CRD5460-1

Power Meter Reference Design Board and Software

Features

- Operates with PC or Stand-Alone without PC
- On-board LCD Data Display or Optional Electro-Mechanical Totalizer
- Lab Windows/CVI™ PC Evaluation Software
  - Register Setup & Chip Control
  - Current, Voltage and Energy Display
  - Power Factor Analysis
- Non-Volatile energy accumulation via EEPROM
- Non-Volatile Calibration Values via EEPROM
- Current Transformer and VoltageTransformer Interface
- Header for optional external Analog input
- RS-232 Serial Communication with PC
- Optional Peripheral Daughter Board Header

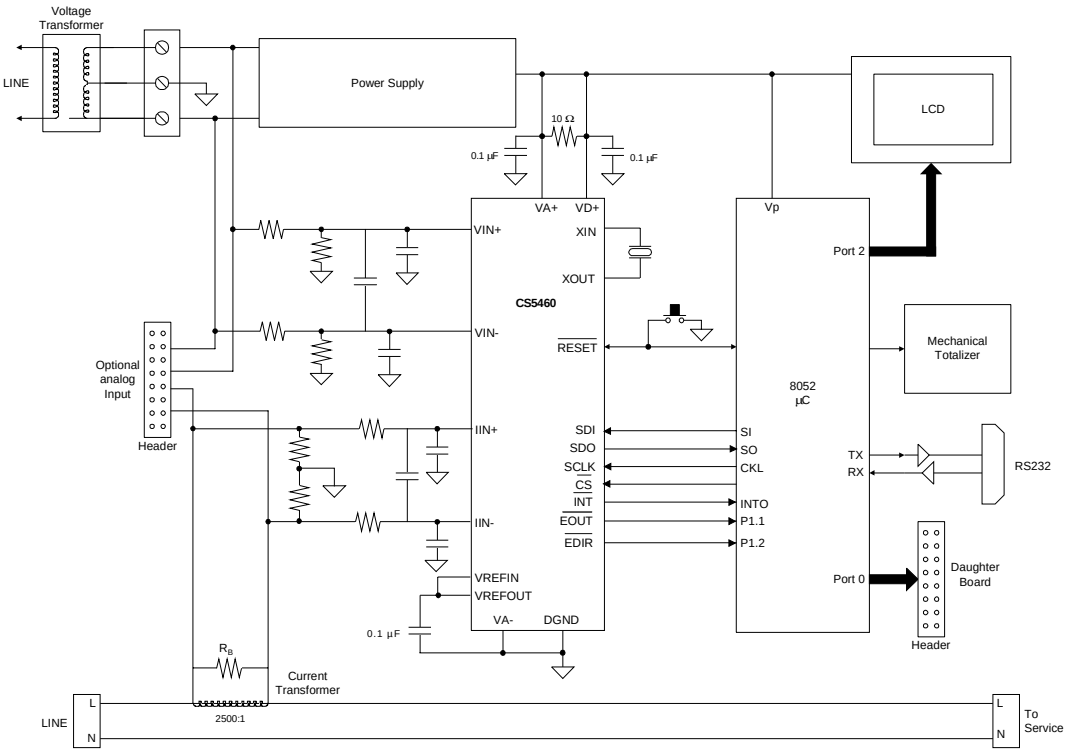
General Description

The CRD5460 is a stand-alone reference design intended to demonstrate the functionality and performance of the CS5460. It is recommended that the CS5460 Data Sheet be used along with this document. The CRD5460 is intended to be a starting point for designers who are developing Power Metering applications using the CS5460 chip.

The CRD5460 includes firmware that can monitor and display energy, RMS current/voltage, and power factor. An optional mechanical totalizer can be substituted for the LCD display for those applications involving only the non-volatile accumulation of energy usage. The CRD5460 functions in a Stand-Alone Mode, or with a PC connected. PC-mode is included as a diagnostic tool.

ORDERING INFORMATION

CRD5460-1 Reference Design Board



Preliminary Product Information | This document contains information for a new product. Cirrus Logic reserves the right to modify this product without notice.

CIRRUS LOGIC®
P.O. Box 17847, Austin, Texas 78760
(512) 445 7222 FAX: (512) 445 7581
http://www.cirrus.com

Copyright © Cirrus Logic, Inc. 2000
(All Rights Reserved)

TABLE OF CONTENTS

Contacting Cirrus Logic Support
For a complete listing of Direct Sales, Distributor, and Sales Representative contacts, visit the Cirrus Logic web site at:
http://www.cirrus.com/corporate/contacts/

IBM, AT and PS/2 are trademarks of International Business Machines Corporation.
Windows is a trademark of Microsoft Corporation.
Lab Windows and CVI are trademarks of National Instruments.
SPI™ *is a trademark of Motorola.*
Microwire™ *is a trademark of National Semiconductor.*

Preliminary product information describes products which are in production, but for which full characterization data is not yet available. Advance product information describes products which are in development and subject to development changes. Cirrus Logic, Inc. has made best efforts to ensure that the information contained in this document is accurate and reliable. However, the information is subject to change without notice and is provided "AS IS" without warranty of any kind (express or implied). No responsibility is assumed by Cirrus Logic, Inc. for the use of this information, nor for infringements of patents or other rights of third parties. This document is the property of Cirrus Logic, Inc. and implies no license under patents, copyrights, trademarks, or trade secrets. No part of this publication may be copied, reproduced, stored in a retrieval system, or transmitted, in any form or by any means (electronic, mechanical, photographic, or otherwise) without the prior written consent of Cirrus Logic, Inc. Items from any Cirrus Logic website or disk may be printed for use by the user. However, no part of the printout or electronic files may be copied, reproduced, stored in a retrieval system, or transmitted, in any form or by any means (electronic, mechanical, photographic, or otherwise) without the prior written consent of Cirrus Logic, Inc.Furthermore, no part of this publication may be used as a basis for manufacture or sale of any items without the prior written consent of Cirrus Logic, Inc. The names of products of Cirrus Logic, Inc. or other vendors and suppliers appearing in this document may be trademarks or service marks of their respective owners which may be registered in some jurisdictions. A list of Cirrus Logic, Inc. trademarks and service marks can be found at http://www.cirrus.com.

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

The CRD5460 power meter reference design board provides a quick means of prototyping with the CS5460 Power/Energy Measurement IC. The CRD5460 board comes populated with a CS5460 Power/Energy IC, an 87C52 micro with firmware, an LCD display module, an EEPROM, current transformer, and a RS-232 level shifter. The reference design is supplied with an external step down transformer capable of providing power to the board as well as a voltage-sense input to the CS5460 chip. The CRD5460 Reference Design is capable of functioning in a stand-alone mode, but can be interfaced to an IBM™ compatible PC via an RS-232 interface. To accomplish this, the board comes equipped with the necessary 87C52 microcontroller firmware and a 9-pin RS-232 cable which physically interfaces the reference design board to the PC. The 87C52 controls the serial communication between the CRD5460 and the PC, enabling quick and easy access to all of the CS5460's registers and functions. Additionally, analysis software for the PC is provided to allow for easy access to the internal registers of the CS5460, and to provide a means to display the computational results that can be read from the CS5460.

2. HARDWARE

2.1 General Description

The schematic (Figures 1, 2 and 3) is partitioned into three main sections: I) Phase 0 with sensors included, II) Digital processor with LCD display, and III) Optional Phase1/Phase2 section for monitoring power in three-phase systems. The Phase 0 section consists of the CS5460, voltage regulator, and current transformer. The digital section consists of the 87C52 microcontroller, 1 64x16 EEPROM, the hardware switches, the reset circuitry, LCD module and the RS-232 interface. The Phase1/Phase2 section contains the necessary circuitry for support of up to two additional CS5460 chips. A center-tapped step-down transformer is provided as a source of AC for power and voltage sensing. An optional 3-terminal header (J1) is available for user-supplied AC power input. Acceptable AC input range to J1/J2 is 14 - 25 VAC, center-tapped.

2.2 Phase 0 Section

The CS5460 is implemented to accurately measure and calculate: Energy, I_{RMS}, and V_{RMS} when operating from a 4.096 MHz crystal.

2.2.1 AC Power

As shown in Figure 1, the reference design is powered by +5VD voltage line that is provided from the voltage regulator U2. This regulator is powered by an external AC voltage transformer that is connected via header J2 or optionally via the screw-terminal header J1. A full-wave rectifier (D1 and D2) is used with a LM7805 voltage regulator to create the +5VD supply voltage.

NOTE: *Special CRD5460 Voltage Transformer: There are two ways to plug the Tamura 818A_0001 voltage transformer (supplied with the CRD5460) into a power outlet. One way is correct. One way is incorrect and will lead to inaccurate results. Correct connection to the power line can be verified by measuring the AC voltage between external earth ground (third wire in power outlet) and AGND on the CRD5460 board. [Before making this measurement, make sure that the RS-232 cable from the PC is DISCONNECTED from the CRD5460's RS-232 port.] If the transformer is connected correctly, there will be near zero volts AC between earth and AGND. If the voltage transformer is connected incorrectly, there will be a weak 75 VAC that is measurable between the CRD5460's AGND and earth ground.*

The CRD5460 Voltage Transformer does not have to be used with the CRD5460. In fact, the Tamura 818A-0001 voltage transformer should never be exposed to more than 150 VAC on the inputs. Therefore, in areas where the nominal line voltage is more than 150VAC, the user must find a suitable transformer, which can be connected to the available J1 header. The output should of this transformer should be 14-25VAC, center tapped.

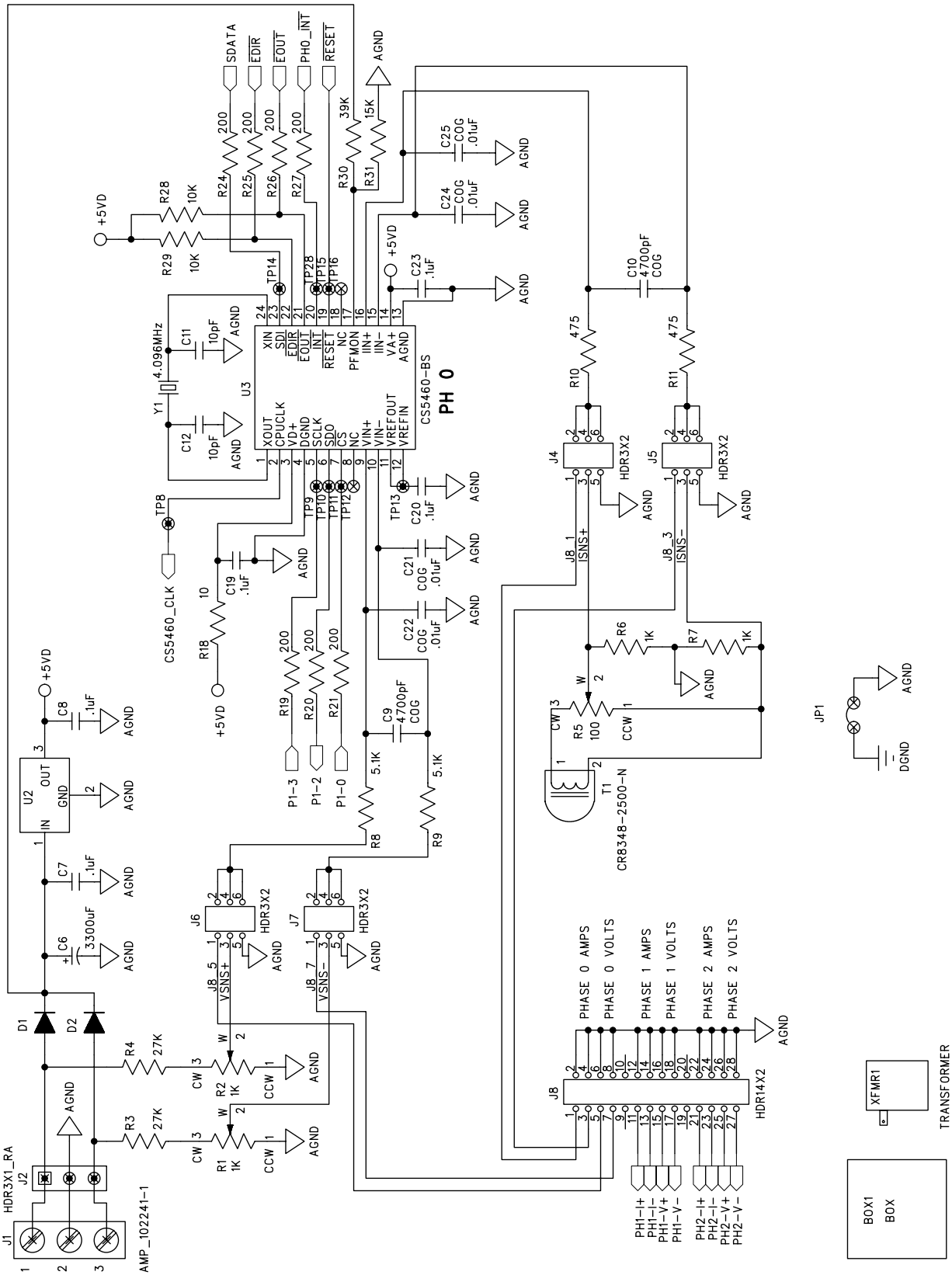


Figure 1. Phase 0 Section

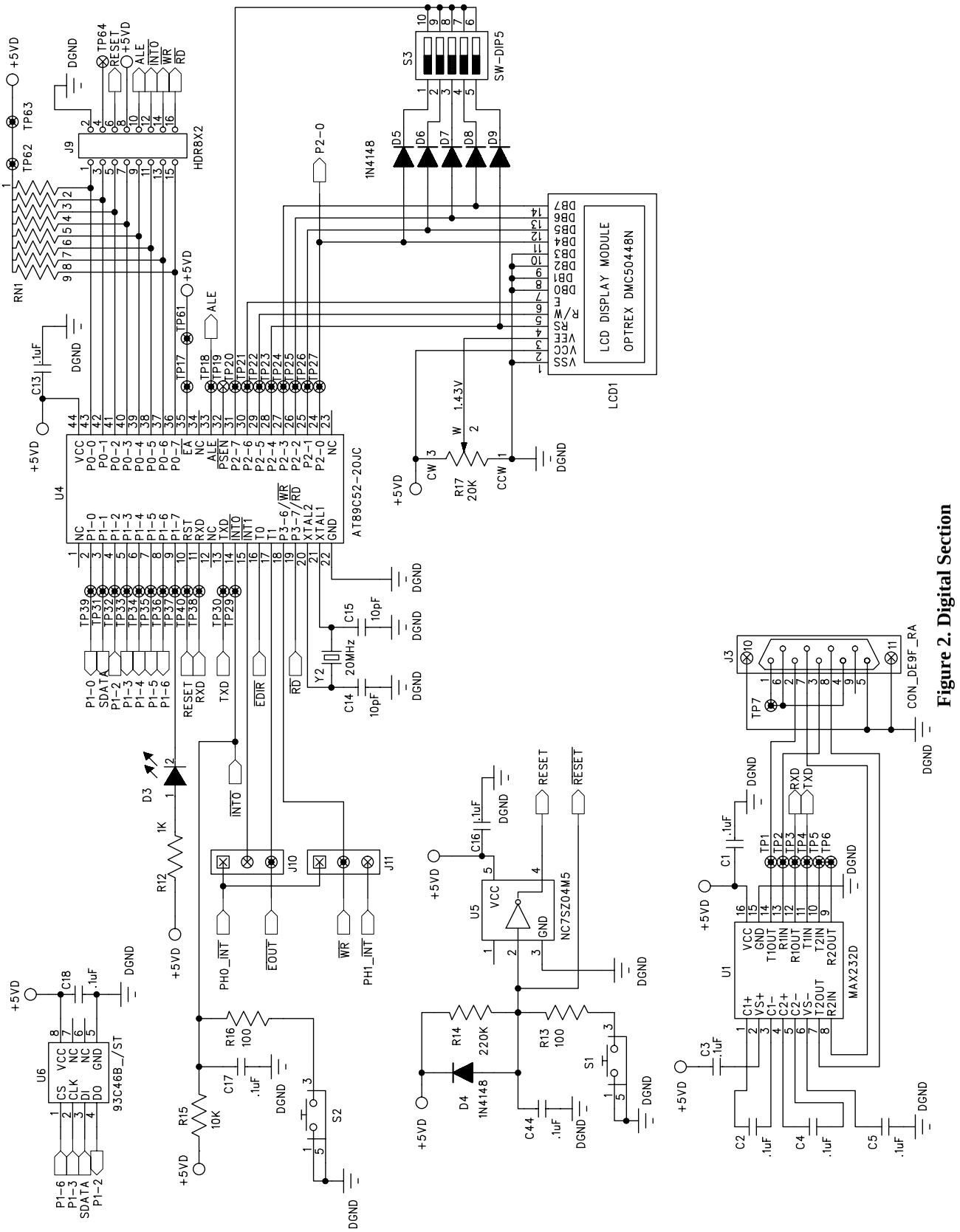


Figure 2. Digital Section

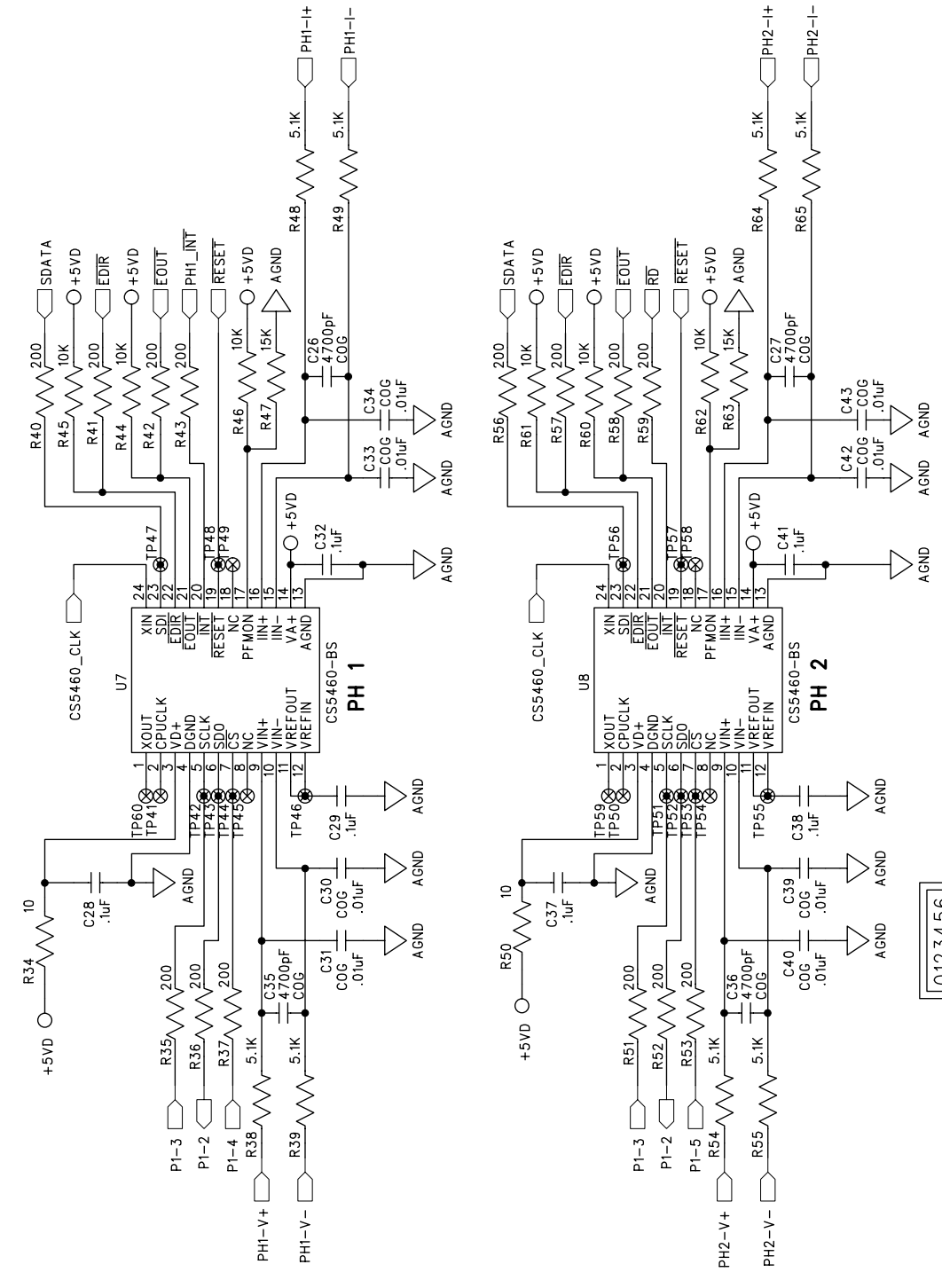


Figure 3. Phase 1/2 section

2.2.2 Line Voltage Sense

The AC voltage from J1/J2 is also used to supply the required voltage sense input to the voltage channel of the CS5460. In order to provide bipolar voltage levels at the VIN+ and VIN- inputs, the voltage transformer has a center-tapped output which are returned to AGND. Each side of the voltage input is then scaled through a simple resistor-divider network formed by R3/R1 and R4/R2. R1 and R2 are both adjustable so that the differential voltage-channel inputs to the CS5460 can be evenly matched. (More about this later). Headers J6 and J7 provide options for sensing the voltage from an external line-voltage sensor which can be connected via J8. J6 and J7 also have options for connections to AGND which can be used during the power meter's calibration process.

NOTE: *Potentiometers R1 and R2 have been provided on the CRD5460-1 Reference Design Board for convenience. It is recommended that production designs use fixed resistors for stability and reliability. The same can be said for R5, which is discussed next.*

2.2.3 Line Current Sense

To monitor the line current, there is a PC-mounted current transformer (T1) with adjustable sense resistor (R5). This configuration provides an isolated means of measuring the current to the load. The user must feed one of the two wires that connect LINE to LOAD straight through T1 one time (no loops). Either neutral side or line side can be used. Jumper headers J4 and J5 are provided to allow for optional inputs from external current sensors via header J8. Note that resistors R6 and R7 are used to provide equal ground-referenced differential input levels at the Iin+ and Iin- pins of the CS5460 from the sensed current. While the common-mode level of the voltage and current inputs does not have to be at AGND, this configuration was chosen to simplify the reference design.

Note that with no current present in the current transformer 1/2-turn winding, a display of negative energy is possible and correct. This is because with no transformer current present, the transformer's output voltage will be hovering right above or right below 0 V. And so the voltage on the current-channel input may easily fall just below AGND potential, or may just as easily rise just above AGND.

NOTE: *After verifying that the voltage transformer is plugged into the wall correctly, the user should thread the line-current loop through the current transformer. The user might thread the current loop wire in the correct or incorrect direction. If, after power is turned on, and WITH SIGNIFICANT CURRENT IN THE LINE, the CRD5460 LCD display indicates "NEG ENGY" (negative energy), then the line-current loop wire is threaded in the wrong direction.*

NOTE: *The CRD5460's maximum input current level for the on-board current transformer (T1) is 5A (RMS). For loads greater than 5A (RMS), an external sensor must be used.*

2.2.4 Input Filtering

Note that for both the voltage channel (VIN+/VIN-) and the current channel (Iin+/Iin-), a simple RC network filters the sensors' outputs to attenuate common-mode/differential-mode interference that might be induced in the sensor leads. For the voltage channel, the 3 dB corner of these filters is ~1.61KHz (common-mode) and ~2.05KHz (differential mode). For the current channel, the 3 dB corner of these filters is ~17.26kHz (common-mode) and ~17.6kHz (differential mode). These filter networks are defined by R8, R9, C9, C21, and C22 for the voltage channel, and by R10, R11, C10, C24 and C25 for the current channel.

2.2.5 Bypassing

Bypass capacitors C19, C20, and C23 are used to stabilize the voltage and de-couple high-frequency noise respectively on the VD+, VREFIN, and VA+ input pins.

2.2.6 Oscillator

Y1, C11, and C12 form the oscillator input circuitry. The clock signal that is generated by the

CS5460 is wired out of the CPUCLK pin to serve as a synchronized clock source for other CS5460s that might be working in parallel (such as a three-phase application).

2.2.7 Power Monitor

R30 and R31 form a resistor-divider, and the ratio of their values was chosen to match the voltage threshold point of the PFMON input pin. If the voltage level at the input of the voltage regulator circuitry becomes too low (i.e., if it goes below threshold), the PFMON input will trip and set a flag in the CS5460 status register indicating that power is about to fail.

2.3 Digital Section

Figure 2 illustrates the schematic of the digital section. It contains the microcontroller, mode dip-switches, RS-232 interface chip, a 64x16 EEPROM, and an LCD module. The mode dip-switches aid in setting the different modes of operation on the reference design board.

2.3.1 RS-232 Interface

The RS-232 interface is configured to communicate at 9600 baud, no parity, 8-bit data, and 1 stop bit. The microcontroller interfaces to the RS-232 chip (U1) through the RXD and TXD lines. U1 provides the RS-232 voltage levels necessary for interfacing to external PC or other devices.

2.3.2 Reset Circuit

The network formed by D4, R13, R14, S1, U5 and C16 is used to generate the hardware RESET signal, which can be initiated by depressing the S1 button or by cycling power. S1 is located on top-side of PCB. Pressing S1 will asynchronously reset the reference design circuit. Reset is described in a later section called *9. Resetting the CRD5460*.

2.3.3 Non-Volatile Memory

U6 is a serial EEPROM which is used to store calibration data for the CS5460. In a metering application, calibration data can be established during the manufacture of each power meter, stored in the EEPROM, and then recalled at a later time when the circuit is powered up. The EEPROM also retains a cumulative energy sum to record the total energy that is used over an extended period of time.

2.3.4 Microcontroller

The U4 microcontroller uses three of its four 8-bit peripheral interfaces (P0-, P1-, and P2-). The P0-peripheral lines can be used for communication to an external device via the daughter board header, J9. The P1- peripheral lines are dedicated for interfacing to the CS5460 power chip(s). The P2- lines (pins 24-31) are used to drive output information to the LCD1 liquid crystal, as well as to periodically interrogate the status of the S3 DIP switch. The S3 DIP switch is used to select one of several different operating modes. The microcontroller derives its clock from a 20.0 MHz crystal.

2.4 Phase1/Phase2 Section

Figure 3 shows the optional second and third phases to allow the reference design to measure polyphase loads. The basic configuration of the extra CS5460's is very similar to the configuration discussed earlier (for Phase 0). The required voltage and current sensors for the one or two phases would need to be mounted external to the reference design and connect via the J8 header. The 3-Phase application is described in more detail in the section titled *11. Three-Phase Operation*.

The schematic also shows the optional mechanical totalizer, that can replace the LCD module when a low cost, non-volatile display is required.

3. GENERAL OPERATION

3.1 Overview of the CS5460

The CS5460 chip is the heart of the CRD5460 Power Meter Reference Design. The CS5460 is a highly integrated device, containing dual ADCs with a computational unit. The CS5460 and CRD5460 data sheets should be read thoroughly and understood before attempting to use the CRD5460 reference design. The CS5460 contains a programmable gain amplifier (PGA), two ΔΣ modulators, two high rate filters, an on-chip reference, and power calculation engine to compute Energy, V_{RMS}, I_{RMS}, and Instantaneous Power. The PGA sets the input levels of the current channel at either 30 mV_{RMS} or 150 mV_{RMS} (for VREFIN = 2.5 V). The on-chip reference can provide the necessary 2.5 V reference. This output (VREFOUT), along with a 0.1 µF capacitor, is used to supply the VREFIN pin with 2.5 V. The ΔΣ modulators and high rate digital filters allow the user to measure instantaneous voltage, current, and power at an output word rate of 4000 Hz when a 4.096 MHz clock source is used. Measurement of I_{RMS}, V_{RMS}, and energy takes place once per second with the default cycle count register setting of 4000.

3.2 Modes of Operation

The CRD5460 Power Meter has two main modes of operation: Stand-Alone Mode and PC Mode. Stand-Alone Mode supports several different operating modes. These modes are controlled by the S3 DIP switch. Stand-Alone Mode allows the CRD5460 to operate by itself, without a computer. The various readings are output to the on-board liquid-crystal display. PC Mode allows the user to operate the power meter from a PC that interfaces to the power meter via the RS-232 connector. The CRD5460 Power Meter was designed so that the user can switch back and forth between PC Mode and Stand-Alone Mode while the meter is operating.

3.2.1 S3 DIP Switch

The modes of operation are controlled by the S3 DIP switch.

3.2.1.1 Stand-Alone Mode

Stand-Alone Mode is initiated by setting the S3 DIP switches to any non-zero setting during power-on or system-reset. (Not all S3 combinations are valid, but if S3 is set to any unused state, the user receives a message on the LCD display that reads "?UNKNOWN DIPVALUE".) Table 1 summarizes the various modes of operation. The DIP switches can be set to various values which determine the particular action that the user wants the meter to perform. Note, however, that in Stand-Alone Mode, manual switching of the DIP switches in order to get from one type of operation to the next can make the CDB5460 power meter go through intermediate operating modes that may not be desirable. For example, if the user is not careful, he/she may accidentally cause the meter to write its most recent calibration values to the on-board EEPROM, overwriting existing calibration values that were thought to be saved. The user should therefore plan a 'safe' switching sequence in order to set the DIP switches from one state to another state without entering any 'undesirable' states in between. Note that in a production meter, the DIP switch functions would be hard-wired and not subject to change.

3.2.1.2 PC Mode

PC mode allows the user to operate the power meter from a PC, which interfaces to the power meter via the RS-232 connector. PC mode is initiated by setting all of the DIP switches on S3 to "0" (closed). In PC mode, the user runs a LabWindows program (provided along with the CRD5460). The CRD5460 PC Mode software is somewhat similar to that available for the CDB5460 Evaluation Board. The CRD5460 software allows the user to read from the various data registers, and to write to the various configuration and status registers in or-

der to control the operation of the CS5460. Voltage, current, power, and energy values are read from the CS5460 data registers and put into PC memory for on-screen viewing. The power-factor and phase-angle is calculated by the software, and these quantities are also displayed.

To exit from PC Mode an return to the Stand-Alone Mode, the user must execute one PC-mode command from the software program in order to get Stand-Alone Mode to start operating. This is done by using the mouse to click on the soft-button labeled "Change DIP SW Then Click to Exit PC Mode." (This button is available on-screen in the PC software program, discussed later).

Note from Table 1 that only certain DIP switch settings should be engaged when the CRD5460 is first powered up, or reset. The valid boot-up/reset switch states are indicated in the last column of Table 1. Note further that the state of the DIP switch settings at power-up/reset determine whether *default* calibration values (from microcontroller firmware) will be loaded into the calibration registers of the 5460, or if calibration values stored in the on-board EEPROM will be downloaded to the CS5460. Any even switch setting (including mode "00000") that is classified as a valid boot-up/reset state will reset the board with default settings determined by the microcontroller firmware. A reset with odd DIP switch settings that are classified as valid boot-up/reset states will cause the CS5460's voltage and current gain registers to be written with settings from the on-board EEPROM. Please see the section titled *9. Resetting the CRD5460* for more specifics on reset of the CRD5460.

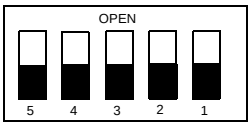
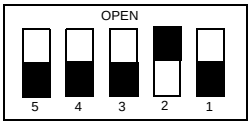
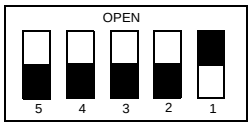
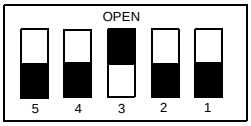
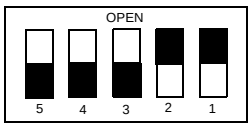
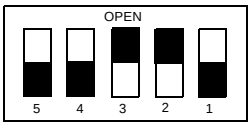
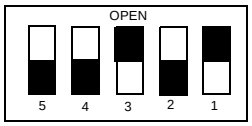
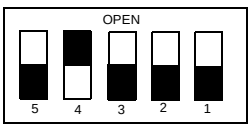
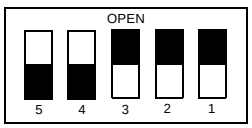
Operating Mode	S3 even: Cal values come from Micro	S3 odd: Cal Values come from EEPROM	Reset/ Boot-up allowed?
"PC Mode" All five switches CLOSED (logic levels: "00000") Output is on PC monitor (All other modes are "Stand-Alone" modes - Output on LCD)			Yes
Display Power Factor (F) and LEAD/LAG status (Output on LCD display). "00001" or "00010" S3-1 OPEN, S3-2 thru S3-5 CLOSED, or S3-2 OPEN, S3-1, S3-3 thru S3-5 CLOSED		-OR- 	Yes
"Cumulative" Energy displayed on top, and Cycle-Count ENERGY displayed on bottom of LCD. "00011" or "00100" S3-1, S3-2, S3-4, S3-5 CLOSED, S3-3OPEN, or S3-1 and S3-2 OPEN, S3-3 thru S3-5 CLOSED		-OR- 	Yes
RMS-Current and RMS-Voltage displayed. "00101" or "00110" S3-2, S3-4, and S3-5 CLOSED, S3-1 and S3-3 OPEN, or S3-1, S3-4 and S3-5 CLOSED and S3-2 and S3-3 OPEN		-OR- 	Yes
"Auto-Cycle" mode--cycles through energy readings, power/power factor, and I(RMS) and V(RMS) on the LCD display. "00111" or "01000" S3-4 and S3-5 CLOSED, S3-1 thru S3-3 OPEN, or S3-1 thru S3-3, and S3-5 CLOSED, S3-4 OPEN.		-OR- 	Yes

Table 1. Modes of Operation Summary

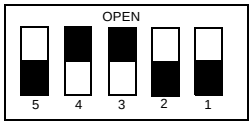
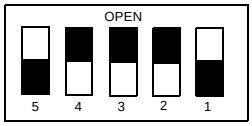
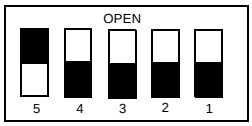
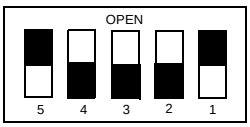
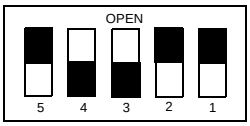
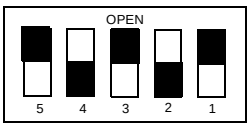
Operating Mode	S3 even: Cal values come from Micro	S3 odd: Cal Values come from EEPROM	Reset/ Boot-up allowed?
Initiate CS5460 voltage gain calibration sequence. “01100” S3-1, S3-2, and S3-5 CLOSED, S3-3 and S3-4 OPEN.			No
Initiate CS5460 current gain calibration sequence. “01110” S3-1 and S3-5 CLOSED, S3-2 thru S3-4 OPEN.			No
Electromechanical counter operation, use default calibration values. “10000” S3-1 thru S3-4 CLOSED, S3-5 OPEN. **Must set J10 shunt to connect pins 2 and 3.**			Yes
Electromechanical counter operation, use calibration values on EEPROM (U6). “10001” S3-2 thru S3-4 CLOSED, S3-1 and S3-5 OPEN. **Must set J10 shunt to connect pins 2 and 3.**			Yes
Reset the non-volatile cumulative energy to zero. “10011” S3-3 and S3-4 CLOSED, S3-1, S3-2, and S3-5 OPEN.			No
Save (write) the voltage/current offset and gain calibration values in CS5460 registers to the EEPROM (U6). “10101” S3-2 and S3-4 CLOSED, S3-1, S3-3, and S3-5 OPEN.			No

Table 1. Modes of Operation Summary (Continued)

Name	Function Description	Default Setting	Default Jumpers
J4	Switches IIN+ pin on the CS5460 between external sense option (J8), on-board sensor, and AGND.	IIN+ Set to on-board current sensor	IIN+ O O J8_1 IIN+ O ▬ O ISNS+ IIN+ O O AGND
J5	Switches IIN- pin on the CS5460 between external sense option (J8), on-board sensor, and AGND.	IIN- Set to on-board current sensor	IIN- O O J8_3 IIN- O ▬ O ISNS- IIN- O O AGND
J6	Switches VIN+ pin on the CS5460 between external sense option (J8), on-board sensor, and AGND.	VIN+ Set to on-board sensor	VIN+ O O J8_5 VIN+ O ▬ O VSNS+ VIN+ O O AGND
J7	Switches VIN- pin on the CS5460 between external sense option (J8), on-board sensor, and AGND.	VIN- Set to on-board sensor	VIN- O O J8_7 VIN- O ▬ O VSNS- VIN- O O AGND
J10	Controls which signal will cause an interrupt [/INT1] on the 87C52 microcontroller: PH0_INT is used for LCD Display, EOUT is used for Electromechanical Counter.	PH0_/INT controls the 87C52’s /INT1 line	O PH0_/INT ▌ /INT1 O /EOUT
J11	Used to control which signal will connect to /WR control line on the 87C52’s P3 register (pin18).	No connection (not used at present).	O PH0_/INT O /WR O PH1_/INT

Table 2. Header/Jumper Descriptions

The execution of the CRD5460 operations is controlled by the 87C52 microcontroller. The firmware in the microcontroller periodically interrogates the S3 DIP switch, and determines which actions need to be taken. Figure 4 is a flow diagram which shows the basic flow of the firmware's main loop. The topics discussed in this section are illustrated by the boxes towards the bottom of the diagram, particularly with the box that says "Read DIP Switch." Also see section *9. Resetting the CRD5460*.

Note from Table 1 that some of the modes concern calibration values. *Gain* calibration values for the current and voltage channels can be obtained and written to the CS5460 gain and registers. These calibration values can be saved to the on-board EEPROM (U6) and then, at a later time, they can be written back to the voltage/current gain calibration registers of the CS5460 by setting the DIP switches to the proper settings. It is recommended that the user perform the AC Calibration Procedure that is described in the section *6. Calibration.*

Also note in Table 1 that two of the modes are dedicated to operation of the optional electromechanical counter. See section *8. Electromechanical Counter* for more details.

3.2.2 Jumper Settings

There are several headers on the board which allow the user to select from several different input sources for the voltage and current channels. Table 2 summarizes the various jumper settings. On-board sensing components are provided to monitor single-phase power line voltage and current, or in the case of external sensors, output of the sensors can be interfaced through the J8 connector on the

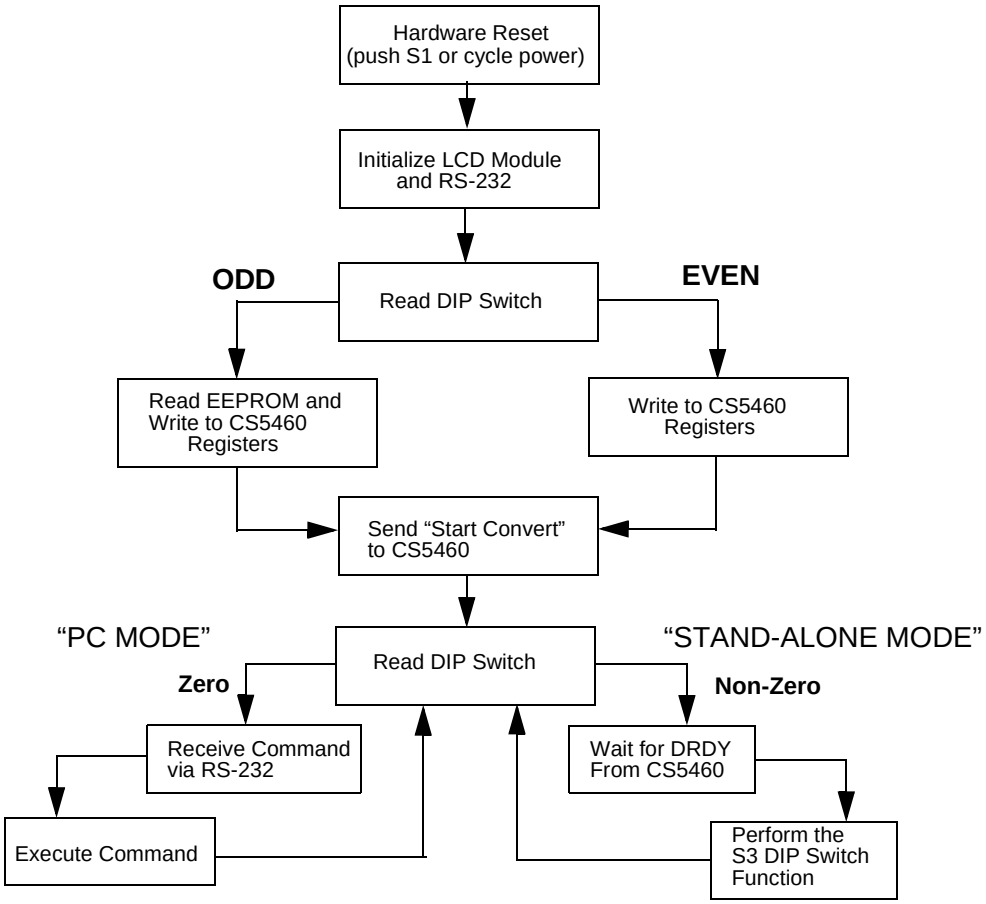


Figure 4. 87C52 Microcode Flow Chart

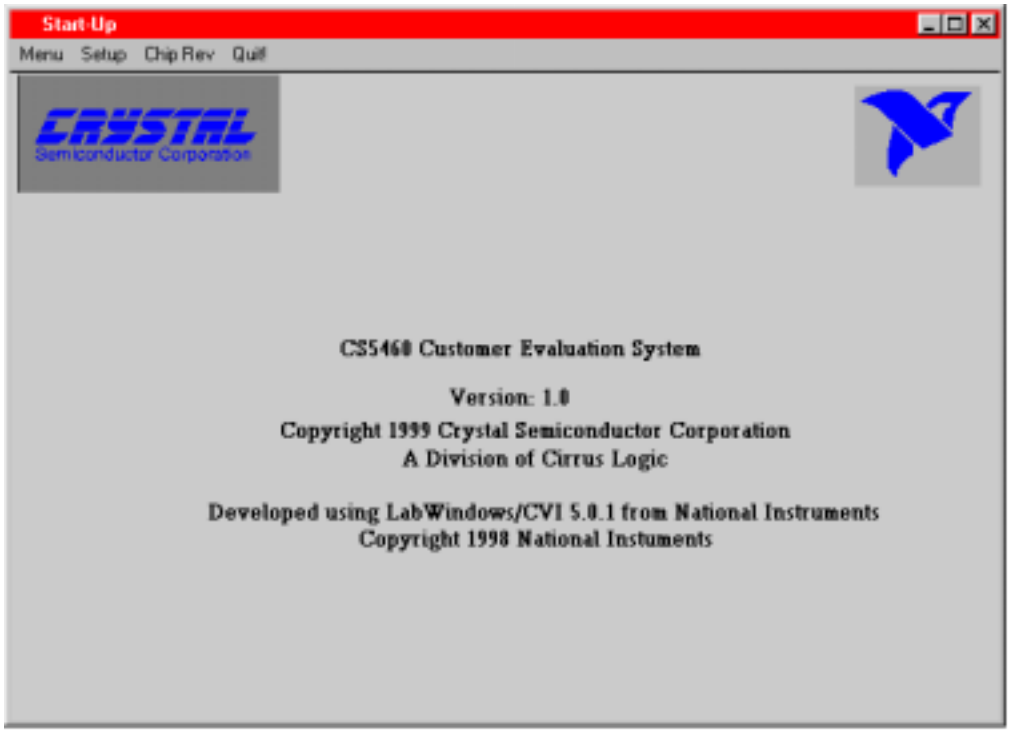


Figure 5. Start-Up Window

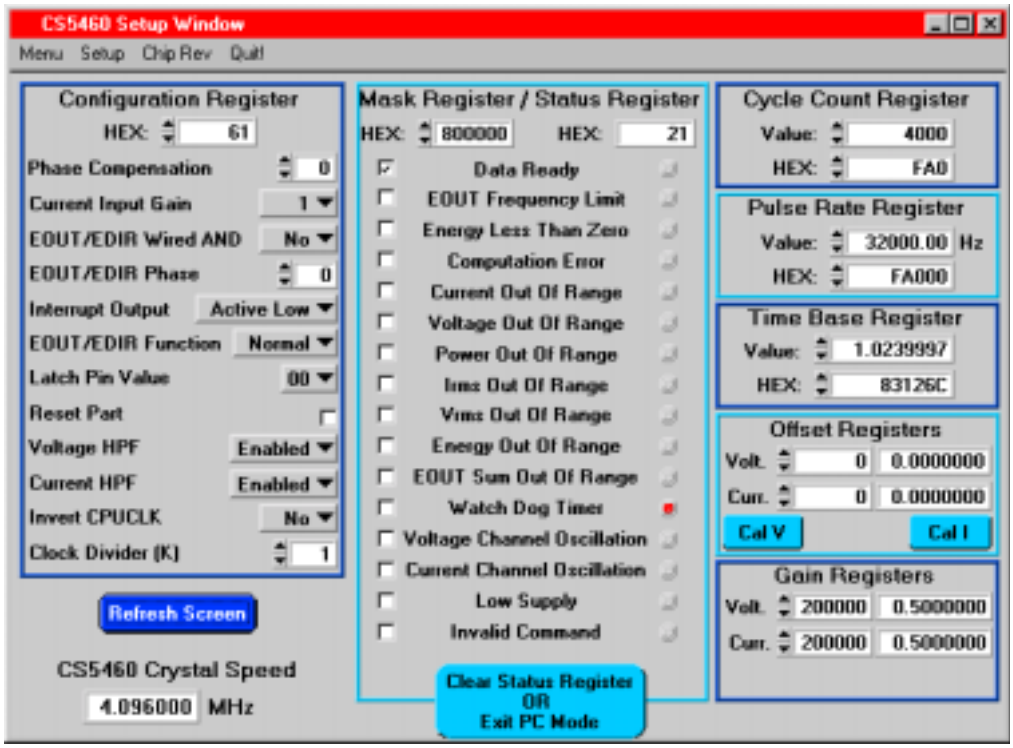


Figure 6. Setup Window

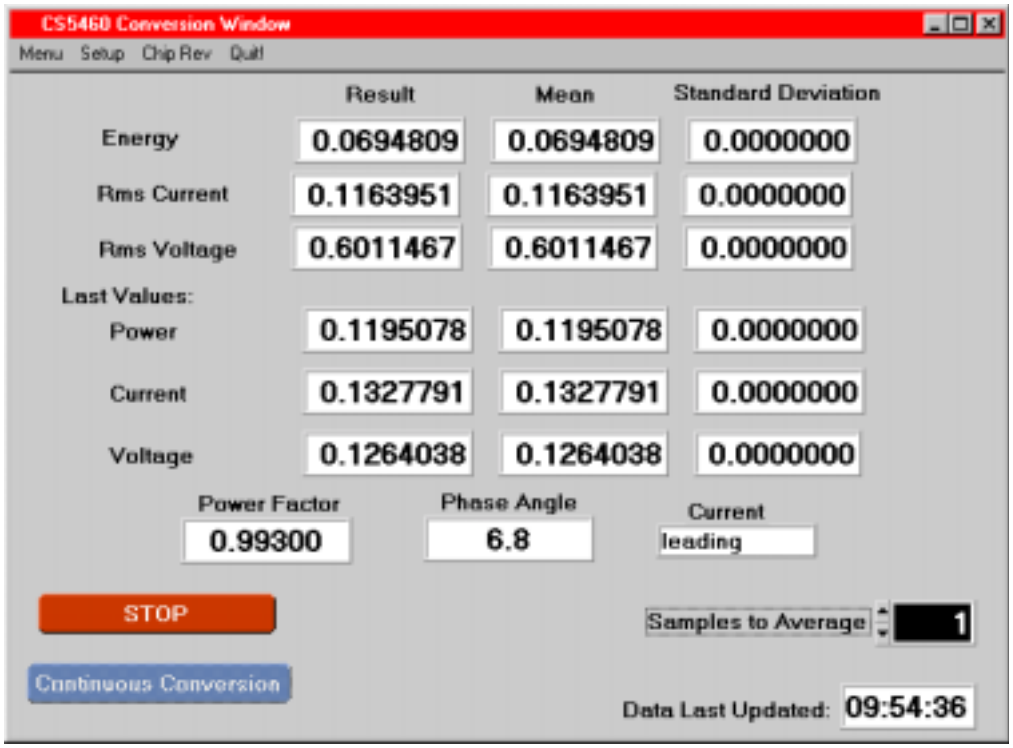


Figure 7. Conversion Window

CRD5460 board. The user must consider the output levels of the sensors being used, to make sure they are within the acceptable input ranges of the CS5460's voltage and current channels. If the user wishes, the jumpers can be set so that the current/voltage inputs are connected to AGND.

NOTE: *This design uses a bipolar differential input sensing configuration, where AGND is at the midpoints of the sensors' input voltage span. Other configurations can be used. Note from the data sheet for the CS5460 that the CS5460 can be configured for differential or single-ended inputs within the acceptable common-mode range of the inputs.*

Looking at Table 2, we see that for normal operation with the LCD display, the user should verify that J10 jumper is connecting pins 1 and 2 of J10, and not pins 2 and 3. (Pins 2 and 3 must be connected when the electromechanical totalizer/counter is used instead of the LCD display.)

4. PC-MODE SOFTWARE

The CRD5460 Reference Design board comes with software and an RS-232 cable to link the evaluation board to a PC. The evaluation software was developed with Lab Windows/CVI™ (Version 5.0), a software development package from National Instruments. The software was designed to run under Windows 95™ or later, and requires about 3MB of hard drive space (2MB for the CVI Run-Time Engine™, and 1MB for the evaluation software). After installing the software, read the readme.txt file for any last minute updates or changes.

Installation Procedure

1) Turn on the PC, running Windows 95™ or later.
2) Insert the Installation Diskette #1 into the PC.
3) Select the Run option from the Start menu.
4) At the prompt, type: A:\SETUP.EXE <enter>.
5) The program will begin installation.
6) If it has not already been installed on the PC, the user will be prompted to enter the directory in which to install the CVI Run-Time Engine™. The Run-Time Engine™ manages exe-

cutables created with Lab Windows/CVI™. If the default directory is acceptable, select OK and the Run-Time Engine™ will be installed there.

7) After the Run-Time Engine™ is installed, the user is prompted to enter the directory in which to install the CRD5460 software. Select OK to accept the default directory.

8) Once the program is installed, it can be run by double clicking on the CRD5460 icon, or through the Start menu. The user should first power up the board and then run the software.

9) Note that, beneath the main installation directory, a sub-directory called "src_code" will be created which contains the source code for the AT89C52 Microcontroller. (This is equivalent to a standard 87C52, which is an 8051 core-based microcontroller.) The source code was compiled using the Franklin Advanced Development System (Release VIII) available from Franklin Software, Inc.

NOTE: *The software is written to run with 640 x 480 resolution; however, it will work with 1024 x 768 resolution. If the user interface seems to be a little small, the user might consider setting the display settings to 640 x 480. (640x480 was chosen to accommodate a variety of computers).*

4.1 Using the PC-Mode Software

Before launching the software, the user should set up the CRD5460 evaluation board with acceptable DIP switch settings on S3 (for PC mode, the DIP settings would be "00000"), and connect it to an open COM port on the PC using the RS-232 serial cable. Then, once the board is powered on (i.e, once the AC voltage supply is connected to the board), the user can start the PC Mode software.

If the board is powered on (or hardware reset) while the DIP switches are set to "00000," then the CS5460 registers will be written with the firmware-controlled default settings. This is because "00000" is considered an even DIP switch setting (see Table 1). If the user wants to run PC Mode using calibration settings from the EEPROM, the user should power up the board in one of the odd DIP switch settings, such as "00001," and then switch to "00000" to initiate PC Mode. See section *9. Resetting the CRD5460* for more specifics on resetting the CRD5460.

When the software is launched, the Start-Up window appears first (Figure 5). This window contains information concerning the software's title, revision number, copyright date, etc. At the top of the screen is a menu bar which displays user options. The menu bar item Menu is initially disabled to prevent conflicts with other serial communications devices, such as the mouse or a modem. After selecting a COM port, the Menu item will become available.

4.2 Selecting and Testing a COM Port

Upon start-up, the user is prompted to select the serial communications port which will interface to the CDB5460 board. To select the COM port, pull down the Setup menu option, and select either COM1 or COM2.

Once the serial link is established between the PC and the evaluation board, the user is ready to access the internal registers of the CS5460, collect data, and perform analysis on the collected data.

4.3 Register Access in the Setup Window

The Evaluation software provides access to the CS5460's internal registers in the Setup Window (Figure 6). The user can enter the Setup Window by pulling down Menu and selecting Setup Window, or by pressing F2 on the keyboard.

In the Setup Window, all of the CS5460's registers are displayed in hexadecimal value, and also decoded to provide easier access. Refer to the CS5460 data sheet for information on register functionality and definitions.

4.3.1 Refresh Screen Button

The user should see several push-buttons icons on the screen at this time. In the lab-Windows environment, left-clicking on these buttons with the mouse will initiate the indicated action. The Refresh Screen button will update the contents of the screen by reading all the register values from the part. This usually takes a couple of seconds, but it is a good idea to press the Refresh Screen button when entering the Setup Window, or after modifying any registers to reflect the current status of the part.

4.3.2 CS5460 Crystal Frequency

The CS5460 accepts a wide range of crystal input frequencies, and can therefore run at many different sample rates. The crystal frequency being used on the CRD5460 runs at 4.096MHz, which is the default setting of this field. The user should never have to change this field unless a new crystal (with different frequency) is mounted on the board.

4.3.3 Configuration Register

In the Configuration Register box, the contents of the Configuration Register can be modified by typing a hexadecimal value in the HEX: box, or by changing any of the values below the HEX: box to the desired settings.

NOTE: *When changing the value of the reset bit to '1' (RS, bit 7 in the Configuration Register), the part will be undergo a 'software reset,' and all registers will return to their default values. Press the Refresh Screen button after performing a reset to update the screen with the new register values.*

NOTE: *Although the CRD5460 software allows the user to modify any of the bits in the Configuration Register, changing certain bits may cause the software and board to behave erratically. For the CRD5460 Power Meter Reference Design to function properly, the Interrupt Output function should be set to the default Active Low, and the Eout / Edir Function should be set to the default Normal.*

4.3.4 Clear Status Register Button

At the bottom of the Configuration Register box, the user should see a button labeled "Clear Status Register." If the S3 DIP switch is set to PC-Mode, then left-clicking on this soft button (with the mouse) will clear the status register, with the exception of the Watch-Dog Timer bit.

4.3.5 Exit PC Mode Button

Another button located at the bottom of the Status Register box is called "Change DIP SW Then EXIT PC mode" If the user wants to get out of PC-Mode and use the CRD5460 in Stand-Alone mode, the user should first change the S3 DIP switch to the desired Stand-Alone setting and then, using the mouse, left-click on this button. Then the PC will relinquish control of the CS5460 and allow the CRD5460 to function in Stand-Alone Mode.

4.3.6 Mask Register / Status Register

The Mask and Status Registers are displayed in hexadecimal and decoded in this box to show the function of each bit. The Mask Register can be modified by typing a value in the HEX: box, or by checking the appropriate check boxes for the bits that are to be masked. The Status Register cannot be directly modified. It can only be reset by left-clicking on the Clear Status Register Button. The HEX: box for this register, and the bit 'indicator lights' are displayed only. An indicator light that is on (red-colored) means that the corresponding bit in the Status Register is set (except the Invalid Command bit, which is inverted). If a bit is not set, then the color of the indicator light will be gray.

NOTE: *The value present in the Mask Register may be changed by the software during certain operations to provide correct functionality of the CRD5460 board. Therefore, the user should not be surprised to find that certain bits in the Status Register have changed state since the last time the user cleared the Status Register.*

4.3.7 Cycle Count / Pulse Rate / Time Base Registers

These three boxes display the values of the Cycle Count, Pulse Rate, and Time Base Registers in both hexadecimal and decimal format. All three registers can be modified by typing a value in the corresponding Value: or HEX: box.

4.3.8 Offset / Gain Registers

In the Offset and Gain Register boxes, the offset and gain registers for both channels are displayed in hexadecimal and decimal. These registers can be modified directly by typing the desired value in the hexadecimal display boxes.

4.3.9 Performing Calibrations

AC-Calibration algorithm can be initiated by the user in Stand-Alone Mode. Refer to section *6. Calibration*.

4.4 Conversion Window

The Conversion Window (Figure 7) allows the user to see the results of continuous conversions on all six data registers, perform data averaging, and view the computed phase angle and power factor. The Conversion Window can be accessed by pulling down the Menu option, and selecting Conversion Window, or by pressing F3 on the keyboard. In general, the readings on the conversion screen represent scaled values, which are dependent on the values in the gain calibration registers.

4.4.1 Continuous Conversions Button

This button initiates the CS5460 IC into continuous conversion mode (C=1, see CS5460 datasheet) whereby it will continuously sample the instantaneous voltage and current from the sensors, and compute the resulting power. The screen is updated at the end of each computation cycle, and so only the most recent instantaneous voltage, current, and power values will be displayed. In addition, the program will update RMS-current, RMS-voltage, and energy calculations after each computation cycle. The duration of each computation cycle is based on the number in the CS5460's cycle-count register. For a cycle-count of 4000, the duration of the computation cycle is one second. Also note that the power factor is computed, as well as the phase angle between current and voltage, with an indicator on whether the current leads or lags the voltage. Finally, the conversion window provides 'mean' and 'standard deviation' values for each quantity so that an average of the results of each computation cycle can be computed. The user can change the value in the "samples to average" box in order to set the number of computation cycles to average. (Remember this only sets the number of computation cycles to average; it does not set the value of the cycle-count register). Clicking on the 'Stop' button terminates this data collection process. There are some speed limitations when operating in continuous mode. For example, the value of the cycle-count register cannot be set to a value which exceeds the thru-put capability of the RS-232 connector. If any of these limitations are exceeded, the user will be prompted to change some settings before proceeding.

5. OVERFLOW ERROR: REVS A-C

Before discussing calibration, the user should be made aware of an error in first two silicon revs (revs A and B) of the CS5460. Refer to the CS5460 Errata for further details. To work around this problem with the CS5460 in the CRD5460 Reference Design, the following deviation is made from normal operation:

The values of the voltage/current gain registers are set to one-half the normal value. The nominal gain setting is 0.5, instead of 1.0. The firmware on the 87C52 micro-controller has been programmed to compensate for this divide-by-two action in the gain registers--Before the values of the CS5460's output registers are displayed to the screen and/or liquid-crystal display, the microcontroller will

multiply the results by 2 in order to get the true voltage/current values (and multiply by 4 to get the energy and power values). Note that the actual results in the CS5460 registers will be at one-half (or one-fourth) of their normal value. The measurement results in the Conversion Window of the PC Mode software do not reflect the actual register values inside the CS5460. Before being displayed, these register values are multiplied by 2 (or 4). However, the register values that are displayed in the Setup Window of the PC Mode software do reflect the actual register values of the CS5460.

6. CALIBRATION

In this section, the 'on-board' calibrations (in Stand-Alone Mode) are discussed. For the CRD5460 Reference Design, performing the on-board AC calibrations in Stand-Alone Mode is the preferred mode of calibration.

It is worth noting here that once a calibration has been completed, the data in the calibration registers of the CS5460 can be saved to the on-board EEPROM for later use. This is discussed in a later section *6.3 Saving Calibration Results***.** (See Table 1 to find the DIP switch setting that will write the calibration values to the EEPROM.)

6.1 Theory of Stand-Alone AC Calibration

The CS5460 IC is designed to perform an automatic internal *offset* calibration and *gain* calibration for both the voltage and current channels. These are DC calibrations. The CRD5460 Reference Design provides an alternative way to calibrate the power meter so that an AC signal can be used as the calibration signal. In practical applications such as this reference design (AC power meter), an AC calibration usually proves to be more useful than a DC calibration. The microcontroller on the CRD5460 has been programmed to sample the AC (RMS) value of the calibration signal that is presented across the voltage and current inputs, and program the voltage/current gain values accordingly. It is assumed that the voltage/current channel high-pass filters are turned on inside the CS5460. This removes all DC-content from the sensed voltage/current signals, and so no offset calibration is performed. This section describes the AC calibration method.

The AC-calibration signals for voltage and current channels must be supplied by the user. The calibration signals can be can be taken off of the on-board voltage and current sensors that interface to the power-line, or alternatively, they can be applied to the CRD5460 through the J8 header inputs. For the calibration procedure described here, we need to make some assumptions about how/where the calibration signals are applied. These assumptions are described in Sections 6.1.1 through 6.1.4. Before the CRD5460 is shipped from the factory, it is calibrated in this way.

6.1.1 Source of Calibration Input Signals

We assume that the voltage and current calibration signals are from a power-line source, and are being fed through the on-board voltage and current sensors. This means that the voltage input is taken to be coming from the line input transformer, and fed to the CRD5460 board through the J2 connector, and the current signal is going through a 1/2-turn winding on the on-board current transformer (T1). The J4, J5, J6, and J7 headers should have their jumpers set to the on-board sensor inputs. (See Table 2.)

6.1.2 Level of Calibration Input Signals

We assume that the nominal full-scale level of the voltage signal will be 150V (RMS) and the full-scale level of the line-current will be 5A (RMS). The user must be able to supply these signal levels. These levels were chosen to support metering of a standard 115VAC line. [Note that, in this document, the term 'VAC' has the same meaning as the term 'RMS-volts.' And so 150VAC = 150 V (RMS).] The maximum level of 150 VAC will al-

low for linearity of 0.1% for line voltages between 75VAC - 150VAC (see CS5460 datasheet on linear accuracy range of voltage channel). The maximum current level of 5A (RMS) allows for linear line-current measurements on the current channel in the load curve range between 0.03A and 5A.

6.1.3 High-Pass Filters Enabled

Because the CRD5460 is intended to be an AC power meter, the CS5460's internal high-pass filters should be on, which renders the CS5460 insensitive to DC input signals. The high-pass filters for the voltage/current channels are turned on by default on the CRD5460 Reference Design at power-up. Since the high-pass filters remove all of the DC content from the sensed voltage and current signals, there is no need to do current/voltage offset calibrations. Therefore, only gain calibrations are performed, and they are AC-gain calibrations, not DC.

6.1.4 CS5460 Input Amplifiers

It is assumed that the input PGAs on the voltage and current channel are both at the "x10" gain setting. This is the default of the CRD5460 at power-up. In the "x10" setting, the maximum input voltage level that should be applied voltage and current channel inputs is ~176 mV (RMS). To allow for some over-ranging capability on the meter, we will adjust the on-board sensor gains so that the full-scale power-line values of 150V (RMS) and 5A (RMS) will present a 150mV (RMS) signal across the inputs of the voltage and current inputs. This will allow for some over-ranging capability. For example, on the voltage channel, the meter will be able to monitor line voltages up to ~176VAC before the meter's input range is exceeded.

6.1.5 AC-Calibration Diagram

Figure 8 is a diagram of a typical calibration setup. The user-supplied variac and adjustable load is one way for the user to provide the 150V calibration signal, as well as the 5A (RMS) calibration signal. The user could adjust the load so that the 150 volts (RMS) source generates 5A (RMS) of current through the line. At 150V, the transformer that comes with the CRD5460 Reference Design can still be used to supply power to the board and to deliver the voltage sense signal. However, the user is advised not to expose this transformer to line-voltages greater than 170VAC for extended periods of time. Note that it is not necessary to provide both calibration signals at the same time.

6.2 Procedure for Stand-Alone AC Calibration

The following AC-Calibration procedure uses a 'safe' switching sequence, which assures that the user will not enter any undesirable modes of operation and/or enter into modes in the wrong order.

1) Turn the R5 POT fully counter-clockwise so that the resistance between terminals 2 and 1 is minimized.
2) Turn the R1 and R2 POTs fully counter-clockwise so that the resistance between terminals 2 and terminal 1 (from wiper to AGND) is minimized.
3) Set the J4 and J5 jumpers so that they connect the current channel inputs of the CS5460 (Iin+ and Iin-) to the on-board current burden resistor (R5). Set the J6 and J7 jumpers so that they connect the voltage channel inputs of the CS5460 (Vin+ and Vin-) to the on-board voltage sensors. (They should now be connected to wipers of R1/R2 POTs.)
4) Activate power on the CRD5460 with the S3 DIP switch set to "00010" (Power Factor and Lead/Lag status displayed). This activation of power will apply the voltage sense signal to the CRD5460. Also, the user should apply the load current signal to the CRD5460 by engaging the load to the line voltage.
5) With a full load current of 5A, measure the voltage across terminals 1 and 2 of the R5 POT,

and adjust R5 until this voltage measures 150mV (RMS).

6) With 150 volts (RMS) across the voltage transformer, measure the voltage across terminals 1 and 2 of the R1 POT with a voltmeter, and adjust R1 until this voltage measures 0.075 V (RMS). Then measure the voltage across terminals 1 and 2 of the R2 POT with a voltmeter, and adjust R2 until this voltage measures 0.075 V (RMS).

7) Set the S3 DIP switches to "00110" (RMS-current and RMS-voltage displayed).

8) Set the S3 SIP switches to "01110." This initiates the current gain calibration. Wait for one computation cycle to complete. The LCD should display the word "IGAINCAL" and also a hex number that is equal to the current gain cal value. (Waiting for 5 seconds will insure that the computation cycle has completed. Depending on the value in the cycle-count register, the minimum waiting time will vary.)

9) Set the S3 DIP switches to "00110" (RMS-current and RMS-voltage displayed).

10) Set the S3 DIP switches to "00100" (cumulative energy and energy for most recent computation cycle displayed).

11) Set the S3 SIP switches to "01100." This initiates the voltage gain calibration. Wait for one computation cycle to complete. The LCD should display the word "VGAINCAL" and also a hex number that is equal to the voltage gain cal value. (Waiting for 5 seconds will insure that the computation cycle has completed. Depending on the value in the cycle-count register, the minimum waiting time will vary.)

12) Set the S3 DIP switches to "00100" (cumulative energy and energy for most recent computation cycle displayed).

The system is now calibrated. A value of 0.6 in the CS5460s RMS-Current register will now indicate that a current of 5A (RMS) is present through the line. A value of 0.6 in the RMS-voltage register will indicate a line voltage of 150V (RMS). With the calibration signals still applied to the CRD5460, the user can verify that the current/voltage measurements were calibrated to 0.6 by setting the S3 DIP switches to "00101" and noting that the RMS current and RMS voltage values should be measuring very near to 0.6. If the user goes into PC Mode and goes to the Setup Window in the PC-Mode software program, and uses the mouse to left-click on the "Refresh Screen" button, the user should see new calibration values in the "Volt." and "Curr." gain register boxes.

Once the CRD5460 is powered down, the values in the CS5460s calibration registers will be lost. If the user wants to save these calibration values to the on-board EEPROM (U6) so that these values can be restored after a power-down, execute the next section *6.3 Saving Calibration Results* immediately. We say 'immediately' because there is a risk of accidentally re-running the calibrations for voltage/current gain if the S3 DIP switch is set back to a calibration setting. The user must be careful not to set the DIP switches back to these calibration modes until the calibration values that were just obtained are no longer needed, or have been saved to the EEPROM.

6.3 Saving Calibration Results

Once calibration values have been obtained, these calibration values (in the gain registers) can be saved for future use by writing them to the on-board EEPROM (U6). Referring to Table 1, setting the S3 DIP switches to certain codes will cause the values in the calibration registers (on the CS5460) to be written to the EEPROM. If the user wants these calibration values to be read back into the CS5460 registers, that can be done by resetting (or re-booting) the CRD5460 with the S3 DIP switch to certain odd-numbered values. This is useful if the user wants to power down the CRD5460 for a

period of time, but does not want to loose the results of a calibration.

To write the calibration register values that are presently in the CS5460 to the EEPROM, the user should follow the following S3 switch sequence. We assume that the user is starting from Stand-Alone Mode and S3 DIP switches are set to "00100," so that the LCD is displaying RMS-Voltage/RMS-Current.

1) Set the S3 DIP switches to "00101" (RMS-current and RMS-voltage displayed).
2) Set the DIP switches to "10101." This will initiate the microcontroller to write the calibration values in the CS5460 to the EEPROM.
3) Set the S3 DIP switches to "00101" (RMS-current and RMS-voltage displayed).
4) Set the S3 DIP switches to "00100" (RMS-current and RMS-voltage displayed).

Once the above procedure has been completed, the user can reboot the CRD5460 in any of the valid odd-numbered DIP switches (see Table 1) and the calibration values will be recalled from the EEPROM and copied the CS5460's current/voltage gain registers. The cal values will remain in the EEPROM until they are overwritten.

6.4 Interpreting Calibration Results

Once the above AC Calibration has been performed, we will know that a value of "0.6" in the CS5460's RMS-Current register will indicate a line current of 5A (RMS), and a value of "0.6" in the RMS-voltage register will indicate 150V (RMS) across the line. Conversion factors can be generated for current and voltage. Define a voltage conversion constant, K_1, whose value would be 150/0.6 = 250. Then define a current conversion constant, K_2, whose value would be 5/0.6 = 8.33. These constants can be used to convert the readings from the instantaneous Current/Voltage and RMS-Current/-Voltage registers into readings that are quantified in volts and amps.

7. CUMULATIVE ENERGY READING

One may have noticed in Table 1 that the "00011" and "00100" DIP switch settings will display two energy readings. The energy value in the bottom-half of the LCD display represents the energy calculated over the most recent calculation cycle. The energy value on the top-half of the LCD display displays the value of the 'cumulative energy'. The cumulative energy represents the total energy that has been monitored by the meter.

The cumulative energy value is stored in the on-board EEPROM (U6). Storing the cumulative energy in this non-volatile memory allows the CRD5460 to be used to monitor energy that is consumed over a long period of time, and keep this information even when the power is turned off. When the CRD5460 is powered back up, the cumulative energy reading is restored. This cumulative energy reading is retained until the EEPROM reset command is given from the S3 DIP switch, or if the register that holds the cumulative energy overflows.

To get a sense of how much energy is represented by the cumulative energy value, note the following: After each computation cycle, the 87C52 microcontroller takes the energy calculation from the CS5460 and adds it to a running energy sum. As the sum gets larger and larger, it will eventually overflow. The cumulative energy value will increment by one every 10th time that the 24-bit energy sum overflows.

To convert the cumulative energy reading to Watt-hrs, the following computation is performed: First we assume that the Stand-Alone AC Calibration procedure was performed (see section *6. Calibration*). After calibration is performed, a value of 0.6 in the RMS-Current register indicates a line current of 5A (RMS) and 0.6 in the RMS-voltage register indicates a line voltage of 150V (RMS). If the voltage and current are in phase (pure resistive line load), then the energy consumed over a one-second

interval is 5A x 150V = 750 Watt-seconds. Since the voltage is in phase with the current, the value in the energy register will be equal to the product of the RMS-Current and RMS-Voltage register values. And so the energy register value is 0.6 x 0.6 = 0.36. The above calibration assumes that the CRD5460 was set to default firmware values, which means that the computation cycle time is one second. Therefore, a value of 0.36 in the energy register (after one computation cycle) corresponds to 750 Watt-seconds of energy.

The user should note that the running energy sum will overflow when the sum reads or exceeds 1.0, and so the energy register will overflow for every 2083.3 Watt-seconds of energy that is consumed, because 2083.3 = 750 x (1/.36) (approximately). The cumulative energy value that is stored on the EEPROM (and displayed on the LCD) increments its register value by one every 10th time that the running energy sum overflows. *Therefore, one increment of the cumulative energy reading corresponds to 10 x 2083.3 Watt-seconds = 20833 Watt-seconds, or 5.785 Watt-hrs.*

As an example, suppose that the cumulative energy reading seen on the top-line of the LCD display reads 00000026. (The energy reading is a decimal number.) The total energy consumed would then be calculated as 26 x 5.785 Watt-hrs = 150.4 Watt-hrs.

One should also note that if the AC-Calibration has been performed, then the real (average) power (in Watts) over one computation cycle is calculated by multiplying the value in the energy register by 2083.3. For this power calculation, the cycle-count register value must be set to 4000.

8. ELECTROMECHANICAL COUNTER

The CRD5460 Reference Design has an optional footprint location for an electromechanical counter/totalizer monitoring device. Instead of the LCD display, the electromechanical counter can be placed on the board as the on-board output device. The user can connect the counter to the board using wires if it is desirable to mount the counter elsewhere. Note that the counter and the LCD display can be wire-connected to the CRD5460 simultaneously, but only one of the two devices can be mounted on the CRD5460 assembly at one time. If the user is interested in obtaining an electromechanical counter, they might considering ordering a GO 635 132 Counter, available from Danaher Controls (5211 Parkcrest Drive Suite 205 Austin, TX 78731 USA).

The electromechanical counter provides a non-volatile way to record and display the accumulated energy. It is driven by the microcontroller. *To use the electromechanical counter, the user must set the J10 jumper to the EOUT side, so that the jumper is connecting pins 2 and 3 of J10.* With the electromechanical counter mounted onto the CRD5460 board, the user should be able to boot up the CRD5460 in an appropriate DIP switch setting for electro-mechanical counter operation. (See Table 1 for proper DIP switch settings).

Referring to Figure 3 of the schematic, we see that the input to the electromechanical counter/totalizer circuitry is from the P3-0 line from the microcontroller. The microcontroller firmware is configured such that the electromechanical counter will increment every time an EOUT pulse is sensed on the microcontroller's /INT1 line. Therefore, the electromechanical counter displays an exact count of the number of EOUT pulses are generated over time.

The electromechanical totalizer reading is similar (but not identical) to the total cumulative energy value that was discussed in the last section. Since the electromechanical counter will increment by one with every EOUT pulse received, we can determine the amount of energy that is represented by one 'count' on the counter by calculating how much energy is represented by one EOUT pulse from the CS5460. This is done by looking at the

contents of the pulse-rate register, the offset/gain registers, and determining the voltage/current levels that are represented by full-scale readings in the instantaneous voltage/current registers. (The full-scale readings are present when the registers measure 0.9999999.)

The CRD5460's boot-up default value for the pulse-rate register is 0x3C0 HEX. This corresponds to a pulse-rate frequency of 30Hz (see CS5460 data sheet). This implies that the EOUT pin will issue 30 pulses per second *if* the voltage and current channels are given full-scale inputs, *and if* the values in the CS5460's voltage/current gain registers are set to unity. If the user calibrates the CRD5460 using the calibration procedure that was described in this document, then the voltage and current RMS registers will read 0.6 when the line-voltage and line-current are at 150V (RMS) and 5A (RMS) respectively. This implies that the full-scale DC line-voltage and line-current values will equal 250 volts and ~8.3333 amps respectively. We multiply this voltage and current together, we see that the instantaneous power in this situation will be ~2.0833 kilo-Watts. If the CS5460 generates 30 pulses over one second, and if the energy consumed over that one-second period is 2.0833 kWatt-sec, then each pulse would represent 69.444 Watt-sec, or 0.000019290 KW-hrs. *However,* these numbers need to be further adjusted: The voltage and current gain registers are nominally 0.5 for the CRD5460 (see section *5. Overflow Error: Revs A-C*), and so we must multiply the above result by (1/0.5)(1/0.5) = 4. *Therefore, the actual amount of energy that is represented by one pulse is 277.778 Watt-sec, or 0.000077161 kW-hrs. This is also the amount of energy that is represented by one increment on the electromechanical counter.* Even though the pulse rate register is set for 30 Hz, the frequency of EOUT would actually be at (30) x (0.5) x (0.5) = 7.5 Hz when the values of the voltage/current registers read at 0.9999999.

Note that in the above discussion, we assumed that the voltage/current gain values were set to the default value of 0.5. Once the user calibrates the CRD5460, the user should note the new calibration values of the voltage/current gain registers, and use these instead of the nominal '0.5' values. The actual gain register values obtained from calibration should be used when calculating the amount of energy per pulse.

One final note about the electromechanical counter is that its maximum increment rate is ~10 increments per second. When using the electromechanical counter, the Pulse-Rate Register value should not be increased to a frequency that would cause more than 10 clicks per second on the counter. If the user used the AC calibration process described in this document, then the user should remember to always keep the Pulse-Rate Register value below 110Hz. A setting above this frequency may result in inaccurate results from the electromechanical counter.

9. RESETTING THE CRD5460

There are two ways to reset the CRD5460 Reference Design board: hardware reset and software reset. Pressing on S1 (or cycling power) will cause an asynchronous hardware reset of the system. The software reset, by contrast, is a synchronous reset which is initiated through the CRD5460 PC Mode software program. Initiating the software reset is done in the PC Mode software by using the mouse to 'check' on the "Reset Part" box. The "Reset Part" box is one of the options inside the "Configuration Register" box in the Setup Window.

The hardware and software resets are different. The *hardware* reset (pressing S1) is just like recycling the power to the board (power supplied through J2). The user should take note of the state of the S3 DIP switches before resetting the CRD5460, because not all DIP switch settings are valid for a reset. Only certain DIP switch settings allow for a successful system reset. See Table 1. It is also im-

portant to remember that when a hardware reset occurs, the state of the S3 DIP switch determines whether the initial values of the voltage/current gain registers are set from the most-recently saved values in the on-board EEPROM, or by the default values in the micro-controller. If the S3 DIP switch is set to a valid *even* value, then a hardware reset (or power cycle on the board) will cause the microcontroller (U7) to write preset default register values to the CS5460. If the S3 DIP switch is set to a valid *odd* value, then at hardware reset/power cycle, the CS5460 voltage/current gain calibration registers will be set to the values that were last saved to the EEPROM (U6). (Note that the "00000" DIP switch setting is considered an even setting.)

Initiating the *software* reset is equivalent to asserting the RS bit in the Configuration Register of the CS5460. It will cause the CS5460 to use its own hard-coded default register values, which were determined when the part was designed. Note that these values may be different than the preset default register values that would come from the microcontroller/EEPROM during a hardware reset of the CRD5460. Also note that the software reset can only be done from PC mode.

The difference between a hardware and software reset can be seen in Figure 4. The hardware reset is seen at the top. When a hardware reset is initiated, the entire CRD5460 goes through its boot-up sequence. A software reset is initiated as one of the RS-232 commands seen in the lower-left hand corner of Figure 4. It is simply one of the commands that is sent to the microcontroller while running in PC Mode. It merely resets the CS5460, not the entire power meter.

To summarize, there are three different reset scenarios:

1) Asynchronous hardware reset (or power cycling the CRD5460) when the DIP switches are set to an *even* (and valid) value. Initial register values come from microcontroller. They are as follows:

Voltage Offset Register: 0 (decimal)

Voltage Gain Register: 0.5 (decimal)

Current Offset Register: 0 (decimal)

Current Gain Register: 0.5 (decimal)

Cycle-Count Register: 4000 (decimal)

Configuration Register: 0x61 (HEX)

Time Base Register: 0x83126C (HEX)

Pulse Rate Register: 0x3C0 (HEX)

2) Asynchronous hardware reset (or power cycling the CRD5460) when the DIP switches are set to an *odd* (and valid) value. The voltage/current offset and gain calibration register values come from the EEPROM, other register values are set as listed above in 1).

3) Synchronous software reset from the CRD5460 software program. Initial register values come from default values that were designed into the CS5460. See CS5460 datasheet for default values.

Each of these reset situations results in a different unique set of initial register values to be written to the CS5460.

10. COMMUNICATION INTERFACE OPTION

As shown in Figure 2, the control/data lines are connected to the P0-0 thru P0-7 interface lines of the 87C52. An optional daughter card which has a CS8900 Ethernet controller is being developed for use with the CRD5460 reference Design (should be available by Dec. 1999). It is connected via the J9 header.

11. THREE-PHASE OPERATION

A future revision of the 87C52 firmware, CRD5460 PC-Mode software and documentation will provide details on poly-phase operation of the CRD5460 Reference Design.

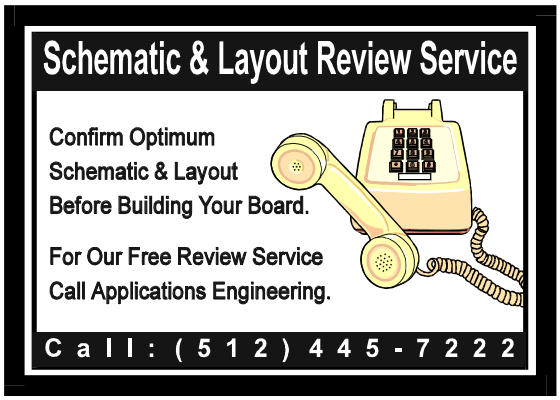
12. SOURCE CODE LISTING

The C-language source code listings for the C-language/assembly language source code for the microcontroller (U7) are included at the end of this document. The microcontroller code is compiled and burned into the 87C52 microcontroller during the manufacture of the CRD5460. The microcode was developed using the Franklin Compiler Developer's Kit (Version 8.63).

13. PCB LAYOUT

The CS5460 should be placed entirely over an analog ground plane with both the VA- and DGND pins of the device connected to the analog plane. Place the analog-digital plane split immediately adjacent to the digital portion of the chip. Figures 9, 10, 11, 12, 13 and 14 show the layout of the CDB5460. A bill of materials is also included.

NOTE: *See Applications Note 18 for more detailed layout guidelines. Before layout, please call for our Free Schematic Review Service.*



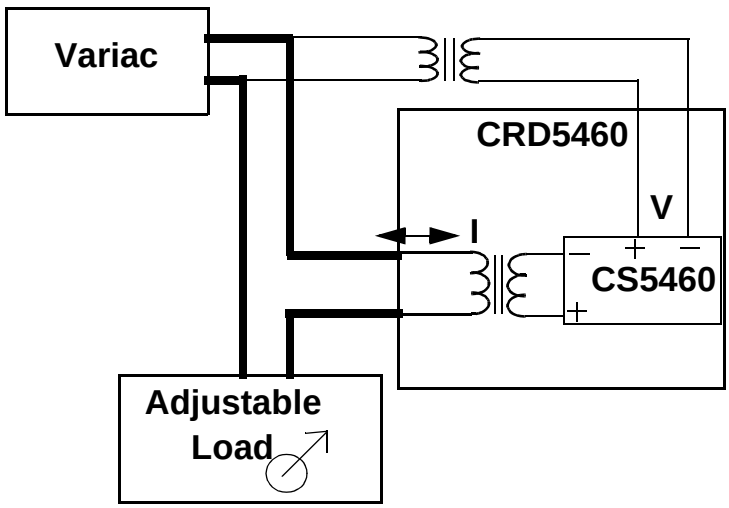


Figure 8. Typical Calibration Setup

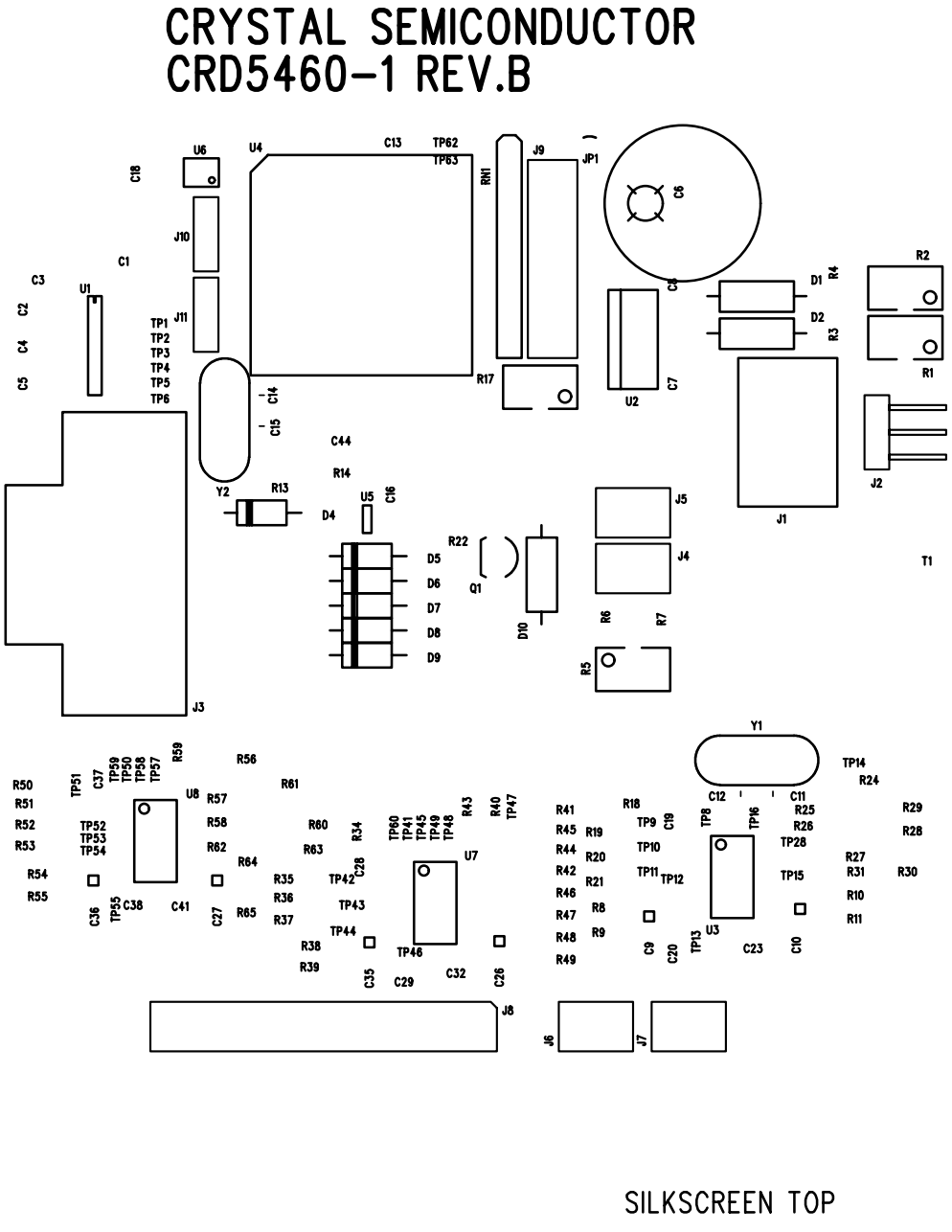


Figure 9. Silkscreen

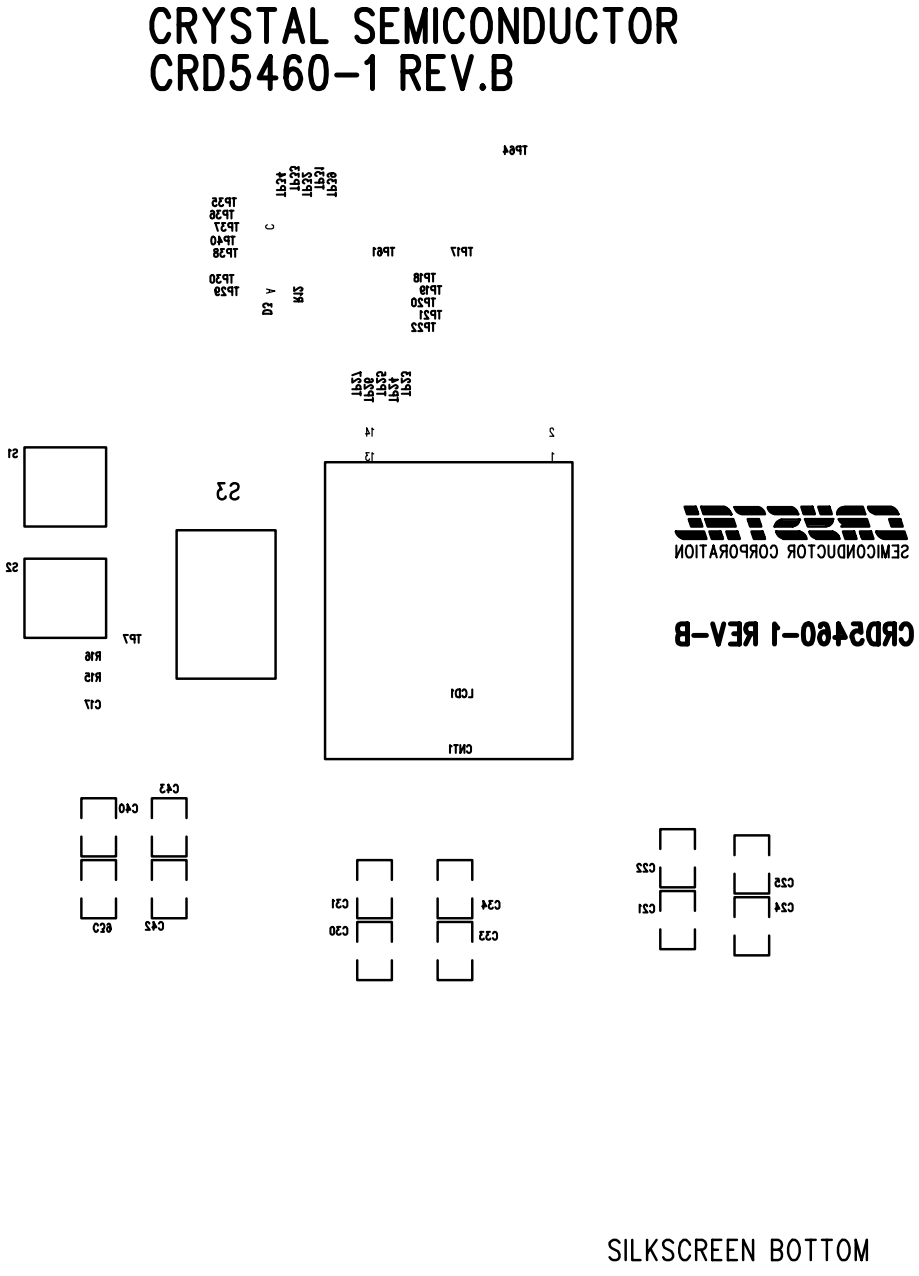


Figure 10. Silkscreen Bottom

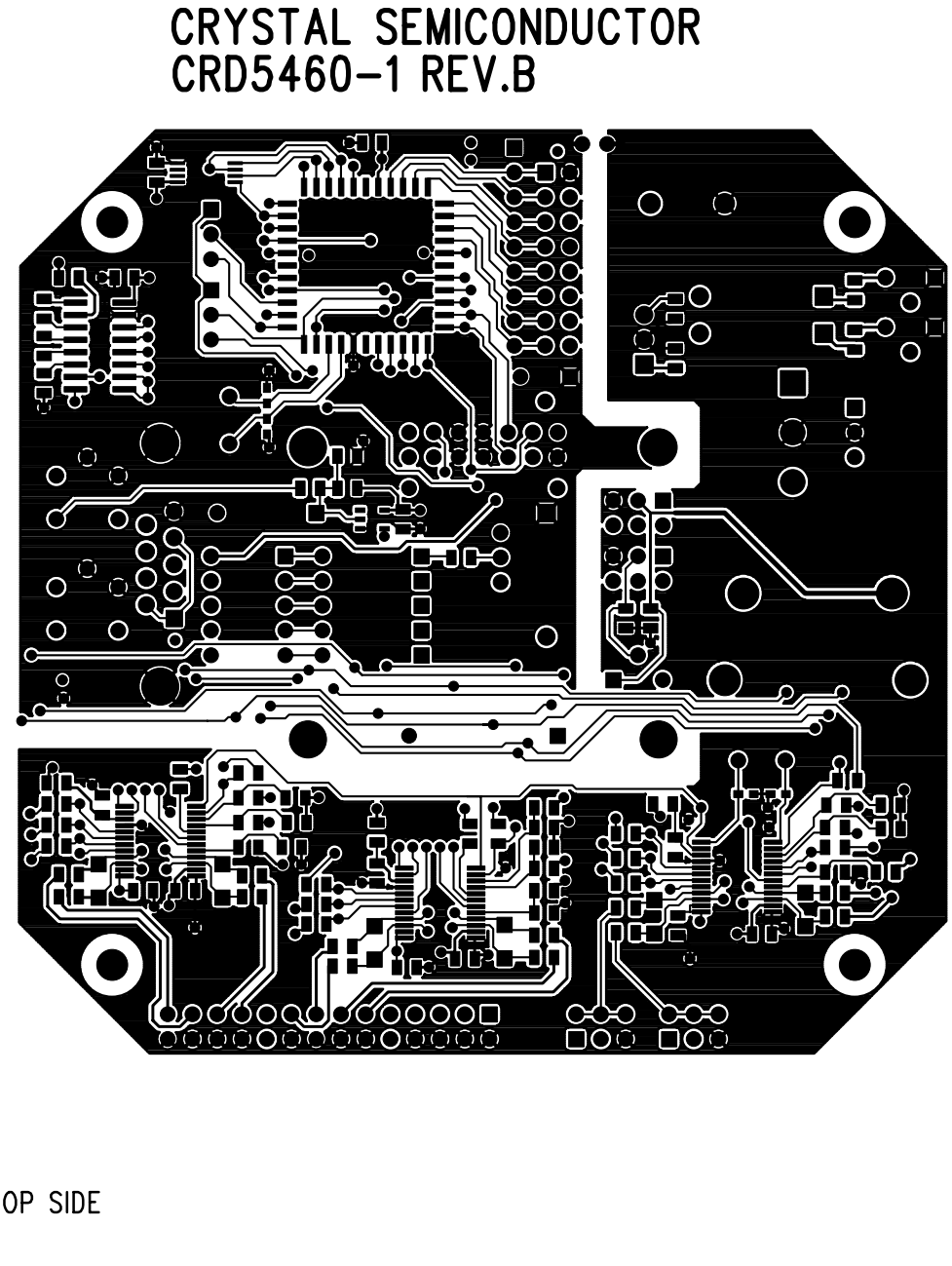


Figure 11. Circuit Side

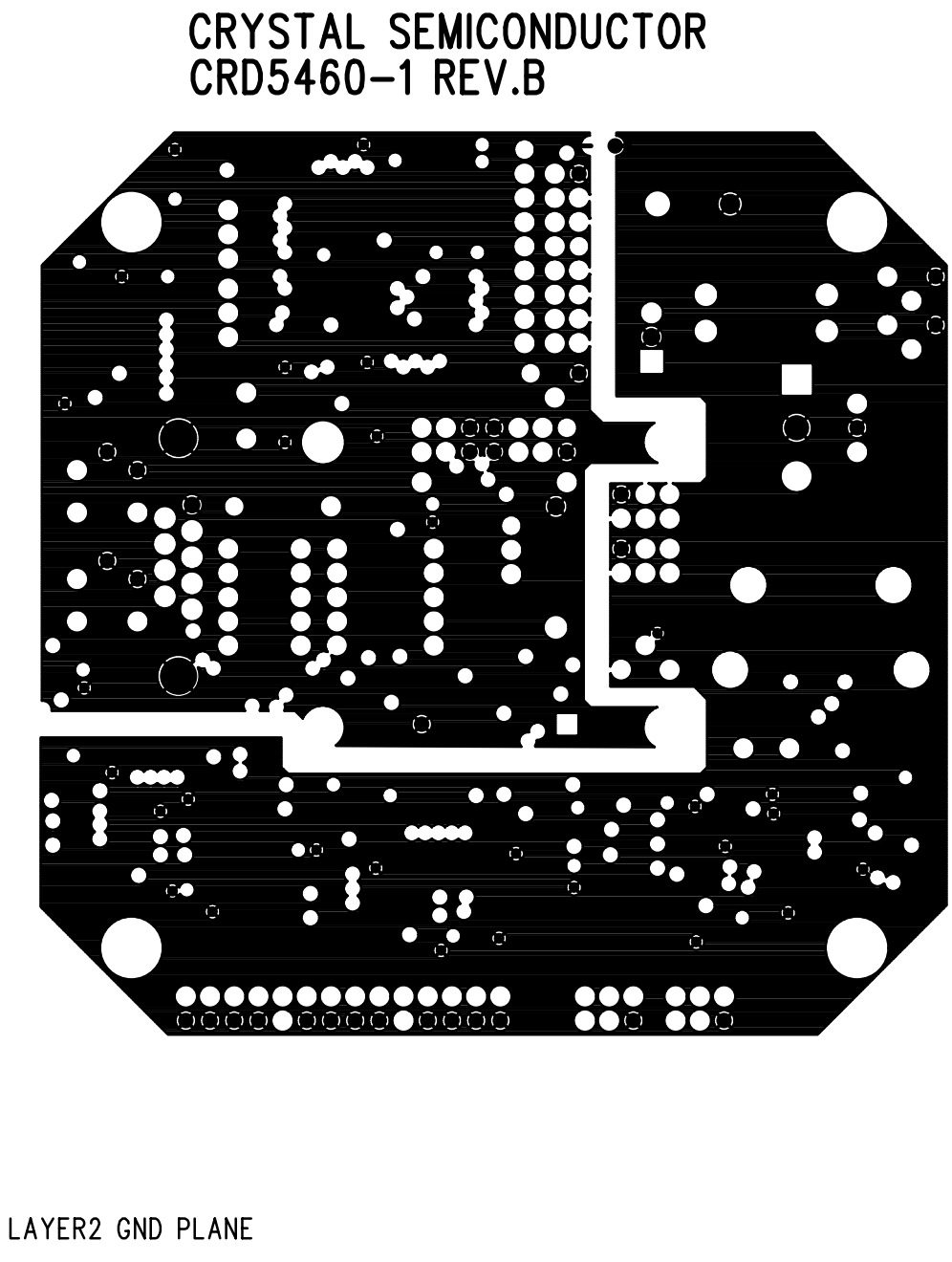


Figure 12. Solder Side

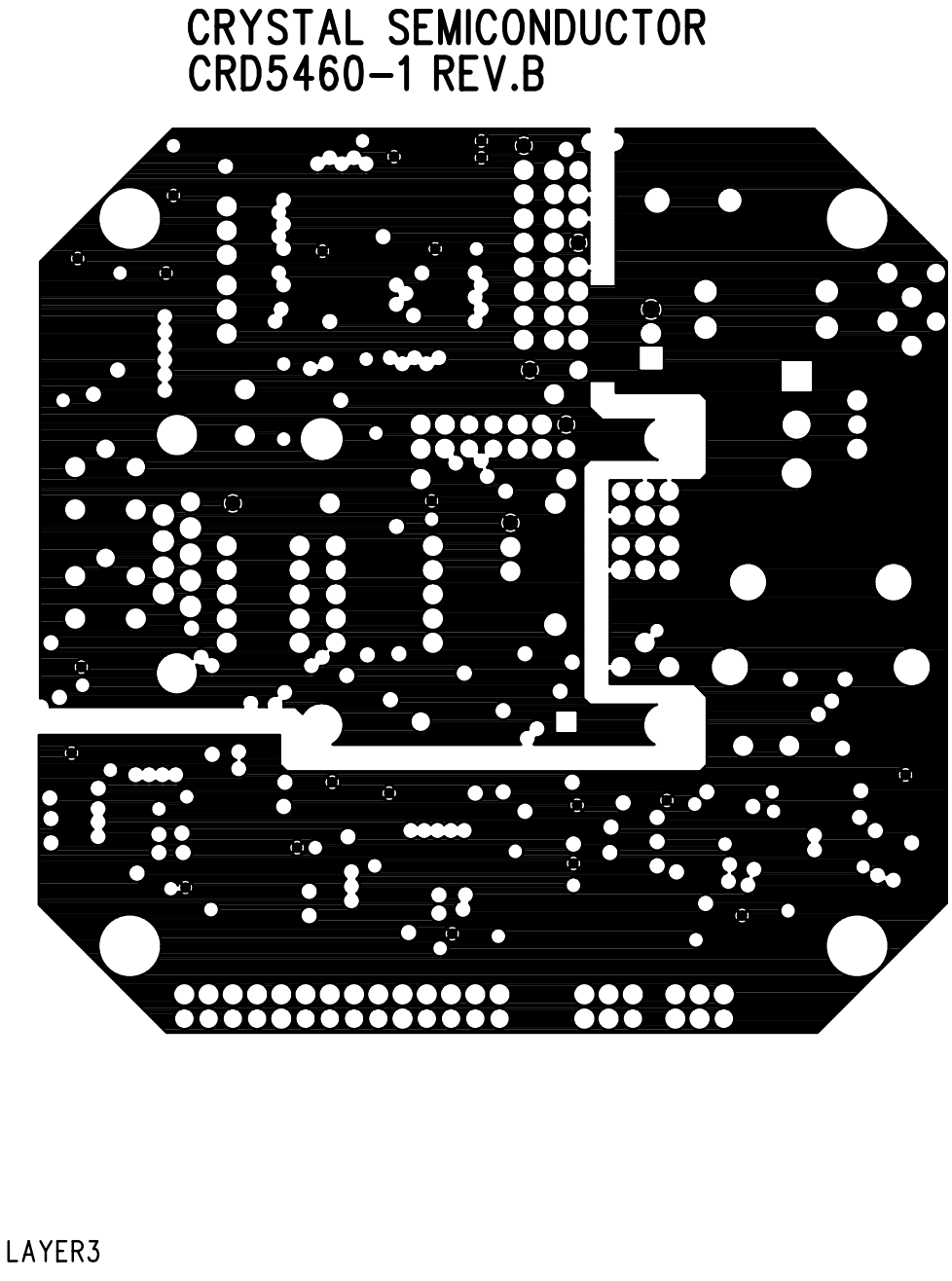


Figure 13. Layer Three

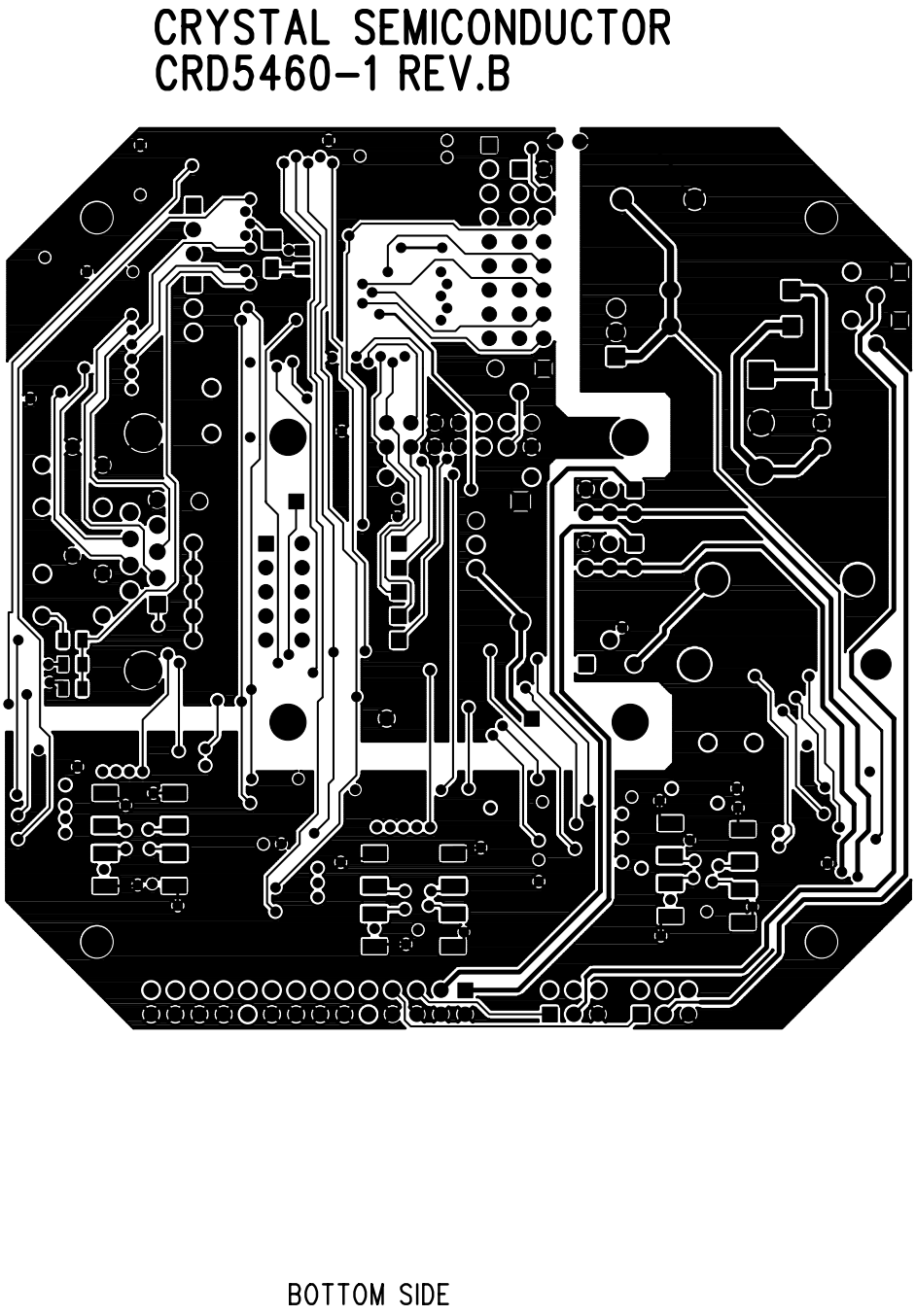


Figure 14. Bottom Side

14. BILL OF MATERIALS

Qty	Reference	Manufacturer	Mfg PN	Description
1	R13	DALE	CRCW0603101FT	RES, 100-OHM, 0603, 1/16W, 1%
3	R6-7 R12	DALE	CRCW06031001F	RES, 1K, 0603, 1/16W, 1%
2	R28-29	DALE	CRCW06031002F	RES, 10K, 0603, 1/16W, 1%
1	R18	DALE	CRCW0603100FT	RES, 10-OHM, 0603, 1/16W, 1%
1	R31	DALE	CRCW06031502F	RES, 15K, 0603, 1/16W, 1%
2	D1-2	MOTOROLA	1N4001	GENERAL PURPOSE RECTIFIER
1	D4	MOTOROLA	1N4148	GENERAL PURPOSE SIGNAL DIODE
7	R19-21, R24-27	DALE	CRCW06032000F	RES, 200-OHM, 0603, 1/16W, 1%
2	R3-4	DALE	CRCW06032702FT	RES, 27K, 0603, 1/16W, 1%
1	R14	DALE	CRCW06032203F	RES, 220K, 0603, 1/16W, 1%
1	R30	DALE	CRCW06033902F	RES, 39K, 0603, 1/16W, 1%
1	RN1	BOURNS	4609X-101-103	RES NETWORK, 10K, BUSSED, SIP-9
4	R8-11	DALE	CRCW06035101F	RES, 5.1K, 0603, 1/16W, 1%
1	U6	MICROCHIP	93C46B /ST	SERIAL EEPROM, 1K, 5.0V, TSSOP8
1	U4	ATMEL	AT89C52-20JC	MICROCONTROLLER, 5V, 8K ROM, PLCC44
1	BOX1	ALTECH	PC1111-7-TO	PLASTIC ENCLOSURE
2	R1-2	BOURNS	3262W-1-102	POT, CERMET, MULTITURN, TOP ADJ, 1K
1	R17	BOURNS	3262W-1-203	POT, CERMET, MULTITURN, TOP ADJ, 20K
1	R5	BOURNS	3262W-1-101	POT, CERMET, MULTITURN, TOP ADJ, 100
1	J3	AMP	748390-5	CONNECTOR, D-SUB, DE9, FEMALE, RT. A
1	T1	CR MAGNETICS	CR8348-2500-N	CURRENT TRANSFORMER
1	U3	CRYSTAL SEMI	CS5460-BS	POWER/ENERGY MONITORING IC
1	LCD1	OPTREX	DMC50448N	LCD MODULE, DIGI-KEY# 73-1106-ND
1	C6	PANASONIC	ECE-A1CU332	CAP, ELEC, 3300uF, 20%, 16V
4	C11-12, C14-15	KEMET	C0603C100J5GAC	CAP, COG, 10pF, 5% 50V, 0603
4	C21-22	KEMET	C1210C103J5GAC	CAP, COG,.01uF, 5% 50V, 1210
	C24-25			
2	C9-10	KEMET	C1206C472J5GAC	CAP, COG, 4700pF, 5% 50V, 1206
2	J10-11			
4	J4-7			
1	J2	SAMTEC	TSW-103-08-G-S-RA	HEADER, RT-ANGLE, 3X1, GOLD PLATED
1	J9	SAMTEC		HEADER,8X2,.1 IN CTR
1	JP1			JUMPER WIRE
1	D3	PANASONIC	LN1351C	LED, SMT 3216, GREEN
1	U2	NATIONAL SEMI	LM7805CT	VOLTAGE REGULATOR, 5V, TO-220
1	U1	TEXAS INST	MAX232D	DUAL RS-232 DRIVER / RECEIVER
1	U5	FAIRCHILD	NC7SZ04M5	SINGLE INVERTER, SOT23-5 CASE
1	S1	C&K	PTS645TL50	SWITCH, MOMENTARY, PUSHBUTTON
64	TP1-64			
1	XFMR1	TAMURA	818A_0001	WALL TRANSFORMER, CENTER TAPPED
14	C1-5,C7-8,C13,C16,C18-20,C23,C44	KEMET	C0805C104K5RAC	CAP, X7R, .1uF, 10% 50V, 0805
1	Y1	CAL CRYSTAL		CRYSTAL, HC49S CASE, 4.096MHz
1	Y2	ACME	GENERIC	CRYSTAL, HC49S CASE, 20MHz
1	R16	DALE	CRCW0603101FT	RES, 100-OHM, 0603, 1/16W, 1%
7	R15, R44-46, R60-62	DALE	CRCW06031002F	RES, 10K, 0603, 1/16W, 1%

Table 3. Bill of Materials

Qty	Reference	Manufacturer	Mfg PN	Description
2	R34 R50	DALE	CRCW0603100FT	RES, 10-OHM, 0603, 1/16W, 1%
2	R47 R63	DALE	CRCW06031502F	RES, 15K, 0603, 1/16W, 1%
1	D10	MOTOROLA	1N4001	GENERAL PURPOSE RECTIFIER
5	D5-9	MOTOROLA	1N4148	GENERAL PURPOSE SIGNAL DIODE
14	R35-37,R40-43,R51-53,R56-59	DALE	CRCW06032000F	RES, 200-OHM, 0603, 1/16W, 1%
1	R22	DALE	CRCW06032201F	RES, 2.2K, 0603, 1/16W, 1%
1	Q1	MOTOROLA	2N3906	TRANSISTOR, PNP, TO-92 CASE
8	R38-39,R48-49,R54-55,R64-65	DALE	CRCW06035101F	RES, 5.1K, 0603, 1/16W, 1%
1	J1	AMP	10224-1	TERMINAL BLOCK, 3 TERM, .2" CENTERS
2	U7-8	CRYSTAL SEMI	CS5460-BS	POWER/ENERGY MONITORING IC
8	C30-31,C33-34,C39-40,C42-43	KEMET	C1210C103J5GAC	CAP, COG,.01uF, 5% 50V, 1210
4	C26-27, C35-36	KEMET	C1206C472J5GAC	CAP, COG, 4700pF, 5% 50V, 1206
1	J8	SAMTEC		HEADER, 14X2
1	CNT1	HENGSTLER	635-1	MECHANICAL COUNTER, 5V
1	S3	GRAYHILL	76SB05	5 POSITION DIP SWITCH
1	S2	C&K	PTS645TL50	SWITCH, MOMENTARY, PUSHBUTTON
7	C17,C28-29,C32,C37-38,C41	KEMET	C0805C104K5RAC	CAP, X7R, .1uF, 10% 50V, 0805

Table 3. Bill of Materials (Continued)

15. MICROCONTROLLER SOURCE CODE

15.1 “nopt”

```
#pragma debug           /* default is no-debug*/
#pragma code            /*  ASSEMBLY source lines in list file   */

#include <intrins.h>    /*REQD FOR   _lrol_() */
#include "nopt.h"       /*prototypes, sfr equates, sbit declarations*/

   /*globals */
char  command,gpc,
                high_c, mid_c,low_c,                    /* for reading EEPROM  */
                high_byte, mid_byte, low_byte,/* from "read_register()" etc. */
                temp1, dipval,                                      /* dip switch variables */
                qty,                          /* for counting cumulative energy */
      NUMERATOR[6],                                     /* gets modified  by LONG_DIVIDE  */
      DENOMINATOR[6],                       /* not  modified  by LONG_DIVIDE  */
      NUMERATOR_BYTES,              /* QUANTITY OF NUM or DENOM BYTES */
      QUOTIENT_BYTES,         /* QUANTITY OF QUOTIENT BYTES     */
      QUOTIENT[6];            /* LONG_DIVIDE OUTPUT ARRAY       */
bdata int calls;
long sum,reg_a;
int s, max=32760, ind=200;

union dual {char auc[4]; long ans; };
union dual var;

union both {char ach[2]; int wht;};
union both bwe;

union pair {long arr[8]; int grp[16];};
union pair xyz;


main()
{

EA = 0x00; /* Disable all interrupts */
Delay(max);  /*20 milliseconds */
Delay(max);  /*20 milliseconds */
Delay(max);  /*20 milliseconds */
Delay(max);  /*20 milliseconds */
Delay(max);  /*20 milliseconds */
Delay(max);  /*20 milliseconds */

P1                      = 0xb7;  /*eeprom not selected*/
COMM     = 0x00;                                                        /* LED on*/
P3       = 0xFF; /*Set up port 3 to all inputs or special functions*/
SCON     = 0x72; /*8 bit UART*/
                        /*Set TI to enable TXSER UART function*/
                        /*Transmission Rate is 9600,N,8,1*/
TMOD    = 0x20; /*Use TIMER1, 8-bit auto-reload */
TH1     = 0xF5; /*Initialize to auto-reload to F5 for 20.000MHz external clock*/
PCON    = 0x80; /* when set, run at 9600 Baud, when clear, 4800 baud */
TCON      = 0x40; /* Turn internal UART timer ON */
                                /*Reset Serial Port*/
CS = 0x00;
SDI = 0x01;      /*SET SDI PIN*/
for(s=0;s<31;s++)/*Clock in 3 bytes of FF and then seven 1’s*/
       {
       SCLK = 0x01;    /*SET SCLK*/
       SCLK = 0x00;    /*CLR SCLK*/
       }
SDI = 0x00;        /*Now clock in a single 0*/
SCLK = 0x01;       /*SET SCLK*/
SCLK = 0x00;       /*CLR SCLK*/
CS = 0x01;
                        Delay(max);
                        Delay(max);
```

```
                        Delay(max);
                        Delay(max);
                        Delay(max);   /* Allow LED to be seen*/
                        Delay(max);

COMM    = 0x01;                                                         /* LED off*/

dipval = RD_DIP();/*READ THE 5 POSIT DIP SWITCH*/
switch(dipval)
       {
       case 0x01:  /*STANDALONE modes using FLASH Cal values*/
       case 0x03:
       case 0x05:
   case 0x07:
       case 0x09:
       case 0x11:
   lcdmi();
   usefv();
   hcreg();
       break;


                                                        /* MODES  using  DEFAULT Cal Values */
       case 0x00:
       case 0x02:
       case 0x04:
       case 0x06:
       case 0x08:
       case 0x0a:
       case 0x10:
       lcdmi();
   hcreg();
       high_c = 0x20;          /*gain one half */
       mid_c = 0;
       low_c = 0;
       command =  0x44;        /*write, current gain reg*/
       write_to_register(command,low_c,mid_c,high_c);
       high_c = 0x20;                       /*gain one half */
       mid_c = 0;
       low_c = 0;
       command =  0x48;        /*write, voltage gain reg*/
       write_to_register(command,low_c,mid_c,high_c);
       break;


       default:
       undip();
       break;
       }  /*END OF first SWITCH ON DIPVAL*/

write_to_register(0x74, 0x00, 0x00, 0x80);/* write, int mask reg */
CS   = 0x00;                                            /*Clear CSb*/
transfer_byte(0xE8);               /* Continuous Conversion  */
CS  = 0x01;                                                     /* Set CSb */

while(1)
   {
       while(dipval == 0)           /*PC mode, no cumulative energy*/
                {
                write_to_register(0x5E, 0x00, 0x00, 0x80); /* Clear interrupt */
                command = RXSER();      /*Get F9, etc from PC, (waits on RS-232)*/
                decode_command(command);/*one F9 call waits on STOP RS-232 before return*/
                dipval = RD_DIP();
      if(dipval)
       {
                        write_to_register(0x74, 0x00, 0x00, 0x80);/* write, int mask reg */
                        CS   = 0x00;                                    /*Clear CSb*/
                        transfer_byte(0xE8);                    /* Continuous Conversion  */
                        CS  = 0x01;                                     /* Set CSb */
                        }
                } /*end of PC mode*/
```

```
       calls=0;
       sum = 0;
               do  {
         if(calls < 16)
            {
            sum = sum + early();
            calls++;
            }
        }
               while (INTB != 0);     /*P3-3 low TO GET PAST HERE*/
               switch(dipval)          /* the value at reset*/
                        {
                        case 0x01:          /*POWER FACTOR */
                        case 0x02:
                        gete();
                        pwfk();
        break;

                        case 0x03:          /*  DISPLAY total cumulative energy*/
                        case 0x04:
                        gete();
                        cume();
                        break;

                        case 0x05:          /*Display RMS volts, RMS amps */
                        case 0x06:
                        gete();
                        rmsd();
                        break;

                        case 0x07:         /*automatic cycle through modes*/
                        case 0x08:
                        gete();
                        temp1++;
                        if(temp1 > 29)temp1 = 0;
                        if(temp1 < 10) pwfk();
                        if((temp1 > 9) && (temp1 < 20))cume();
        if((temp1 >19) && (temp1 < 30))rmsd();
                        break;

                        case 0x10:                    /* Electromechanical counter, NO cumulative energy
to flash  */
         case 0x11:
         MECH = 0x00;/* move INT1 jumper (J10) to EOUT */
                        Delay(6000);
                        MECH = 0x01;
                        break;


       case 0x0c:                                              /*calibrate voltage gain*/
                        read_register(0x18);    /*for Vrms */
                        shif();                  /* gain APPROX half, volts x 2*/
                        INS_LCD(0X01);                                  /*TOP LINE*/
                        Delay(ind);
                        TXSER(dipval);
                        WR_LCD(0x56);                               /* "V" */
                        TXSER(0x56);
                        WR_LCD(0x47);                               /* "G" */
                        TXSER(0x47);
                        WR_LCD(0x41);                               /* "A" */
                        TXSER(0x41);
                        WR_LCD(0x49);                               /* "I" */
                        TXSER(0x49);
                        WR_LCD(0x4E);                               /* "N" */
                        TXSER(0x4E);
                        WR_LCD(0x43);                               /* "C" */
                        TXSER(0x43);
                        WR_LCD(0x41);                               /* "A" */
                        TXSER(0x41);
                        WR_LCD(0x4C);                               /* "L" */
                        TXSER(0x4C);
```

```
                    TXSER(0x0D);
                    TXSER(0x0A);
                    NUMERATOR_BYTES=6;
                    QUOTIENT_BYTES =3;
      NUMERATOR[0]=0X33;   /*LS*/
      NUMERATOR[1]=0X33;
      NUMERATOR[2]=0X33;
      NUMERATOR[3]=0X33;
                    NUMERATOR[4]=0X13;
                    NUMERATOR[5]=0X00;   /*MS*/
                    DENOMINATOR[0]=0X00;   /*LS*/
                    DENOMINATOR[1]=0X00;
                    DENOMINATOR[2]=low_byte;
                    DENOMINATOR[3]=mid_byte;
                    DENOMINATOR[4]=high_byte;
                    DENOMINATOR[5]=0X00;        /*MS*/
                    LONG_DIVIDE();
      if(OV)
                            {
                            QUOTIENT[0]=0X00;
                            QUOTIENT[1]=0X00;
                            QUOTIENT[2]=0X20;
                            }
              write_to_register(0x48,QUOTIENT[0],QUOTIENT[1],QUOTIENT[2]);  /*low,mid,high*/
                    INS_LCD(0XC0);                          /*BOTTOM LINE*/
                    Delay(ind);
                    TXSER(0x20);
                    WR_LCD(0x30);                           /* "0" */
                    TXSER(0x30);
                    WR_LCD(0x58);                           /* "X" */
                    TXSER(0x58);
                    LCD_3(QUOTIENT[2],QUOTIENT[1],QUOTIENT[0]);              /*high,mid,low*/
                    TXSER(0x0D);
                    TXSER(0x0A);
      while(RD_DIP()==0x0c);/*wait for dip switch change */
      break;


case 0x0e:/*calibrate current gain*/
                    read_register(0x16);     /*for Irms */
                    shif();                  /*cut gain in half, curr x 2*/
                    INS_LCD(0X01);                                  /*TOP LINE*/
                    Delay(ind);
                    TXSER(dipval);
                    WR_LCD(0x49);                           /* "I" */
                    TXSER(0x49);
                    WR_LCD(0x47);                           /* "G" */
                    TXSER(0x47);
                    WR_LCD(0x41);                           /* "A" */
                    TXSER(0x41);
                    WR_LCD(0x49);                           /* "I" */
                    TXSER(0x49);
                    WR_LCD(0x4E);                           /* "N" */
                    TXSER(0x4E);
                    WR_LCD(0x43);                           /* "C" */
                    TXSER(0x43);
                    WR_LCD(0x41);                           /* "A" */
                    TXSER(0x41);
                    WR_LCD(0x4C);                           /* "L" */
                    TXSER(0x4C);
                    TXSER(0x0D);
                    TXSER(0x0A);
                    NUMERATOR_BYTES=6;
                    QUOTIENT_BYTES =3;
      NUMERATOR[0]=0X33;   /*LS*/
      NUMERATOR[1]=0X33;
      NUMERATOR[2]=0X33;
      NUMERATOR[3]=0X33;
                    NUMERATOR[4]=0X13;
                    NUMERATOR[5]=0X00;   /*MS*/
```

```
                    DENOMINATOR[0]=0X00;   /*LS*/
                    DENOMINATOR[1]=0X00;
                    DENOMINATOR[2]=low_byte;
                    DENOMINATOR[3]=mid_byte;
                    DENOMINATOR[4]=high_byte;
                    DENOMINATOR[5]=0X00;        /*MS*/
                    LONG_DIVIDE();
  if(OV)
                            {
                            QUOTIENT[0]=0X00;
                            QUOTIENT[1]=0X00;
                            QUOTIENT[2]=0X20;
                            }
              write_to_register(0x44,QUOTIENT[0],QUOTIENT[1],QUOTIENT[2]);  /*low,mid,high*/
                    INS_LCD(0XC0);                          /*BOTTOM LINE*/
                    Delay(ind);
                    TXSER(0x20);
                    WR_LCD(0x30);                           /* "0" */
                    TXSER(0x30);
                    WR_LCD(0x58);                           /* "X" */
                    TXSER(0x58);
                    LCD_3(QUOTIENT[2],QUOTIENT[1],QUOTIENT[0]);              /*high,mid,low*/
                    TXSER(0x0D);
                    TXSER(0x0A);
  while(RD_DIP()==0x0e);/*wait for dip switch change */
  break;


 case 0x15:          /*save cal values to FLASH*/
                   read_register(0x02);    /*for I offset val */
                   WR_EE(0x00,0x00,high_byte); /*high_byte ends up at eeprom addr 1 */
                   WR_EE(0x01,mid_byte,low_byte); /*the rest at addr 2 and 3 */
                   read_register(0x04);    /*for I gain val */
                   WR_EE(0x02,0x00,high_byte); /*high_byte ends up at eeprom addr 5 */
                   WR_EE(0x03,mid_byte,low_byte); /*the rest at addr 6 and 7 */
                   read_register(0x06);    /*for V offset val */
                   WR_EE(0x04,0x00,high_byte); /*high_byte ends up at eeprom addr 9 */
                   WR_EE(0x05,mid_byte,low_byte); /*the rest at addr a and b */
                   read_register(0x08);    /*for V gain val */
                   WR_EE(0x06,0x00,high_byte); /*high_byte ends up at eeprom addr d */
                   WR_EE(0x07,mid_byte,low_byte); /*the rest at addr e and f */
                   INS_LCD(0X01);                                   /*TOP LINE*/
                   Delay(ind);
                   TXSER(dipval);
                   WR_LCD(0x53);                                    /* "S" */
                   TXSER(0x53);
                   WR_LCD(0x41);                                    /* "A" */
                   TXSER(0x41);
                   WR_LCD(0x56);                                    /* "V" */
                   TXSER(0x56);
                   WR_LCD(0x45);                                    /* "E" */
                   TXSER(0x45);
                   WR_LCD(0x44);                                    /* "D" */
                   TXSER(0x44);
                   INS_LCD(0XC0);                           /*BOTTOM LINE*/
                   Delay(ind);
                   TXSER(0x20);
                   WR_LCD(0x43);                                    /* "C" */
                   TXSER(0x43);
                   WR_LCD(0x41);                                    /* "A" */
                   TXSER(0x41);
                   WR_LCD(0x4C);                                    /* "L"  */
                   TXSER(0x4c);
                   WR_LCD(0x20);                                    /* "SP" */
                   TXSER(0x20);
                   WR_LCD(0x56);                                    /* "V" */
                   TXSER(0x56);
                   WR_LCD(0x41);                                    /* "A" */
                   TXSER(0x41);
                   WR_LCD(0x4C);                                    /* "L" */
                   TXSER(0x4c);
```

```
                        WR_LCD(0x53);                                   /* "S" */
                        TXSER(0x53);
                        TXSER(0x0D);
                        TXSER(0x0A);
         while(RD_DIP()==0x15);  /*wait for dip switch change */
         break;


        case 0x13:/* clear out energy accum registers */
                        WR_EE(0X08,0x00,0x00);     /*total bin MS*/
                        WR_EE(0x09,0x00,0x00);     /*total bin LS*/
                        RD_EE(0x08);              /*most sig cum energy to  high_c ,mid_c ,low_c*/
                        INS_LCD(0X01);                                  /*TOP LINE*/
                        Delay(ind);
                        TXSER(dipval);
                        WR_LCD(0x41);                                   /* "A" */
                        TXSER(0x41);
                        WR_LCD(0x43);                                   /* "C" */
                        TXSER(0x43);
                        WR_LCD(0x43);                                   /* "C" */
                        TXSER(0x43);
                        WR_LCD(0x55);                                   /* "U" */
                        TXSER(0x55);
                        WR_LCD(0x4D);                                   /* "M" */
                        TXSER(0x4d);
                        WR_LCD(0x20);                                   /* "SP" */
                        TXSER(0x20);
                        WR_LCD(0x43);                                   /* "C" */
                        TXSER(0x43);
                        INS_LCD(0XC0);                          /*BOTTOM LINE*/
                        Delay(ind);
                        TXSER(0x20);
                        WR_LCD(0x30);                           /* "0" */
                        TXSER(0x30);
                        WR_LCD(0x58);                           /* "X" */
                        TXSER(0x58);
                        LCD_3(high_c,mid_c,low_c);
                        TXSER(0x0d);
                        TXSER(0x0a);
         while(RD_DIP()==0x13);  /*wait for dip switch change */
         break;



        default:    /*default for  switch on dipval*/
        undip();
        break;
                        } /*end of switch on dipval */
        write_to_register(0x5E, 0x00, 0x00, 0x80); /* Clear CS5460 interrupt */
        dipval=RD_DIP();
        }                   /*end of while one*/
}       /*end of main*/

void Delay(int fin)
{
for(s=0;s<fin;s++);
}


void write_to_register(char command,char low,char mid,char high)   /*write to wildcat*/
{
CS   = 0x00;                                /* Clear CSb */
transfer_byte(command);
transfer_byte(high);
transfer_byte(mid);
transfer_byte(low);
CS   = 0x01;                                /* Set CSb */
}

void read_register(char command)         /*read wildcat*/
{
```

```
CS   = 0x00;                                                    /*Clear CSb*/
transfer_byte(command);
high_byte       = receive_byte();   /*Receive Bytes*/
mid_byte        = receive_byte();
low_byte        = receive_byte();
CS   = 0x01;                                        /*Set CSb*/
}

void decode_command(char command)
{
data int j;
data unsigned char cnt;

switch (command)
       {
       /*THE FOLLOWING CASES WRITE TO CS5460 REGISTERS*/
       case 0x40:                          /* Config Register */
       case 0x42:                           /*  Current Offset Register */
       case 0x44:                          /* Current Gain Register */
       case 0x46:                          /*   Voltage Offset Register */
       case 0x48:                          /* Voltage Gain Register */
       case 0x4A:                          /* Cycle Count Register */
       case 0x4C:                          /* Pulse Rate Register */
       case 0x4E:                          /* Last Current Value */
       case 0x50:                          /* Last Voltage Value */
       case 0x52:                          /* Last Power Value */
       case 0x54:                          /* Last Total Energy Value */
       case 0x56:                          /* Last RMS Current Value */
       case 0x58:                          /* Last RMS Voltage Value */
       case 0x5A:                          /* Timebase Calibration Register */
       case 0x5E:                          /* Error/Status Register */
       case 0x74:                          /* Interrupt Mask Register */
       case 0x5C:                          /* RoEo Register */
       case 0x70:                          /* PDEF Register */
       case 0x72:                          /* PwrDn Register */
       case 0x76:                          /* Test Register */
       low_byte    = RXSER();  /*Receive data low byte first*/
       mid_byte    = RXSER();  /*from the PC*/
       high_byte   = RXSER();
       write_to_register(command,low_byte,mid_byte,high_byte);
       break;
                /*THE FOLLOWING CASES ARE FOR READING THE CS5460*/
       case 0x00:                          /* Config Register */
       case 0x02:                          /* Current Offset Register */
       case 0x04:                          /* Current Gain Register */
       case 0x06:                          /* Voltage Offset Register */
       case 0x08:                          /* Voltage Gain Register */
       case 0x0A:                          /* Cycle Count Register */
       case 0x0C:                          /* Pulse Rate Register */
       case 0x0E:                          /* Last Current Value */
       case 0x10:                          /* Last Voltage Value */
       case 0x12:                          /* Last Power Value */
       case 0x14:                          /* Last Total Energy Value */
       case 0x16:                          /* Last RMS Current Value */
       case 0x18:                          /* Last RMS Voltage Value */
       case 0x1A:                          /* Timebase Calibration Register */
       case 0x1E:                          /* Error/Status Register */
       case 0x34:                          /* Interrupt Mask Register */
       case 0x1C:                          /* RoEo Register */
       case 0x30:                          /* PDEF Register */
       case 0x32:                          /* PwrDn Register */
       case 0x36:                          /* Test Register */
       read_register(command); /* Read register's content */
       TXSER(low_byte);        /* Transfer bytes to PC*/
       TXSER(mid_byte);
       TXSER(high_byte);
       break;

                case 0xF9:   /*PC GUI related continuous convert with lead/lag */
                         write_to_register(0x74, 0x00, 0x00, 0x80);/* mask  for DRDY bit */
                         CS   = 0x00;                                    /*Clear CSb*/
```

```
                        transfer_byte(0xE8);                    /* Continuous Conversion  */
                        CS   = 0x01;                            /* Set CSb */
                        do {
                                cnt = 128;  /*cnt >128 is "current lagging" for Labwindows*/
                                calls=0;
                                sum = 0;
                                do  {
             if(calls < 16)
             {
              sum = sum + early();       /*long plus integer*/
            calls++;
               }
             }
                                while (INTB != 0);
                                for (temp1 = 0x0E; temp1 <= 0x18; temp1 = temp1 + 2)
                                        {
                                        read_register(temp1);  /* Get Value */
                                        TXSER(low_byte);              /* Send info to PC */
                                        TXSER(mid_byte);
                                        TXSER(high_byte);       /*total 18 bytes sent up*/
                                        }

                                if(sum & 0x80000000)cnt++;     /*LAGGING*/
          else cnt--;                  /*LEADING*/
                                TXSER(cnt);             /*send the lead/lag info up*/
                                write_to_register(0x5E, 0x00, 0x00, 0x80); /* Clear "data ready" */
                        }
                        while (RI != 1);              /* Do until PC sends another byte */
                        CS   = 0x00;                            /*Clear CSb*/
                        transfer_byte(0xA0);          /* Send HALT  */
                        CS   = 0x01;                            /* Set CSb */
                        write_to_register(0x5E, 0x00, 0x00, 0x80); /* Clear interrupt */
                        RI = 0;                                         /* throw out byte */
                        break;


       default:
       break;
       } /* END switch on command */
}      /* END decode_command*/



/* ROUTINE TO MAKE BCD AND ASCII  */
/* NOT SUITABLE  FOR FRACTIONS */
/* ALWAYS OUTPUTS 8 DIGITS, with any leading ZEROs */
void LCD_D(void)
{
char j,n;
xyz.arr[0]=       1L;
xyz.arr[1]=      10L;
xyz.arr[2]=     100L;
xyz.arr[3]=    1000L;
xyz.arr[4]=   10000L;
xyz.arr[5]=  100000L;
xyz.arr[6]= 1000000L;
xyz.arr[7]=10000000L;
var.auc[3]=low_byte;
var.auc[2]=mid_byte;
var.auc[1]=high_byte;
var.auc[0]=0;

for(n=7;n >= 0;n--)
       {
       j=0;
       if(var.ans >= (xyz.arr[n]))
                {
          do
                        {
                var.ans = var.ans - xyz.arr[n];
```

```
                    j++;
                          }
                  while(var.ans >= xyz.arr[n]);
                  }
        WR_LCD(j + 0x30);
        TXSER(j + 0x30);
        }
TXSER(0x0D);
TXSER(0x0A);
}


/*USED TO CONVERT TWO BYTES TO FRACTIONAL*/
void LCD_F(void)
{
int n;
char j,m;
long dfr,d50k;
d50k = 50000L;
xyz.grp[0]=        1;
xyz.grp[1]=        3;
xyz.grp[2]=        6;
xyz.grp[3]=       12;
xyz.grp[4]=       24;
xyz.grp[5]=       49;
xyz.grp[6]=       98;
xyz.grp[7]=      195;
xyz.grp[8]=      391;
xyz.grp[9]=      781;
xyz.grp[10]=    1562;
xyz.grp[11]=    3125;
xyz.grp[12]=    6250;
xyz.grp[13]=   12500;
xyz.grp[14]=   25000;
bwe.ach[0]=high_byte;
bwe.ach[1]=mid_byte;
n=1;
dfr = 0;
for(j=0; j < 15; j++)
        {
        if(bwe.wht & n)
                  {
                  dfr = dfr + (long)xyz.grp[j];
                  }
   n = n << 1;
   }
if(bwe.wht & n)
dfr = dfr + d50k;

WR_LCD(0X2E);      /*DOT*/
TXSER(0X2E);

xyz.arr[0]=       1L;
xyz.arr[1]=      10L;
xyz.arr[2]=     100L;
xyz.arr[3]=    1000L;
xyz.arr[4]=   10000L;
for(m=4; m >= 0; m--)
        {
        j=0;
        if(dfr >= xyz.arr[m])
                  {
           do
                          {
                     dfr = dfr - xyz.arr[m];
                     j++;
                          }
                  while(dfr >= xyz.arr[m]);
                  }
        WR_LCD(j + 0x30);
        TXSER(j + 0x30);
```

```
       }
TXSER(0x0D);
TXSER(0x0A);
}



/*ROUTINE USED FOR POS SIGNED DATA, TO SHIFT 3 BYTES LEFT ONE BIT*/
void shif()
{
var.auc[3]=low_byte;
var.auc[2]=mid_byte;
var.auc[1]=high_byte;
var.auc[0]=0;
var.ans = _lrol_(var.ans,1);
high_byte=var.auc[1];
mid_byte= var.auc[2];
low_byte= var.auc[3] & 0xfe;
}

/*************************POWER FACTOR ROUTINE ***********************************/
void pwfk()
                        {
                        /*set up to multiply (vrms*irms) using existing globals*/
                        read_register(0x16);/*IRMS*/
                        shif();                                         /* curr gain was cut in half,
so times two */
                        NUMERATOR[0]=low_byte;
        NUMERATOR[1]=mid_byte;
        NUMERATOR[2]=high_byte;
                        read_register(0x18);/*VRMS*/
                        shif();                                         /*volt gain was cut in half, so
times two */
                        DENOMINATOR[0]= low_byte;
                        DENOMINATOR[1] = mid_byte;
                        DENOMINATOR[2] = high_byte;
                        MULT24();
                                                                        /*set up to divide (ener-
gy/(vrms*irms)*/
                        DENOMINATOR[5]=QUOTIENT[5];
                        DENOMINATOR[4]=QUOTIENT[4];             /*Vrms x Irms */
                        DENOMINATOR[3]=QUOTIENT[3];
                        DENOMINATOR[2]=QUOTIENT[2];
                        DENOMINATOR[1]=QUOTIENT[1];             /*Vrms x Irms */
                        DENOMINATOR[0]=QUOTIENT[0];

                        NUMERATOR_BYTES = 6;
       QUOTIENT_BYTES=2;
                        read_register(0x14);          /*energy to low_byte, mid_byte, high_byte*/
                        if(high_byte & 0x80)
                                {
                                nege();
           }
                        shif();                 /*signed to unsigned*/
                        shif();
                        shif();                 /*cut gain in half, energy x 4*/
                        NUMERATOR[5]=high_byte;
                        NUMERATOR[4]=mid_byte;
                        NUMERATOR[3]=low_byte;

                        LONG_DIVIDE();   /*energy/(vrms*irms)*/
        if(OV)
                                {
                                QUOTIENT[1]=0XFF;
                                QUOTIENT[2]=0XFF;
                                }
                        INS_LCD(0X01);                                  /*TOP LINE*/
                        Delay(ind);
                        TXSER(dipval);
                        WR_LCD(0x46);                                   /* "F" */
```

```
                        TXSER(0x46);
                        WR_LCD(0x20);                                   /* "SPACE" */
                        TXSER(0x20);
                        high_byte=QUOTIENT[1];
                        mid_byte =QUOTIENT[0];
                        LCD_F();
                        INS_LCD(0XC0);                          /*BOTTOM LINE*/
                        Delay(ind);
                        TXSER(0x20);
                        if(sum & 0x80000000)
                                {
                                WR_LCD(0x49);                           /* "I" */
                                TXSER(0x49);
                                WR_LCD(0x20);                           /* "SPACE" */
                                TXSER(0x20);
                                WR_LCD(0x4C);                           /* "L" */
                                TXSER(0x4C);
                                WR_LCD(0x41);                           /* "A" */
                                TXSER(0x41);
                                WR_LCD(0x47);                           /* "G" */
                                TXSER(0x47);
                                WR_LCD(0x53);                           /* "S" */
                        TXSER(0x53);
                                TXSER(0x0D);
                                TXSER(0x0A);
                                }
        else
        {
                                WR_LCD(0x49);                           /* "I" */
                                TXSER(0x49);
                                WR_LCD(0x20);                           /* "SPACE" */
                                TXSER(0x20);
                                WR_LCD(0x4C);                           /* "L" */
                                TXSER(0x4C);
                                WR_LCD(0x45);                           /* "E" */
                                TXSER(0x45);
                                WR_LCD(0x41);                           /* "A" */
                                TXSER(0x41);
                                WR_LCD(0x44);                           /* "D" */
                                TXSER(0x44);
                                WR_LCD(0x53);                           /* "S" */
                        TXSER(0x53);
                                TXSER(0x0D);
                                TXSER(0x0A);
           }
                        }

void cume()                      /*Display total cumulative energy*/
                        {
                        RD_EE(0x08);             /*reads total cum energy 0x08 and 0x09 to high_c etc*/
                        INS_LCD(0X01);                                  /*TOP LINE no label*/
                        Delay(ind);
                        TXSER(dipval);
                        low_byte=low_c;
                        mid_byte=mid_c;
                        high_byte=high_c;
                        LCD_D();                                        /*  "87654321" cumulative energy */
                        read_register(0x14);    /*cycle energy to high_byte,mid_byte,low_byte*/
                        if(high_byte & 0x80)
                                {
                                nege();
           }
                        shif();                                          /* x 2, for SIGNED output*/
                        shif();
                        shif();                  /*gain is half, energy x 4*/
                        INS_LCD(0XC0);                          /*BOTTOM LINE*/
                        Delay(ind);
                        TXSER(0x20);
                        WR_LCD(0x45);                           /* "E"  */
                        TXSER(0x45);
                        WR_LCD(0x20);                           /* "SPACE" */
```

```
                        TXSER(0x20);
                        LCD_F();                                        /*one calc cycle energy */
                        }

void rmsd()
                        {
                        read_register(0x16);     /*for Irms, high_byte,mid_byte,low_byte */
                        shif();                   /*gain was cut in half, curr x 2 */
                        INS_LCD(0X01);                                  /*TOP LINE*/
                        Delay(ind);
                        TXSER(dipval);
                        WR_LCD(0x49);                                   /* "I" */
                        TXSER(0x49);
                        WR_LCD(0x20);                                   /* "SPACE" */
                        TXSER(0x20);
        LCD_F();
                        read_register(0x18);    /*for Vrms */
                        shif();                  /*gain was cut in half, volts x 2*/
                        INS_LCD(0XC0);                          /*BOTTOM LINE*/
                        Delay(ind);
                        TXSER(0x20);
                        WR_LCD(0x56);                           /* "V" */
                        TXSER(0x56);
                        WR_LCD(0x20);                           /* "SPACE" */
                        TXSER(0x20);
                        LCD_F();
                        }


void nege(void)
{
temp1=RD_DIP();
INS_LCD(0X01);                              /*TOP LINE*/
Delay(ind);
TXSER(dipval);
WR_LCD(0x4E);                       /* "N" */
TXSER(0x4E);
WR_LCD(0x45);                       /* "E" */
TXSER(0x45);
WR_LCD(0x47);                       /* "G" */
TXSER(0x47);
WR_LCD(0x20);                       /* "SP" */
TXSER(0x20);
WR_LCD(0x45);                       /* "E" */
TXSER(0x45);
WR_LCD(0x4E);                       /* "N" */
TXSER(0x4E);
WR_LCD(0x47);                       /* "G" */
TXSER(0x47);
WR_LCD(0x59);                       /* "Y" */
TXSER(0x59);
TXSER(0x0D);
TXSER(0x0A);
Delay(max);
Delay(max);
Delay(max);
}

void undip(void)
       {
                temp1=RD_DIP();
                INS_LCD(0X01);                                  /*TOP LINE*/
                Delay(ind);
                TXSER(dipval);
                WR_LCD(0x3F);                                   /* "?" */
                TXSER(0x3F);
                WR_LCD(0x55);                                   /* "U" */
                TXSER(0x55);
                WR_LCD(0x4E);                                   /* "N" */
                TXSER(0x4E);
                WR_LCD(0x4B);                                   /* "K" */
```

```
                TXSER(0x4B);
                WR_LCD(0x4E);                                   /* "N" */
                TXSER(0x4E);
                WR_LCD(0x4F);                                   /* "O" */
                TXSER(0x4F);
                WR_LCD(0x57);                                   /* "W" */
                TXSER(0x57);
                WR_LCD(0x4E);                                   /* "N" */
                TXSER(0x4E);
                TXSER(0x20);            /*RS-232 space*/
                INS_LCD(0XC0);                      /*BOTTOM LINE*/
                Delay(ind);
                TXSER(0x20);
                WR_LCD(0x44);                                   /* "D" */
                TXSER(0x44);
                WR_LCD(0x49);                                   /* "I" */
                TXSER(0x49);
                WR_LCD(0x50);                                   /* "P" */
                TXSER(0x50);
                WR_LCD(0x56);                                   /* "V" */
                TXSER(0x56);
                WR_LCD(0x41);                                   /* "A" */
                TXSER(0x41);
                WR_LCD(0x4C);                                   /* "L" */
                TXSER(0x4C);
                WR_LCD(0x55);                                   /* "U" */
                TXSER(0x55);
                WR_LCD(0x45);                                   /* "E" */
                TXSER(0x45);
                TXSER(0x0D);
                TXSER(0x0A);
                while((RD_DIP() == temp1));/*wait for dip switch change */
                dipval = RD_DIP();
                }

void lcdmi()
        {
        INIT_LCD(0X03);    /*START LCD MODULE INIT*/
        Delay(max);           /* > 4.1 milliseconds */
        INIT_LCD(0X03);
        Delay(max);           /* > 100 microseconds */
        INIT_LCD(0X03);
        Delay(max);           /* > 100 microseconds */
        INIT_LCD(0X02);
        Delay(max);           /* > 100 microseconds */
        INS_LCD(0X28);     /*  possible to poll busy flag */
        Delay(ind);
        INS_LCD(0X01);
        Delay(ind);
        INS_LCD(0X06);
        Delay(ind);
        INS_LCD(0X0C);     /*END OF LCD MODULE INIT*/
        Delay(ind);
        }

void usefv()
        {
        RD_EE(0x00);
        command =  0x42;        /*write, current offset reg*/
        write_to_register(command,low_c,mid_c,high_c);

        RD_EE(0x02);
        command =  0x44;        /*write, current gain reg*/
        write_to_register(command,low_c,mid_c,high_c);

        RD_EE(0x04);
        command =  0x46;        /*write, voltage offset reg*/
        write_to_register(command,low_c,mid_c,high_c);

        RD_EE(0x06);
        command =  0x48;        /*write, voltage gain reg*/
```

```
        write_to_register(command,low_c,mid_c,high_c);
        }

void hcreg()
        {
        command =   0x40;        /*write, config reg*/
        low_c =  0x61;        /* 0x000061*/
        mid_c =  0x00;
        high_c = 0x00;
        write_to_register(command,low_c,mid_c,high_c);

        command =   0x4A;        /*write, cycle cnt reg*/
        low_c =  0xa0;        /*0x000fa0 = 4000*/
        mid_c =  0x0f;
        high_c = 0x00;
        write_to_register(command, low_c, mid_c, high_c);

        command =   0x5A;        /*write, time base reg*/
        low_c =  0x6c;        /*0x7fffff =  unity*/
        mid_c =  0x12;        /*0x83126C = 1.024 */
        high_c = 0x83;
        write_to_register(command,low_c, mid_c, high_c);

        command =   0x4C;        /*write, pulse rate reg*/
        low_c =  0xc0;        /*0x0fa000 = default  */
        mid_c =  0x03;        /*0x0003c0 = 30 */
        high_c = 0x00;         /*0x0000a0 = 5 */
        write_to_register(command, low_c, mid_c, high_c);
        }



void    gete()
{
read_register(0x14);    /*cycle energy to high_byte,mid_byte,low_byte*/
if(high_byte & 0x80)nege();
shif();                                                    /* x 2, for SIGNED output*/
shif();
shif();                  /*gain is half, energy x 4*/
var.auc[3]=low_byte;     /*CYCLE energy*/
var.auc[2]=mid_byte;
var.auc[1]=high_byte;
var.auc[0]=0;
reg_a = reg_a + var.ans; /* accumulate LS cycle energy */
var.ans = reg_a;       /*any overflow into var.auc[0]*/
if(var.auc[0])     /* if accumulation over 24 bits  */
        {
        qty++;          /*increment overflow count */
        var.auc[0]=0;   /* overflow amount deleted */
        reg_a = var.ans;    /*reg_a  overflow deleted */
        if(qty > 9)
                {
                qty = 0;                   /*most variables global*/
                RD_EE(0x08);              /*reads total cum energy 0x08 and 0x09*/
                var.auc[3] = low_c;
                var.auc[2] = mid_c;
                var.auc[1] = high_c;
                var.ans++;
                high_c=var.auc[1];
                mid_c =var.auc[2];
                low_c =var.auc[3];
                WR_EE(0X08,0x00,high_c);
                WR_EE(0x09,mid_c,low_c);
                }
        }
}
```

15.2 “early”

```
$DEBUG
```

```
TCOD    SEGMENT CODE    ;  TCOD RELOCATABLE
PUBLIC  EARLY             ; Make subroutine global
EXTERN DATA (CALLS)
RSEG    TCOD     ; Assemble to a certain segment
EARLY:
       CLR      0D5H                        ; START WITH CLEAR "POS" BIT
       CLR      0D1H           ; START WITH CLEAR "POSNEG" BIT
   CLR 090H                       ; CHIP SELECT BAR
       MOV      R1,#0H
loop1:
       MOV      A,#10H      ; Read Command for channel A (VOLTAGE)
       RLC      A                                      ; Rotate into Carry
       MOV      P1.1, C           ; Put MSB on SDI
       SETB     P1.3                       ; Toggle SCLK
       CLR      P1.3                       ; Toggle SCLK

       RLC      A                                      ; Rotate into Carry
       MOV      P1.1, C           ; Put bit 6 on SDI
       SETB     P1.3                       ; Toggle SCLK
       CLR      P1.3                       ; Toggle SCLK

       RLC      A                                      ; Rotate into Carry
       MOV      P1.1, C           ; Put bit 5 on SDI
       SETB     P1.3                       ; Toggle SCLK
       CLR      P1.3                       ; Toggle SCLK

       RLC      A                                      ; Rotate into Carry
       MOV      P1.1, C           ; Put bit 4 on SDI
       SETB     P1.3                       ; Toggle SCLK
       CLR      P1.3                       ; Toggle SCLK

       RLC      A                                      ; Rotate into Carry
       MOV      P1.1, C  ; Put bit 3 on SDI
       SETB     P1.3                       ; Toggle SCLK
       CLR      P1.3                       ; Toggle SCLK

       RLC      A                                      ; Rotate into Carry
       MOV      P1.1, C           ; Put bit 2 on SDI
       SETB     P1.3                       ; Toggle SCLK
       CLR      P1.3                       ; Toggle SCLK

       RLC      A                                      ; Rotate into Carry
       MOV      P1.1, C           ; Put bit 1 on SDI
       SETB     P1.3                       ; Toggle SCLK
       CLR      P1.3                       ; Toggle SCLK

       RLC      A                                      ; Rotate into Carry
       MOV      P1.1, C           ; Put LSB on SDI
       SETB     P1.3                       ; Toggle SCLK
       CLR      P1.3                       ; Toggle SCLK

;READ CHANNEL A HIGH BYTE (NO WRITING WILDCAT ON SPI)

       SETB     P1.1                       ; Set SDI to avoid reading config register..
       MOV      C, P1.2           ; Put SDO into Carry
       RLC      A                                      ; Rotate MSB into Acc
       SETB     P1.3                       ; Toggle SCLK
       CLR      P1.3                       ; Toggle SCLK

       MOV      C, P1.2           ; Put SDO into Carry
       RLC      A                                      ; Rotate Bit 6 into Acc
       SETB     P1.3                       ; Toggle SCLK
       CLR      P1.3                       ; Toggle SCLK

       MOV      C, P1.2           ; Put SDO into Carry
       RLC      A                                      ; Rotate Bit 5 into Acc
       SETB     P1.3                       ; Toggle SCLK
       CLR      P1.3                       ; Toggle SCLK

       MOV      C, P1.2           ; Put SDO into Carry
       RLC      A                                      ; Rotate Bit 4 into Acc
```

```
      SETB     P1.3                      ; Toggle SCLK
      CLR      P1.3                      ; Toggle SCLK

      MOV      C, P1.2           ; Put SDO into Carry
      RLC      A                                  ; Rotate Bit 3 into Acc
      SETB     P1.3                      ; Toggle SCLK
      CLR      P1.3                      ; Toggle SCLK

      MOV      C, P1.2           ; Put SDO into Carry
      RLC      A                                  ; Rotate Bit 2 into Acc
      SETB     P1.3                      ; Toggle SCLK
      CLR      P1.3                      ; Toggle SCLK

      MOV      C, P1.2           ; Put SDO into Carry
      RLC      A                                  ; Rotate Bit 1 into Acc
      SETB     P1.3                      ; Toggle SCLK
      CLR      P1.3                      ; Toggle SCLK

      MOV      C, P1.2           ; Put SDO into Carry
      RLC      A                                  ; Rotate LSB into Acc
      SETB     P1.3                      ; Toggle SCLK
      CLR      P1.3                      ; Toggle SCLK
      MOV      R2, A                     ;  CHAN A HI BYTE


      ; **** READ MID BYTE OF CHANNEL A, SEND READ COMMAND FOR CHANNEL B ****
MOV  B,#0EH                      ; "READ CURRENT" COMMAND
      MOV      C, P1.2           ; Put SDO into Carry
      RLC      A                                  ; Rotate Bit 7 into Acc
      MOV      C, 0F7H           ; Get Channel B command bit 7
      MOV      P1.1, C           ; Put bit on SDI
      SETB     P1.3                      ; Toggle SCLK
      CLR      P1.3                      ; Toggle SCLK

      MOV      C, P1.2           ; Put SDO into Carry
      RLC      A                                  ; Rotate Bit 6 into Acc
      MOV      C, 0F6H           ; Get Channel B command bit 6
      MOV      P1.1, C           ; Put bit on SDI
      SETB     P1.3                      ; Toggle SCLK
      CLR      P1.3                      ; Toggle SCLK

      MOV      C, P1.2           ; Put SDO into Carry
      RLC      A                                  ; Rotate Bit 5 into Acc
      MOV      C, 0F5H           ; Get Channel B command bit 5
      MOV      P1.1, C           ; Put bit on SDI
      SETB     P1.3                      ; Toggle SCLK
      CLR      P1.3                      ; Toggle SCLK

      MOV      C, P1.2           ; Put SDO into Carry
      RLC      A                                  ; Rotate Bit 4 into Acc
      MOV      C, 0F4H           ; Get Channel B command bit 4
      MOV      P1.1, C           ; Put bit on SDI
      SETB     P1.3                      ; Toggle SCLK
      CLR      P1.3                      ; Toggle SCLK

      MOV      C, P1.2           ; Put SDO into Carry
      RLC      A                                  ; Rotate Bit 3 into Acc
      MOV      C, 0F3H           ; Get Channel B command bit 3
      MOV      P1.1, C           ; Put bit on SDI
      SETB     P1.3                      ; Toggle SCLK
      CLR      P1.3                      ; Toggle SCLK

      MOV      C, P1.2           ; Put SDO into Carry
      RLC      A                                  ; Rotate Bit 2 into Acc
      MOV      C, 0F2H           ; Get Channel B command bit 2
      MOV      P1.1, C           ; Put bit on SDI
      SETB     P1.3                      ; Toggle SCLK
      CLR      P1.3                      ; Toggle SCLK

      MOV      C, P1.2           ; Put SDO into Carry
      RLC      A                                  ; Rotate Bit 1 into Acc
```

```
        MOV     C, 0F1H         ; Get Channel B command bit 1
        MOV     P1.1, C         ; Put bit on SDI
        SETB    P1.3                    ; Toggle SCLK
        CLR     P1.3                    ; Toggle SCLK

        MOV     C, P1.2         ; Put SDO into Carry
        RLC     A                               ; Rotate LSB into Acc
        MOV     C, 0F0H         ; Get Channel B command bit 0
        MOV     P1.1, C         ; Put bit on SDI
        SETB    P1.3                    ; Toggle SCLK
        CLR     P1.3                    ; Toggle SCLK
        MOV     R3, A                   ; MID BYTE CHAN A


; **** GET HIGH BYTE OF CHANNEL  B, SEND STATUS (5E) ****

        MOV     C, P1.2         ; Put SDO into Carry
        RLC     A                               ; Rotate MSB into Acc
        CLR     P1.1                    ; First Command bit = 0
        SETB    P1.3                    ; Toggle SCLK
        CLR     P1.3                    ; Toggle SCLK

        MOV     C, P1.2         ; Put SDO into Carry
        RLC     A                               ; Rotate Bit 6 into Acc
        SETB    P1.1                    ; Second Command Bit = 1
        SETB    P1.3                    ; Toggle SCLK
        CLR     P1.3                    ; Toggle SCLK

        MOV     C, P1.2         ; Put SDO into Carry
        RLC     A                               ; Rotate Bit 5 into Acc
        CLR     P1.1                    ; Third Command Bit = 0
        SETB    P1.3                    ; Toggle SCLK
        CLR     P1.3                    ; Toggle SCLK

        MOV     C, P1.2         ; Put SDO into Carry
        RLC     A                               ; Rotate Bit 4 into Acc
        SETB    P1.1                    ; Fourth Command Bit = 1
        SETB    P1.3                    ; Toggle SCLK
        CLR     P1.3                    ; Toggle SCLK

        MOV     C, P1.2         ; Put SDO into Carry
        RLC     A                               ; Rotate Bit 3 into Acc
                                                        ; Fifth Command Bit = 1
        SETB    P1.3                    ; Toggle SCLK
        CLR     P1.3                    ; Toggle SCLK

        MOV     C, P1.2         ; Put SDO into Carry
        RLC     A                               ; Rotate Bit 2 into Acc
                                                        ; Sixth Command Bit = 1
        SETB    P1.3                    ; Toggle SCLK
        CLR     P1.3                    ; Toggle SCLK

        MOV     C, P1.2         ; Put SDO into Carry
        RLC     A                               ; Rotate Bit 1 into Acc
                                                        ; Seventh Command Bit = 1
        SETB    P1.3                    ; Toggle SCLK
        CLR     P1.3                    ; Toggle SCLK

        MOV     C, P1.2         ; Put SDO into Carry
        RLC     A                               ; Rotate LSB into Acc
        CLR     P1.1                    ; Eighth Command Bit = 0
        SETB    P1.3                    ; Toggle SCLK
        CLR     P1.3                    ; Toggle SCLK
        MOV     R4, A                   ; HIGH BYTE CHAN B (CURRENT)

        ; **** GET MIDDLE BYTE OF CHANNEL  B,  SEND HIGH STATUS (80) ****
        MOV     C, P1.2         ; Put SDO into Carry
        RLC     A                               ; Rotate MSB into Acc
        SETB    P1.1                    ; First bit = 1
        SETB    P1.3                    ; Toggle SCLK
        CLR     P1.3                    ; Toggle SCLK
```

```
        MOV     C, P1.2         ; Put SDO into Carry
        RLC     A                                   ; Rotate Bit 6 into Acc
        CLR     P1.1                        ; Second Bit = 0
        SETB    P1.3                        ; Toggle SCLK
        CLR     P1.3                        ; Toggle SCLK

        MOV     C, P1.2         ; Put SDO into Carry
        RLC     A                                   ; Rotate Bit 5 into Acc
                                                    ; Third Bit = 0
        SETB    P1.3                        ; Toggle SCLK
        CLR     P1.3                        ; Toggle SCLK

        MOV     C, P1.2         ; Put SDO into Carry
        RLC     A                                   ; Rotate Bit 4 into Acc
                                                    ; Fourth Bit = 0
        SETB    P1.3                        ; Toggle SCLK
        CLR     P1.3                        ; Toggle SCLK

        MOV     C, P1.2         ; Put SDO into Carry
        RLC     A                                   ; Rotate Bit 3 into Acc
                                                    ; Fifth Bit = 0
        SETB    P1.3                        ; Toggle SCLK
        CLR     P1.3                        ; Toggle SCLK

        MOV     C, P1.2         ; Put SDO into Carry
        RLC     A                                   ; Rotate Bit 2 into Acc
                                                    ; Sixth Bit = 0
        SETB    P1.3                        ; Toggle SCLK
        CLR     P1.3                        ; Toggle SCLK

        MOV     C, P1.2         ; Put SDO into Carry
        RLC     A                                   ; Rotate Bit 1 into Acc
                                                    ; Seventh Bit = 0
        SETB    P1.3                        ; Toggle SCLK
        CLR     P1.3                        ; Toggle SCLK

        MOV     C, P1.2         ; Put SDO into Carry
        RLC     A                                   ; Rotate LSB into Acc
                                                    ; Eighth Bit = 0
        SETB    P1.3                        ; Toggle SCLK
        CLR     P1.3                        ; Toggle SCLK
        MOV     R5, A                       ; MID BYTE CHAN B (CURRENT)

; **** SEND MID STATUS BYTE (00) ****
        SETB    P1.3            ; Toggle SCLK
        CLR     P1.3            ; Toggle SCLK
                                                    ; Bit 6 = 0
        SETB    P1.3            ; Toggle SCLK
        CLR     P1.3            ; Toggle SCLK
                                                    ; Bit 5 = 0
        SETB    P1.3            ; Toggle SCLK
        CLR     P1.3            ; Toggle SCLK
                                                    ; Bit 4 = 0
        SETB    P1.3            ; Toggle SCLK
        CLR     P1.3            ; Toggle SCLK
                                                    ; Bit 3 = 0
        SETB    P1.3            ; Toggle SCLK
        CLR     P1.3            ; Toggle SCLK
                                                    ; Bit 2 = 0
        SETB    P1.3            ; Toggle SCLK
        CLR     P1.3            ; Toggle SCLK
                                                    ; Bit 1 = 0
        SETB    P1.3            ; Toggle SCLK
        CLR     P1.3            ; Toggle SCLK
                                                    ; LSB = 0
        SETB    P1.3            ; Toggle SCLK
        CLR     P1.3            ; Toggle SCLK

; **** SEND LOW STATUS BYTE (00) ****
        SETB    P1.3            ; Toggle SCLK
```

```
       CLR      P1.3              ; Toggle SCLK
                                                    ; Bit 6 = 0
       SETB     P1.3              ; Toggle SCLK
       CLR      P1.3              ; Toggle SCLK
                                                    ; Bit 5 = 0
       SETB     P1.3              ; Toggle SCLK
       CLR      P1.3              ; Toggle SCLK
                                                    ; Bit 4 = 0
       SETB     P1.3              ; Toggle SCLK
       CLR      P1.3              ; Toggle SCLK
                                                    ; Bit 3 = 0
       SETB     P1.3              ; Toggle SCLK
       CLR      P1.3              ; Toggle SCLK
                                                    ; Bit 2 = 0
       SETB     P1.3              ; Toggle SCLK
       CLR      P1.3              ; Toggle SCLK
                                                    ; Bit 1 = 0
       SETB     P1.3              ; Toggle SCLK
       CLR      P1.3              ; Toggle SCLK
                                                    ; LSB = 0
       SETB     P1.3              ; Toggle SCLK
       CLR      P1.3              ; Toggle SCLK
;volts R2:R3    current R4:R5

   INC  R1
        MOV      ACC,R1
        JB                ACC.7, QRT

   MOV  A,R2
        JNB      ACC.7,POS         ;IF VOLTS POS, JUMP TO POS:
        JNB      0D5H,SAV                  ;IF VOLTS STARTED NEG, GO TO SAV
        SETB     0D1H              ;SET POSNEG BIT
        JMP      SAV                                ;GO TO THE SAV
POS:
        SETB     0D5H                               ;SET "POS" BIT
        JNB      0D1H,SAV                  ;UNLESS "POSNEG" SET, GO TO SAV
        JNB      CALLS.0,QRT
        MOV      A,R5                               ;CURRENT LS
   MOV  R7,A          ;current LS
   MOV  A,R4                              ;CURRENT MS
   MOV  R6,A                              ;CURRENT MS
QRT:
   SETB 090H                              ;RAISE CHIP SELECT BAR
        RET
SAV:
        MOV      A,R5                               ;CURRENT LS
   MOV  R7,A          ;current LS
   MOV  A,R4                              ;CURRENT MS
   MOV  R6,A                              ;CURRENT MS
        LJMP     LOOP1                              ;LOOP1 IS TOO FAR FOR RELATIVE JUMP
END
```

15.3 “Init3ph”

```
;THIS ROUTINE IS THE NIBBLE WRITER USED IN INITIALIZATION OF THE LCD

$DEBUG
USING   0               ; Use register bank 0
TCOD    SEGMENT CODE    ; Relocatable, called from C, with identical name

PUBLIC  _INIT_LCD       ; Make subroutine global

RSEG    TCOD            ;Assemble to a certain segment
_INIT_LCD:
   MOV    ACC, R7   ;Move passed byte to ACC
   ANL    ACC,#0FH  ;CLEAR HI NIBBLE OF ACCUM
   SETB   ACC.7     ;HI-Z THE DIP SWITCH
   MOV    P2,ACC    ;MOVE ACCUM TO PORT 2
   SETB   P2.6      ;RAISE E CLOCK
   CLR    P2.6      ;LOWER E CLOCK
   RET              ;Exit subroutine
```

```
END
```

15.4 “Ins3ph”

```
;THIS ROUTINE IS PASSED AN INSTRUCTION WHICH IS WRITTEN TO THE LCD

$DEBUG
USING   0               ; Use register bank 0
TCOD    SEGMENT CODE    ; Relocatable, called from C, with identical name

PUBLIC  _INS_LCD         ; Make subroutine global

RSEG    TCOD            ;Assemble to a certain segment
_INS_LCD:
   MOV     ACC, R7      ;Move passed byte to ACC
   SWAP    A          ;PUT  HI NIBBLE AT LOW END
   ANL    ACC,#0FH      ;CLEAR HI NIBBLE OF ACCUM
   SETB   ACC.7         ;HI Z DIP SWITCHES
   MOV    P2,ACC        ;MOVE ACCUM TO PORT 2
   SETB   P2.6          ;RAISE E CLOCK
   CLR    P2.6          ;LOWER E CLOCK

   MOV    ACC, R7       ;Move passed byte to ACC
   ANL    ACC,#0FH      ;CLEAR HI NIBBLE OF ACCUM
   SETB   ACC.7         ;HI Z DIP SWITCHES
   MOV    P2,ACC        ;MOVE ACCUM TO PORT 2
   SETB   P2.6          ;RAISE E CLOCK
   CLR    P2.6          ;LOWER E CLOCK


;checks for not busy. This code works but has been replaced by a delay (for 200)
;       SETB     P2.0                                    ;bit zero will be read
;       SETB     P2.1                                    ;bit one will be read
;       SETB     P2.2                                    ;bit two will be read
;       SETB     P2.3                                    ;bit 3 for busy bit
;       SETB     P2.5                                    ;R/W TO READ
;LA1:
;       SETB     P2.6                                    ;RAISE E CLOCK
;       MOV      C,P2.3                      ;BUSY BIT TO CARRY FLAG
;       CLR      P2.6                                    ;LOWER E CLOCK
;   SETB   P2.6          ;RAISE E CLOCK FOR LOWER NIBBLE
;   CLR    P2.6          ;LOWER E CLOCK FOR LOWER NIBBLE
;       JC                LA1                                       ;AGAIN IF CARRY SET
   RET             ; Exit subroutine
END
```

15.5 “lcd3”

```
;THIS ROUTINE IS PASSED AN INSTRUCTION WHICH IS WRITTEN TO THE LCD

$DEBUG
USING   0               ; Use register bank 0
TCOD    SEGMENT CODE    ; Relocatable, called from C, with identical name

PUBLIC  _INS_LCD         ; Make subroutine global

RSEG    TCOD            ;Assemble to a certain segment
_INS_LCD:
   MOV     ACC, R7      ;Move passed byte to ACC
   SWAP    A          ;PUT  HI NIBBLE AT LOW END
   ANL    ACC,#0FH      ;CLEAR HI NIBBLE OF ACCUM
   SETB   ACC.7         ;HI Z DIP SWITCHES
   MOV    P2,ACC        ;MOVE ACCUM TO PORT 2
   SETB   P2.6          ;RAISE E CLOCK
   CLR    P2.6          ;LOWER E CLOCK

   MOV    ACC, R7       ;Move passed byte to ACC
   ANL    ACC,#0FH      ;CLEAR HI NIBBLE OF ACCUM
   SETB   ACC.7         ;HI Z DIP SWITCHES
   MOV    P2,ACC        ;MOVE ACCUM TO PORT 2
   SETB   P2.6          ;RAISE E CLOCK
```

```
   CLR    P2.6          ;LOWER E CLOCK


;checks for not busy. This code works but has been replaced by a delay (for 200)
;       SETB     P2.0                                    ;bit zero will be read
;       SETB     P2.1                                    ;bit one will be read
;       SETB     P2.2                                    ;bit two will be read
;       SETB     P2.3                                    ;bit 3 for busy bit
;       SETB     P2.5                                    ;R/W TO READ
;LA1:
;       SETB     P2.6                                    ;RAISE E CLOCK
;       MOV      C,P2.3                       ;BUSY BIT TO CARRY FLAG
;       CLR      P2.6                                    ;LOWER E CLOCK
;   SETB   P2.6          ;RAISE E CLOCK FOR LOWER NIBBLE
;   CLR    P2.6          ;LOWER E CLOCK FOR LOWER NIBBLE
;       JC                LA1                                    ;AGAIN IF CARRY SET
   RET             ; Exit subroutine
END
```

15.6 “longdiv”

```
;This program is set up as a function for calling from Franklin C compiler
;The EXTERN DATA here should be GLOBAL in the C program.  See Bob’s write-up
;on the use and limitations of this general purpose routine

$DEBUG

TCOD    SEGMENT   CODE   ;Relocatable, called from C, with identical name

EXTRN DATA(NUMERATOR, DENOMINATOR, QUOTIENT)
EXTRN NUMBER(NUMERATOR_BYTES, QUOTIENT_BYTES)
PUBLIC LONG_DIVIDE, TIMES_TWO_AND_COMPARE

RSEG TCOD      ;Assemble to a certain segment
       BYTE_COUNT                  EQU      R2
       BIT_COUNT                   EQU      R3
       HIGHEST_NUMERATOR_BYTEEQU   R4      ;pointer
       HIGHEST_DENOMINATOR_BYTEEQUR5      ;pointer


LONG_DIVIDE:
       MOV      A,#NUMERATOR
       MOV      R0,#NUMERATOR_BYTES
       ADD      A,@R0
       DEC      A
       MOV      R4,A
       MOV      A,#DENOMINATOR
       MOV      R0,#NUMERATOR_BYTES
       ADD      A,@R0
       DEC      A
       MOV      R5,A
       MOV      R0,#QUOTIENT_BYTES
       MOV      A,@R0
       MOV      B,#8
       MUL      AB
       JNB      OV,$+4   ;to MOV bit_count
       RET
       MOV      BIT_COUNT,A        ;BITS IN QUOTIENT

ALGORITHM:
       CALL     TIMES_TWO_AND_COMPARE
       MOV      F0,C                        ;TEMP SAVE qn IN F0
;  SHIFT qn INTO QUOTIENT:
       MOV      R0,#QUOTIENT_BYTES
   MOV  A,@R0
       MOV      BYTE_COUNT,A         ;BOB’S INDIRECT LOAD
       MOV      R0,#QUOTIENT
Q_SHIFT:
       MOV      A,@R0
       RLC      A
       MOV      @R0,A
```

```
        INC     R0
        DJNZ    BYTE_COUNT,Q_SHIFT

        JNB     F0,BIT_COUNT_TEST
        MOV     R0,#NUMERATOR_BYTES
        MOV     A,@R0
        MOV     BYTE_COUNT,A        ;BOB'S INDIRECT LOAD
        MOV     R0,#NUMERATOR
        MOV     R1,#DENOMINATOR
        CLR     C
SUBTRACT:
        MOV     A,@R0
        SUBB    A,@R1
        MOV     @R0,A
        INC     R0
        INC     R1
        DJNZ    BYTE_COUNT,SUBTRACT
BIT_COUNT_TEST:
        DJNZ    BIT_COUNT,ALGORITHM
        CLR     OV
        RET

TIMES_TWO_AND_COMPARE:
        MOV     R0,#NUMERATOR_BYTES
        MOV     A,@R0
        MOV     BYTE_COUNT,A      ;6, Bob's indirect load
        MOV     R0,#NUMERATOR     ;pointer to LS num byte
        CLR     C

LEFT_SHIFT:               ;SHIFTS ENTIRE 6-BYTE NUM (one bit left)
        MOV     A,@R0             ;move byte  to accum
        RLC     A                 ;shift accum left
        MOV     @R0,A             ;put accum back
        INC     R0                ;point to next higher byte
        DJNZ    BYTE_COUNT,LEFT_SHIFT   ;dec r2, jump back six times
        JC      ERROR                                          ;CY = 1 if TIMES_TWO indicates overflow

        MOV     R0,#NUMERATOR_BYTES
        MOV     A,@R0
        MOV     BYTE_COUNT,A                          ;6, BOB'S INDIRECT,  load BYTE_COUNT again
        MOV     A,HIGHEST_NUMERATOR_BYTE      ;R4
        MOV     R0,A                        ; R0 points highest num
        MOV     A,HIGHEST_DENOMINATOR_BYTE    ;R5
        MOV     R1,A                        ;R1 points highest den


COMPARISON:
        MOV     A,@R0
        MOV     B,@R1
        CJNE    A,B,DONE  ;SETS CY IF NUMERATOR < DENOMINATOR
                                                ;CLEARS CY IF NUMERATOR >= DENOMINATOR
        DEC     R0
        DEC     R1
        DJNZ    BYTE_COUNT,COMPARISON

DONE:   CPL     C                 ;CLEARS CY IF NUMERATOR < DENOMINATOR
                                                ;SETS CY IF NUMERATOR >= DENOMINATOR
        RET                                 ;THUS CY = qn

ERROR:  SETB    OV
        POP     ACC
        POP     ACC
        RET
END
```

15.7 "mult24"

```
;USING LONG DIV VARIABLE NAMES FOR THIS MULTIPLY ROUTINE
$DEBUG

TCOD    SEGMENT   CODE   ;Relocatable, called from C, with identical name
```

```
EXTRN DATA(NUMERATOR, DENOMINATOR, QUOTIENT)  ;ARRAY POINTERS
PUBLIC MULT24

RSEG TCOD      ;Assemble to a certain segment
MULT24:
        MOV       R0,#NUMERATOR
        MOV       A,@R0
   MOV  R1,#DENOMINATOR
   MOV  B,@R1
   MUL  AB                ; "NUMERATOR[0]" x "DENOMINATOR[0]"
   MOV  QUOTIENT,A
   MOV  (QUOTIENT + 1),B
   INC  R0                                                    ;ONLY R0 INCREMENTED HERE
        MOV       A,@R0
   MOV  B,@R1
   MUL  AB                ; "NUMERATOR[1]" x "DENOMINATOR[0]"
   ADD  A,(QUOTIENT + 1)  ;ADD , MAYBE SET C FLAG
   MOV  (QUOTIENT + 1),A  ;
   MOV  A,B               ;
   ADDC A,#0                                          ;ADD ANY C FLAG
   MOV  (QUOTIENT + 2),A;
   INC  R0                ;ONLY R0 INCREMENTED HERE
        MOV       A,@R0             ;
   MOV  B,@R1             ;
   MUL  AB                ; "NUMERATOR[2]" x "DENOMINATOR[0]"
   ADD  A,(QUOTIENT + 2)  ;ADD , MAYBE SET C FLAG
   MOV  (QUOTIENT + 2),A  ;
   MOV  A,B               ;
   ADDC A,#0                                          ;ADD ANY C FLAG
   MOV  (QUOTIENT + 3),A;

   DEC  R0
   DEC  R0
        INC       R1

        MOV       A,@R0
        MOV       B,@R1
   MUL  AB                ; "NUMERATOR[0]" x "DENOMINATOR[1]"
   ADD  A,(QUOTIENT + 1)  ;ADD , MAYBE SET C FLAG
   MOV  (QUOTIENT + 1),A  ;
   MOV  A,B               ;
   ADDC A,(QUOTIENT + 2);ADD ANY C FLAG
   MOV  (QUOTIENT + 2),A;
   INC  R0                ;ONLY R0 INCREMENTED HERE
        MOV       A,@R0
   MOV  B,@R1
   MUL  AB                ; "NUMERATOR[1]" x "DENOMINATOR[1]"
   ADD  A,(QUOTIENT + 2)  ;ADD , MAYBE SET C FLAG
   MOV  (QUOTIENT + 2),A  ;
   MOV  A,B               ;
   ADDC A,(QUOTIENT + 3);ADD ANY C FLAG
   MOV  (QUOTIENT + 3),A;
   INC  R0                ;ONLY R0 INCREMENTED HERE
        MOV       A,@R0             ;
   MOV  B,@R1             ;
   MUL  AB                ; "NUMERATOR[2]" x "DENOMINATOR[1]"
   ADD  A,(QUOTIENT + 3)  ;ADD , MAYBE SET C FLAG
   MOV  (QUOTIENT + 3),A  ;
   MOV  A,B               ;
   ADDC A,#0                                          ;ADD ANY C FLAG
   MOV  (QUOTIENT + 4),A;

   DEC  R0
   DEC  R0
        INC       R1

        MOV       A,@R0
        MOV       B,@R1
   MUL  AB                ; "NUMERATOR[0]" x "DENOMINATOR[2]"
   ADD  A,(QUOTIENT + 2)  ;ADD , MAYBE SET C FLAG
```

```
   MOV  (QUOTIENT + 2),A  ;
   MOV  A,B               ;
   ADDC A,(QUOTIENT + 3);ADD ANY C FLAG
   MOV  (QUOTIENT + 3),A;
   INC  R0                ;ONLY R0 INCREMENTED HERE
        MOV      A,@R0
   MOV  B,@R1
   MUL  AB                ; "NUMERATOR[1]" x "DENOMINATOR[2]"
   ADD  A,(QUOTIENT + 3)  ;ADD , MAYBE SET C FLAG
   MOV  (QUOTIENT + 3),A  ;
   MOV  A,B               ;
   ADDC A,(QUOTIENT + 4);ADD ANY C FLAG
   MOV  (QUOTIENT + 4),A;
   INC  R0                ;ONLY R0 INCREMENTED HERE
        MOV      A,@R0             ;
   MOV  B,@R1             ;
   MUL  AB                ; "NUMERATOR[2]" x "DENOMINATOR[2]"
   ADD  A,(QUOTIENT + 4)  ;ADD , MAYBE SET C FLAG
   MOV  (QUOTIENT + 4),A  ;
   MOV  A,B               ;
   ADDC A,#0                                          ;ADD ANY C FLAG
   MOV  (QUOTIENT + 5),A;


        RET

END
```

15.8 “nopt prototypes”

```
/*PROTOTYPES*/

void gete(void);
void hcreg(void);
void usefv(void);
void lcdmi(void);
void undip(void);
void nege(void);
void rmsd(void);
void cume(void);
void pwfk(void);
void Delay(int);
void shif(void);
void LCD_D(void);
void LCD_F(void);
void decode_command(char command);
void write_to_register(char command,char low,char mid, char high);
void read_register(char command);
extern int early(void);
extern void TXSER(char);
extern char RXSER(void);
extern char receive_byte(void);
extern void transfer_byte(char);
extern void LCD_3(char,char,char);
extern void WR_LCD(char);
extern void INS_LCD(char);
extern voidLONG_DIVIDE(void);
extern void MULT24(void);
extern void WR_EE(char,char,char);
extern char RD_DIP(void);
extern void RD_EE(char);

extern void INIT_LCD(char);


/*  BYTE Register equates for the register ports */
sfr P0   = 0x80;
sfr P1   = 0x90;
sfr P2   = 0xA0;
sfr P3   = 0xB0;
sfr PSW  = 0xD0;
```

```
sfr ACC  = 0xE0;
sfr B    = 0xF0;
sfr SP   = 0x81;
sfr DPL  = 0x82;
sfr DPH  = 0x83;
sfr PCON = 0x87;
sfr TCON = 0x88;
sfr TMOD = 0x89;
sfr TL0  = 0x8A;
sfr TL1  = 0x8B;
sfr TH0  = 0x8C;
sfr TH1  = 0x8D;
sfr IE   = 0xA8;
sfr IP   = 0xB8;
sfr SCON = 0x98;
sfr SBUF = 0x99;

/*Interface Equates*/
sbit    CS   =   0x90; /* 5460 Chip Select */
sbit    SDI  =   0x91; /* 5460 Serial Data In */
sbit    SDO  =   0x92; /* 5460 Serial Data Out */
sbit    SCLK =   0x93; /* 5460 Serial Clock */
sbit    DCLK=    0x94; /* 5460 Clock output */
sbit    INTRQ=   0xB2; /* USB Interrupt */
sbit    INTB=    0xB3; /* 5460 Interrupt */
sbit    EDIR=    0xB4; /* 5460 Energy Direction signal */
sbit      EOUT   =        0xB5; /* 5460 Energy Output signal */
sbit    COMM   =  0x97;
sbit      MECH  =   0xa0;  /*for the mechanical counter p2.0*/
/*  BIT Register  */

/*  Accumulator  */
sbit ABIT7 = 0xE7;
sbit ABIT6 = 0xE6;
sbit ABIT5 = 0xE5;
sbit ABIT4 = 0xE4;
sbit ABIT3 = 0xE3;
sbit ABIT2 = 0xE2;
sbit ABIT1 = 0xE1;
sbit ABIT0 = 0xE0;

/*  B Register  */
sbit BBIT7 = 0xF7;
sbit BBIT6 = 0xF6;
sbit BBIT5 = 0xF5;
sbit BBIT4 = 0xF4;
sbit BBIT3 = 0xF3;
sbit BBIT2 = 0xF2;
sbit BBIT1 = 0xF1;
sbit BBIT0 = 0xF0;



/*  PSW   */
sbit CY   = 0xD7;
sbit AC   = 0xD6;
sbit F0   = 0xD5;
sbit RS1  = 0xD4;
sbit RS0  = 0xD3;
sbit OV   = 0xD2;
sbit P    = 0xD0;

/*  TCON  */
sbit TF1  = 0x8F;
sbit TR1  = 0x8E;
sbit TF0  = 0x8D;
sbit TR0  = 0x8C;
sbit IE1  = 0x8B;
sbit IT1  = 0x8A;
sbit IE0  = 0x89;
sbit IT0  = 0x88;
```

```
/*  IE   */
sbit EA   = 0xAF;
sbit ES   = 0xAC;
sbit ET1  = 0xAB;
sbit EX1  = 0xAA;
sbit ET0  = 0xA9;
sbit EX0  = 0xA8;

/*  IP   */
sbit PS   = 0xBC;
sbit PT1  = 0xBB;
sbit PX1  = 0xBA;
sbit PT0  = 0xB9;
sbit PX0  = 0xB8;

/*  P3  */
sbit RD   = 0xB7;
sbit WR   = 0xB6;
sbit T1   = 0xB5;
sbit T0   = 0xB4;
sbit INT1 = 0xB3;
sbit INT0 = 0xB2;
sbit TXD  = 0xB1;
sbit RXD  = 0xB0;

/*  SCON  */
sbit SM0  = 0x9F;
sbit SM1  = 0x9E;
sbit SM2  = 0x9D;
sbit REN  = 0x9C;
sbit TB8  = 0x9B;
sbit RB8  = 0x9A;
sbit TI   = 0x99;
sbit RI   = 0x98;
```

15.9 “rdee”

```
;THIS ROUTINE reads the flash eeprom. void RD_EE(char addr);
;PASSED ADDR FOR MS WORD, NEXT WORD ALSO READ
$DEBUG
USING   0               ; Use register bank 0
TCOD    SEGMENT CODE    ;Relocatable, called from C, with identical name

PUBLIC  _RD_EE          ; Make subroutine global

EXTRN DATA (HIGH_C, MID_C, LOW_C)

RSEG    TCOD            ; Assemble to a certain segment
_RD_EE:
  CLR           P1.7                                    ;TURN ON LED
  SETB  P1.0            ;CSB 5460 PH-0
  SETB  P1.4           ;CSB 5460 PH-1
  SETB  P1.5           ;CSB 5460 PH-2
  CLR           P1.3                                    ;EEPROM CLOCK
  SETB  P1.2                                  ;PULLUP RESISTOR FOR DO

;"READ" START BIT
  SETB  P1.6        ;CS HI
  SETB  P1.1                        ;DI  HI
  NOP
  SETB  P1.3                        ;CLK HI
  NOP
  CLR           P1.3                        ;CLK LOW
        MOV     ACC,R7            ;passed addr
        ORL     A,#80H            ;"READ" CODE

        MOV   R1,#08H
LBL3:
        ACALL     LBL2                      ;WRITE CODE, ADDR
        DJNZ      R1,LBL3
```

```
        MOV   R1,#08H
LBL4:
        ACALL     LBL1                        ;READ 31..24 (ZEROS)
        DJNZ      R1,LBL4

        MOV   R1,#08H
LBL5:
        ACALL     LBL1                        ;READ DATA 23..16
        DJNZ      R1,LBL5
   MOV  HIGH_C,ACC

        MOV   R1,#08H
LBL6:
        ACALL     LBL1                        ;READ DATA 15..8
        DJNZ      R1,LBL6
   MOV  MID_C,ACC

        MOV   R1,#08H
LBL7:
        ACALL     LBL1                        ;READ DATA 7..0
        DJNZ      R1,LBL7
   MOV  LOW_C,ACC

        CLR       P1.6                        ;CS  LOW
        SETB      P1.7                        ;LED OFF
        RET               ;Exit  subroutine

LBL1:
  SETB  P1.3                          ;CLK HI
        MOV       C,P1.2              ;DO TO FLAG
        RLC       A                                   ;FLAG TO BIT 0
  CLR             P1.3                        ;CLK LOW
  RET

LBL2:
        RLC       A                                   ;LEFT BIT  TO FLAG
        MOV       P1.1,C              ;FLAG TO DI
        NOP
  SETB  P1.3                          ;CLK HI
  NOP
  CLR             P1.3                        ;CLK LOW
  RET

END
```

15.10 “rdip”

```
;THIS ROUTINE reads the dip sw
$DEBUG
USING   0               ; Use register bank 0
TCOD    SEGMENT CODE    ; Relocatable, called from C, with identical name

PUBLIC  RD_DIP         ; Make subroutine global

RSEG    TCOD            ; Assemble to a certain segment
RD_DIP:

  MOV             ACC,#1Fh
  MOV             P2,ACC
        NOP
  MOV    ACC,P2        ;READ DIP
  SETB  P2.7                                  ;HI Z DIPS
  ANL    ACC,#1FH      ;MASK OFF DIP BITS
  MOV             R7,ACC                      ; RETURN THIS
  RET                   ;Exit subroutine
END
```

15.11 “receive”

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

```
;* Routine     - RECEIVE_BYTE
;* Input       - none
;* Output      - Byte received is placed in R7
;* Description - This subroutine receives 1 byte from converter
;*****************************************************************
;

; The function prototype is:   char RECEIVE_BYTE(void);

$DEBUG
USING   0               ; Use register bank 0
TCOD    SEGMENT CODE    ; Relocatable, called from C, with identical name

PUBLIC  RECEIVE_BYTE; Make subroutine global

RSEG    TCOD     ; Assemble to a certain segment
RECEIVE_BYTE:
        MOV     R1,#08  ; Set count to 8 to receive byte
        SETB       P1.1    ; Set SDI when not in use

LOOP:   ; Receive the byte
        MOV     C,P1.2  ; Move bit to carry
        RLC     A       ; Rotate A in preparation for next bit
        SETB    P1.3    ; Set SCLK
        CLR     P1.3    ; Clear SCLK
        DJNZ    R1,LOOP ; Decrement byte, repeat loop if not zero

        MOV     R7,A    ; Byte to be returned is placed in R7
        RET             ; Exit subroutine
END
```

15.12 “rxser”

```
;*****************************************************************
;* Routine     - RXSER
;* Input       - none
;* Output      - Byte received is placed in R7
;* Description - This subroutine receives 1 byte from converter
;*   via UART.  It uses the RS-232 serial protocol to transmit
;*   one byte from a PC/UART system to the 8051.
;*****************************************************************
;

; The function prototype is:   char RXSER(void);

$DEBUG
USING   0               ; Use register bank 0
TCOD    SEGMENT CODE    ; Relocatable, called from C, with identical name

PUBLIC  RXSER           ; Make subroutine global

RSEG    TCOD            ; Assemble to a certain segment
RXSER:
        JNB     SCON.0,$ ; Poll RI
        MOV     R7,SBUF  ; Place received byte in R7
        CLR     SCON.0   ; Reset RI bit
        RET
END
```

15.13 “transfer”

```
;*****************************************************************
;* Routine     - transfer_byte
;* Input       - Byte to be transmitted is placed in Accumulator
;* Output      - None
;* Description - This subroutine sends 1 byte to converter
;*****************************************************************
;

;The function prototype is:   void TRANSFER_BYTE(char);

$DEBUG
USING   0               ; Use register bank 0
TCOD    SEGMENT CODE    ; Relocatable, called from C, with identical name
```

```
PUBLIC  _TRANSFER_BYTE   ; Make subroutine global

RSEG    TCOD            ; Assemble to a certain segment
_TRANSFER_BYTE:
       MOV     A, R7    ; Move byte to be transmitted to ACC
       MOV     R1,#08   ; Set count to 8 to transmit byte
       CLR     P1.3     ; Clear SCLK

loop:   ; Send Byte
       RLC     A        ; Rotate Accumulator, send MSB 1st
       MOV     P1.1,C   ; Transmit MSB first through C bit
       SETB    P1.3     ; Set SCLK
       CLR     P1.3     ; Clear SCLK
       DJNZ    R1,loop  ; Decrement byte, repeat loop if not zero
       SETB    P1.1     ; Reset SDI to one when not transmitting
       RET              ; Exit subroutine
END
```

15.14 “txser”

```
;*************************************************************
;* Routine     - TXSER
;* Input       - Byte to be transmitted is placed in R7
;* Output      - None
;* Description - This subroutine transfers 1 byte from converter
;*   via UART.  It uses the RS-232 serial protocol to transmit
;*   one byte from a 80C51 to the PC/UART system. To
;*   function properly, the programmer must first initialize the
;*   TI bit in the SCON register to 0X01.
;*************************************************************
;

;The function prototype is:   void TXSER(char);

$DEBUG
USING   0               ; Use register bank 0
TCOD    SEGMENT CODE    ; Relocatable, called from C, with identical name

PUBLIC  _TXSER           ; Make subroutine global

RSEG    TCOD            ;Assemble to a certain segment
_TXSER:  JNB     SCON.1,$        ; Poll TI
       CLR     SCON.1          ; Reset TI
       MOV     SBUF, R7       ; Move byte to output register
       RET                     ; Exit subroutine
END
```

15.15 “wr3ph”

```
;THIS ROUTINE IS PASSED ASCII WHICH IS WRITTEN TO LCD DISPLAY AT THE CURSOR
$DEBUG
USING   0               ; Use register bank 0
TCOD    SEGMENT CODE    ; Relocatable, called from C, with identical name

PUBLIC  _WR_LCD         ; Make subroutine global

RSEG    TCOD            ;Assemble to a certain segment
_WR_LCD:
   MOV     ACC, R7      ;Move passed byte to ACC
   SWAP    A                       ;PUT  HI NIBBLE AT LOW END
   ANL    ACC,#0FH      ;CLEAR HI NIBBLE OF ACCUM
   SETB   ACC.7         ;Bit for HI-Z  DIP SWITCH
   SETB   ACC.4         ;RS HI FOR ASCII
   MOV    P2,ACC        ;MOVE ACCUM TO PORT 2
   SETB   P2.6          ;RAISE E CLOCK
   CLR    P2.6          ;LOWER E CLOCK

   MOV    ACC, R7       ;Byte to ACC again, no swap
   ANL    ACC,#0FH      ;CLEAR HI NIBBLE OF ACCUM
   SETB   ACC.7         ;Bit for HI-Z THE DIP SWITCH
   SETB   ACC.4         ;RS HI FOR ASCII
```

```
   MOV    P2,ACC        ;MOVE ACCUM TO PORT 2
   SETB   P2.6          ;RAISE E CLOCK
   CLR    P2.6          ;LOWER E CLOCK
;check for not busy. This code works, but has been replaced by the txser delay
;       SETB     P2.3                                    ;DATA7 LINE IS FOR INPUT
;       SETB     P2.5                                    ;R/W TO READ
;LAB1:
;       SETB     P2.6                                    ;RAISE E CLOCK
;       MOV      C,P2.3                    ;BUSY BIT TO CARRY FLAG
;       CLR      P2.6                                    ;LOWER E CLOCK
;   SETB   P2.6          ;RAISE E CLOCK FOR LOWER NIBBLE
;   CLR    P2.6          ;LOWER E CLOCK FOR LOWER NIBBLE
;       JC                LAB1                                    ;AGAIN IF CARRY SET
   RET                  ;Exit subroutine
END
```

15.16 “Wree”

```
;THIS ROUTINE writes 16 BITS TO eeprom.  void WR_EE(addr,ZERO,HI)  OR   void WR_EE(addr,MID,LOW);
$DEBUG
USING   0               ; Use register bank 0
TCOD    SEGMENT CODE    ; Relocatable, called from C, with identical name

PUBLIC  _WR_EE         ;  passing = underscore

RSEG    TCOD            ; Assemble to a certain segment
_WR_EE:
  CLR           P1.7                                    ;TURN ON LED
  SETB   P1.0           ;CSB 5460 PH-0
  SETB  P1.4           ;CSB 5460 PH-1
  SETB  P1.5           ;CSB 5460 PH-2
  CLR           P1.6                                    ;GET FLASH CS LOW
  CLR           P1.3                                    ;EEPROM CLOCK  LOW
  SETB  P1.2                                    ;PULLUP RESISTOR FOR SDO
  NOP
  NOP
; EWEN START BIT
  SETB  P1.6           ;CS HI
  SETB  P1.1                                    ;DI HI
  NOP
  SETB  P1.3                                    ;CLK HI
  NOP
  CLR           P1.3                                    ;CLK LOW
       MOV      ACC,#30H     ;"EWEN" CODE,ADDR
       MOV   R1,#08H
LBL3:
       ACALL    LBL1
       DJNZ     R1,LBL3
       CLR      P1.6                          ;CS LOW
   NOP
;"WRITE" START BIT
  SETB  P1.6        ;CS HI
  SETB  P1.1                          ;DI  HI
  NOP
  SETB  P1.3                          ;CLK HI
  NOP
  CLR           P1.3                          ;CLK LOW
       MOV      ACC,R7             ;ADDR BYTE (from C PROGRAM)
       ORL      A,#40H             ;"WRITE" CODE to bits 7, 6
       MOV   R1,#08H
LBL4:
       ACALL    LBL1
       DJNZ     R1,LBL4

       MOV      ACC,R5             ;THE MS BYTE
       MOV   R1,#08H
LBL5:
       ACALL    LBL1
       DJNZ     R1,LBL5

       MOV      ACC,R3    ;LS BYTE
```

```
       MOV   R1,#08H
LBL6:
       ACALL     LBL1
       DJNZ      R1,LBL6

       CLR       P1.6                        ;CS  LOW
       SETB      P1.3                                ;CLK HI
       SETB      P1.6                        ;CS HI
       CLR       P1.3                                ;CLK LOW

LBL2:
       SETB      P1.3                                ;CLK HI
       NOP
       CLR               P1.3                                    ;CLK LOW
       JNB       P1.2,LBL2 ;WAIT FOR BUSYBAR HI
       CLR       P1.6                        ;CS  LOW


       SETB      P1.7                        ;TURN OFF LED
       RET               ;Exit  subroutine

LBL1:
       RLC       A                                   ;LEFT BIT  TO FLAG
       MOV       P1.1,C             ;FLAG TO DI
  NOP
  SETB  P1.3                       ;CLK HI
  NOP
  CLR           P1.3                        ;CLK LOW
  RET
END
```

• Notes •


SMART
Analog
™