Software Compile-Time Options | 63 |
| A | Single Frequency Power Line Communication Schematics | 65 |
| B | Glossary | 69 |

List of Figures

| | | |
|------|---|----|
| 1-1 | Single-Frequency PLC Modem Platform | 10 |
| 2-1 | CEA-709 PHY Block Diagram | 15 |
| 2-2 | Transmit Waveform | 16 |
| 2-3 | Tri-Level Pulse Waveform..... | 16 |
| 2-4 | Harmonic Distortion vs Pulse Width and Filter Cutoff Frequency..... | 17 |
| 2-5 | 1/3 Pulse Width Waveform Construction..... | 18 |
| 2-6 | Basic Sallen-Key Circuit | 18 |
| 2-7 | Transmit Low-Pass Filter Amplifier..... | 19 |
| 2-8 | Sampling Effect | 20 |
| 2-9 | Receiver Signal Processing..... | 21 |
| 2-10 | Response of Digital PLL in Receiver | 22 |
| 2-11 | Multiple Feedback Bandpass Amplifier..... | 23 |
| 2-12 | eZDSP Stack Header Pin Definitions..... | 26 |
| 3-1 | CEA-709 PHY Block Diagram | 28 |
| 3-2 | Transmit Waveform | 30 |
| 3-3 | Sampling Effect..... | 31 |
| 3-4 | Receiver Signal Processing..... | 31 |
| 3-5 | Response of Analog and Digital Filters in Receiver | 32 |
| 3-6 | System Frequency Response After the Correlation Filter | 33 |
| 3-7 | Data Detection Waveforms | 34 |
| 3-8 | Multiple Feedback Bandpass Amplifier..... | 35 |
| 3-9 | ezDSP Stack Header Pin Definitions | 38 |
| 4-1 | Transmit ISR Flow Diagram | 43 |
| 4-2 | PSK Receive ISR Flow Diagram | 45 |
| 4-3 | DPSK Receive ISR Flow Diagram | 46 |
| 4-4 | PSK Signal Processing | 47 |
| A-1 | PLC Single - Driver Schematic | 66 |
| A-2 | PLC Single - IOs and LEDs Schematic | 67 |

List of Tables

| | | |
|------|--|----|
| 2-1 | CEA-709 Packet Description | 14 |
| 2-2 | Transmit Amplifier Component Values+..... | 20 |
| 2-3 | Receive Amplifier Component Values | 23 |
| 2-4 | F2812 Controller Hardware Utilization..... | 24 |
| 2-5 | RS-232 Connector Usage..... | 25 |
| 3-1 | CEA-709 Packet Description | 29 |
| 3-2 | Receive Amplifier Component Values | 36 |
| 3-3 | F2812 Controller Hardware Utilization..... | 36 |
| 3-4 | RS-232 Connector Usage..... | 37 |
| 4-1 | DPSK System Timing..... | 40 |
| 4-2 | PSK System Timing..... | 40 |
| 4-3 | Main Loop | 42 |
| 4-4 | Command List | 48 |
| 4-5 | (0001h) Read Memory..... | 48 |
| 4-6 | (0002h) Write Memory..... | 49 |
| 4-7 | (0003h) Read Status | 49 |
| 4-8 | (000Ah) Direct Lamp Control | 50 |
| 4-9 | (000Bh) Lamp Control | 50 |
| 4-10 | (000Ch) Send PLC Message | 51 |
| 4-11 | (000Dh) Configure PLC Bus Flooder | 51 |
| 4-12 | (0020h) Set Echo Acknowledge Address | 52 |
| 4-13 | (0021h) Echo Request..... | 52 |
| 4-14 | (0022h) Echo Acknowledge..... | 52 |
| 4-15 | PSK Statistics Array Assignments | 53 |
| 4-16 | DPSK Statistics Array Assignments | 53 |
| 4-17 | F2812 Controller Hardware Utilization..... | 54 |
| 4-18 | DSP MIPS Utilization | 56 |
| 4-19 | Task MIPS Load..... | 57 |
| 4-20 | DSP Software Functions | 57 |
| 4-21 | Function Call Hierarchy..... | 61 |
| 4-22 | Compile Flags | 63 |

Preface

This user's guide provides the specifications to set up and use the single-frequency power line (SFPL) modem and describes the hardware and software used with this modem in the TMS320F28x family of fixed-point digital signal controllers.

About this Manual

This book contains the following chapters, each of which includes demonstration units that consist of an eZdsp controller developer's board and an analog interface daughtercard.

- Chapter 1: Single-Frequency Power Line Communication (SFPLC) Introduction
- Chapter 2: Single-Frequency Power Line Communication (SFPLC) Hardware Description - Phase-Shift Key (PSK)
- Chapter 3: Single-Frequency Power Line Communication (SFPLC) Hardware Description - Differential Phase-Shift Key (DPSK)
- Chapter 4: Single-Frequency Power Line Communication (SFPLC) Software Architecture
- Chapter 5: Single-Frequency Power Line Communication Schematics

Related Documentation from Texas Instruments

The following documents describe the TMS320C28x™ devices and related support tools. Copies of these documents are available on the Internet at www.ti.com. *Tip:* Enter the literature number in the search box provided at www.ti.com.

SPRS174: — [TMS320F2810](#), [TMS320F2811](#), [TMS320F2812](#), [TMS320C2810](#), [TMS320C2811](#), [TMS320C2812 Digital Signal Processors](#)

Contains the electrical and timing specifications for these devices, as well as signal descriptions and pinouts for all of the available packages.

Trademarks

All other trademarks are the property of their respective owners.

Single-Frequency Power Line Communication (SFPLC) Introduction

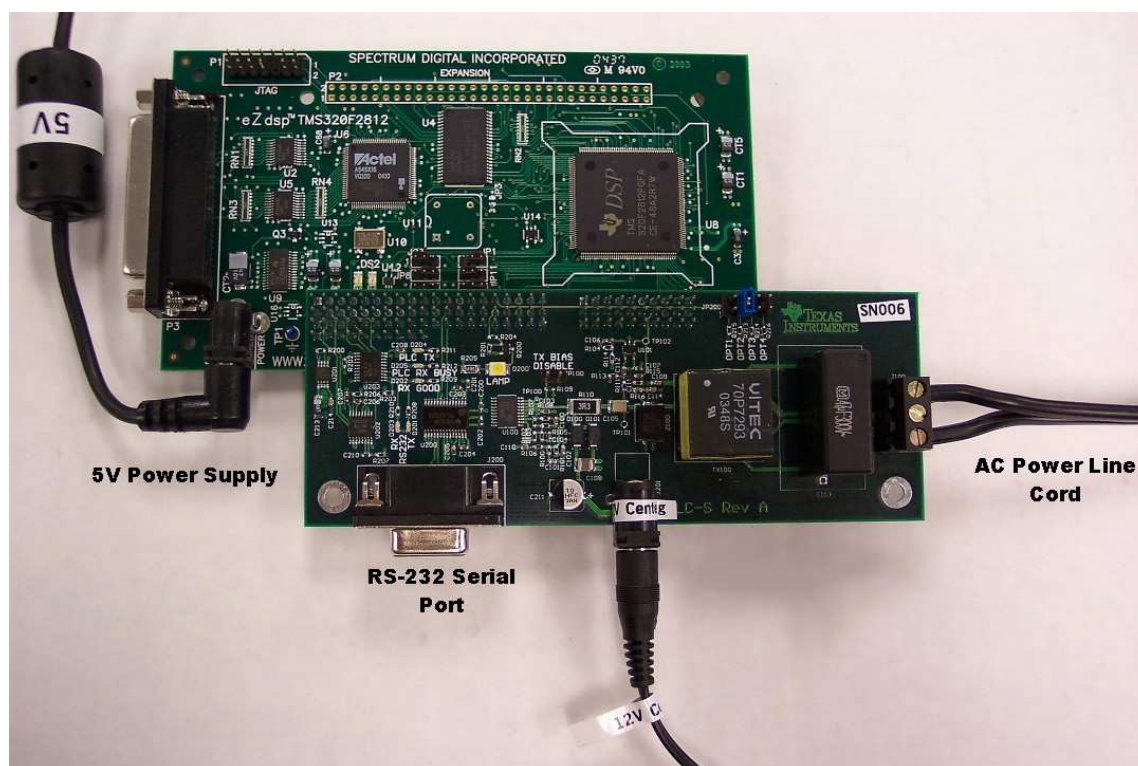
This chapter introduces the single-frequency power line communication modem and describes its setup and usage.

| Topic | Page |
|---|-------------|
| 1.1 System Overview | 10 |
| 1.2 Setting Up the eZdsp Board | 11 |
| 1.3 Loading the DSP Firmware | 11 |
| 1.4 PC Software | 12 |
| 1.5 Status LED Definitions | 12 |

1.1 System Overview

This section describes the setup and usage of the single-frequency power line modem demonstration platform. Each demonstration unit consists of an F2812 eZdsp controller developer's board and an analog daughtercard. The daughtercard contains electronics that shape and amplify the power line communications (PLC) transmit signal, and filter the PLC receive signal. These electronics also manage RS-232 serial port communication, status lights, and a white LED. [Figure 1-1](#) provides a view of the platform.

Figure 1-1. Single-Frequency PLC Modem Platform



The modem communicates with a host PC via an RS-232 serial interface.

WARNING

High Voltage

When connected to the power line, whether 115V or 230V, the line voltage is present on the daughtercard at the screw terminal and across the high voltage AC blocking capacitor. In addition, the AC blocking cap can retain these high voltages for some time after connection to the power line is removed. Use appropriate care when handling and probing the PCBA.

1.2 Setting Up the eZdsp Board

1.2.1 eZdsp Sockets and Jumpers

The PLC analog board plugs into sockets on the eZdsp board. Since the eZdsp expansion ports are not constructed at the factory, these sockets must be soldered onto the eZdsp board before use. Also, because the eZdsp board is shipped from the factory without power supplied to the P4 connector, a zero-ohm resistor must be soldered in the JP4 jumper location on the backside of the eZdsp board, to supply 5V power to the modem board.

1.2.2 Modem Board Jumpers

The option jumpers labeled OPT1 through OPT4 are connected to the DSP's GPIO port. The demo software uses OPT1 and OPT2 to set the device address, as shown here:

| Address | OPT1 | OPT2 |
|---------|------|------|
| 256 | Off | Off |
| 257 | Off | On |
| 258 | On | Off |
| 259 | On | On |

The TX_BIAS_DISABLE jumper is normally not populated. Placing a jumper on it will disable the transmitter for testing purposes. This jumper only controls the driver hardware; the DSP does not detect its setting.

OPT3 and OPT4 are used for test purposes and are not needed for this demo.

1.2.3 Cables

The power types available for the modem are an AC power line and an RS-232 cable.

AC Power Line: To connect the AC power line, you should attach it to the large screw terminals on the right end of the modem board. Then, connect the power leads to the two outside terminals (polarity is not critical). The center terminal is provided for an optional earth ground.

RS232 (Optional): To communicate to the PC software, attach a standard RS-232 cable to the connector.

1.3 Loading the DSP Firmware

The DSP firmware must be loaded into the TMS320F2812 controller. Software developers can compile code in Code Composer Studio IDE™ and load it into the DSP's RAM using the JTAG connection.

For standalone use, you must compile the firmware to run from FLASH and burn the resulting .OUT file into the DSP's FLASH (this .OUT file may have already been compiled for you). Contact your Texas Instruments representative for details.

Code Composer Studio provides a FLASH burn utility to burn the firmware into the FLASH. Follow these steps:

1. Put jumper JP7 into the right-hand position (away from the JTAG connector).
2. Attach the JTAG connector and 5V power supply to the eZdsp board.
3. Start Code Composer Studio. Use the FLASH burn utility to burn the firmware to the DSP FLASH.
4. Shut down Code Composer Studio and disconnect the 5V power supply. Optionally, unplug the JTAG connector.
5. Move jumper JP7 to the left-hand position, and reconnect the 5V power supply.
6. If everything is working properly, after about two seconds the white LED should turn on, and the PLC activity lights should start pulsing once per second.

1.4 PC Software

1.4.1 Software Installation

The power line communication (PLC) control software is a graphical user interface (GUI) created in Matlab® Version 6 (release 12.0). If you do not have Matlab Version 6.0, install the Matlab libraries by clicking on the mglarchive.exe file. This step installs two subdirectories, BIN and Toolbox, and places the necessary Matlab library files (DLLs) in them.

The PLC control software has been archived in a self-extracting ZIP format. To install the software, double-click the ZIP file and unzip the files to the directory C:\TI_Platform\Ballast. The following files will be copied into that directory:

- Serial port libraries (*.dll)
- Matlab setup files (mglarchive.exe and UserSettings.mat)
- Control executables (PLC_Control.bat and PLC_Control.exe)

If you do not have Matlab Version 6.0, install the Matlab libraries by clicking on the mglArchive.exe file. This step installs two subdirectory trees, BIN and TOOLBOX, and places the necessary Matlab library files (DLLs) in them.

1.4.2 Starting the PLC Control Software

Before running the PLC software, connect the PC serial port and power supplies to the demonstration board. To run the PLC software, double-click on the PLC_Control.bat file. This sets your PATH so that the control software can find the Matlab library files, change the current directory to C:\TI_Platform\Ballast, and run PLC_CONTROL.EXE.

1.5 Status LED Definitions

The PLC daughterboard contains three PLC status LEDs.

| Label | Color | IO Port | Description |
|---------|--------|---------|---|
| RX Good | Green | GPIOA8 | The latest PLC packet was received successfully |
| RX Busy | Yellow | GPIOA9 | A PLC packet is being received |
| TX | Red | GPIOA10 | A PLC packet is being transmitted |

Single-Frequency Power Line Communication (SFPLC) Hardware Description - Phase Shift Key

This chapter describes the single-frequency power-line communications (SFPLC) hardware for binary phase-shift keying.

| Topic | Page |
|---|-----------|
| 2.1 Introduction..... | 14 |
| 2.2 System Overview | 14 |
| 2.3 Transmit Hardware Description | 16 |
| 2.4 Receiver Hardware Description..... | 20 |
| 2.5 DSP Processor Utilization | 24 |
| 2.6 Auxiliary Circuits on the PLC Modem Platform | 24 |

2.1 Introduction

A printed circuit board assembly (PCBA), along with an eZdsp controller developers board and the appropriate software, demonstrates a power line communications (PLC) modem that conforms to the CEA-709 standard which calls out a physical layer interface that uses binary phase-shift keying (BPSK) to modulate a 131.5 kHz carrier on the power line. This process produces a 5.5 kb/s communication channel on the power line. The described daughtercard contains electronics that shape and amplify the PLC transmit signal and filter the PLC receive signal. In addition, the daughtercard includes electronics to manage RS-232 serial port communication, status lights, and a white LED whose intensity may be controlled by issuing commands via the PLC or the RS-232. [Figure 1-1](#) shows the diagram of the hardware. [Table 2-1](#) summarizes the packet description of the CEA-709 standard.

Table 2-1. CEA-709 Packet Description

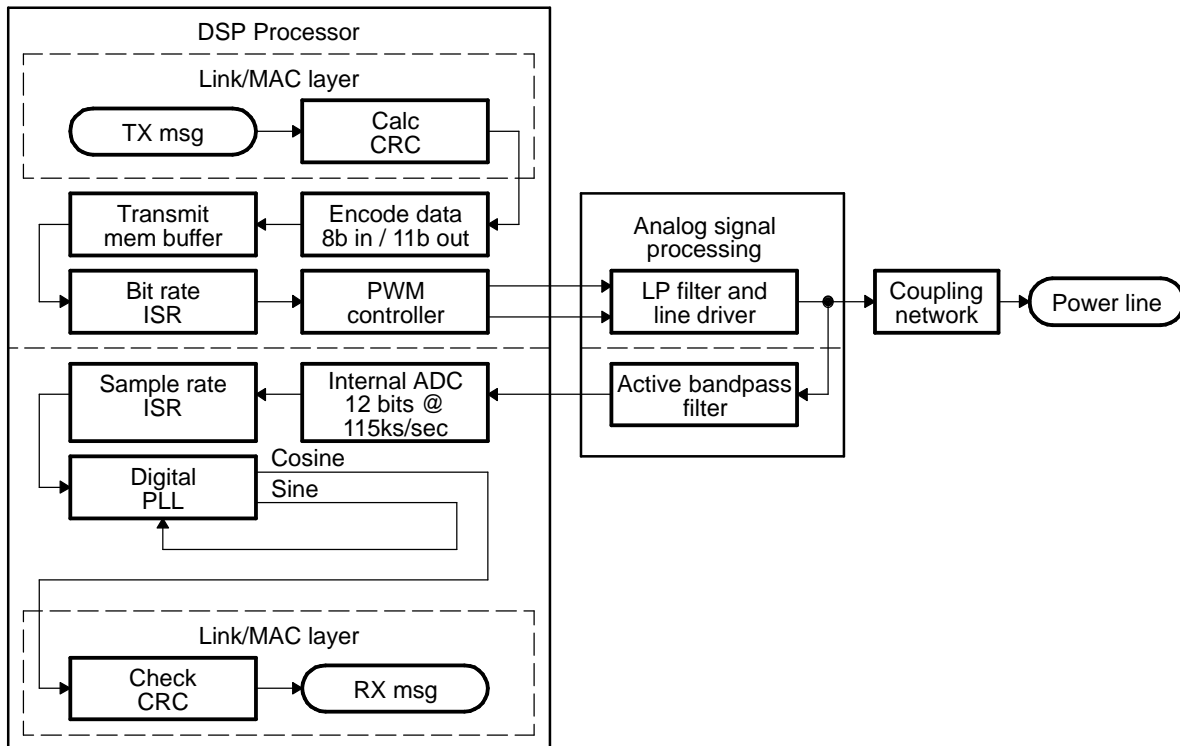
| | | |
|----------------------|---------|--------------------------|
| Fcarrier | 131.579 | kHz |
| Cycles/bit | 24 | |
| Encoded bit rate | 5.482 | kb/s |
| Message data payload | 128 | bytes (for example) |
| Bit encoding rate | 8/11 | bits |
| | 0.727 | |
| L2HDR | 8 | bits |
| NPDU | 1024 | bits |
| CRC | 16 | bits |
| Packet data payload | 1441 | total bits |
| bit sync | 24 | (1010101010101010101010) |
| word sync | 11 | (11001111011) |
| data payload | 1441 | |
| end of packet (EOP) | 11 | (11100110011) |
| end of packet (EOP) | 11 | (11100110011) |
| MAC layer packet | 1498 | bits |
| Effective data rate | 3.748 | kb/s |

Instructions for setting up and using the demonstration platform may be found in [Chapter 1, SFPLC Introduction](#). Details of the software design can be found in [Chapter 4, SFPLC Software Architecture](#).

2.2 System Overview

The CEA-709 standard defines a physical layer interface that uses binary phase-shift keying at 131.5 kHz to perform digital communication using power lines and the communication medium. This demonstration uses a Texas Instruments C28x™ controller to perform most of the signal processing with the hardware interface to the line implemented using an OPA561 line driver, an OPA535 low noise amplifier, and a passive coupling network. The complete CEA-709 protocol stack is a substantial topic; however, this document primarily discusses the physical layer. Within the demonstration software the MAC, link, network, transport, and application layers are combined in the command handler portion of the software. [Figure 2-1](#) shows a block diagram of the CEA-709 modem function.

Figure 2-1. CEA-709 PHY Block Diagram



2.2.1 Signal Reception

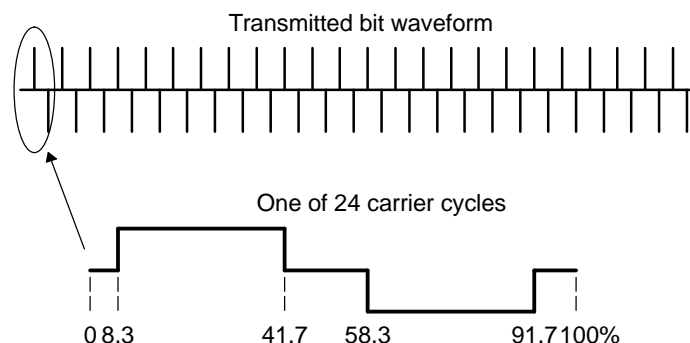
The 131.5 kHz modulated signal is detected by first removing the 50/60Hz power-line voltage at the coupling network and then filtering the signal with a second order-active bandpass filter, which is constructed using an OPA353 amplifier. The OPA353 is a good choice in this application because it has both low-noise voltage performance at 5 nV/ $\sqrt{\text{Hz}}$ and low-quiescent current at 5.2 mA. The output of the bandpass filter is connected to one of the channels of the DSP's analog-to-digital converter (ADC) where it is sampled at a slightly higher level than the carrier frequency: approximately 25 samples per bit or 137.0 ks/s. The signal sample sequence is processed by a FIR matched filter and the output of this filter is used to perform timing recovery and data detection.

In operation, the DSP generates an interrupt at the completion of each A/D conversion. Each sample is then compared to the output of the digital PLL to estimate the phase of the receive signal. At the bit rate, 5.5 kHz, if the phase is less than ± 90 degrees, a zero is assumed to be received; otherwise a one is assumed to be received. The received bit sequence is compared to bit pattern for the "bit sync" field which is at the beginning of a transmitted data packet. When the bit sync pattern is recognized, the modem then begins looking for the "word sync" pattern. The word sync pattern defines the start of message data and also defines the polarity of the message data.

Since the receiving unit does not know when a transmitted packet will arrive, the 115 kHz interrupt must run all the time, and its timing must remain at 115 kHz in order for the PLL algorithm to work. So, for systems based on a processor that does not have a hardware DMA, this interrupt will need to be the highest priority interrupt.

After the data portion of the packet is determined and each bit has been detected, each data byte is decoded from the 11-bit codeword for that byte. The parity bit for each byte is compared to the calculated parity for the transmitted data. The data is passed on from the PHY layer to the MAC layer. If the CRC checksum calculated from the received data agrees with the CRC word that was transmitted, the message is passed on from the MAC/link layer to the network layer. Figure 2-2 shows an example of signal reception detection.

Figure 2-2. Transmit Waveform



2.2.2 Signal Transmission

For applications where the total harmonic distortion (THD) requirements are not too stringent (50 dB or less), the transmit signal can be generated directly from the PWM output of the DSP. Each carrier cycle of each bit is generated with a positive and negative pulse, where the pulse width is 1/3 of the cycle period. The timing can easily be programmed into the PWM hardware registers. Each bit is defined to be 24 cycles, so the PWM controller is allowed to run for 24 cycles, and then an interrupt reassigns the PWM outputs based on the polarity of the bit being transmitted.

To generate the transmit waveform, the message data to be transmitted is passed from the application layer to the session layer, to the transport layer, to the network layer, to the MAC/link layer, and then to the PHY. At the link layer, the CRC word for the message data is calculated and appended to the data. The MAC layer holds the data until the PHY has determined when the channel is available. This is done by looking for the presence of the preamble pattern on the power line.

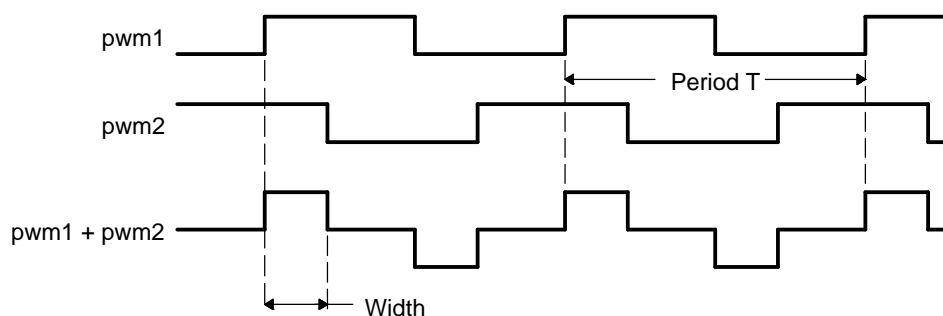
You will use two PWM outputs from the TMS320F28xx controller. The PWM outputs from the processor are used to generate a tri-level pulse train; the power amplifier is configured as a low-pass filter to remove unwanted harmonics.

2.3 Transmit Hardware Description

2.3.1 PWM Generation of the Transmit Waveform

This PLC modem demonstration uses an efficient, low-cost technique to generate the transmit waveform. By utilizing two pulse-width modulation (PWM) outputs from a DSP, a tri-level signal waveform is generated by summing the two waveforms as shown in Figure 2-3.

Figure 2-3. Tri-Level Pulse Waveform



This tri-level waveform is then low-pass filtered to produce a sinewave. Different pulse widths will produce different harmonic frequency content. In order to minimize harmonics that the filter needs to remove, the optimum pulse width is used. This can be found by writing out the formula for the Fourier series for a symmetric pulse ⁽¹⁾ where T is the fundamental frequency period and w is the pulse width.

⁽¹⁾ CRC handbook of Standard Mathematical Tables and Formula, 30th Edition, CRC Press.

$$f(t) = \frac{4}{T} \sum_{n=1}^{\infty} \left[\sin\left(\frac{n\pi}{2}\right) \frac{\sin\left(\frac{n\pi w}{T}\right)}{\frac{n\pi w}{T}} \sin\left(\frac{n2\pi}{T} t\right) \right] \quad (2-1)$$

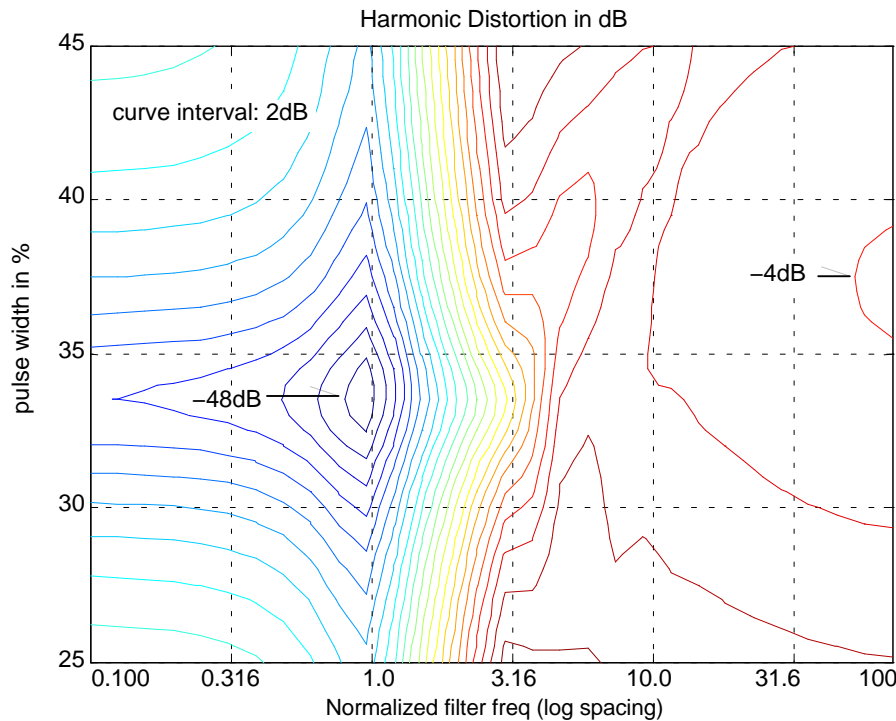
Equation 2-1 can be split into two pieces: the fundamental content of the waveform is given when $n = 1$ and the harmonics are given when $n \geq 2$. Define the measure of goodness as the ratio of the power of the fundamental over the power of the summed harmonics. This is shown in Equation 2-2.

$$THD = \frac{\int_0^T [f(t, n = 1)]^2 dt}{\sum_{n=2}^{\infty} \left[\int_0^T [f(t, n)]^2 dt \right]} \quad (2-2)$$

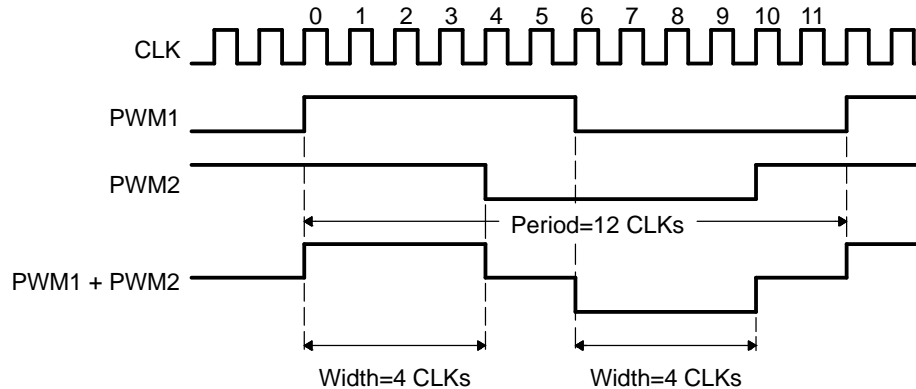
Using these equations and solving for the minimum total harmonic distortion (THD), we find that the optimum pulse width is approximately 37% of the period T . However, this does not take into account the effect of the low-pass filtering the waveform. If a second-order, low-pass filter is applied to the waveform, a different result is found. Figure 2-4 shows a contour plot of simulated THD while varying pulse width and filter cutoff frequency. You can see that when the filter cutoff frequency is high and it attenuates the harmonics very little, the optimum pulse width is 37% as Equation 2-1 predicts. However, at lower filter cutoff frequencies, the optimum shifts to approximately 33% of the pulse period.

The Q of the 2nd order low-pass filter used in these simulations was set to 2.3. This value is a tradeoff between improved THD and the natural response time constant. With a large Q, the THD would be even better, but the circuit rings from one transmitted bit time into the next bit time, which causes inter-symbol interference.

Figure 2-4. Harmonic Distortion vs Pulse Width and Filter Cutoff Frequency



Therefore, the optimum design sets the digital positive and negative pulse widths to 1/3 of the pulse period, and sets the low-pass filter corner frequency to the same frequency as the digital pulse train. The 1/3 pulse width can easily be constructed using a timing clock that is 12 times the transmit waveform frequency. This is shown in Figure 2-5.

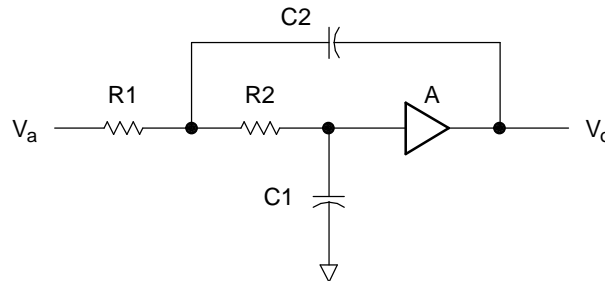
Figure 2-5. 1/3 Pulse Width Waveform Construction


The "best" tri-level digital signal can be obtained by defining a clock that is 12 times the desired transmit carrier frequency, constructing a square wave consisting of six clock periods high and six clock periods low, and then adding to it the negative of that square wave, delayed by four clocks.

To obtain a sinewave from these digital values, an analog circuit that sums the two signals and then low-pass filters the harmonics, must be constructed. The amplifier in the next section performs these operations (see [Figure 2-6](#)).

2.3.2 Transmitter Lowpass Amplifier Equations

Intro to the figure goes here.

Figure 2-6. Basic Sallen-Key Circuit


The transmit amplifier is based on a Sallen-Key filter; the transfer function for this circuit can be expressed as:

$$V_{out} = \frac{A}{akR^2C^2} \frac{V_a}{s^2 + \frac{1+k+ak(1-A)}{akRC} s + \frac{1}{akR^2C^2}} \quad (2-3)$$

where $R_1 = kR$, $R_2 = R$, $C_1 = C$, $C_2 = aC$

For an amplifier with a gain of 2, V_{out} can be expressed as follows $V_{out} = \frac{2\omega^2 V_a}{s^2 + \frac{\omega}{Q} s + \omega^2}$ where

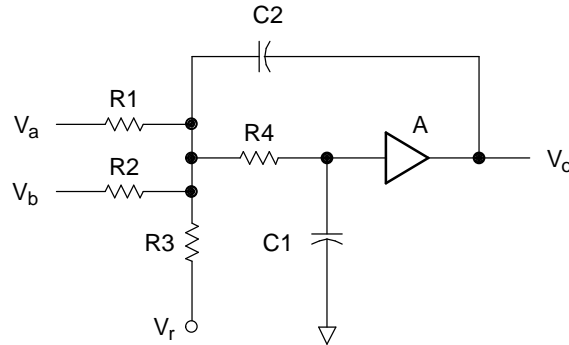
$$\omega = \frac{1}{\sqrt{akRC}} \text{ and } Q = \sqrt{a} \frac{\sqrt{k}}{(1+k)}$$

The peaking in the filter is maximized when Q is maximized and Q is maximized when the quotient

$$\frac{\sqrt{k}}{(1+k)} = 1 \quad . (k \text{ is the ratio of } R_1 \text{ to } R_2)$$

Therefore the resistors R_1 and R_2 in [Figure 2-6](#) are typically set equal in a Sallen-Key filter circuit and the Q is set by adjusting the ratio of the capacitor values. For the PLC transmit circuit, the resistor values must be defined so that the same condition is effectively met.

Figure 2-7. Transmit Low-Pass Filter Amplifier



The transmit amplifier has two inputs that add and filter the signal from the two PWM outputs of the processor. It is desirable to have a fair amount of peaking at the transmit frequency. The more peaking the amplifier provides at the transmit frequency, the more relative attenuation there will be at the harmonic frequencies. Therefore, set the parallel combination of R_1 , R_2 and R_3 equal to R_4 so that a large Q can

easily be obtained. Define $R_4 = R$, then $\frac{1}{R} = \frac{1}{R_1} + \frac{1}{R_2} + \frac{1}{R_3}$ and $R = \frac{R_1 R_3}{2R_3 + R_1}$

Further, define an attenuation factor K such that $\frac{1}{K} = \frac{R_3}{2R_3 + R_1}$. This is the attenuation of the input at the first voltage node in the circuit. Then, define the resistors in terms of R and K .

$$R_1 = KR, R_2 = KR, R_3 = \frac{K}{K-2}R, R_4 = R \quad (2-3)$$

Defining the capacitors as: $C_1 = C$, $C_2 = aC$ now express the transmit amplifier transfer function in terms of A , K , a , R and C

$$V_{out} = \frac{A}{KaR^2C^2} \frac{V_a}{s^2 + \frac{2+a(1-A)}{aRC}s + \frac{1}{aR^2C^2}} = \frac{A}{K} \frac{\omega^2 V_a}{s^2 + \frac{\omega}{Q}s + \omega^2}$$

where $\omega = \frac{1}{\sqrt{a}RC}$ and $Q = \frac{\sqrt{a}}{2 + a(1 - A)}$

For a given Q we can solve for the capacitor ratio: $a = \frac{1 \pm \sqrt{8Q^2(A - 1) + 1}}{2Q^2(A - 1)^2} + \frac{2}{(A - 1)}$

For an amplifier gain of $A = 2$, and picking the smaller solution for a this reduces to:

$$a = 2 + \frac{1 - \sqrt{8Q^2 + 1}}{2Q^2}$$

Finally, at $s = \omega_0$ the transfer function gain is $G(j\omega_0) = -j\frac{AQ}{K}$ Now, all the information needed to define the component values for the transmit amplifier is complete. Table 2-2 shows the component values.

Table 2-2. Transmit Amplifier Component Values+

| | Parameter | Value | Units |
|-----------------------------|-----------|--------|-------|
| Input values | A | 2 | |
| | K | 3.66 | |
| | a | 1.46 | |
| | R | 16.9 | k |
| | C | 56 | pF |
| Calculated parameters | A/K | 0.546 | |
| | Q | 2.26 | |
| | Fc | 138.97 | kHz |
| Calculated component values | R1, R2 | 61.9 | k |
| | R3 | 37.23 | k |
| | R4 | 16.9 | k |
| | C1 | 56 | pF |
| | C2 | 82 | pF |
| | R5 | 1.5 | k |
| | RF | 1.5 | k |

2.4 Receiver Hardware Description

2.4.1 Signal Selection Circuits

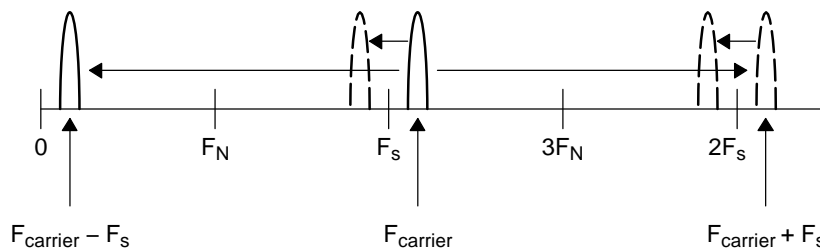
The following steps describe how to convert sine-wave signals to a specific frequency.

First, define a digital communication system where the carrier frequency is 132 kHz and each transmitted bit consists of 24 cycles of a sine wave at the carrier frequency. Then the phase of each bit field can be set to 0 or 180 degrees to encode the bit information.

By sampling the received signal at 21/24ths of the carrier frequency or 115.5 kHz, you can "down-convert" the signal from 132 kHz to an intermediate frequency which is three times the 5.5 kHz bit rate. One way to look at how this works is to think of the ADC as mixing or multiplying the incoming carrier sinewave signal with the sample rate clock. In trigonometry, the result of multiplying two sine wave signals will be a signal containing the sum and difference of the frequency of the two sine waves. Figure 2-8 shows that because of the way the frequencies "fold" around the sample rate and the Nyquist frequency, both the sum and difference end up at $F_s - F_{carrier}$ or 16.5 kHz.

Note that if traditional sampling theory were used, the signal would have to be sampled at greater than $2\sqrt{2} F_c$ or 373 kHz, or 3.2 times the rate used in this system.

Figure 2-8. Sampling Effect



The resulting digital values from the undersampled ADC have a frequency content of 16.5 kHz plus the additive noise from the communication channel. To detect the transmitted signal, the phase of the IF signal in the discrete received signal values must be detected. This is done by first driving a digital phase lock loop (PLL) with the received samples. When the output of the PLL is locked synchronously with the

received signal, an estimate of the complex phase between the PLL and the received signal is generated by the PLL. The real part of the complex phase is the cosine sum, and will be either a large positive value when a 0 (zero) has been received, or a large negative value when a 1 has been received. The complex part of the phase is the sine sum. This represents the phase error and is fed back to the PLL to adjust the sine output so that it tracks the received signal. [Figure 2-9](#) and [Figure 2-10](#) illustrate this process.

Figure 2-9. Receiver Signal Processing

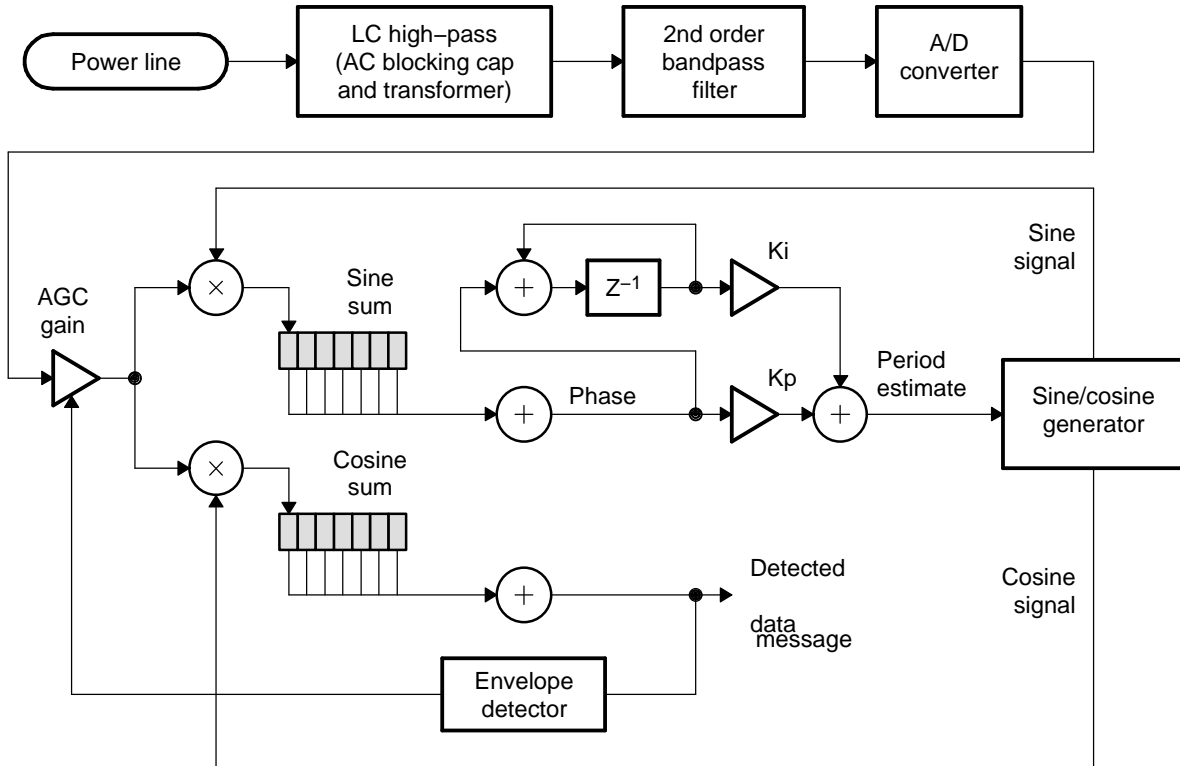
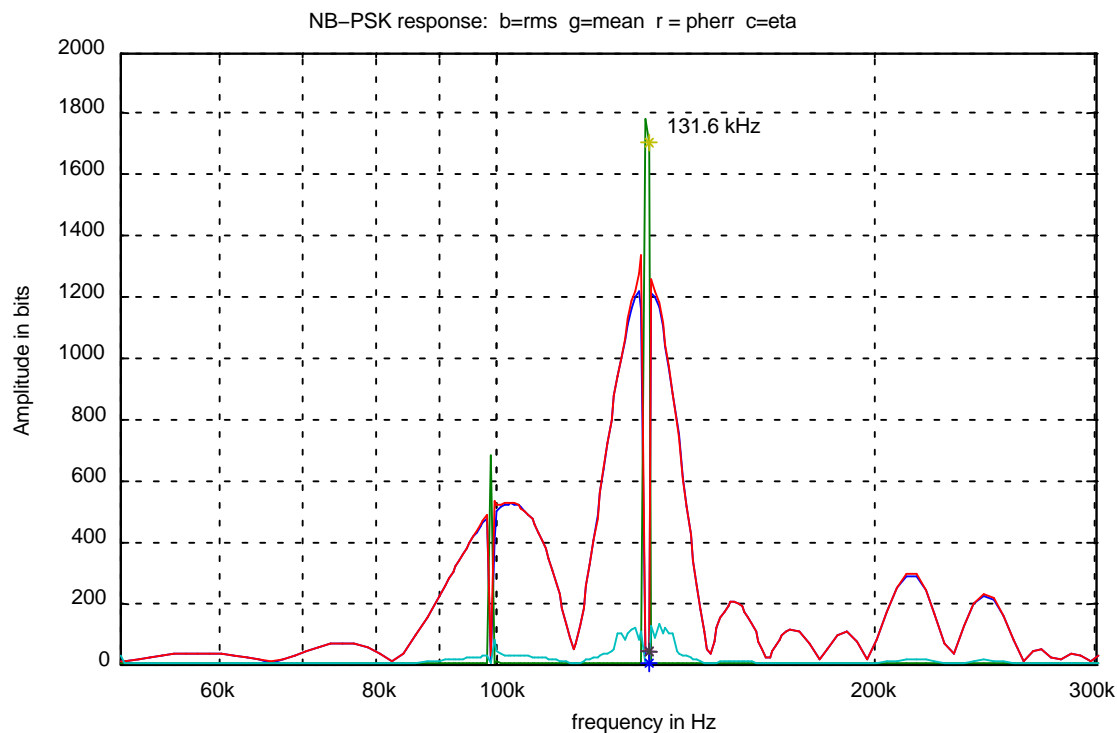


Figure 2-10. Response of Digital PLL in Receiver


2.4.2 Signal Detection Logic

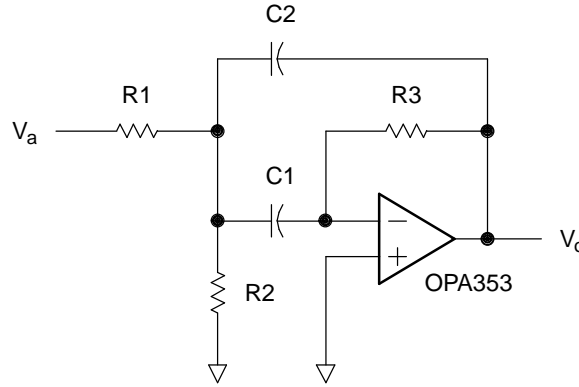
Once a sufficient amount of noise from the received signal has been removed, you can detect the message data encoded in the signal. To do this, these operations must be performed:

1. Detect the presence of the carrier
2. Detect the bit field boundaries
3. Detect the word sync pattern
4. Detect each data bit
5. Detect the parity bit and suffix bits
6. Detect the end-of-packet (EOP) pattern

2.4.3 Analog Bandpass Filter Circuit

Figure 2-11 shows the second-order bandpass filter used to attenuate out-of-band frequency content in the received signal. The amplifier is an OPA353 low-noise amplifier. The equations used to select component values follow.

Figure 2-11. Multiple Feedback Bandpass Amplifier



First, define the resistive and capacitive components as a function of R and C where $R_1 = R$, $R_2 = R/(a - 1)$, $R_3 = gR$, $C_1 = C$, $C_2 = C$ then

$$V_{out} = \frac{A}{Q} \frac{\omega_0 S V_a}{s^2 + \frac{\omega_0}{Q} s + \omega_0^2}$$

where $\omega_0 = \sqrt{\frac{a}{g}} \frac{1}{RC}$ and $Q = \frac{1}{2} \sqrt{ag}$

At the center frequency ω_0 the output amplitude is

$$V_{out}(j\omega_0) = -\frac{1}{2} g V_a = -A V_a$$

Then, solve for the resistor ratios a and g in terms of the peak amplifier gain A and the desired Q :

$$g = 2A$$

$$a = \frac{2Q^2}{A}$$

Finally, choose F_c , A , Q , and C to calculate the resistor component values:

$$R_1 = \frac{Q}{A} \frac{1}{C\omega_0}$$

$$R_2 = \frac{Q}{2Q^2 - A} \frac{1}{C\omega_0}$$

$$R_3 = 2Q \frac{1}{C\omega_0}$$

A tabular representation is shown in Table 2-3 below.

Table 2-3. Receive Amplifier Component Values

| | |
|-------|-----------|
| F_c | 135.2 kHz |
| Q | 2.06 |
| A | 1.0 |
| R_1 | 35.7 k |
| R_2 | 4.75 k |
| R_3 | 71.5 k |
| C_1 | 68 p |

**Table 2-3. Receive Amplifier Component Values
(continued)**

| | |
|-------|------|
| C_2 | 68 p |
|-------|------|

2.5 DSP Processor Utilization

The table below provides details about the functions, registers and pins of the F2812 controller hardware.

Table 2-4. F2812 Controller Hardware Utilization

| Function | Registers | Pins |
|---------------------------------|-----------|---------|
| Used in Real Application | | |
| PLC Tx Waveform | | |
| Tx+ output | CMP1 | PWM1 |
| Tx- output | CMP2 | PWM3 |
| Polarity control | ACTRA | |
| Period register | T1PR | |
| Tx Bias Enable | | GPIOA11 |
| ADC Rx Input | | |
| ADC Input | ADCIN6 | ADCINA6 |
| Period register | T2PR | |
| USED ONLY FOR DEMO BOARD | | |
| LED Intensity | | |
| PWM output | CMP5 | PWM9 |
| Period register | T3PR | |
| RS-232 Interface | | |
| Serial Data Tx | | SCITXDA |
| Serial Data Rx | | SCIRXDA |
| RTS | | GPIOF0 |
| CTS | | GPIOF1 |
| Status LEDs | | |
| Rx Good | | GPIOA8 |
| Rx Busy | | GPIOA9 |
| Tx | | GPIOA10 |
| Option Jumpers | | |
| Opt1 | | GPIOB9 |
| Opt0 | | GPIOB10 |
| Opt2 | | GPIOB11 |
| Opt3 | | GPIOB12 |

2.6 Auxiliary Circuits on the PLC Modem Platform

2.6.1 RS-232 Interface

The daughterboard contains a 9-pin D-shell connector and an SN75LV4737 transceiver. Signals RD, TD, RTS and CTS, are provided at the connector in the standard RS-232 configuration for a modem. This means that a standard, non-crossover serial port cable can be used to interface from a PC to the PLC modem platform.

The D203 and D201 LEDs are also provided on the board. D203 is a red LED and indicates activity on the RD line, which is used to transfer data from the modem to the PC. D201 is a green LED and indicates activity on the TD, which is used to transfer data from the PC to the modem. [Table 2-5](#) provides more detail about the RS-232 interface.

Table 2-5. RS-232 Connector Usage

| Pin | Modem Schematic | PC Signal Name |
|-----|-----------------|----------------|
| 1 | - | CD |
| 2 | TX | RD |
| 3 | RX | TD |
| 4 | - | DTR |
| 5 | GND | GND |
| 6 | - | DSR |
| 7 | CTS | RTS |
| 8 | RTS | CTS |
| 9 | - | RI |

2.6.2 Status LEDs

A red, a yellow, and a green LED are also provided to display status from the DSP. They drive a SN74LVT240 buffer and connect to the DPS GPIO pins through the I/O interface connector, P200.

The red LED is lit when the modem is transmitting a packet on the power line. The yellow LED is lit when the modem is receiving a packet on the power line. The green LED is lit and stays lit for a programmable period after a packet has been received, and the 16-bit CRC indicates that the packet was received error-free.

2.6.3 White LEDs

As a demonstration of processing a command received over the power line or over the RS-232 PC connection, the daughterboard contains a white LED. The intensity of the white LED can be controlled through one of the PWM channels on the DSP. It is connected to the PWM ch-9 output through pin 32 on the I/O interface connector.

2.6.4 Power Supplies

Both a 5V and a 12V supplier are required for the modem demonstration. The 5V supply furnishes power for the eZdsp board. The eZdsp controller in turn supplies 5V power to the daughterboard through the I/O interface connector. The 12V supply furnishes power for the transmit amplifier on the daughterboard. Both connectors accept a 5.5mm power connector. The 12V power supply connection is clearly labeled on the modem board.

2.6.5 Stack Headers

Figure 2-12 shows the orientation for the pins on the analog interface board that connect to the eZDSP board.

Figure 2-12. eZDSP Stack Header Pin Definitions

| I/O Interface | | | Analog Interface | | | OPT jumpers |
|---------------|-----------|----|------------------|-----------|----|-------------|
| 2 3... | P203 (P4) | 20 | 2 3... | P202 (P5) | 10 | 2 4 6 8 |
| 2 4 6... | P200 (P8) | 40 | 2 4 6... | P201 (P9) | 20 | 3 5 7 |
| 3 5... | | 39 | 3 5... | | 19 | |
| 2 3... | P204 (P7) | 10 | | | | |

| | | | |
|-------------|-----------|----------|---------|
| PLC_RX_GOOD | P200.6 | UART_RX | P200.4 |
| PLC_RX_BUSY | P200.7 | UART_TX | P200.3 |
| PLC_TX | P200.8 | UART_CTS | P200.24 |
| | | UART_RTS | P200.23 |
| PWM1, PWM3 | P200.9,11 | | |
| TX_ENABLE | P200.17 | | |
| ADCIN_A6 | P201.14 | | |

Single Frequency Power Line Communication (SFPLC) Hardware Description - Differential PSK

This chapter describes the single-frequency, power-line communications (PLC) hardware for differential phase shift keying. Most of the information contained in this chapter is the same as the information contained in Chapter 2; however, Section 4.2 and its subsections have been added for this discussion.

| Topic | Page |
|---|-----------|
| 3.1 Introduction..... | 28 |
| 3.2 System Overview | 28 |
| 3.3 Transmit Hardware Description | 30 |
| 3.4 Receiver Hardware Description..... | 30 |
| 3.5 DSP Processor Utilization | 36 |
| 3.6 Auxiliary Circuits on the PLC Modem Platform | 37 |

3.1 Introduction

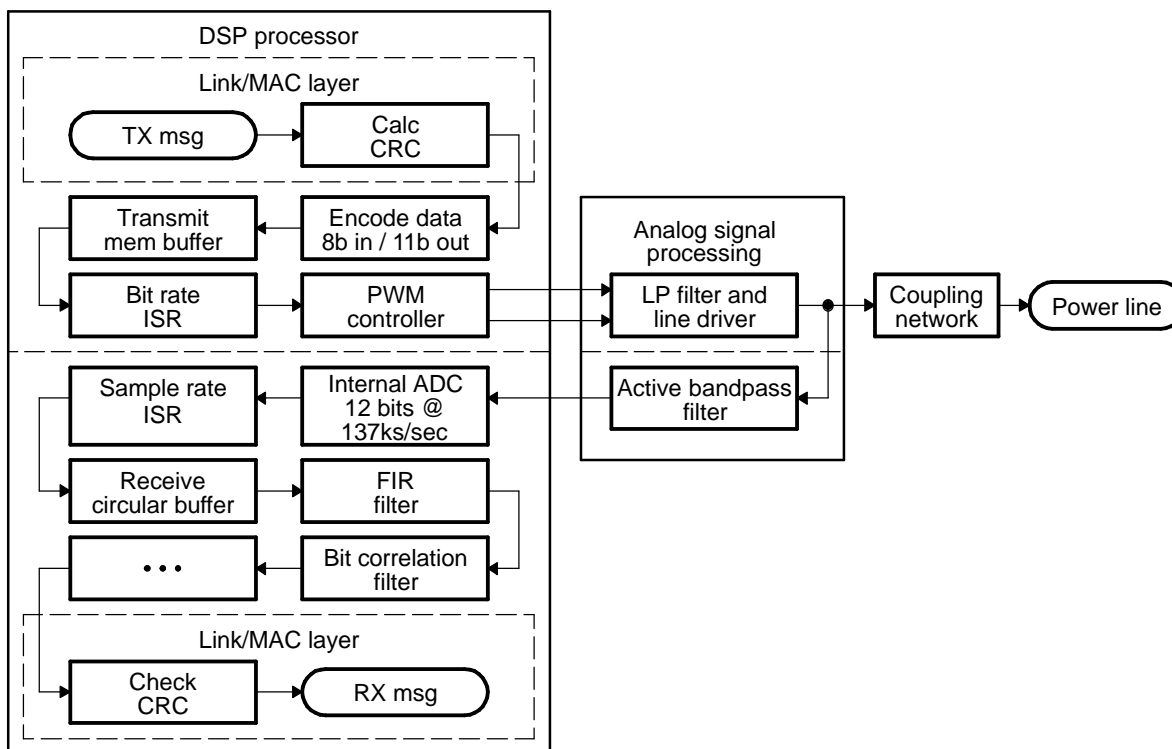
A printed circuit board assembly (PCBA), along with an eZdsp controller developers board and the appropriate software, demonstrates a power-line communications (PLC) modem that conforms to the CEA-709.2 Standard. The Standard calls out a physical layer interface that uses binary phase shift keying (BPSK) to modulate a 131.5 kHz carrier on the power line. This produces a 5.5 kb/s communication channel on the power line. The described daughtercard contains electronics that shape and amplify the PLC transmit signal and filter the PLC receive signal. In addition, the daughtercard includes electronics to manage RS-232 serial port communication, status lights, and a white LED. The intensity of the white LED may be controlled by issuing commands via the PLC or the RS-232. Please refer to [Figure 1-1](#).

Instructions for setting up and using the demonstration platform may be found in [Chapter 1, SFPLC Introduction](#). Details of the software design can be found in [Chapter 4, SFPLC Software Architecture](#).

3.2 System Overview

The CEA-709 Standard defines a physical layer interface that uses binary phase shift keying at 131.5 kHz to perform digital communication using power lines as the communication medium. This demonstration uses a Texas Instruments TMS320C28xx DSP to perform most of the signal processing with the hardware interface to the line implemented using OPA561 line driver, an OPS535 low noise amplifier, and a passive coupling network. The complete CEA-709 protocol stack is a substantial topic; this document primarily discusses the physical layer. Within the demonstration software the MAC, Link, Network, Transport, and Application layers are combined in the command handler portion of the software. [Figure 3-1](#) shows a block diagram of the CEA-709 modem function.

Figure 3-1. CEA-709 PHY Block Diagram



[Table 3-1](#) summarizes the construction of a packet based on the CEA-709 Standard.

Table 3-1. CEA-709 Packet Description

| | | |
|----------------------|---------|----------------------------|
| Fcarrier | 131.579 | kHz |
| Cycles/bit | 24 | |
| Encoded bit rate | 5.482 | kb/s |
| Message data payload | 128 | bytes (for example) |
| Bit encoding rate | 8/11 | bits |
| | 0.727 | |
| L2HDR | 8 | bits |
| NPDU | 1024 | bits |
| CRC | 16 | bits |
| Packet data payload | 1441 | total bits |
| bit sync | 24 | (101010101010101010101010) |
| word sync | 11 | (11001111011) |
| data payload | 1441 | |
| end of packet (EOP) | 11 | (11100110011) |
| end of packet (EOP) | 11 | (11100110011) |
| MAC layer packet | 1498 | bits |
| Effective data rate | 3.748 | kb/s |

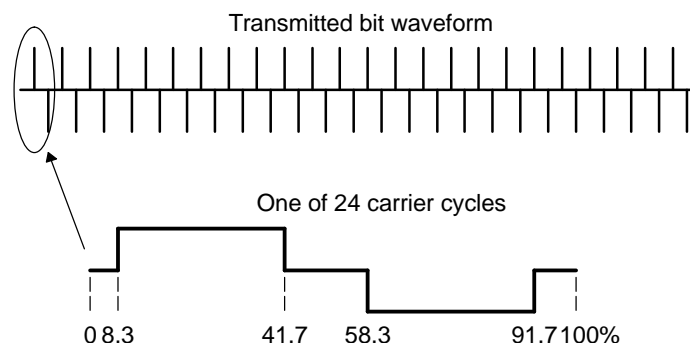
3.2.1 Signal Reception

The 131.5 kHz modulated signal is detected by first removing the 50/60Hz power-line voltage at the coupling network and then filtering the signal with a second order active bandpass filter. This filter is constructed using an OPA353 amplifier. The OPA353 is a good choice in this application because it has both low noise voltage performance at 5 nV/ $\sqrt{\text{Hz}}$ and low quiescent current at 5.2 mA. The output of the bandpass filter is connected to one of the channels of the DSP's ADC where it is sampled at slightly higher level than the carrier frequency: approximately 25 samples per bit or 137.0 ks/s. The signal sample sequence is processed by a FIR-matched filter and the output of this filter is used to perform timing recovery and data detection.

In operation, the DSP has a short interrupt that continually transfers data from the ADC to a receive sample buffer. This buffer is set up as a circular buffer. At the bit rate, 5.5 kHz, the FIR routine is executed and an algorithm to detect the presence of the "bit sync" field is executed. If the bit sync detection algorithm determines that a packet has arrived, the bit window is aligned based on the "bit sync" field and the start of data is determined by detecting the "word sync" field.

Since the receiving unit doesn't know when a transmitted packet is going to arrive, the 137 kHz interrupt needs to run all the time, and its timing needs to be consistently at 137 kHz in order for the FIR filter detection algorithm to work. So, for systems based on a processor that does not have a hardware DMA, this interrupt will likely need to be the highest priority interrupt.

After the data portion of the packet is determined and each bit has been detected, each data byte is decoded from the 11-bit code word for that byte. The parity bit for each byte is compared to the calculated parity for the transmitted data. The data is passed on from the PHY layer to the MAC layer. If the CRC checksum calculated from the received data agrees with the CRC word that was transmitted, the message is passed on from the MAC/Link layer to the Network layer. [Figure 3-2](#) shows a view of the transmitted waveform.

Figure 3-2. Transmit Waveform


3.2.2 Signal Transmission

For applications where the total harmonic distortion (THD) requirements are not too stringent, (50 dB or less), the transmit signal can be generated directly from the PWM output of the DSP. Each carrier cycle of each bit is generated with a positive and negative pulse, where the pulse width is 1/3 of the cycle period. This timing can easily be programmed into the PWM hardware registers. Each bit is defined to be 24 cycles, so the PWM controller is allowed to run for 24 cycles; then an interrupt reassigns the PWM outputs based on the polarity of the bit being transmitted.

To generate the transmit waveform, the message data to be transmitted is passed from the application layer to the session layer, to the transport layer, to the network layer, to the MAC/Link layer, and then to the PHY. At the link layer the CRC word for the message data is calculated and appended to the data. The MAC layer holds the data until the PHY has determined if the channel is available. This is done by looking for the presence of the preamble pattern on the power line.

The PWM outputs from the TMS320F28xx controller are used to generate a tri-level pulse train and the PWM outputs from an OPA561 power amplifier/line driver are configured as a low pass filter to remove unwanted harmonics.

3.3 Transmit Hardware Description

See [Section 2.3](#) for information regarding this description.

3.4 Receiver Hardware Description

3.4.1 Signal Selection Circuits

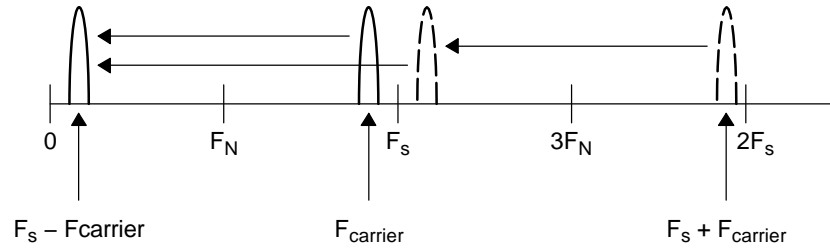
The following steps describe how to convert sine-wave signals to a specific frequency.

First, define a digital communication system where the carrier frequency is 132 kHz and each transmitted bit consists of 24 cycles of a sine wave at the carrier frequency. Then the phase of each bit field can be set to 0 or 180 degrees to encode the bit information.

By sampling the received signal at 25/24ths of the carrier frequency or 137.5 kHz, you can "down-convert" the signal from 132 kHz to the bit rate of 5.5 kHz. One way to look at how this works is to think of the ADC as mixing or multiplying the incoming carrier sinewave signal with the sample rate clock. In trigonometry, the result of multiplying two sine wave signals will be a signal containing the sum and difference of the frequency of the two sine waves. [Figure 3-3](#) shows that because of the way the frequencies "fold" around the sample rate and the Nyquist frequency both the sum and difference end up at $F_s - F_{\text{carrier}}$ or 5.5 kHz.

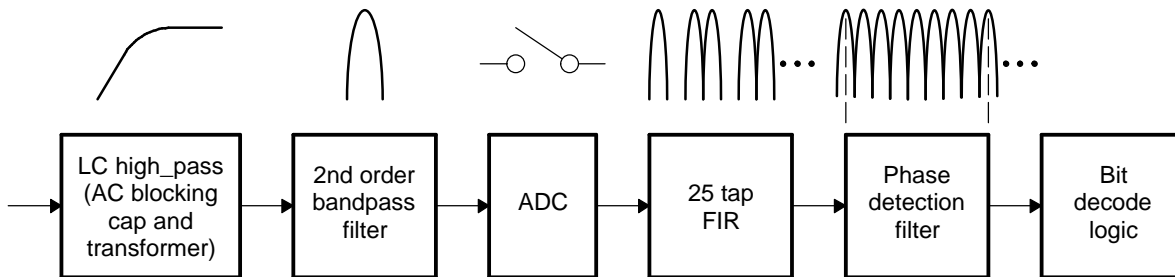
Note that if the traditional sampling theory is used, the signal must be sampled at greater than $2\sqrt{2} F_c$ or 373 kHz, or 2.7 times the rate used in this system.

Figure 3-3. Sampling Effect



The resulting digital values from the undersampled ADC have a frequency content of 5.5 kHz plus the additive noise from the communication channel. To detect the transmitted signal, you must detect the 5.5 kHz content in the discrete received signal values. This is done by first applying a narrow band analog filter to the received signal, down-sampling the signal in the ADC, and then digitally filtering the signal with a finite impulse response (FIR) filter. [Figure 3-4](#) depicts the process.

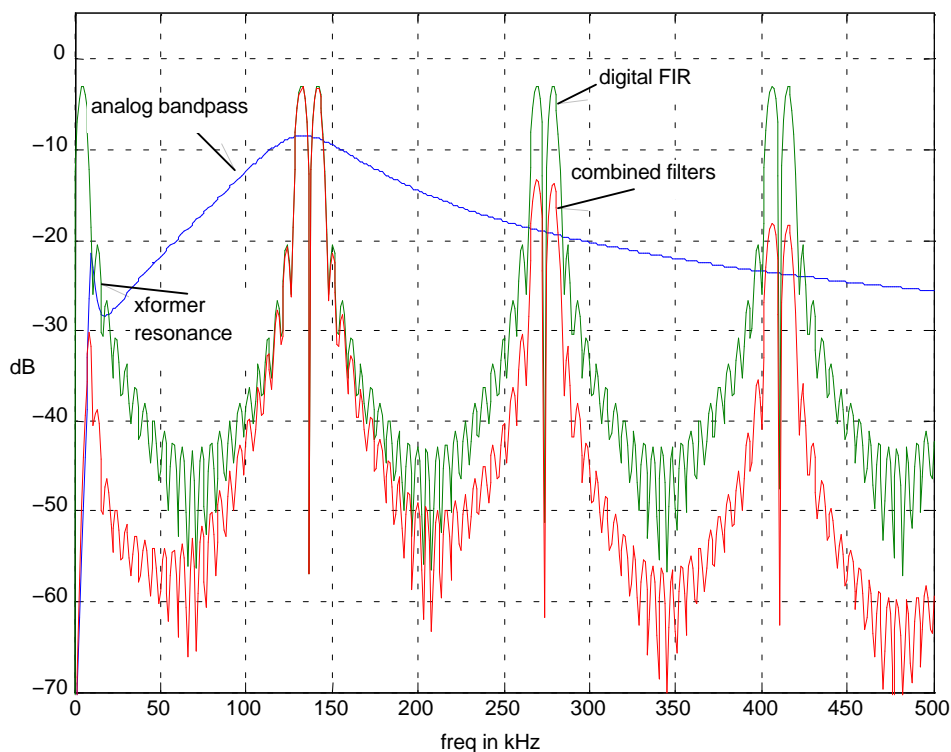
Figure 3-4. Receiver Signal Processing



To get the maximum sensitivity out of the FIR filter, it should have as many taps as possible. Since there are 25 samples in each bit field time, the FIR is defined to have 25 taps. The maximum response from the filter will be when the tap weights match the signal you are trying to detect. In this case, the tap weights are defined by a sine wave with a frequency of $F_s/25$.

$$tap_n = \sin\left(2\pi \frac{n}{25}\right) \text{ where } [0... 24] \tag{3-1}$$

Using the tap weights in [Equation 3-1](#) the response of the system can be calculated for input signals with frequencies from 0 to 500 kHz. Note that the FIR filter will respond to signals with frequencies greater than F_N by aliasing those signals to the $0..F_N$ baseband. Because of this, you need to attenuate signals at other than the carrier frequency $F_{carrier} = 132$ kHz. [Figure 3-5](#) shows the response of the analog portion of the system, the digital FIR response, and the combined system response where the analog circuits attenuate signals that the FIR would otherwise pass through the system.

Figure 3-5. Response of Analog and Digital Filters in Receiver


3.4.2 Signal Detection Logic

Once a sufficient amount of noise from the received signal has been removed, you can detect the message data encoded in the signal. To do this, you need to perform several operations:

1. Detect the presence of the carrier
2. Detect the bit field boundaries
3. Detect the word sync pattern
4. Detect each data bit
5. Detect the parity bit and suffix bits
6. Detect the end-of-packet (EOP) pattern

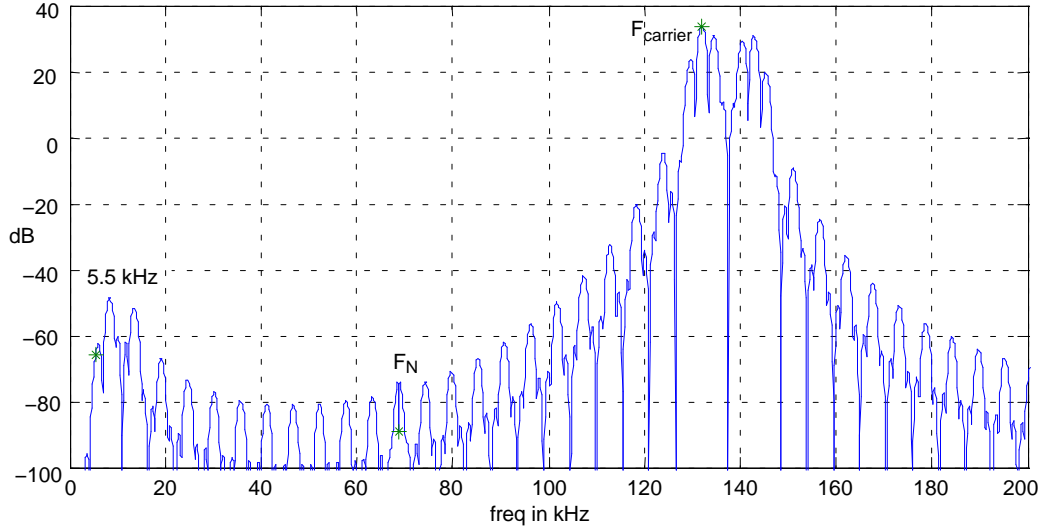
3.4.2.1 Detection Method Overview

The receiver detection logic is clocked at very nearly the same frequency as the remote transmitter. However, the two clocks are not exactly at the same frequency and the phase between the two clocks is not synchronized. Because of this, the receiver uses a differential phase detection technique to encode the transmitted signal and decode the received signal.

The phase of each bit field is determined by applying the output of the FIR filter to a correlation filter. The correlation filter is a FIR structure that uses the previous 25 samples as the tap weights for the current 25 samples. The output of the correlation filter indicates the differential phase of each bit field. If the phase of the current bit field is the same as the previous bit field, the output of the correlation filter will be positive. If the phase of the previous bit field is different, the output will be negative.

The net result of the analog filtering, the FIR digital filtering and the correlation filter is a system bandwidth that is shown in [Figure 3-6](#). There is more than 80 dB of attenuation of interference and noise below 100 kHz, which is the noisiest part of a spectrum on a typical power line.

Figure 3-6. System Frequency Response After the Correlation Filter



3.4.2.1.1 Detecting the Presence of the Carrier

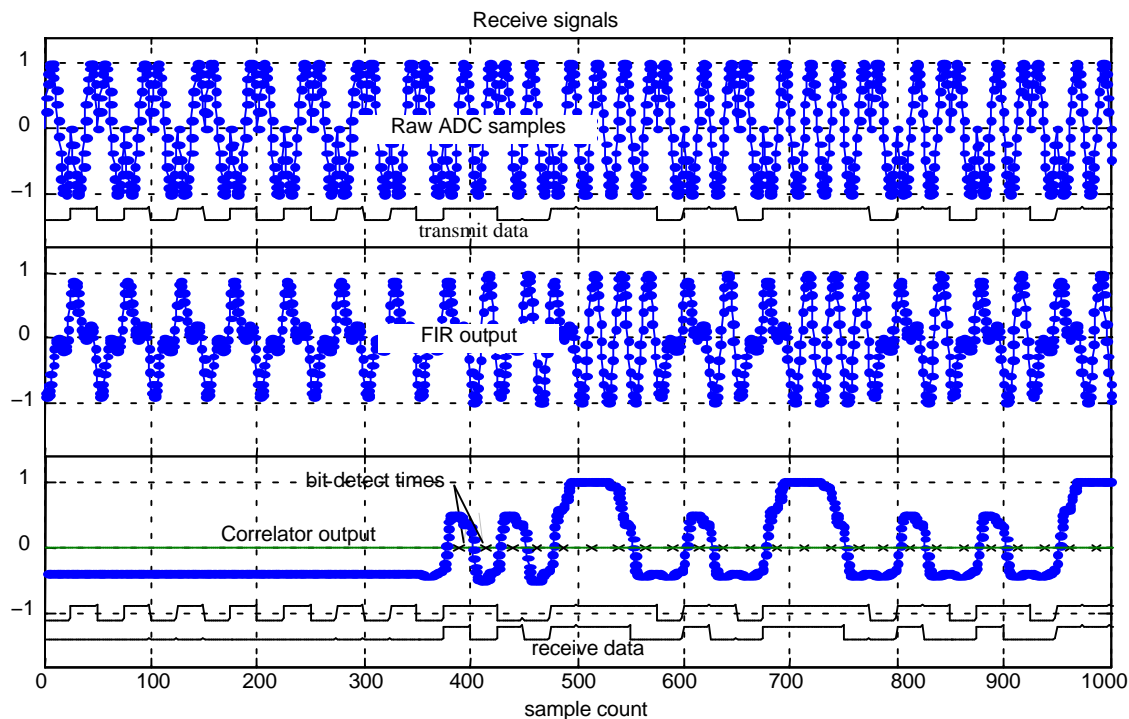
The preamble portion of the transmitted packet contains an alternating phase for each bit field. This will look like a string of all negative outputs from the correlation filter (see the bottom plot in [Figure 3-7](#)). The packet detection logic keeps track of the number of consecutive negative outputs from the correlation filter. When this number is greater than a threshold, the presence of a packet is declared.

3.4.2.1.2 Detecting the Bit Field Boundaries

Once the bit sync field has been detected, the bit window timing is determined by looking for the correlator output to go from negative to positive. The center of the bit window will be the sample that is 1/2 of the 25 sample bit time or 12 samples from this point.

3.4.2.1.3 Detecting the Word Sync Pattern

The CEA-709 spec defines the word sync pattern as [1 1 0 0 1 1 1 1 0 1 1]. The output of the correlation filter is a sample on an interval of 25 samples starting with the bit window center defined in the previous step. A negative value means that the phase of the carrier in the current bit window is different from the phase of the carrier in the previous bit window, and is decoded as a 1. A positive output from the correlation filter means the phase of the carrier in the current bit window is the same as the phase in the previous bit window, and is decoded as a 0. The decoded bits are shifted into a memory buffer. When the 11-bit word sync pattern is detected, we know that what follows are a series of 11-bit data words.

Figure 3-7. Data Detection Waveforms


3.4.2.1.4 Detecting the Data Words

At the transmitter, the message data is encoded so that after the differential phase process in the receiver, the original data is recovered. This encoding is done at the transmitter because it is a cumulative mathematical process to "undo" the differential phase detection. That is, the process of calculating the original state of a received bit after doing differential phase detection is dependent on all the previously transmitted bits. So, if any one of those bits were in error, every bit after that point will be in error as well.

We can get around this shortcoming by encoding the data at the transmitter, where, since noise has not been introduced, there is no chance of making an error. The data is encoded so that the output of the correlation filter, which measures differential phase, produces the original message data directly. [Example 3-1](#) show this algorithm.

To "precode" the transmit data to compensate finding the differential phase in the receiver, you need to sum the bits. For binary data, summing is performed by doing an exclusive OR

Example 3-1. Transmit Encode Algorithm (Simplified)

```

u16 encodeWord(u16 msgWord)
{
    u16 b, bit;
    u16 parity = 0;
    u16 encWord = 0;
    u16 prevBit = 1;           // all words end in a '1'
    for( b = 7; b >= 0; b--)
    {
        // do cumulative phase encoding
        bit = (msgWord & 0x07) >> 7;
        msgWord = msgWord << 1;
        bit = bit ^ prevBit;
        encWord |= bit << b;
        prevBit = bit;
        parity ^= bit;
    }
}
    
```

Example 3-1. Transmit Encode Algorithm (Simplified) (continued)

```

encWord &= (parity<<2) + 0x01;    // assign parity & postfix bits
return encWord;
  
```

Note that each data word ends in a '1' so the transmit encoder is not dependent on the previous word. This allows the encoder algorithm to be implemented as a simple lookup table

```

*encdata++ = uTxPrecodeTable[*msg++];    // 8 bits of user data in, 11 bits of pre-coded data out
  
```

3.4.2.1.5 Detecting the Data Words

Each 11-bit data word consists of a 8-bit encoded data byte followed by a postfix consisting of a parity bit and the suffix '0 1'

| | | | | | | | | | | |
|----------------|-------|-------|-------|-------|-------|-------|-------|--------|---|---|
| Bit 7 (MSB) | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 | Parity | 0 | 1 |
|----------------|-------|-------|-------|-------|-------|-------|-------|--------|---|---|

At the receiver, the parity and postfix are checked and stripped off before the data is shifted into the receive memory buffer.

3.4.2.1.6 Detecting the End of Packet (EOP) Words

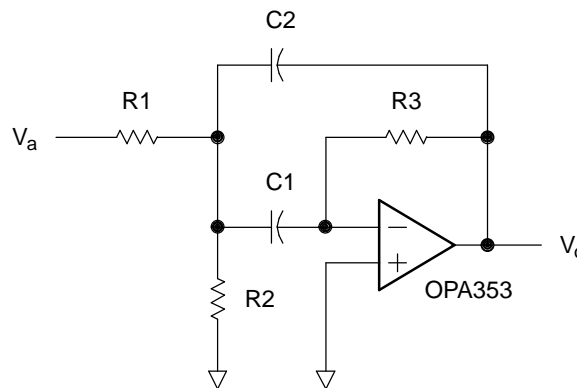
The CEA-709 spec specifies that the transmitted packet contains two EOP words. These 11-bit words are defined as [1 1 1 0 0 1 1 0 0 1 1]. Note that the word sync pattern and the EOP pattern end in a [1 1], whereas the data always ends in a [0 1]. In this way, a data sequence should not be confused as the word sync or EOP pattern. This is only true when the data received is synchronized to the correct word boundaries and is a significant shortcoming of the CEA-709 spec.

Once the data for a packet is detected, it is passed to the MAC layer in the communication protocol.

3.4.3 Analog Bandpass Filter Circuit

Figure 3-8 shows the 2nd order bandpass filter used to attenuate out-of-band frequency content in the received signal. The amplifier is an OPA353 low-noise amplifier. The equations used to select component values follow.

Figure 3-8. Multiple Feedback Bandpass Amplifier



Define the resistive and capacitive components as a function of R and C where $R_1 = R$, $R_2 = R/(a - 1)$, $R_3 = gR$, $C_1 = C$, $C_2 = C$ then

$$V_{out} = \frac{A}{Q} \frac{\omega_0 S V_a}{s^2 + \frac{\omega_0}{Q} s + \omega_0^2}$$

where $\omega_0 = \sqrt{\frac{a}{g}} \frac{1}{RC}$ and $Q = \frac{1}{2} \sqrt{ag}$

At the center frequency ω_0 the output amplitude is

$$V_{out}(j\omega_0) = -\frac{1}{2} g V_a = -A V_a$$

Then we can solve for the resistor ratios a and g in terms of the peak amplifier gain A and the desired Q :

$$g = 2A$$

$$a = \frac{2Q^2}{A}$$

Finally, choosing F_c , A , Q , and C we can calculate the resistor component values:

$$R_1 = \frac{Q}{A} \frac{1}{C\omega_0}$$

$$R_2 = \frac{Q}{2Q^2 - A} \frac{1}{C\omega_0}$$

$$R_3 = 2Q \frac{1}{C\omega_0}$$

Table 3-2. Receive Amplifier Component Values

| | |
|-------|-----------|
| F_c | 135.2 kHz |
| Q | 2.06 |
| A | 1.0 |
| R_1 | 35.7 k |
| R_2 | 4.75 k |
| R_3 | 71.5 k |
| C_1 | 68 p |
| C_2 | 68 p |

3.5 DSP Processor Utilization

The table below provides details about the functions, registers, and pins of the F2812 controller hardware.

Table 3-3. F2812 Controller Hardware Utilization

| Function | Registers | Pins |
|---------------------------------|-----------|---------|
| USED IN REAL APPLICATION | | |
| PLC Tx Waveform | | |
| Tx+ output | CMP1 | PWM1 |
| Tx- output | CMP2 | PWM3 |
| Polarity control | ACTRA | |
| Period register | T1PR | |
| Tx Bias Enable | | GPIOA11 |
| ADC Rx Input | | |
| ADC Input | ADCIN6 | ADCINA6 |
| Period register | T2PR | |

Table 3-3. F2812 Controller Hardware Utilization (continued)

| Function | Registers | Pins |
|-------------------------------------|-----------|---------|
| Used Only for the Demo Board | | |
| LED Intensity | | |
| PWM output | CMP5 | PWM9 |
| Period register | T3PR | |
| RS-232 Interface | | |
| Serial Data Tx | | SCITXDA |
| Serial Data Rx | | SCIRXDA |
| RTS | | GPIOF0 |
| CTS | | GPIOF1 |
| Status LEDs | | |
| Rx Good | | GPIOA8 |
| Rx Busy | | GPIOA9 |
| Tx | | GPIOA10 |
| Option Jumpers | | |
| Opt1 | | GPIOB9 |
| Opt0 | | GPIOB10 |
| Opt2 | | GPIOB11 |
| Opt3 | | GPIOB12 |

3.6 Auxiliary Circuits on the PLC Modem Platform

3.6.1 RS-232 Interface

The daughterboard contains a 9-pin D-shell connector and a SN75LV4737 transceiver. RS-232 signals RD, TD, RTS, and CTS are provided at the connector in the standard configuration for a modem. This means that a standard, non-crossover serial port connector can be used to interface from a PC to the PLC modem platform.

The D203 and D201 LEDs are also provided on the board. D203 is a red LED and indicates activity on the RD line, which is used to transfer data from the modem to the PC. D201 is a green LED and indicates activity on the TD, which is used to transfer data from the PC to the modem. [Table 3-4](#) provides more detail for the RS-232 connector.

Table 3-4. RS-232 Connector Usage

| Pin | Modem Schematic | PC Signal Name |
|-----|-----------------|----------------|
| 1 | - | CD |
| 2 | TX | RD |
| 3 | RX | TD |
| 4 | - | DTR |
| 5 | GND | GND |
| 6 | - | DSR |
| 7 | CTS | RTS |
| 8 | RTS | CTS |
| 9 | - | RI |

3.6.2 Status LEDs

A red, a yellow, and a green LED are also provided to display status from the DSP. They are driven by a SN74LVT240 buffer and connect to the DSP GPIO pins through the I/O interface connector, P200.

The red LED is lit when the modem is transmitting a packet on the power line. The yellow LED is lit when the modem is receiving a packet on the power line. The green LED is lit and stays lit for a programmable period after a packet has been received and the 16-bit CRC indicates that the packet was received error-free.

3.6.3 White LEDs

As a demonstration of processing a command received over the power line or over the RS-232 PC connection, the daughterboard contains a white LED. The intensity of the white LED can be controlled through one of the PWM channels on the DSP. It is connected to the PWM ch-9 output through pin 32 on the I/O interface connector.

3.6.4 Power Supplies

Both a 5V and a 12V supplier are required for the modem demonstration. The 5V supply furnishes power for the eZdsp board. The eZdsp controller in turn supplies 5V power to the daughterboard through the I/O interface connector. The 12V supply furnishes power for the transmit amplifier on the daughterboard. Both connectors accept a 5.5mm power connector.

3.6.5 Stack Headers

Figure 3-9 shows the pin orientation for the pins on the analog interface board that connect to the eZdsp board.

Figure 3-9. ezDSP Stack Header Pin Definitions

| I/O Interface | | | Analog Interface | | | OPT jumpers |
|---------------|-----------|----|------------------|-----------|----|-------------|
| 2 3... | P203 (P4) | 20 | 2 3... | P202 (P5) | 10 | 2 4 6 8 |
| 2 4 6... | P200 (P8) | 40 | 2 4 6... | P201 (P9) | 20 | 3 5 7 |
| 3 5... | | 39 | 3 5... | | 19 | |
| 2 3... | P204 (P7) | 10 | | | | |
| PLC_RX_GOOD | P200.6 | | UART_RX | P200.4 | | |
| PLC_RX_BUSY | P200.7 | | UART_TX | P200.3 | | |
| PLC_TX | P200.8 | | UART_CTS | P200.24 | | |
| PWM1, PWM3 | P200.9,11 | | UART_RTS | P200.23 | | |
| TX_ENABLE | P200.17 | | | | | |
| ADCIN_A6 | P201.14 | | | | | |

Single Frequency Power Line Communication (SFPLC) Software Architecture

This chapter describes the software architecture and design of the single-frequency power-line modem demonstration platform. Each demonstration unit consists of an eZdsp controller developer's board plus a daughtercard. The daughtercard contains electronics that shape and amplify the power-line communications (PLC) transmit signal, filter the PLC receive signal, and manage RS-232 serial port communication, status lights, and a white LED. The intensity of the white LED may be controlled by issuing commands via the PLC or the RS-232.

The target processor for the software is the F28x generation family of fixed-point DSP controllers.

| Topic | Page |
|---|------|
| 4.1 System Overview | 40 |
| 4.2 Modem Software (TMS320F2812 Controller) | 41 |
| 4.3 DSP Command Interface | 48 |
| 4.4 Status Reporting | 52 |
| 4.5 Lamp Control | 53 |
| 4.6 DSP Processor Utilization | 54 |
| 4.7 DPSK Sytem Software Functions | 57 |
| 4.8 Software Compile-Time Options | 63 |

4.1 System Overview

The single-frequency power-line modem is designed to demonstrate the use of single-frequency binary phase-shift keying in a power-line modem to control a remote device. In the demonstration software, the remote device is a dimmable lamp fixture. See [Figure 1-1](#).

4.1.1 PLC Timing and Frequency

Two signal processing systems have been developed: a differential phase shift key (DPSK) system that uses a two-stage FIR filter structure to detect received data, and a phase-shift key (PSK) system that uses a digital phase lock loop (D-PLL) structure to detect received data. The DPSK system uses a sample rate of 25/24ths of the carrier frequency. The PSK system uses a sample rate of 21/24ths of the carrier frequency.

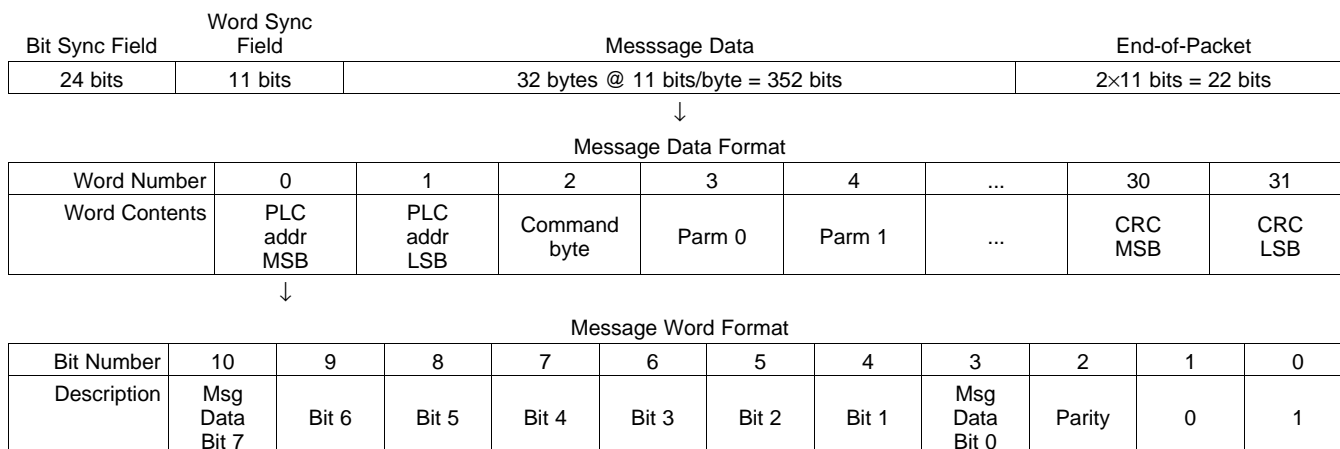
Table 4-1. DPSK System Timing

| | Frequency in MHz | Time in μ sec | Clock Ticks @ 150 MIPS |
|----------------------------|------------------|-------------------|---------------------------|
| Transmit Carrier Frequency | 131.579 | 7.600 | 1140 |
| Receive Sample Rate | 137.112 | 7.293 | 1094 |
| Bit time | 5.482 | 182.4 | 27360 |
| Byte time | 0.498 | 2006.4 | 300960 |

Table 4-2. PSK System Timing

| | Frequency in MHz | Time in μ sec | Clock Ticks @ 150 MIPs |
|----------------------------|------------------|-------------------|---------------------------|
| Transmit Carrier Frequency | 131.579 | 7.600 | 1140 |
| Receive Sample Rate | 115.119 | 8.687 | 1303 |
| Bit time | 5.482 | 182.4 | 27363 |
| Byte time | 0.498 | 2006.6 | 300993 |

4.1.2 Message Composition



| | |
|-----------------|-------------------------------|
| Bit Sync Field | 1010 1010 1010 1010 1010 1010 |
| Word Sync Field | 1100 1111 011 |
| EOP Field | 1110 0110 011 |
| data | xxxx xxxx P01 |

Total packet time = 24 + 11 + 32×11 + 2×11 = 409 bits = 74.602 msec

4.2 Modem Software (TMS320F2812 Controller)

4.2.1 System Initialization

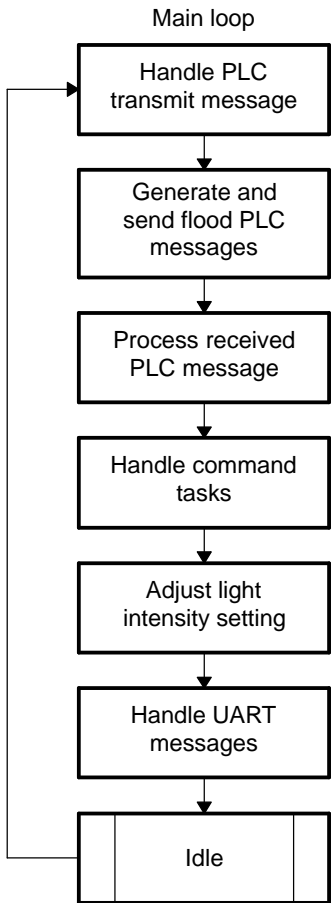
Before entering the main loop, the main() function initializes the hardware and software. Follow these steps:

1. Disable interrupts; clear all interrupt flag registers
2. Initialize system control registers, PLL, watchdog, clocks to default state
3. Set each GPIO pin to operate as either general-purpose I/O or to perform a specific dedicated function.
4. Initialize the peripheral interrupt expansion (PIE) control registers and the interrupt service routine table.
5. If executing from FLASH, copy certain program functions from FLASH to RAM for faster execution.
6. Configure all peripherals:
 - external interface
 - CPU Timers - generate period interrupts
 - serial communications interface (SCI)
 - analog-to-digital converters (ADC)
 - event manager timers - controls period of PWM transmit waveform and receiver ADC sampling rate
7. Initialize variables
 - global variables
 - CRC Table
 - UART message arrays
 - diagnostic trace buffer
 - lamp control variables
8. Read the device address from the jumpers.
9. Arm the ADC to start converting input signals when triggered by the Event Manager.
10. Enable interrupts.
11. Enter the MainLoop() function, which does not return.

4.2.2 Main Loop

The main loop is run once per ADC interrupt. In each pass through the loop, the functions listed in [Table 4-3](#) are performed. Each function in the main loop is intended to perform its function (or a part of it) quickly and then exit. A state variable or some other flag is set for functions which require more processing than is reasonable in a single pass through the loop, so that processing continues where it left off.

Table 4-3. Main Loop

| | Function Name | Operation |
|---|---------------------------|--|
|  <p>Main loop</p> | GenerateFloodPLCMessage() | If the PLC bus has been idle long enough, this function builds a message packet that can be used to load the PLC bus with traffic. Useful for system debug, but should be turned off in a field-deployed system. |
| | FillTxBuffer() | Pending transmit messages, whether they are real or fake, are formatted and copied from the TXUserDataArray to the TxDataArray. |
| | ProcessRxPlcMsg() | Recently completed receive message packets are processed to look for errors and passed to the command handler if necessary. |
| | TaskCommand() | Handle active commands. |
| | ControlLamp() | Adjust lamp intensity setting. |
| | HandleUART() | Handle incoming/outgoing serial data. |
| | IDLE | Drops the processor into a low-power mode until an interrupt wakes it up. |

4.2.3 Interrupts

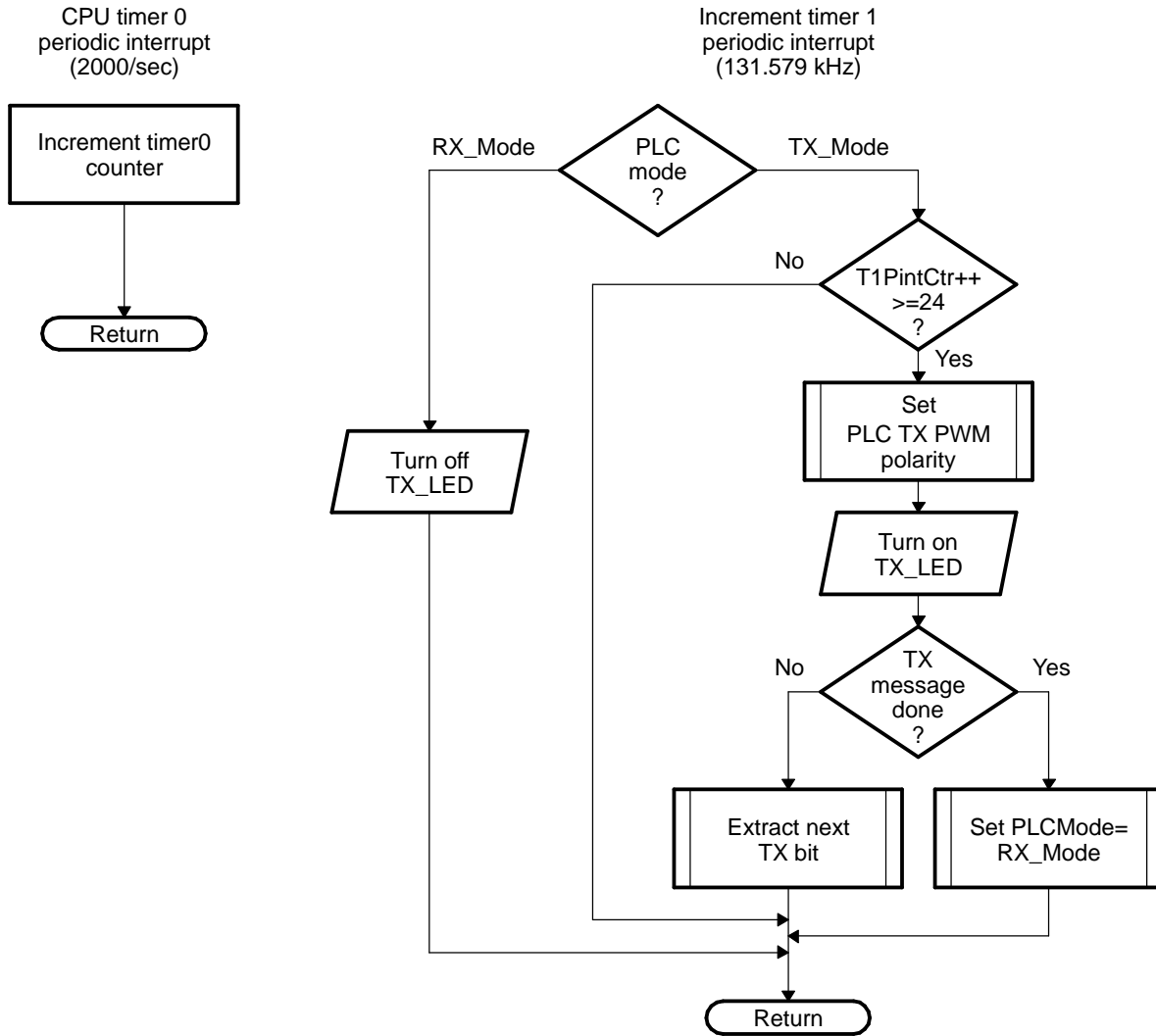
The following are interrupts of the modem software:

- ISRTimer0() - Implements the main system periodic timer. Occurs twice per 1000 Hz control period.
- ADCINT_ISR () - Handle incoming samples from the A/D converter. Runs at 137.061 kHz for the DPSK system and 115.119 kHz for the PSK system.
- T1PINT_ISR - Controls polarity of transmit waveform. Runs once per transmit period of 131.579 kHz.

4.2.3.1 Periodic Interrupts

The CpuTimer0 timer in the DSP is configured to generate periodic interrupts 2000 times per second. The interrupt service routine (ISR) for this interrupt is very short. It increments a counter (CpuTimer0.InterruptCount) and returns. This counter can be used with the ElapsedTime() function to control timing intervals. These interrupts are shown in the diagram in [Figure 4-1](#).

Figure 4-1. Transmit ISR Flow Diagram



4.2.3.2 Transmit PWM and Timer Configuration

The PLC transmitter uses one hardware Event Manager in the F28x controller to generate a clock at 131.579 kHz (2.5 MHz / 19) running in continuous up/down counting mode. This clock is fed to two full-compare units. One compare unit is set to generate an output with 33.3% duty cycle, the other is set for 66.6% duty cycle. The outputs of the two compare units are summed together by the analog circuitry on the power line driver amplifier. Once configured at start-up, the timer and the compare units are not touched. By adding these two signals together and then filtering the sum at the transmit line driver, a 131.579 kHz sinewave carrier signal is produced.

To modulate the carrier, the polarity of the signal is set once per bit time by modifying the Compare Action Control Register (ACTRA). In this manner, the clock remains aligned throughout the transmit cycle with no discontinuities. The polarity modulation can be performed as a separate interrupt minor timing variations in the polarity modulation will have little effect on the transmitted signal quality.

On systems with limited resources, the PLC transmit function could be accomplished using a single general-purpose timer PWM output. In this case the period is still set to 131.579 kHz but the duty cycle is fixed at 50%. The timer could be configured to run in up/down, up, or down-counting mode. Polarity modulation requires modifying the GPTCONA register once per bit time.

The dual PWM implementation is preferred over the single PWM because it generates less out-of-band interference. The duty cycles on the dual-PWM implementations were chosen so that the third harmonic of the carrier frequency is essentially eliminated and higher harmonics are somewhat attenuated.

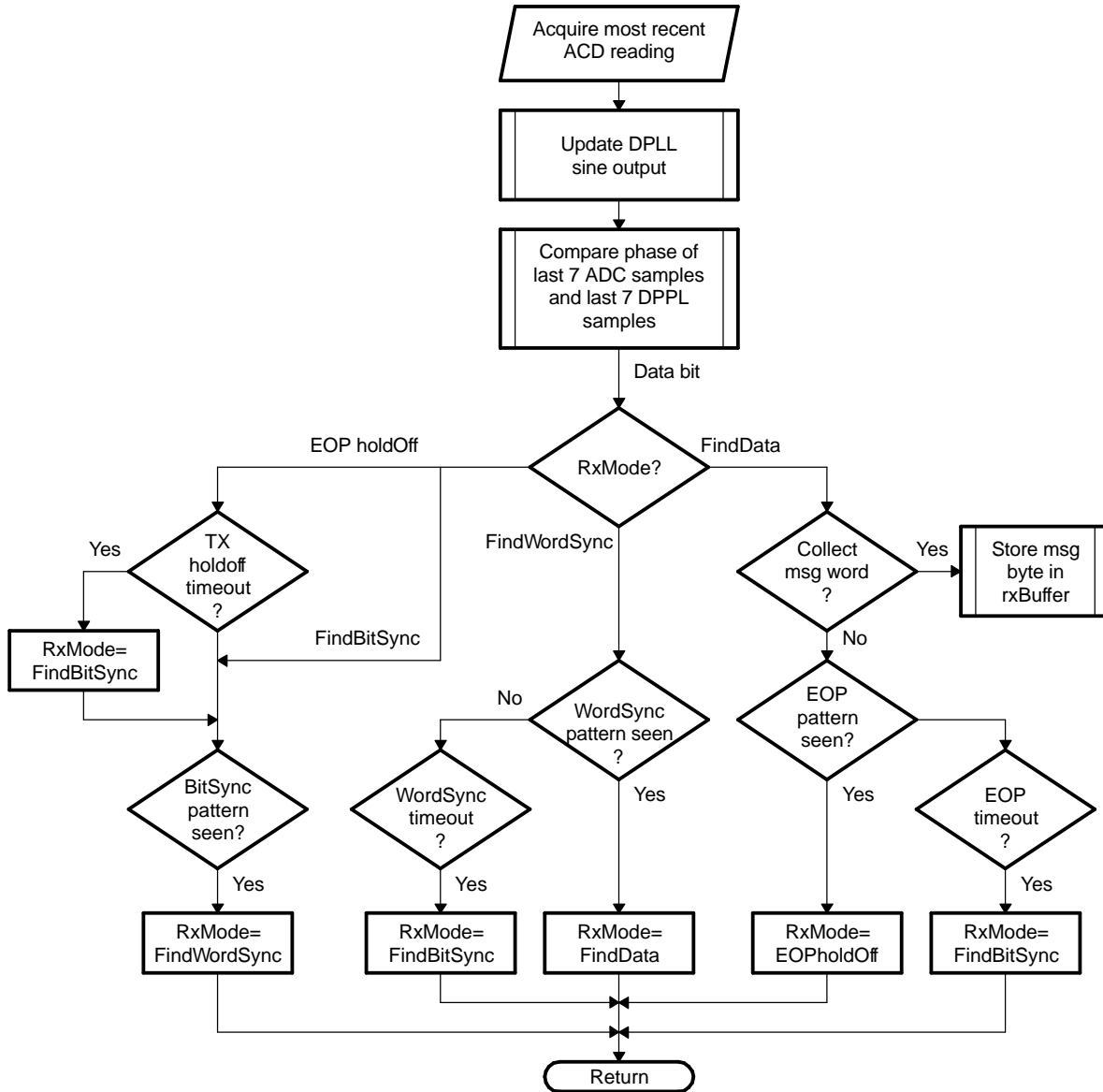
4.2.3.3 Receive Timer Configuration

The receive function samples the transmitted signal at a sample rate that is far below the Nyquist minimum sampling rate of 2x the carrier frequency. This undersampling technique generates intermediate frequency that is used to detect the transmitted message data. It is critical that the receive clock period be very consistent during the entire time that a message packet is being received.

4.2.3.3.1 PSK System Timing

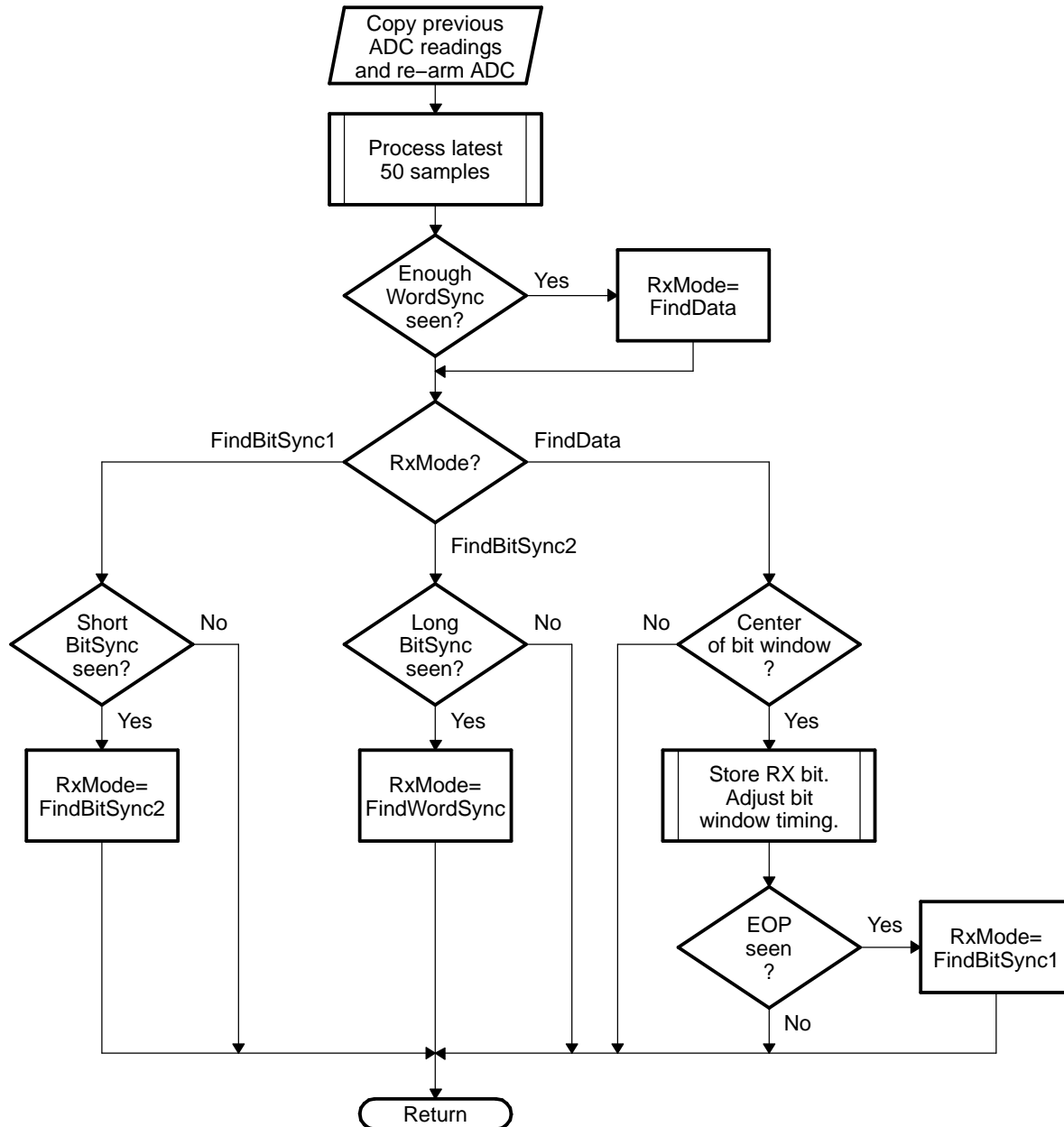
For the PSK system the A/D converter is driven by the receive timer which runs at 0.875 times the transmit carrier frequency. In this manner, 21 receive waveform samples will be collected for each 24-cycle transmit bit time. These 21 samples will look like three cycles of a 16.447 kHz sinewave. Each sample is compared to a synthesized sinewave from a digital phase lock loop (PLL). By synchronizing the PLL signal to the receive samples the phase between the two signals is used to estimate the transmitted data. The flow diagram in [Figure 4-2](#) illustrates PSK system timing.

Figure 4-2. PSK Receive ISR Flow Diagram



4.2.3.3.2 DPSK System Timing

The A/D converter is driven by the receive timer that runs at a fixed frequency which is 1.04167 times the transmit carrier frequency. In this manner, 25 receive waveform samples will be collected for each 24-cycle transmit bit time. The 25 samples will look like a single period of a 5.482 kHz sinewave. These 25 samples are fed into a 25 tap 5.482 kHz FIR filter, by a 25 tap correlation filter to estimate the transmitted data bits. The flow diagram in [Figure 4-3](#) illustrates DPSK system timing.

Figure 4-3. DPSK Receive ISR Flow Diagram


4.2.3.4 Receiver Signal Processing

The ADC interrupt is triggered by the completion of the A/D conversion process. The interrupt is handled by the ADCINT_ISR() function. This ISR function reads the most recent ADC value and then calls the data detection routines. The routine to collect the ADC readings is called SmoothADCResults(). After a receive sample is returned, ArmAllSensors() is called to re-arm the A/D converters to await triggering for the next sample time.

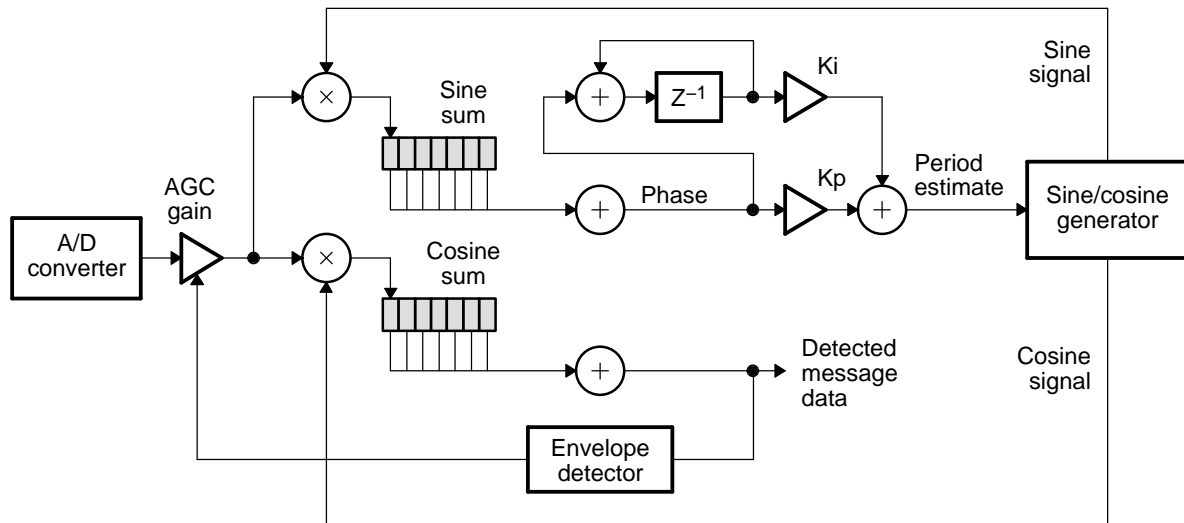
4.2.3.4.1 PSK Signal Processing

The function runPLL() is the heart of the PSK processing. The process is as follows:

1. Multiply the incoming ADC sample value by an automatic gain control (AGC) state variable.
2. Calculate a new synthesized sinewave sample value based on the current IF time-period estimate.

3. Multiply the normalized ADC sample with the latest synthesized sine and cosine values, and the digital PLL and place the results into a circular buffer. Then sum the contents of the buffers to produce a complex estimate of the phase between the PLL sine wave and the received signal. Since the data is modulated at the transmitter as a carrier phase of either zero or 180 degrees, the normalized cosine phase values should be either +1 or -1 and the sine phase values should be near zero when the synthesized and received signals are locked. Therefore, the sine phase values represent the PLL phase error and are used to generate the period estimate for the sinewave generator. This closes the loop on the PLL.
4. The cosine phase value is returned from runPLL() and used to estimate the message data transmitted. [Figure 4-4](#) illustrates this process.

Figure 4-4. PSK Signal Processing



4.2.3.4.2 DPSK Signal Processing

The function FilterCorrRx() is the heart of the DPSK processing. During every sample, it does the following:

- Uses the FIR2() assembly function to filter the most recent ADC results through a matched-response FIR filter.
- Uses the AutoCorr() assembly function to correlate the most recent period of the FIR filter output with the previous period. Positive correlator outputs indicate that the present bit matches the previous bit. Negative correlator outputs indicate a bit change between the current and previous bit.
- Once the receiver timing is properly synchronized to the transmitter, this step will only need to be performed once per bit time, every 25 samples. Until the timing is synchronized, this step must be performed more frequently. In the demonstration software it is run every sample time to aid in diagnostics and debugging.

Later activity depends on which portion of the packet is being received:

- During the FIND_BITS SYNC portion, the correlator output is examined for a long string of mostly negative values, indicating the presence of the bit sync field.
- During the FIND_ZEROCROSS portion, a sustained switch to positive correlator output values indicates that the word sync field may be arriving soon.
- During the FIND_WORDS SYNC portion, the recently received data is compared to the known word sync pattern. The sample number in which a valid pattern match is first seen is recorded. When the valid pattern match is no longer seen, the sample number is also recorded. The center of the bit timing window is then set to be midway between the two recorded sample numbers.
- During the FIND_DATA portion, the received signal at the center of the bit timing window is stored in the rxDataArray buffer by the StoreRxBit() function. After every 11 bits, the rxDataArray value is parsed into data bytes, parity bits, and start/stop bits. The data bytes are then stored in the rxUserDataArray buffer.

DSP Command Interface

- During this time, the end-of-pattern (EOP) field is sought amongst the data using the HammingDistance() function. When the EOP pattern is seen, or when the maximum allowed message length has been exceeded, the message is complete and the software switches back to FIND_BITSYNC mode.

The completed message is then checked for errors using the CRC and the parity bits. Valid messages whose target address matches the receiver's address are parsed and stored in the upCommand buffer which will later be interpreted by the TaskCommand() function in the MainLoop().

4.3 DSP Command Interface

Commands to various devices on the PLC bus can be using the graphical user interface on a host PC, which then sends the command to the PLC modem board via an RS-232 serial interface. The DSP translates the RS-232 message into the proper format and sends it over the PLC bus. The RS-232 serial port operates at 115.2 kbps, with 8 data bits, 1 stop bit, and no parity.

The basic flow of a command is as follows:

- The host (the PC) initiates a command by sending 32 words (64 bytes) of serial data that constitute a command (described below).
- The F2812 controller running on the demonstration platform receives and interprets the serial data, executes the command, and sends back a one word return code (2 bytes), followed by any requested data (all returned data in 16-bit word format also).

The following commands in [Table 4-4](#) are available to configure and monitor the system, as well as exercise most of its functions.

Table 4-4. Command List

| | |
|-------|------------------------------|
| 0001h | Read Memory |
| 0002h | Write Memory |
| 0003h | Read Status |
| 000Ah | Direct Lamp Control |
| 000Bh | Lamp Control |
| 000Ch | Send PLC Message |
| 000Dh | Configure PLC Bus Flooder |
| 000Fh | Get local PLC Address |
| 0020h | Set Echo Acknowledge Address |
| 0021h | Echo Request |
| 0022h | Echo Acknowledge |

Table 4-5. (0001h) Read Memory

| Parm No. | Description | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------|--|--------|--------|--------|--------|--------|--------|-------|-------|--|--|--|--|--|--|--|--|-------|-------|-------|-------|-------|-------|-------|-------|--|--|--|--|--|--|------|--|
| 0 | Command number = 0001h | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 1 | Control flags | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | <table border="1" style="width: 100%; text-align: center;"> <tr> <td>Bit 15</td> <td>Bit 14</td> <td>Bit 13</td> <td>Bit 12</td> <td>Bit 11</td> <td>Bit 10</td> <td>Bit 9</td> <td>Bit 8</td> </tr> <tr> <td style="width: 25px; height: 15px;"></td> <td style="width: 25px; height: 15px;"></td> <td style="width: 25px; height: 15px;"></td> <td style="width: 25px; height: 15px;"></td> <td style="width: 25px; height: 15px;"></td> <td style="width: 25px; height: 15px;"></td> <td style="width: 25px; height: 15px;"></td> <td style="width: 25px; height: 15px;"></td> </tr> <tr> <td>Bit 7</td> <td>Bit 6</td> <td>Bit 5</td> <td>Bit 4</td> <td>Bit 3</td> <td>Bit 2</td> <td>Bit 1</td> <td>Bit 0</td> </tr> <tr> <td style="width: 25px; height: 15px;"></td> <td style="width: 25px; height: 15px;"></td> <td style="width: 25px; height: 15px;"></td> <td style="width: 25px; height: 15px;"></td> <td style="width: 25px; height: 15px;"></td> <td style="width: 25px; height: 15px;"></td> <td style="width: 25px; height: 15px;">MODE</td> <td style="width: 25px; height: 15px;"></td> </tr> </table> | Bit 15 | Bit 14 | Bit 13 | Bit 12 | Bit 11 | Bit 10 | Bit 9 | Bit 8 | | | | | | | | | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 | | | | | | | MODE | |
| Bit 15 | Bit 14 | Bit 13 | Bit 12 | Bit 11 | Bit 10 | Bit 9 | Bit 8 | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | MODE | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 2 | Count | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 3-21 | Address specification list. | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

The Read Memory command is used to view the internal DSP data memory.

Parm 1, the flags register, controls how to read memory.

- MODE 0 = Addresses specified in Parms 3-31.
 1 = Block read. Read starting at address in Parm3.

Parm 2, the count value, specifies the number of DSP memory addresses to be read. Since the DSP is a 16-bit machine, the count specified refers to 16-bit words (not bytes).

If MODE = 0, Count determines the number of words to be read (≤ 29). The desired addresses to be read should be listed in Parms 3 through 31.

If MODE = 1, a block read will be performed beginning at the address specified in Parm3. 'Count' words will be read.

Table 4-6. (0002h) Write Memory

| Parm No. | Description | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------|--|--------|--------|--------|--------|--------|--------|-------|-------|--|--|--|--|--|--|--|--|-------|-------|-------|-------|-------|-------|-------|-------|--|--|--|--|--|--|------|--|
| 0 | Command number = 0002h | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 1 | Control flags | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | <table border="1" style="width: 100%; text-align: center;"> <tr> <td>Bit 15</td> <td>Bit 14</td> <td>Bit 13</td> <td>Bit 12</td> <td>Bit 11</td> <td>Bit 10</td> <td>Bit 9</td> <td>Bit 8</td> </tr> <tr> <td style="height: 15px;"></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td>Bit 7</td> <td>Bit 6</td> <td>Bit 5</td> <td>Bit 4</td> <td>Bit 3</td> <td>Bit 2</td> <td>Bit 1</td> <td>Bit 0</td> </tr> <tr> <td style="height: 15px;"></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td>MODE</td> <td></td> </tr> </table> | Bit 15 | Bit 14 | Bit 13 | Bit 12 | Bit 11 | Bit 10 | Bit 9 | Bit 8 | | | | | | | | | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 | | | | | | | MODE | |
| Bit 15 | Bit 14 | Bit 13 | Bit 12 | Bit 11 | Bit 10 | Bit 9 | Bit 8 | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | MODE | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 2 | Count | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 3-21 | These parms are split into two word pairs: <ul style="list-style-type: none"> • Low word: Address • High word: Data | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

The Write Memory command is used to modify the internal DSP data memory.

- MODE 0 = Address mode
 1 = Block fill mode

In address mode, up to fourteen words may be written per command (exact number specified by Count). The parameters 3-31 are paired into address/data sets.

In block fill mode, an entire address range can be set to the same value (e.g., 0000, FFFF, DEAD,...). Parm3 specifies the starting address to be written to, Parm4 specifies the 16-bit data word to be written, and Count specifies the number of addresses to write. Any size block is allowed.

Table 4-7. (0003h) Read Status

| Parm No. | Description |
|----------|------------------------|
| 0 | Command number = 0002h |
| 1 | Control flags |

DSP Command Interface

This command is used by the GUI program to request status from the modem. The status consists of counters used to measure the bit error rate.

Reset flag 1: Reset (fill with zero) the contents of the status arrays ulPlcStats and ulBerStats.
 else: Place the contents of the ulPlcStats and ulBerStats arrays on the Rs-232 UART FIFO for read by the PC.

The format of data returned is:

| Byte No. | Description |
|----------|-------------|
| 0 | Return code |
| 1 | Count |
| 2-N | ulPlcStats |
| N-M | ulBerStats |

Table 4-8. (000Ah) Direct Lamp Control

| Parm No. | Description |
|----------|------------------------|
| 0 | Command number = 000Ah |
| 1 | Lamp intensity setting |

This command directly sets the lamp intensity setpoint level. The lamp intensity will fade to the new setpoint level at a rate determined by the Fade Time variable. If the command setting is between zero and the lamp minimum level, the intensity will stop at the minimum level. If the command setting is between 254 and the lamp maximum level, the intensity will stop at the maximum level. If the command setting is 0, the lamp will fade to the minimum level, then turn off.

If this command is issued through the RS-232 port, the device attached to the RS-232 will respond, regardless of its address. If this command was passed inside a PLC packet, only the device with the matching address will respond.

Table 4-9. (000Bh) Lamp Control

| Parm No. | Description |
|----------|------------------------|
| 0 | Command number = 000Bh |
| 1 | DALI command number |

This command controls the lamp using one of the special DALI commands, which can be used to set the intensity to a previously memorized setting, adjust fade rate and fade time, memorize settings, or adjust the light intensity.

If this command is issued through the RS-232 port, the device attached to the RS-232 will respond, regardless of its address. If this command is passed inside a PLC packet, only the device with the matching address will respond.

| DALI Command No. | Command Name | Description |
|------------------|---------------------------------------|---|
| 0 | OFF | Turn lamp off instantly. |
| 1 | UP | Fade up for 200 ms if already lit, at rate determined by Fade Rate. |
| 2 | DOWN | Fade down for 200 ms if already lit at rate determined by Fade Rate. Do not turn off. |
| 3 | STEP_UP | Step up one step if already lit |
| 4 | STEP_DOWN | Step down one step if already lit. |
| 5 | RECALL_MAX_LEVEL | Set intensity to MaxLevel, without fading |
| 6 | RECALL_MIN_LEVEL | Set intensity to MinLevel, without fading |
| 7 | STEP_DOWN_AND_OFF | Step down one step. Turn off if already at MinLevel |
| 8 | ON_AND_STEP_UP | Step up one step. If lamp was off, ignite it first. |
| 9-15 | --- RESERVED --- | |
| 16-31 | GO_TO_SCENE x, {x=0..15} | Fade to new scene level. If lamp was off, ignite it first. |
| 32 | RESET | |
| 33 | STORE_ACTUAL_LEVEL_IN_THE_DTR | |
| 34-41 | --- RESERVED --- | |
| 42 | STORE_THE_DTR_AS_MAX_LEVEL | |
| 43 | STORE_THE_DTR_AS_MIN_LEVEL | |
| 44 | STORE_THE_DTR_AS_SYSTEM_FAILURE_LEVEL | |
| 45 | STORE_THE_DTR_AS_POWER_ON_LEVEL | |
| 46 | STORE_THE_DTR_AS_FADE_TIME | |
| 47 | STORE_THE_DTR_AS_FADE_RATE | |
| 48 - 63 | --- RESERVED --- | |
| 64-79 | STORE_THE_DTR_AS_SCENE x, {x=0..15} | |
| 80-95 | REMOVE_FROM_SCENEx, {x=0..15} | |
| 96-111 | ADD_TO_GROUP g, {g = 0..15} | |
| 112-127 | REMOVE_FROM_GROUP g, {g = 0..15} | |
| 128-255 | Additional DALI Commands | Not implemented in demo unit. |

Table 4-10. (000Ch) Send PLC Message

| Parm No. | Description |
|----------|-------------------------|
| 0 | Command number = 000Ch |
| 1 | Target address Hi |
| 2 | Target address Lo |
| 3 | Embedded command number |
| 4 | Parameter 0 |

This command is used to initiate a PLC Bus message to another node. Only the node matching the address will respond.

The embedded command number can be any valid command. Generally, it will be 0x0A: Direct Lamp Control or 0x0B: Lamp Control.

Table 4-11. (000Dh) Configure PLC Bus Flooder

| Parm No. | Description |
|----------|--------------------------------|
| 0 | Command number = 000Dh |
| 1 | Flood Rate: Packets per Second |

The Configure PLC Bus Flooder command is used to control the rate at which fake PLC packets are generated to flood with PC bus with traffic. This is useful for exercising the bus under high-traffic conditions and for testing transmission error rates. It would rarely be used in a field-deployed installation.

Table 4-12. (0020h) Set Echo Acknowledge Address

| Parm No. | Description | | | | | | | | |
|----------|---|----------|-----------|-------|-------|--|--|----------|-----------|
| 0 | Command number = 0020h | | | | | | | | |
| 1 | Control flags | | | | | | | | |
| | <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 33%; text-align: center;">Bit 15-8</td> <td style="width: 33%; text-align: center;">Bit 7-2</td> <td style="width: 10%; text-align: center;">Bit 1</td> <td style="width: 10%; text-align: center;">Bit 0</td> </tr> <tr> <td style="text-align: center;"> </td> <td style="text-align: center;"> </td> <td style="text-align: center;">BER Read</td> <td style="text-align: center;">BER Reset</td> </tr> </table> | Bit 15-8 | Bit 7-2 | Bit 1 | Bit 0 | | | BER Read | BER Reset |
| Bit 15-8 | Bit 7-2 | Bit 1 | Bit 0 | | | | | | |
| | | BER Read | BER Reset | | | | | | |
| 2 | Slave address | | | | | | | | |
| 3-21 | Flood rate (1 to 15 Hz) | | | | | | | | |

This command defines the address to send an Echo command to. Each packet transmitted at the Flood rate contains a Echo Request command which is addressed to the modem defined using this command.

Table 4-13. (0021h) Echo Request

| Parm No. | Description |
|----------|-----------------------------|
| 0 | Command number = 0021h |
| 1 | Sender's address, high byte |
| 2 | Sender's address, low byte |
| 3 | Ack command (0022h) |

When sending this command, set the address (bytes 0 and 1 of the transmit buffer) to the address specified by the Set Echo Acknowledge Address command, followed by the above command format. Increment the Echo Sent element in the BerStats counter array.

When receiving this command, send an Echo Acknowledge (0022h) command on the power line, addressed to the sender. Increment the Echo Request element in the BerStats counter array.

Table 4-14. (0022h) Echo Acknowledge

| Parm No. | Description |
|----------|---|
| 0 | Command number = 0022h |
| 1 | Sender's address, high byte |
| 2 | Sender's address, low byte |
| 3 | Status of original Echo command reception |

When sending this command, set the address (bytes 0 and 1 of the transmit buffer) to the address specified by the last Echo Request command received.

When receiving this command, Increment the Echo Ack element in the BerStats counter array.

4.4 Status Reporting

4.4.1 PLC Statistics

The array ulPlcStats is used to store information about the performance of the PLC communication channel. The statistics array assignments for PSK and DPSK are outlined in [Table 4-15](#) and [Table 4-16](#), respectively.

Table 4-15. PSK Statistics Array Assignments

| Name | Description |
|--------------------|--|
| TX_CNT | Count of packets transmitted. |
| TX_COLLISION | Count of BitSync patterns seen while in transmit mode (Not use when modem enabled to receive its own transmit signal.) |
| RX_CNT | Count of packets received before CRC is checked. |
| RX_GOOD | Count of packets received with good CRC. |
| RX_PREDET_COUNT | Count of BitSync patterns recognized |
| RX_SYNCDET_COUNT | Count of WordSync patterns recognized |
| RX_EOP_COUNT | Count of End-of-Packet (EOP) patterns recognized |
| RX_ERR_WORDSYNC_TO | Count of events timed out looking for WordSync |
| RX_EOP_TIMEOUT | Count of events timed out looking for the EOP pattern |
| RX_MSGLEN_ERROR | Count of events where more bytes were received than can fit in the receive buffer. |
| RX_ERR_CRC | Count of packets where the CRC sent does not match the CRC calculated from the received data |
| RX_ERR_PARITY | Count of words where the sent parity does not match the parity calculated from the received data. |

Table 4-16. DPSK Statistics Array Assignments

| Name | Description |
|--------------------|--|
| TX_CNT | Packets transmitted. |
| TX_COLLISION | Invalid packets seen while transmitting. |
| RX_CNT | Packets that started to be received. |
| RX_GOOD | Packets received successfully. |
| RX_ERR_ZEROCROSS | Packets that did not have a valid zero-crossing between the bit sync and word sync fields. |
| RX_ERR_WORDSYNC_TO | Packets where a valid word sync pattern was not seen shortly after the zero crossing. |
| RX_ERR_WORDSYNC | Packets where the word sync pattern was invalid. |
| RX_ERR_EOP | Packets where the End-Of-Packet signal was not seen. |
| RX_ERR_CRC | Data error detected by Cyclic Redundancy Code. |
| RX_ERR_PARITY | Parity error detected in data field. |
| RX_ERR_PATTERN | Content of flood message did not match known pattern. |
| RX_ERR_MSGLEN | Message was invalid length. |

4.4.2 Status LED Definitions

The PLC daughterboard contains three PLC status LEDs.

| LABEL | COLOR | IO PORT | DESCRIPTION |
|---------|--------|---------|--|
| RX_GOOD | Green | GPIOA8 | The latest PLC packet was received successfully. |
| RX_BUSY | Yellow | GPIOA9 | A PLC packet is being received. |
| TX | Red | GPIOA10 | A PLC packet is being transmitted. |

4.5 Lamp Control

The demonstration platform uses a white LED to simulate the function of a fluorescent lamp ballast. The lamp is controlled by commands to the CmdLamp() function. The commands closely resemble those used in the DALI lighting control standard.

The intensity setpoint and fade rate are set using commands received by the CmdLamp() function. The

commanded fade rate is used to calculate the interval between fading steps. The elapsed time since the last fading step is monitored in the MainLoop() function, and when a new fading step is due, the ControlLamp() function is called. The lamp intensity setting is moved toward the target lamp setpoint at a rate determined by the Fade Rate and/or Fade Time. The ControlLamp() function then calls the SetLampIntensity() function.

The SetLampIntensity() function translates the intensity setpoint level, which ranges from 0 to 254, into a lamp voltage between 0% and 100%. The voltage levels are logarithmically spaced, ranging from a minimum of 0.1% at setting level 1 to 100% at setting level 254. Setting level 0 yields zero output voltage. The duty cycle of the 15-bit PWM DAC hardware is then programmed to generate the proper voltage level. For fluorescent lamps that utilize a fixed duty cycle and adjust the lamp current by frequency control, the PWM DAC period register could be changed instead of the duty cycle register.

Some lamps may be unable to operate properly at very low voltage settings. For these lamps, the DALI standard allows a programmable minimum intensity level. The CmdLamp() and ControlLamp() functions ensure that intensity settings between zero and the minimum are handled properly.

The DALI standard includes provisions for igniting fluorescent lamps. The demonstration platform software includes stub functions to handle the ignition delays.

4.6 DSP Processor Utilization

The following table, [Table 4-17](#), describes the functions, registers, and pins for the F2812 controller hardware utilization.

Table 4-17. F2812 Controller Hardware Utilization

| Function | Registers | Pins |
|---------------------------------|-----------|---------|
| USED IN REAL APPLICATION | | |
| PLC Tx Waveform | | |
| Tx+ output | CMP1 | PWM1 |
| Tx- output | CMP2 | PWM3 |
| Polarity control | ACTRA | |
| Period register | T1PR | |
| Tx Bias Enable | | GPIOA11 |
| ADC Rx Input | | |
| ADC Input | ADCIN6 | ADCINA6 |
| Period register | T2PR | |
| USED ONLY FOR DEMO BOARD | | |
| LED Intensity | | |
| PWM output | CMP5 | PWM9 |
| Period register | T3PR | |
| RS-232 Interface | | |
| Serial Data Tx | | SCITXDA |
| Serial Data Rx | | SCIRXDA |
| RTS | | GPIOF0 |
| CTS | | GPIOF1 |
| Status LEDs | | |
| Rx Good | | GPIOA8 |
| Rx Busy | | GPIOA9 |
| Tx | | GPIOA10 |
| Option Jumpers | | |
| Opt1 | | GPIOB9 |
| Opt0 | | GPIOB10 |
| Opt2 | | GPIOB11 |

Table 4-17. F2812 Controller Hardware Utilization (continued)

| Function | Registers | Pins |
|----------|-----------|---------|
| Opt3 | | GPIOB12 |

4.6.1 PSK System Memory Usage

4.6.1.1 Memory Configuration

The following table provides the information needed for memory configuration for the PSK system.

SaveTrace Enabled, SLOW_BETWEEN_MSGS=True, CORR_LEN=FIR_LEN, TRACE_BUFF_LEN=0x1000

| | Name | Origin | Length | Used | Usage |
|--------------------------|---------------|-------------|--------------|----------|----------------------|
| PAGE 0: | PRAMH0 | 003f8000 | 00002000 | 000012b4 | Program |
| PAGE 1: | RAMM0A | 00000000 | 00000020 | 00000000 | Build Info |
| | RAMM0B | 00000020 | 000003e0 | 000003b0 | Data (mostly .const) |
| | RAMM1 | 00000400 | 00000400 | 00000400 | Stack |
| | ECAN_AMBOX | 00006100 | 00000100 | 00000100 | CRC table |
| | DRAML0 | 00008000 | 00000400 | 000003f6 | General variables |
| | DRAML1 | 00008400 | 00001c00 | 00001a00 | Trace Buffer |
| | FLASHA | 003f6000 | 00001ff6 | 00000000 | |
| | CSM_PWL | 003f7ff8 | 00000008 | 00000008 | |
| | Length | Used | %Used | | |
| Program Memory | 8192 | 4788 | 58% | | |
| Data Memory | 10240 | 3238 | 32% | | |
| Data Mem /w Trace Buffer | 10240 | 9894 | 97% | | |

The demo program currently occupies 4788 words of program RAM and 3238 words of data RAM, for a total of 8026 words of RAM. The trace buffer, used for development diagnostics, is sized to use the remaining data memory.

4.6.1.2 RAM Reduction

The demo code was developed using an eZdsp card with a TM320F2812 device. We are considering reducing the memory usage to enable use of a lower-cost DSP in the TMS320F280x generation. The trace buffer, the F2806 controller becomes a viable option.

| | TMS320F2801 | TMS320F2806 | TMS320F2808 |
|------------------------|-------------|-------------|-------------|
| Frequency (MHz) | 100 | 100 | 100 |
| RAM (Words) | 6K | 10K | 18K |
| Flash (Words) | 16K | 32K | 64K |
| OTP ROM (Words) | 1K | 1K | 1K |
| Boot Loader Available | YES | YES | YES |
| PWM Channels | 8 | 16 | 16 |
| 12-bit A/D (#Channels) | 16 | 16 | 16 |
| Timers | 3/3/3 | 6/6/3 | 6/6/3 |

If most of the program was moved to FLASH memory, it might be possible to squeeze the application into an F2801. The wait states in the FLASH would make the processor too slow to run all the code from FLASH, but about 800 words of time-critical code could be run from RAM at full speed.

The following list include other ideas for shrinking RAM requirements:

- The 256-word CRC table is generated during DSP initialization by the InitCRCtable() function and remains constant thereafter. This could be computed at compile time and stored in FLASH.
- The 256-word Intensity table in command.c is constant and could be moved to FLASH.
- The UART functions are required for the bus master, but would be unneeded in a slave device like ballast. Eliminating the UART functions and their variables would save about 400 words of data memory and 250 words of program memory.
- The 1k stack is far larger than needed; 128 words would probably be sufficient.

With all of these changes and only the time-critical code running from RAM, the RAM requirement could be brought below 4K for the program + data variables.

The program could eliminate about 0.5K of demo-only code, but would probably need to add about 0.5K more to the command handler. About 1K of the PLC code would be common with the ballast control / PFC code.

4.6.2 Processor MIPS Utilization

Table 4-18 shows the DSP workload measured 22Feb 2005 using a TMS320F2812 controller running at 150 MIPS. The measurements are $\pm 0.5\%$ and all code was executing from RAM. Diagnostic traces were disabled.

Table 4-18. DSP MIPS Utilization

| | No Packet (Looking) | | Processing a Packet | |
|-----------------------|---------------------|------|---------------------|------|
| | Percent | MIPS | Percent | MIPS |
| DPSK Algorithm | | | | |
| main loop + INT | 38.50% | 57.8 | 55.50% | 83.3 |
| TX/RX INT only | 30.90% | 46.4 | 48.50% | 72.8 |
| PSK Algorithm | | | | |
| main loop + INT | 41.20% | 61.8 | 36.80% | 55.2 |
| TX/RX INT only | 35.30% | 53.0 | 31.10% | 46.7 |

The PSK algorithm combines the transmit and receive interrupt which makes the idle time (i.e., the non-PLC time), much more consistent. The reason the PSK algorithm consumes more MIPS when looking for a packet than when processing a packet, is that the signal gain adjustment function (AGC) is held off once a packet is found.

The most processor-intensive operation for the DPSK version takes place during the ADC interrupt handler, in the FilterCorrRx() function. At every ADC interrupt time (137.061 kHz), the ADC must be re-armed and the most recent sample must pass through a 25-tap FIR filter. This activity takes about 293 clock ticks, or 40.2 MIPS. Next, the FIR output is run through a 25-tap correlator and a block of decision logic. These functions take 349 clock ticks, or 47.9 MIPS. These functions are executed every sample time when a message is being processed, but is only run every 5th sample time between messages while listening for incoming traffic.

The baseline code in MainLoop() is synchronized with the ADCInt_ISR so that each task is only run every 5th sample (every 7th sample in the PSK system), and none of them run during the same sample as the correlator. This limits both the average and the peak workload. Table 4-19 shows the total time spent doing each task; it includes both the time spent in the ADCINT_ISR and the time spent in MainLoop.

Table 4-19. Task MIPS Load

| ADCIntCntr | Task | Clock Ticks | MIPS |
|------------|------------------|-------------|------|
| 0 | Correlator | 460 | 63.1 |
| 1 | HandleUART | 436 | 59.8 |
| 2 | TaskCommand | 347 | 47.6 |
| 3 | ControlLamp | 411 | 56.4 |
| 4 | GenerateFakeMsgs | 391 | 53.6 |

4.6.3 FLASH Memory

FLASH memory on the F28x devices is limited to about 25 MIPS. Most of the code can run from FLASH to save RAM space, but certain time-critical functions must be run from RAM for speed. These functions include MainLoop, and all the functions called during the ADC interrupt handler: ADCInt_ISR, FilterCorrRx, SmoothADCResults, ArmAllSensors, FIR2, and AutoCorr.

4.7 DPSK System Software Functions

4.7.1 Function Names and Descriptions

Table 4-20 provides the DPSK system software function names and descriptions, and Table 4-21 provides a listing of the function call hierarchy.

Table 4-20. DSP Software Functions

| Function Name | Description |
|------------------------|---|
| main.c | |
| BootCopy | Copies the InitFlash function from FLASH into RAM so that it can adjust the Wait States of the flash. (Don't try to adjust Flash wait states while running from flash) Only used when FLASH is used. |
| ElapsedTime | Returns elapsed time between two events. |
| GenerateFakePLCMessage | Generates a fake message for the PLC to send. This function is required only for development and testing. |
| ISRTimer0 | Implements the main system periodic timer. |
| MainLoop | Executes tasks using infinite loop. |
| main | Performs setup functions, then enters infinite loop to execute tasks. |
| command.c | |
| CmdConfigFlooder | Sets the flood rate at which Fake PLC messages are generated. |
| CmdLamp | Implements the commands that control the lamp intensity and fade rate. The commands used are very similar to the DALI lighting control standard. |
| CmdPLCCommand | Allows a host connected to this board via RS-232 to issue commands over the PLC bus. |
| CmdReadMemory | Reads memory in the DSP and sends value to DSP via UART. |
| CmdWriteMemory | Modifies the DSP memory based on commands received from UART. |
| ControlLamp | Adjusts the intensity of the lamp, moving it toward its setpoint at the proper fade rate. |
| InitLampVars | Initializes the lamp command variables to their reset values. |
| SetLampIntensity | Sets the lamp PWM DAC to set the lamp intensity. |
| TaskCommand | Executes commands and responds to the MCU. This task gets posted after an MCU interrupt specifying a command occurs. This function reads the parms acknowledges the command, then executes the command and responds to the MCU when complete. While primarily intended for commands received from the UART, commands may also be received from the PLC. |
| plc.c | |
| ADCINT_ISR | Services the A/D converter's conversion complete interrupt. The most recently measured samples are filtered and the A/D is armed to be triggered by an Event Manager timer event. |

Table 4-20. DSP Software Functions (continued)

| Function Name | Description |
|----------------------|---|
| ExtractNextTxBit | Extracts the next bit from the transmit message bit string. |
| FillTxBuffer | Fills the transmit buffer using the proper header, user data (with parity and start/stop bits), and trailer. |
| FilterCorrRx | Passes the most recently received ADC samples through a matched-response FIR filter. The filter output is then correlated against a delayed copy of the filter output to identify periodic patterns and phase reversal. This function is called from the ADC interrupt ISR. |
| HammingDistance | Calculates the number of bits that are different between a received signal and a desired pattern. |
| ProcessRxPlcMsg | This function processes the message in the rxUserDataArray looking for parity errors and extracts message contents. |
| SaveTrace | Stores data to the trace buffer. |
| SetPWMPolarity | Sets the polarity of the PWM output. |
| SetPlcMode | Configures the power-line communication mode and sets the proper variables and registers to switch between transmit and receive. |
| SetRxMode | Sets the receive mode used by the power-line communication system and changes the appropriate variables. |
| StoreRxBit | Looks at the samples in the receive buffer to determine which bit value was just received. |
| T1PINT_ISR | Servises the Event Manager Timer 1 Period Interrupt. During transmit, every 24 periods the polarity of the transmit waveform is adjusted. |
| sensor.c | |
| ArmAllSensors | Configures the A/D converters and sequencers to read all the sensors in a rapid burst, triggered by the Event Manager. |
| ConfigureADCs | Powers up the analog circuitry for the ADCs. It should be called during board initialization. |
| ReadAllSensors | Configures the A/D converters and sequencers to read all the sensors in a rapid burst. |
| SmoothSensor | Averages multiple ADC readings into a single value. Optionally, the highest and lowest values may be removed before averaging to eliminate noise spikes. |
| crc.c | |
| CalcCRC | Calculates CRC check bytes. |
| InitCRCtable | Initializes theCRC-16 table |
| dacout.c | |
| WritePWMDAC | Sends a requested value to the PWM DACs. Used to control lamp voltage. |
| diag.c | |
| CmdDiagTraceConfig | Configures the trace buffer. |
| InitDiagTrace | Initializes the diagnostic trace buffer to begin tracing immediately and stop when the trace buffer is full. Initializing the trace buffer in this manner is useful for bringup. |
| RunTrace | Spews diagnostics if appropriate. - NOT USED - |
| gpio.c | |
| InitGpio | Initializes the GPIO registers. The function determines which IO pins should be used for their predefined peripheral output, and which are to be GPIO. |
| sci.c | |
| InitSci | Initializes the SCI ports to a known state. |
| timer.c | |
| ConfigureDog | Disables the WatchDog timer and sets its prescale ratio. |
| ConfigureGPTimers | Configures the GP timer periods. |
| DelayNms | Delays N milliseconds. It locks out interrupts for 1 μ s at a time, so use cautiously in time-critical routines. |
| DelayNus | Delays N microseconds. It locks out interrupts for 1 μ s at a time, so use cautiously in time-critical routines. |
| FeedDog | Resets the WatchDog timer. |
| SleepDog | Disables the WatchDog timer. |
| WakeDog | Enables the WatchDog timer. |

Table 4-20. DSP Software Functions (continued)

| Function Name | Description |
|---------------------------|---|
| uart.c | |
| HandleUART | Sends/receives data to the UART. It is called once each servo period. Because the defined command interface only allows one command to be issued at a time, the flow of UART data consists of a command coming in to the DSP, and then a response going out. A second command can not be sent until the reply from the first is received. So, if there is data to be sent, we don't need to worry about incoming data. Additionally, if we make it into this routine with nothing to go out, since we don't have multiple prioritized tasks to interrupt us, we will not have to worry about sending anything for the rest of the time (we can just sit and wait to see if there's anything to read). |
| InitializeUARTArray | Initializes the UART, so that it comes up with nothing in the buffer to send out. |
| WriteUART | Acts as an interface between the code and the task which sends data to the UART (TaskWriteUART). This routine accepts a count, and a pointer to the start of data. It then inserts an entry into the array UARTDataOut which TaskWriteUART uses to send out data. Note that it will store the count and pointer, not buffer the actual data, so the calling function must keep the variable around (static or global), or else use the function WriteUARTValue(). |
| WriteUARTValue | Sends a single value to the UART. The calling function does not have to worry about memory management. This function takes the passed value, assigns it to a variable, then sends a pointer to that variable along with the destination FPGA address and a count of one, to the WriteUART() function. The "variable" used, is an element in a circular buffer. The buffer is the same length as the UARTDataOut array, so this shouldn't ever have values overwritten (we should get an error from WriteUART() first). |
| vardefs.c | |
| InitializeGlobals | Assigns initialization values to global variables. This function gets called from main() as part of the setup process. |
| DSP28_CpuTimers.c | |
| ConfigCpuTimer | Initializes the selected timer to the period specified by the "Freq" and "Period" parameters. The "Freq" is entered as "MHz" and the period in "µSeconds". The timer is held in the stopped state after configuration. |
| InitCpuTimers | Initializes CPU timer0 to a known state. |
| DSP28_PieCtrl.c | |
| InitPieCtrl | Initializes the Peripheral Interrupt Expansion control registers to a known state. |
| DSP28_PieVect.c | |
| InitPieVectTable | Initializes the PIE vector table to a known state. This function must be executed after boot time. |
| DSP28_SysCtrl.c | |
| InitFlash | Initializes the Flash Control registers This function MUST be executed out of RAM. Executing it out of OTP/Flash will yield unpredictable results. |
| InitSysCtrl | Initializes the System Control registers to a known state. |
| DSP28_Xintf.c | |
| InitXintf | Initializes the External Interface to a known state. |
| DSP28_DefaultIsr.c | |
| CAPINT1_ISR | |
| CAPINT2_ISR | |
| CAPINT3_ISR | |
| CAPINT4_ISR | |
| CAPINT5_ISR | |
| CAPINT6_ISR | |
| CMP1INT_ISR | |
| CMP2INT_ISR | |
| CMP3INT_ISR | |
| CMP4INT_ISR | |

Table 4-20. DSP Software Functions (continued)

| Function Name | Description |
|----------------------|--------------------|
| CMP5INT_ISR | |
| CMP6INT_ISR | |
| DATALOG_ISR | |
| ECAN0INTA_ISR | |
| ECAN1INTA_ISR | |
| EMUINT_ISR | |
| ILLEGAL_ISR | |
| INT13_ISR | |
| INT14_ISR | |
| MRINTA_ISR | |
| MXINTA_ISR | |
| NMI_ISR | |
| PDPINTA_ISR | |
| PDPINTB_ISR | |
| PIE_RESERVED | |
| RTOSINT_ISR | |
| SCIRXINTA_ISR | |
| SCIRXINTB_ISR | |
| SCITXINTA_ISR | |
| SCITXINTB_ISR | |
| SPIRXINTA_ISR | |
| SPITXINTA_ISR | |
| T1CINT_ISR | |
| T1OFINT_ISR | |
| T1UFINT_ISR | |
| T2CINT_ISR | |
| T2OFINT_ISR | |
| T2PINT_ISR | |
| T2UFINT_ISR | |
| T3CINT_ISR | |
| T3OFINT_ISR | |
| T3PINT_ISR | |
| T3UFINT_ISR | |
| T4CINT_ISR | |
| T4OFINT_ISR | |
| T4PINT_ISR | |
| T4UFINT_ISR | |
| TINT0_ISR | |
| USER0_ISR | |
| USER10_ISR | |
| USER11_ISR | |
| USER1_ISR | |
| USER2_ISR | |
| USER3_ISR | |
| USER4_ISR | |
| USER5_ISR | |

Table 4-20. DSP Software Functions (continued)

| Function Name | Description |
|---------------|-------------|
| USER6_ISR | |
| USER7_ISR | |
| USER8_ISR | |
| USER9_ISR | |
| WAKEINT_ISR | |
| XINT1_ISR | |
| XINT2_ISR | |
| rsvd_ISR | |

Table 4-21. Function Call Hierarchy

| Level | Function |
|---------------|------------------------|
| main() | |
| 1 | main |
| 2 | InitSysCtrl |
| 2 | InitGpio |
| 2 | InitPieCtrl |
| 2 | InitPieVectTable |
| 2 | BootCopy |
| 2 | InitFlash |
| 2 | InitCRCTable |
| 2 | InitXintf |
| 2 | InitCpuTimers |
| 2 | InitSci |
| 2 | ConfigureADCs |
| 3 | DelayNus |
| 2 | ConfigureGPTimers |
| 3 | SleepDog |
| 3 | FeedDog |
| 2 | InitializeGlobals |
| 2 | InitializeUARTArray |
| 2 | InitDiagTrace |
| 2 | InitLampVars |
| 2 | ConfigCpuTimer |
| 2 | ArmAllSensors |
| 2 | MainLoop |
| 3 | ElapsedTime |
| 3 | GenerateFakePLCMessage |
| 3 | FillTxBuffer |
| 4 | CalcCRC |
| 4 | SetPlcMode |
| 5 | SetPWMPolarity |
| 5 | ExtractNextTxBit |
| 5 | ArmAllSensors |
| 3 | ProcessRxPlcMsg |
| 4 | CalcCRC |

Table 4-21. Function Call Hierarchy (continued)

| Level | Function |
|---------------------|-----------------------|
| 3 | TaskCommand |
| 4 | CmdReadMemory |
| 5 | WriteUARTValue |
| 6 | WriteUART |
| 4 | CmdWriteMemory |
| 5 | WriteUARTValue |
| 6 | WriteUART |
| 4 | CmdLamp |
| 5 | InitLampVars |
| 5 | WriteUARTValue |
| 6 | WriteUART |
| 4 | CmdPLCCommand |
| 5 | WriteUARTValue |
| 6 | WriteUART |
| 4 | CmdConfigFlooder |
| 5 | WriteUARTValue |
| 6 | WriteUART |
| 4 | WriteUARTValue |
| 5 | WriteUART |
| 3 | ControlLamp |
| 4 | SetLampIntensity |
| 3 | HandleUART |
| 4 | InitSci |
| 3 | FeedDog |
| 3 | HandlePLC |
| 4 | SetPlcMode |
| 5 | SetPWMPolarity |
| 5 | ExtractNextTxBit |
| 5 | ArmAllSensors |
| 4 | ExtractNextTxBit |
| ADCINT_ISR() | |
| 1 | ADCINT_ISR |
| 2 | FilterCorrRx |
| 3 | ArmAllSensors |
| 3 | SmoothADCResults |
| 3 | FIR2 |
| 3 | AutoCorr |
| 3 | SetRxMode |
| 3 | StoreRxBit |
| 3 | HammingDistanceEOPEOP |
| 3 | SaveTrace |
| T1PINT_ISR() | |
| 1 | T1PINT_ISR |
| 2 | SetPWMPolarity |
| 2 | SetPlcMode |
| 3 | SetPWMPolarity |

Table 4-21. Function Call Hierarchy (continued)

| Level | Function |
|-------|------------------|
| 3 | ExtractNextTxBit |
| 2 | ExtractNextTxBit |

4.8 Software Compile-Time Options

The DSP software contains many compiler flags to allow for customization of the software to the intended application. Some of the important compile flags are listed in [Table 4-22](#).

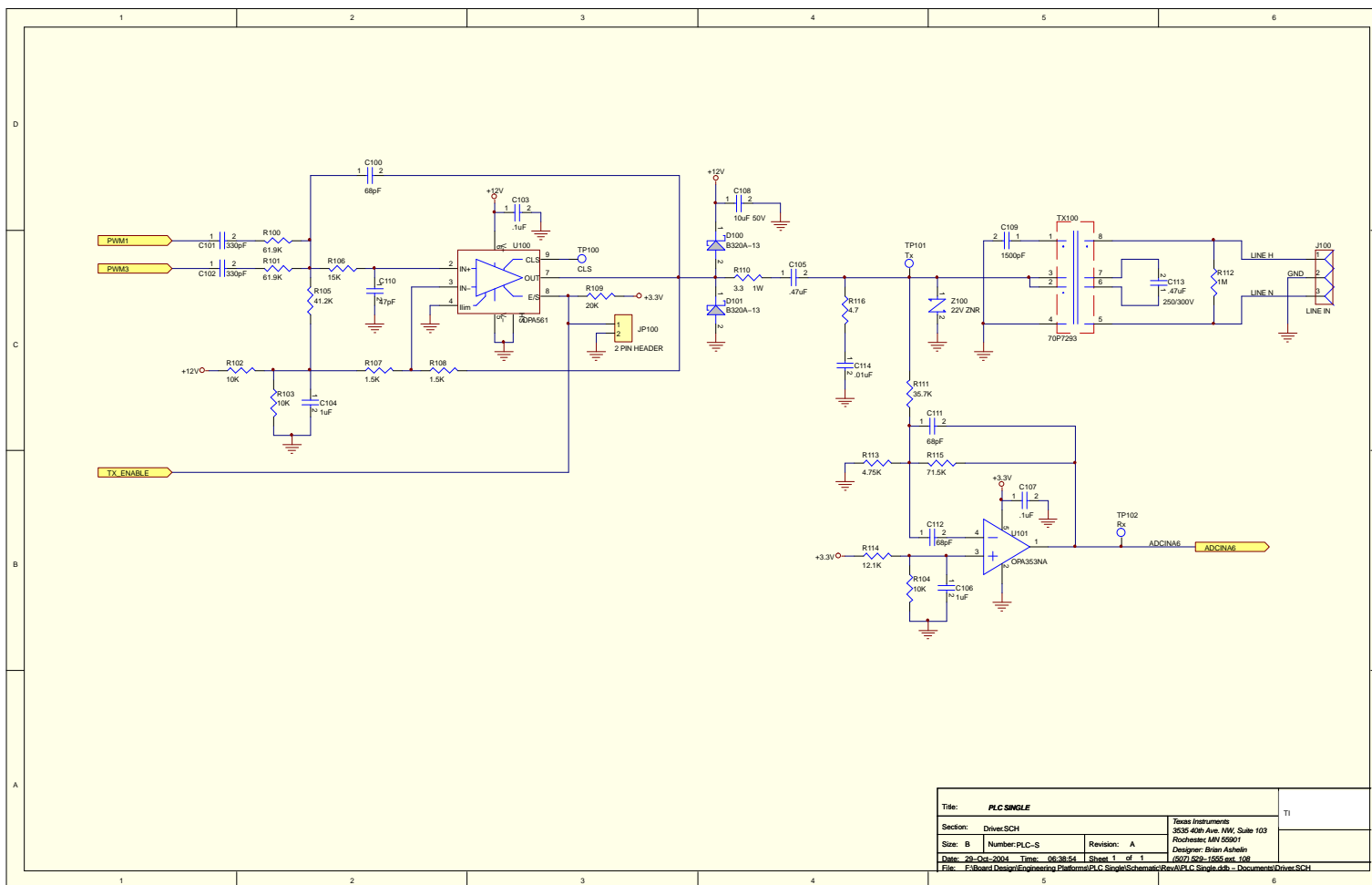
Table 4-22. Compile Flags

| Compile Flag | Default Value | Description |
|-------------------|---------------|---|
| TRACE_BUF_LEN | 0x1000 | Sets size of diagnostics buffer used by SaveTrace function. When set to 0, the SaveTrace function is not compiled to save space and execution time. |
| MIPS | 150 | Sets MIPS rate for DSP. Valid values are 150 and 120. |
| USE_CRC | True | Use Cyclic Redundancy Code to verify validity of data. |
| XOR_TX_DATA | True | When true, pre-codes data so that the differential receiver outputs the correct data with no additional processing. |
| XOR_RX_DATA | ~True | When true, accepts data that has not been differentially precoded and converts it to be read correctly. More susceptible to noise than XOR_TX_DATA, but ensures compatibility with generic Lonworks transmitters. |
| SLOW_BETWEEN_MSGS | True | Reduces the MIPS load between messages by only running the correlator during every 5th sample time. |
| FIR_LEN | 25 | Number of ADC samples in one PLC bit time. |
| CORR_LEN | FIR_LEN | Number of samples used for correlation. Valid values are 25 and 5. Using 5 is not recommended because it degrades the data error rate severely. |
| OVERSAMPLE_RATE | 4 | Sets the number of ADC readings that are combined together to make one measurement. Valid values are 1 through 16. Recommended values are 3, 4, 6, and 10. |

Single Frequency Power Line Communication Schematics

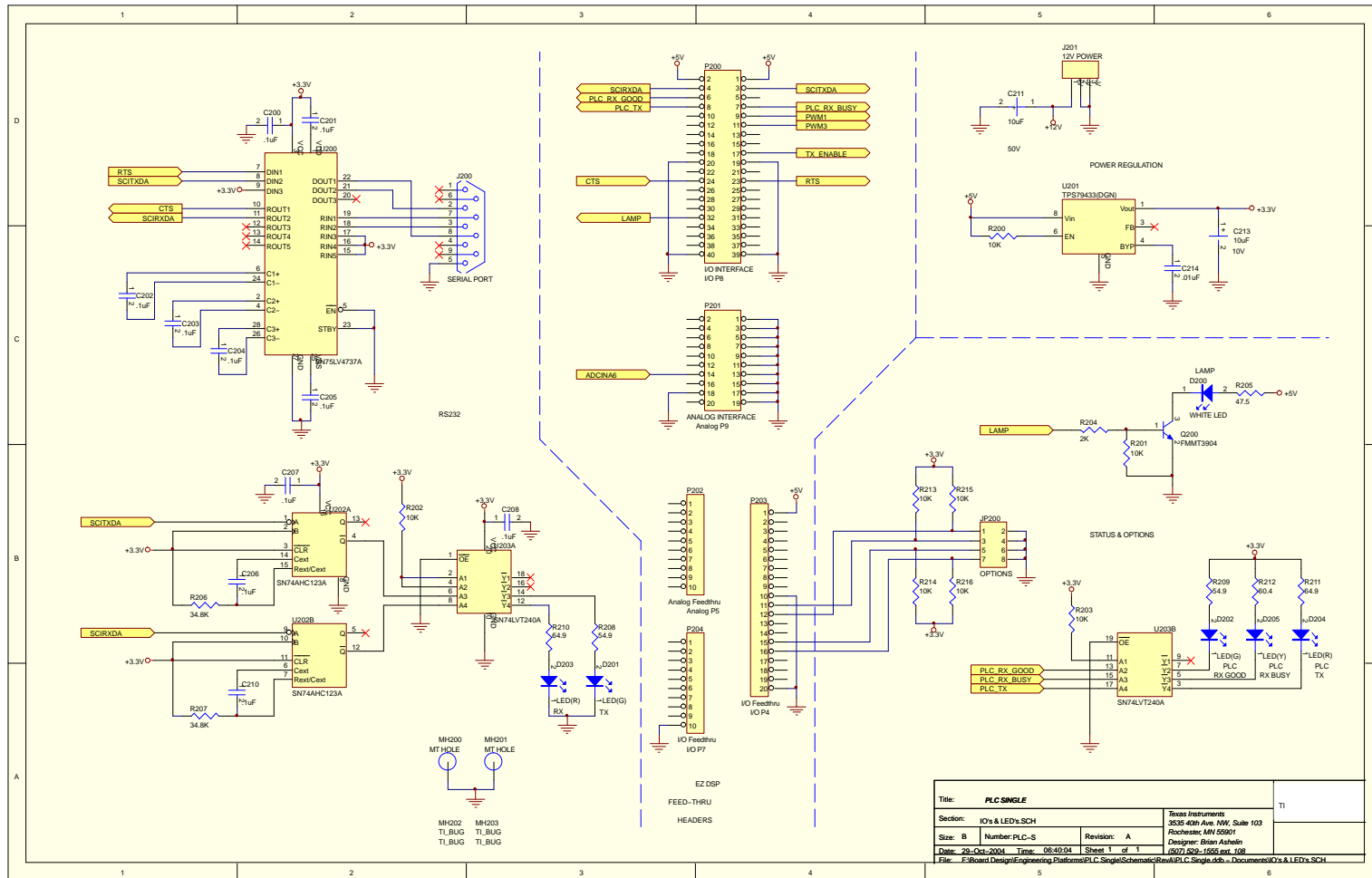
This chapter contains the schematics for the single-frequency power line modem.

Figure A-1. PLC Single - Driver Schematic



| | | | |
|--|--|--------------|------------------------------|
| Title: PLC SINGLE | | | T1 |
| Section: DriverSCH | Texas Instruments 3635 40th Ave. NW, Suite 103 Richfield, MN 55091 | | |
| Size: B | Number: PLC-S | Revision: A | Designer: Brian Ashelin |
| Date: 28-Oct-2004 | Time: 08:28:34 | Sheet 1 of 1 | Rev: 10/27/2004-10:00 am JAB |
| File: F:\Board Design\Engineering\218186\PLC Single Schematic\RevA\PLC Single-Sub - Document\DriverSCH | | | |

Figure A-2. PLC Single - IOs and LEDs Schematic



Glossary

- ADC** — Analog-to-Digital Converter
- DALI** — Digital Addressable Lighting Interface
- DSP** — Digital Signal Processor
- EVA** — Event Manager A
- EVB** — Event Manager B
- FIR** — Finite Impulse Response
- GPIO** — General Purpose Input/Output
- ISR** — Interrupt Service Routine
- MIPS** — Million Operations Per Second
- PIE** — Peripheral Interrupt Expansion
- PLC** — Power Line Communications
- SCI** — Serial Communications Interface
- NPDU** — Network Protocol Data Unit
- CRC** — Cyclical Redundancy Checksum

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

| Products | | Applications | |
|------------------|--|---------------------|--|
| Amplifiers | amplifier.ti.com | Audio | www.ti.com/audio |
| Data Converters | dataconverter.ti.com | Automotive | www.ti.com/automotive |
| DSP | dsp.ti.com | Broadband | www.ti.com/broadband |
| Interface | interface.ti.com | Digital Control | www.ti.com/digitalcontrol |
| Logic | logic.ti.com | Military | www.ti.com/military |
| Power Mgmt | power.ti.com | Optical Networking | www.ti.com/opticalnetwork |
| Microcontrollers | microcontroller.ti.com | Security | www.ti.com/security |
| | | Telephony | www.ti.com/telephony |
| | | Video & Imaging | www.ti.com/video |
| | | Wireless | www.ti.com/wireless |

Mailing Address: Texas Instruments
Post Office Box 655303 Dallas, Texas 75265