READ ME: Late-Breaking Updates to the DCPB Manual

All Customers

• C11 (next to the DB9 serial connector) should be replaced with a pair of header stakes and a shorting block (supplied for kits; installed on assembled units). When you want to program the BS2, install the shorting block on this header to allow the STAMP2 software to reset the BS2 for downloading. When you want to communicate with the DCPB using a terminal program, remove the shorting block to prevent the terminal software from causing unintended resets of the BS2. This change is in response to several customer reports of unwanted resets and communication problems caused by noise coupled into the BS2's ATN line through C11. The jumper arrangement, while less convenient, is significantly less vulnerable to noise.

• When you connect the 9V battery clip or other power source to the board be sure that the polarity is correct! The red wire (+ voltage) must connect to the post marked + on the DCPB. Prolonged connection (more than a second or two) of a reversed power source will fry the voltage regulator, possibly damaging other components (including the BS2) in the process. If you're prone to connecting first and looking later, consider leaving the header socket connected. Connect and disconnect the battery at the polarized battery clip instead.

Kit Customers

• Read the assembly notes on page 5 of the manual carefully. If you install the DCPB's voltage regulator U1—and we suggest that you do unless you *absolutely require* a 55µA reduction in current draw—do *not* install D4. The only reason to install D4 is if you do decide to leave out U1 and C8 in order to use the BS2's regulator. If you install both D4 and U1, you are asking for trouble, and there will be two voltage regulators with slightly different output voltages fighting each other for control of the +5-volt rail. This will cause instability of the power supply and excessive current draw.

Contents

Introductory Material

Warranty ii
Disclaimer of Liability ii
Trademarks ii
BS2 Data Collection Proto Board 1
Prerequisites for Using 1
Technical Support 1
Contacting Us (Scott Edwards Electronics) 1
Contacting Parallax (Manufacturer of the BS2) 1

Hardware Description

Overview 2
BS2-IC 2
Main Power Supply 2
Peripheral Power Supply 2
Synchronous Serial Bus 2
Figure 1. Schematic diagram of the Data Collection Proto Board 3
Solid-State Data Storage 4
Real-time Clock Calendar 4
Analog Inputs 4

Assembly

Assembling the Data Collection Proto Board 5
Power Supply Options 5
Table 1. Components of the Data Collection Proto Board 6
Figure 2. Parts placement diagram 7

Use and Programming

User’s Perspective on the Demonstration Program 8
Programmer’s Perspective on the Demonstration Program 10
Program Listing: Example Data Logger 13

Component Specifications

National Semiconductor LP2950 and 2951 5-volt Regulators 25
LTC1298 2-channel, 12-bit Analog-to-Digital Converter w/Serial Interface 31
Xicor X25640P 8192-byte Serial EEPROMs (applies to X25128P) 51
NJU6355 Real-time Clock/Calendar w/Serial Interface 61

BS2 Data Collection Proto Board

Thank you for purchasing the BS2 Data Collection Proto Board, a carrier board with a suite of peripherals for the Parallax BASIC Stamp® II. The Data Collection Proto Board is a prefabricated hardware/software basis for applications requiring

- Up to 32kB of rewritable, nonvolatile EEPROM data storage
- Real-time clock/calendar
- Two-channel, 12-bit analog-to-digital converter
- Software-controlled peripheral power supply
- Support for three-wire synchronous serial bus devices
- Eight uncommitted digital input/output lines
- Grid-of-holes prototyping areas for additional circuitry
- BASIC-programmable microcontroller (BS2-IC, sold separately)
- BASIC-language subroutines and examples for using all peripherals

Although the board is ideally suited for data-logging applications, it isn't limited to them. Any BS2 control or monitoring application that could benefit from a high-precision ADC, real-time clock, or deep solid-state storage capabilities is a good candidate for construction on the Data Collection Proto Board.

Prerequisites for Using the Data Collection Proto Board

The Data Collection Proto Board is not an off-the-shelf data logger. It is a starting point for user-executed microcontroller designs. To use the board, you need a BS2 microcontroller and programming package, and knowledge of your application and the appropriate sensors, measurement methods, electronics, mathematics, etc.

Technical Support

If you need help with the Data Collection board, contact the manufacturer, Scott Edwards Electronics. If you need technical support for the BS2, contact its manufacturer, Parallax, Inc.

Contacting Us

Scott Edwards Electronics
PO Box 160
Sierra Vista, AZ 85636-0160 USA
ph: 520-459-4802; fax 520-459-0623
e-mail: 72037.2612@compuserve.com

Contacting Parallax

Parallax, Inc.
3805 Atherton Road, #102
Rocklin, CA 95765 USA
ph: 916-624-8333; fax 916-624-8003; BBS: 916-624-7101
e-mail: info@parallaxinc.com; file transfer via Internet: ftp.parallaxinc.com

Overview

The Data Collection Proto Board brings together a BASIC-programmable microcontroller (the BS2-IC) and a set of peripheral devices that extend its capabilities, particularly in the area of collecting, time-tagging, and recording data. Here's a quick tour of the board, shown schematically in figure 1:

BS2-IC. The BS2-IC is a single-board-computer (SBC) in 24-pin DIP format. Based on a 20-MHz RISC controller, the PIC16C57, the BS2 runs a control-oriented dialect of the popular BASIC programming language. The BS2 can execute 3000 BASIC instructions per second, and its repertoire includes synchronous and asynchronous (RS-232-style) serial I/O; frequency and pulse I/O; pseudo-sinewave and DTMF generation; frequency counting; 16-bit integer math; appliance-control X-10 output; and a host of others. The BS2, which normally draws about 8 mA, has short- and long-term sleep modes to further conserve power.

The BS2-IC is programmed through a temporary serial connection to a PC running the manufacturer's host software. Its onboard 2kB EEPROM can hold a program of approximately 500 BASIC instructions.

Because of its IC-style packaging, the BS2-IC requires a carrier board for convenient connection to peripherals, sensors, and outputs. The manufacturer (Parallax) offers a generic carrier board, consisting of a programming connector, battery clips, reset button, and grid-of-holes prototyping area. The Data Collection Proto Board extends this concept by putting commonly needed peripherals like power supplies, analog input, clock, and solid-state data storage onto the carrier board.

Main (Unswitched) Power Supply. The BS2-IC has an onboard 5V, 50mA low-dropout voltage regulator. This regulator is entirely adequate for powering the BS2 itself, but does not offer enough current capacity for some peripherals. Early BS2 models experienced a higher-than-normal failure rate of this onboard regulator. So, for flexibility and reliability, the Proto Board provides a 5V, 100mA regulator, consisting of U1 and associated components.

Peripheral (Switched) Power Supply. The Proto Board places a second 5-V, 100mA power supply under the control of a BS2 I/O pin (P9). This lets an application conserve battery power by shutting down portions of the circuit under program control. Since the installed peripherals (EEPROM, ADC, clock) draw at most a few mA, almost all of the regulator's capacity is available for user circuits.

Synchronous Serial Peripheral Bus. The BS2 has instructions for communicating with serial peripherals with SPI/Microwire-type three-wire interfaces. The Proto Board takes advantage of these instructions by assigning two I/O pins (P15 and P14) as dedicated clock and data lines to be shared by all standard or user-installed peripherals. To talk to a peripheral on the bus, the program activates the device's chip-select or enable line, then shifts data in or out using the BS2 instructions Shiftin and Shiftout.

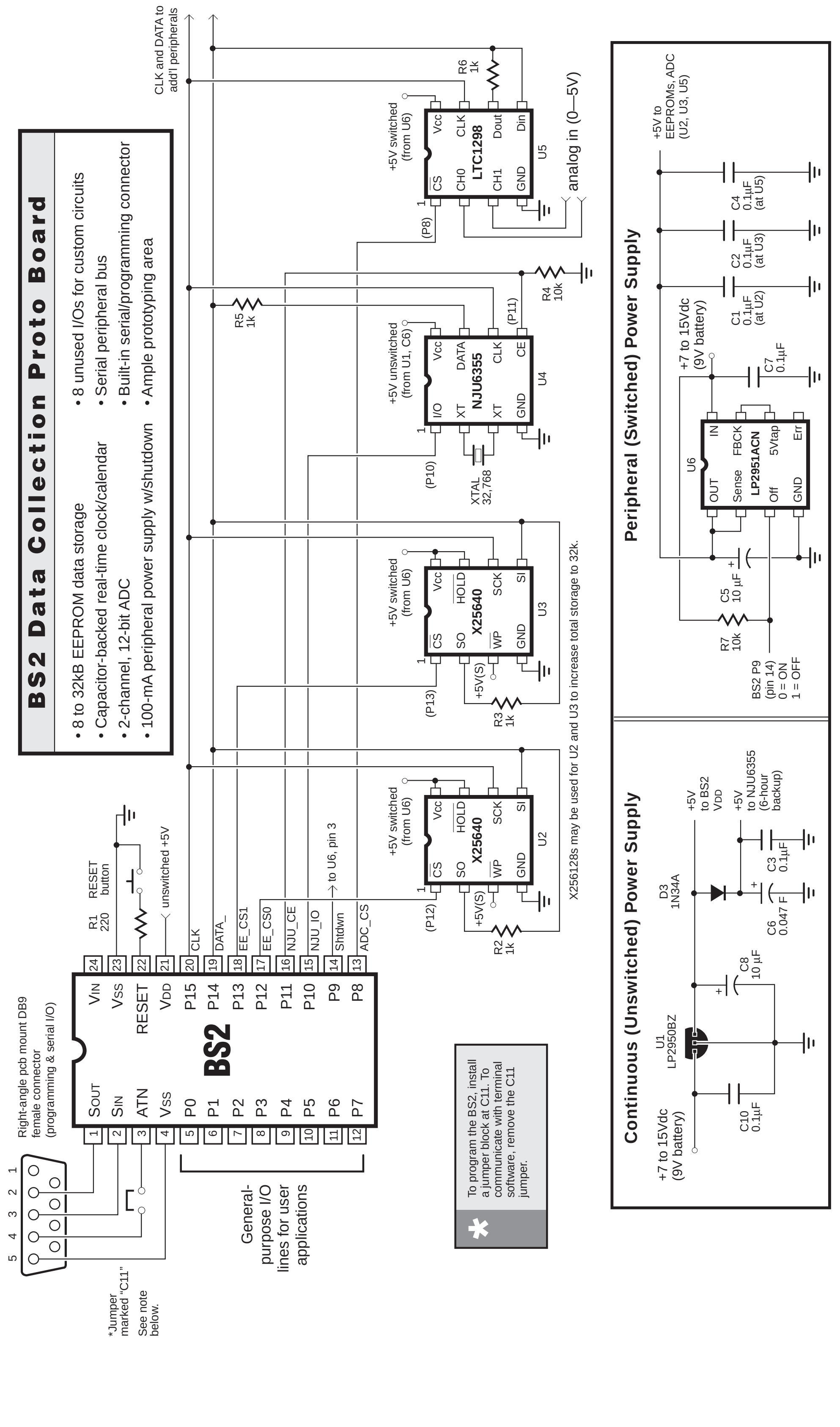

BS2 Data Collection Proto Board
• 8 to 32kB EEPROM data storage
• Capacitor-backed real-time clock/calendar
• 2-channel, 12-bit ADC
• 100-mA peripheral power supply w/shutdown
• 8 unused I/Os for custom circuits
• Serial peripheral bus
• Built-in serial/programming connector
• Ample prototyping area
Right-angle pcb mount DB9 female connector (programming & serial I/O)
*Jumper marked "C11" See note below.
To program the BS2, install a jumper block at C11. To communicate with terminal software, remove the C11 jumper.
General-purpose I/O lines for user applications
BS2
SOUT SIN ATN VSS P0 P1 P2 P3 P4 P5 P6 P7
VIN VSS RESET VDD P15 P14 P13 P12 P11 P10 P9 P8
R1 220
RESET button
unswitched +5V
to U6, pin 3
CLK DATA_ EE_CS1 EE_CS0 NJU_CE NJU_IO Shtdwn ADC_CS
CLK and DATA to add'l peripherals
+5V switched (from U6)
X25640 U2
X25640 U3
R2 1k
R3 1k
+5V(S)
(P12) (P13)
X256128s may be used for U2 and U3 to increase total storage to 32k.
+5V unswitched (from U1, C6)
NJU6355 U4
XTAL 32,768
(P10) (P11)
R4 10k
R5 1k
LTC1298 U5
(P8)
R6 1k
analog in (0—5V)
Continuous (Unswitched) Power Supply
+7 to 15Vdc (9V battery)
C10 0.1µF
U1 LP2950BZ
C8 10 µF
C6 0.047 F
D3 1N34A
C3 0.1µF
+5V to BS2 VDD
+5V to NJU6355 (6-hour backup)
Peripheral (Switched) Power Supply
+7 to 15Vdc (9V battery)
C7 0.1µF
U6 LP2951ACN
C5 10 µF
R7 10k
BS2 P9 (pin 14) 0 = ON 1 = OFF
C1 0.1µF (at U2)
C2 0.1µF (at U3)
C4 0.1µF (at U5)
+5V to EEPROMs, ADC (U2, U3, U5)

Figure 1. Schematic diagram of the Data Collection Proto Board

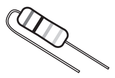
Solid-State Data Storage. The Proto Board accommodates 8 to 32kB of EEPROM (electrically erasable, programmable, read-only memory) for storing data—sort of a solid-state disk drive. The board hold two Xicor X25640P (8kB each) or two X25128P (16kB each). These devices may be read and written like RAM, but retain their contents with power off. This makes it possible to recover data recorded by a system that has experienced power failure or even physical damage. EEPROMs do eventually wear out after repeated writes, and should be replaced after 100,000 write cycles. Put in perspective, writing each address of 32kB of EEPROM at a rate of one byte per second would take almost 104 years to hit the nominal wear-out point.

Real-time Clock/Calendar. An NJU6355 clock/calendar chip keeps track of the current time, date, and day of the week in BCD (binary-coded decimal) format. Data from the clock may be used to time tag data recorded to EEPROM, or factored into program decisions (e.g., weekday vs. weekend in a setback thermostat). Power to the clock chip is backed up by a large capacitor, allowing it to keep time for 6 to 24 hours with system power removed. Timekeeping is unaffected by shutdown of the peripheral power supply.

Analog Inputs. An LTC1298 12-bit ADC (analog-to-digital converter) provides two voltage-measuring inputs with 1.22-millivolt resolution (5 volts range/4096 parts). Under software control, the inputs may be used as separate single-ended inputs (referenced to ground) or as one differential input. The ADC has a sample-and-hold feature that takes a snapshot of the input voltage at the instant of the measurement request, allowing it to sample noisy, or fast-changing inputs. Measurements are referenced to the peripheral power supply, which is exceptionally stable.

Assembling the Data Collection Proto Board

Table 1 lists the components of the board by part number and description. Labels on the circuit board, and figure 2, show where and how the components should be installed. Please review the following construction notes before installing the components:



- The resistors and diodes are mounted on end. To prepare them for installation, bend them into hairpin shapes as shown at right.
- Each diode location is marked with a printed stripe showing where the banded end (cathode) of the diode should go.
- Two different glass-body diodes are included in this package, a 1N34/270 and a 1N4148. The markings on the body are hard to read, but the 1N34/270 is much larger than the 1N4148.
- The locations of polarized capacitors such as C5 and C8 have a + marking by one pad. On the capacitors themselves, the + lead is distinguished by a + mark on the body, a longer lead, or a – mark on the *other* lead.
- IC outlines are marked with a notch at the pin-1 end. Install sockets so that their notches match the printed outlines. (Do *not* solder ICs other than U1 directly to the board; use the supplied sockets.)
- Install U1 so that its flat side matches the flat side of the outline printed on the board.
- The pads marked D1, D2, D4, and C9 are normally left empty.
- The crystal (small metal cylinder, marked XTAL on the board) may be installed standing up or lying down (flat to the board). A piece of adhesive foam tape may be used to secure and cushion the crystal in the lay-down position. Figure 2 shows the crystal installed with the foam tape.
- Depending on the option you choose for the power supply (discussed below), you may have one component left over, a 1N4148 diode.

Power Supply Options. The BS2 can operate from the Proto Board's regulated power supply, or from its own onboard regulator. The table below shows the relative virtues of the two power supplies.

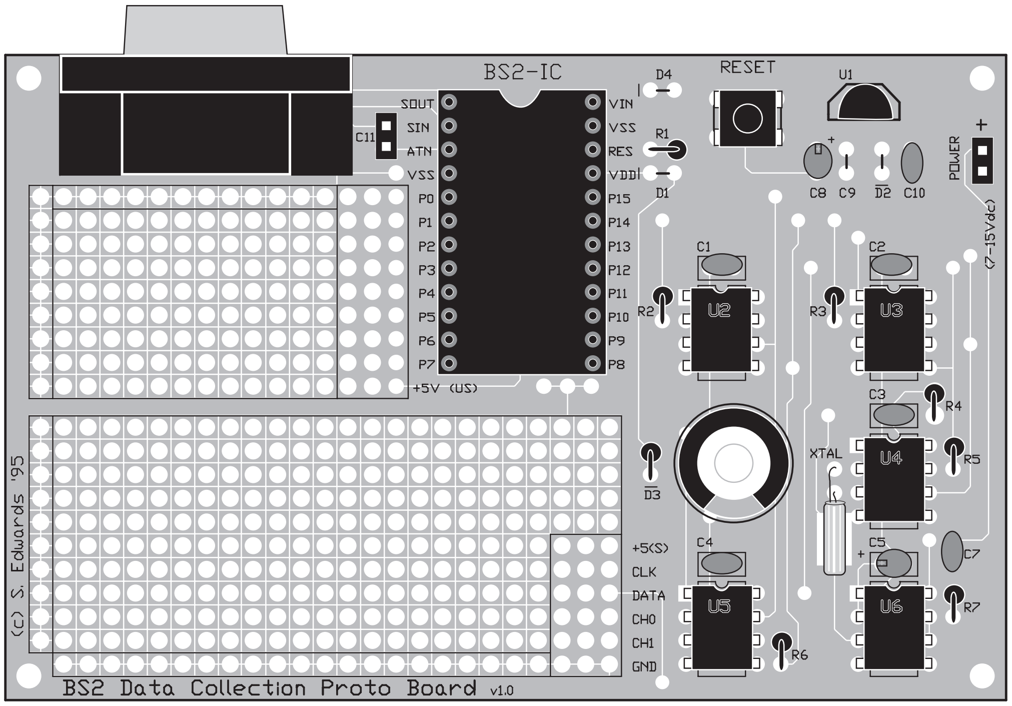
	BS2 Supply	**Proto Board Supply**
Output current (max):	50 mA	100 mA
Quiescent current:	20 µA	75 µA
Reversed input:	vulnerable to damage	resistant to damage

If you want to use the built-in BS2 power supply in order to reduce current draw in sleep mode, assemble the Proto Board without U1 and C8 and install a wire jumper or 1N4148 diode at location D4. The wire jumper will give you more battery life, but the diode will protect the fragile built-in voltage regulator from accidental reversed battery connection.

Table 1. Components of the Data Collection Proto Board

Qty	Designator(s)	Part/Value	Description	Vendor	Vendor PN
1	U1	LP2950BZ	5-volt regulator	Digi-Key	LP2950BZ-ND
2	U2, U3	X25640	8kB serial EEPROM	S. Edwards Elec.	16 kB EEPROM
2	U2, U3	X25128	16kB serial EEPROM	S. Edwards Elec.	32 kB EEPROM
1	U4	NJU6355ED	Clock/calendar	Digi-Key	NJU6355ED-ND
1	U5	LTC1298CN8	12-bit ADC	Digi-Key	LTC1298CN8-ND
1	U6	LP2951ACN	5-volt regulator	Digi-Key	LP2951ACN-ND
1	XTAL	32768 TF XTAL	32.768kHz crystal	Jameco	14584
6	C1–4, C7,10	0.1µF	50 WVDC cap	Hosfelt	15-696
2	C8,5	10µF	10–16 WVDC tant. cap	Digi-Key	P2026-ND
1	C6	0.047F	5.5 WVDC backup cap	Digi-Key	P6951-ND
1	R1	220Ω, 1/8W	5% carbon film resistor	Digi-Key	220EBK-ND
4	R2,3,5,6	1k, 1/8W	5% carbon film resistor	Digi-Key	1KEBK-ND
2	R4,7	10k, 1/8W	5% carbon film resistor	Digi-Key	10KEBK-ND
1	D3	1N34A or 1N270	Germanium diode	Jameco	35941
1	D4 (see pg. 5)	1N4148	Silicon diode	Jameco	36038
1	J1	DB9 fem	Right-angle DB9 fem	Jameco	104951
2	J2, C11*	1x2 hdr	2-contact male header	Jameco	103341
1	—	Jumper	Shorting block	Jameco	19140
1	—	DIP24 MP	24-pin DIP socket	Jameco	39351
5	—	DIP8	8-pin DIP socket	Jameco	112205
1	S1	pushbutton	6mm pushbutton	Digi-Key	P8037S-ND
1	—	9V snap w/hdr	Battery snap w/6" leads	Jameco	109153
1	pcb	BS2 DCPB	printed circuit board	S. Edwards Elec.	BS2 DCPB

*In the original DCPB, a 0.1-µF capacitor was installed at C11 to switch between programming and communication through the built-in DB9 connector. In later models, this cap is replaced with a header/jumper combination. Install the jumper to program; remove it to communicate serially through the DB9. This arrangement is more reliable.



D1, D2, D4, and C9 are normally not installed.
The banded (cathode) end of D3 goes to the pad nearest the D3 label.
All ICs except U1 are mounted in sockets. Notches mark the pin-1 ends.
Capacitors C5, C6, and C8 are polarized; make sure that their + leads go to the + pads.
C11 is *not* a capacitor; it's a 2-contact header block.

Figure 2. Parts placement diagram.

Using and Programming the Data Collection Proto Board

The Data Collection Proto Board comes with an example program on disk. The purpose of this program is twofold: (1) It allows you to test and demonstrate the board's peripherals, and (2) It provides a set of useful subroutines for incorporation into your own programs. Scott Edwards Electronics grants purchasers of any Data Collection Proto Board package that includes this disk unlimited rights to use, modify, and distribute this program or programs incorporating routines from this disk without further permission or payment.

User's Perspective on the Demonstration Program. To get acquainted with the example program, install a BS2-IC in the Proto Board socket, connect the Proto Board to your PC and power (red wire is +), install the jumper at C11, and download the program to the BS2 as described in the Parallax documentation.

Next, boot up terminal software. Set the terminal software for 9600 baud, no parity, 8 data bits, 1 stop bit. Also configure it to always execute a linefeed when a carriage return is received (in ProComm, press ALT-F3).

Remove the jumper from C11. Connect power to the Proto Board or, if power is already connected, press the RESET button. The following menu/prompt will appear on the screen:

```
BS2 Data Logger
(A)DC (T)ime (C)lock (R)ead (D)ump (E)rase (L)og
>
```

Typing the first letter of any item executes the corresponding instruction. If you leave power connected for more than 60 seconds without pressing any key, the program will begin logging data. If you want to prevent that, periodically press any key to restart the logger's internal countdown. You may also abort any instruction by pressing RESET. Here's a description of each of the menu items:

(A)DC: Press A to view ground-referenced voltage readings (0 to 4095 representing voltages of 0 to 5 volts) from the two channels of the ADC. This is handy for testing sensors or obtaining data for calibration purposes. Data displayed on the terminal screen looks like this:

```
Ch0: 557
Ch1: 3463
```

(T)ime: Press T to view the clock/calendar chip's current reckoning of the time and date. The terminal displays (for example):

```
11:09:14
01/10/96
```

If the time is not set all digits will display as E for "error."

(C)lock: If the time/date displayed by the logger's clock is wrong, pressing C lets you correct it. In the example display below, prompts are in bold type, user entries are light. DOW stands for day-of-week, and you may pick any day as number 1. Sunday or Monday are usual choices.

```
Year (YY): 96
Month (MM): 01
Day (DD): 10
DOW (1-7): 3
Hour (HH): 11
Minute (MM): 10
```

(R)ead: Pressing R causes the logger to read back data from its EEPROM memory. It will stop when it encounters blank memory; locates a bad checksum indicating the possibility of corrupted data; or reaches the end of the EEPROM. For neatly aligned display on the screen and compatibility with spreadsheets, the entries are separated by tab characters.

```
521  3466  10:11:06
522  3464  10:12:04
522  3461  10:13:03
523  3466  10:14:02
523  3462  10:15:00
524  3466  10:15:59
0    0     00:00:00
Stopped. End of data.
```

(D)ump: If you need to examine memory byte-by-byte, pressing D dumps a hexadecimal (hex) listing of the entire contents of memory. Each line begins with a hex number representing the starting point in memory of that line of data.

```
0000:  01 E3 0D 7A 09 01 08 7D 01 E3 0D 83 09 02 07 86
0010:  01 E3 0D 83 09 03 05 85 01 E5 0D 82 09 04 03 85
0020:  01 E4 0D 83 09 05 01 84 01 E5 0D 87 09 05 59 E1 ...
```

You can stop the hex dump by pressing RESET.

(E)rase: Pressing E writes 0s to all bytes of the installed EEPROM. This process can take 5 minutes to complete. When it's over, a new menu/prompt appears on the terminal display.

(L)og: Pressing L bypasses the 60-second delay and starts the logging process immediately. This is primarily useful for test runs of the logger, since disconnecting the serial cable between the PC and logger causes the logger to reset and start another countdown. During the logging process, the device will record measurements from each of the two ADC channels about once a minute, storing the readings, the time, and a checksum into EEPROM memory. The checksum serves as a simple indication of whether or not the data was recorded correctly, like the parity bit in serial communications. Between measurements, the logger goes into battery-saving sleep mode, reducing current draw to less than 1 mA.

Programmer's Perspective on the Demonstration Program. As the schematic in figure 1 shows, the Proto Board consists primarily of a serial bus supporting four peripheral ICs; the two EEPROMs, clock, and ADC. So it makes sense that the primary job of the program is to drive these devices using the BS2 instructions Shiftout and Shiftin.

The other defining feature is the extensive use of Serin and Serout to interact with the user as described in the section above. The program uses PBASIC's serial formatting capabilities to translate between internal data formats (like the binary-coded decimal array of the clock) and more human-friendly forms.

Since the program is extensively commented, I won't repeat those details here. What I will do is explain how to use and build on the peripheral subroutines for your own applications.

EEPROMs. The EEPROM routines let you write a byte of data to any address within the two-chip EEPROM memory. For example, suppose you have 16kB of EEPROM (two X25640P chips) installed. You set the address (EEaddr) to 9000 and gosub EEread to read back the data stored at that address. EEread first determines which EEPROM contains the given address. Since 9000 is greater than the highest address available in the lower EEPROM, EEPROM 0, the subroutine pickEE selects EEPROM 1. Another subroutine, send_addr, strips off the bits of the address exceeding 8191—the largest address in the X25640s—when it sends out the address message to EEPROM 1. So these subroutines cooperate to make two EEPROMs look like one big EEPROM. The address value EEaddr is unaffected by these adjustments. After EEread, EEaddr still contains 9000, but EEdata now contains the byte stored at that location in the combined EEPROM.

In our data-logging example, bytes are written to sequential EEPROM addresses starting at 0 and working upward toward the maximum address of the installed EEPROM. The send_addr subroutine can be instructed to increment (add 1 to) the address after each read or write. This is controlled by the bit EEincr; if it's 1, increment; if 0, don't increment. In the example above, if EEaddr were 9000 and EEincr were 1, after EEread, EEaddr would contain 9001.

The EEPROM routines can also maintain a checksum. What's a checksum? It's a running total of a series of values written to or read from the EEPROM. By comparing the checksum that was written to the checksum calculated when the same bytes were read back, the program can detect most kinds of errors in the data. If the checksums don't match, there's an error in the data or checksum itself.

The example program uses a faulty checksum as one of the conditions for ending a read of the data log. There are only two non-automatic aspects of using checksums: (1) The checksum byte, EEcksum, must be cleared to 0 before a set of data is read or written. (2) Before reading the checksum byte itself, checksum calculation must be turned off by setting the bit EEnoCk to 1.

Clock. The NJU6355 maintains 24-hour time and calendar information in an array of 13 four-bit digits. As long as its supply voltage is 2 volts or better, it keeps very good time. The memory-backup capacitor (C6) on the Proto Board can keep it running for more than 24 hours (at room temperature; less at extremes of heat and cold). Because of this backup capability, you don't have to rush to change the battery or worry about power failures.

The time/date is stored as binary-coded decimal (BCD) digits. Each digit is represented by a four-bit value. Four bits can represent values from 0 to 15, but BCD uses them only to represent the numbers from 0 to 9. It reserves values greater than 9 for error codes or other purposes.

A side effect of this way of storing a series of digits is that the hexadecimal (hex) numbering system is also based on four bits. Where hex overlaps BCD the two are the same; the numbers 0 through 9 hex have the same bit patterns as 0 through 9 BCD. So the example program uses the BS2's hex formatting instructions to read and write the BCD digits of the clock.

Your program can access the digits of the date and time through the array called DTG(). The program includes 13 constants that identify the array entry that contains a particular digit. For instance, the tens digit of the seconds is in DTG(S10s).

This brings up a couple of issues related to time. The program uses the BS2's Sleep function to power down the controller for 60 seconds between samples. Since it records the actual time each sample was taken, the relative inaccuracy of the Sleep interval doesn't matter much.

But suppose you wanted to sample at exact intervals; how would you handle that? Since our units of time don't follow normal math rules (22:25 + 4:40 is *not* 26:65!), adding the interval to the current time and waiting 'til then involves some awkward programming. A method I like works like this: Suppose you want an interval of 1 minute. Record the current minute digit in a variable, then enter a loop in which the BS2 naps for, say, 1.3 seconds at a time. When it wakes up, it gets the current time and compares the minutes digit to the stored digit. If it's the same, it goes back to sleep. If it's different, the program takes a measurement. It records the new minute digit, and goes back into the nap loop.

Using this method, the program will synchronize its sampling with the rollover from one minute to the next; taking a new measurement every minute, on the minute, plus or minus a factor for the nap time between checks of the clock. The sampling won't drift away from the actual clock time.

This technique can be extended to cover other intervals. Want 7 minutes between measurements? Create a variable called (for instance) countDown and set it to 7. Whenever the routine detects a change of the minutes digit, subtract 1 from countDown. If countDown is 0, take a measurement, otherwise go back to the nap loop.

ADC. The ADC is the easiest peripheral to use. There are two settings: mode and channel. To set the mode, you determine whether you want to measure the voltage between one of the channels and ground, or between the two channel inputs. If your measurement will be referenced to ground, you can use the two channels as separate inputs; otherwise you get a single “differential” channel.

The next choice is the channel itself, 0 or 1. For a ground-referenced measurement, this means just what it says—which channel to measure. For a differential measurement, it determines which channel will be considered positive with respect to the other.

Program Listing: Example Data Logger

```
' Program: LOG_1.BS2
' This program demonstrates use of the BS2 Data Collection Proto Board
' in a typical data-logging application. It exercises all of the
' board's built-in peripheral devices, including two 8kB (or 16kB)
' EEPROMs, real-time clock calendar, 12-bit ADC, and switchable
' precision power supply.

' [[ NOTE: lines that were too long were split with the ~ symbol used
' to indicate continuation on the next line.]]

'======================================================================
'        PIN ASSIGNMENTS, SYSTEM CONSTANTS, TEMPORARY VARIABLES
'======================================================================
' The BS2's upper port, pins 8 through 15, is dedicated to the
' operation of the data-collection peripherals. The lower port,
' pins 0 through 7, is available for your application.

CLK     con     15      ' Clock line for all serial peripherals.
DATA_   con     14      ' Data line for all serial peripherals.
EE_CS1  con     13      ' Chip-select line for EEPROM 1 (U3).
EE_CS0  con     12      ' Chip-select line for EEPROM 0 (U2).
NJU_CE  con     11      ' Chip-enable for NJU6355 clock/calendar.
NJU_IO  con     10      ' IO (read/write) for NJU6355; 1=write.
Shtdwn  con     9       ' Shutdown for peripheral power supply; 1=off.
ADC_CS  con     8       ' Chip-select line for LTC1298(88) ADC.
b96     con     $54     ' Baudmode for 9600 bps.
timeout con     60000   ' At startup, wait 60 seconds for serial command.
pwrOn   con     $31     ' Initial state of OUTH at peripheral power-up.
                        ' Turns on LP2951, but deselects all chips.
temp    var     byte    ' Temporary variable used in several routines.

'======================================================================
'                          SAMPLING CONSTANT(S)
'======================================================================
smplSiz con     8       ' Each sample consists of 8 bytes--
                        ' 4 of data (2 ADC results in 16-bit words)
                        ' 3 of time (12 BCD digits for hh:mm:ss)
                        ' 1 checksum.

'======================================================================
'                   EEPROM CONSTANTS AND VARIABLES
'======================================================================
WREN    con     6       ' EEPROM opcode to enable writes.
RDSR    con     5       ' EEPROM opcode to read the status register.
WRDI    con     4       ' EEPROM opcode to disable writes.
ReadEE  con     3       ' EEPROM opcode for read.
WriteEE con     2       ' EEPROM opcode for write.
```

```
' In the lines below, use the EEsize constant that matches the
' EEPROM memory installed on your board. Delete or comment out
' the other EEsize constant. This example uses X25640s, which have
' a capacity of 8191 bytes per chip.
'EEsize con     16383   ' Number of bytes per EEPROM (X25128).
EEsize  con     8191    ' Number of bytes per EEPROM (X25640).
'EEmax  con     32768   ' Total EEPROM capacity (X25128).
EEmax   con     16384   ' Total EEPROM capacity (X25640).
EEtop   con     EEmax-smplSiz    ' Last EE address with room for sample.
EEaddr  var     word    ' EEPROM address for reads and writes.
EEsel   var     bit     ' EEPROM select (0 or 1) for address.
EEdata  var     byte    ' Data written/read to/from EEPROM.
EEstats var     byte    ' Copy of the EEPROM status register.
EEbusy  var     EEstats.bit0     ' EEPROM busy bit (1 = busy).
EEcksum var     byte    ' Checksum for EEPROM reads and writes.
EEincr  var     bit     ' If =1, increment EEaddr after each write.
EEnoCk  var     bit     ' If =1, don't do checksum for 1 EE read.
EEblank var     bit     ' If =1, then a series of EEreads was 0.

'======================================================================
'                LTC1298 ADC CONSTANTS AND VARIABLES
'======================================================================
' The LTC1298 is a 12-bit ADC that measures 0-5 volts, referenced to
' the 5-volt output of the peripheral power supply (LP2951).
' The 1298 has two modes. As a single-ended ADC, it measures the
' voltage at one of its inputs with respect to ground. As a differential
' ADC, it measures the difference in voltage between the two inputs.
' The sglDif bit determines the mode; 1 = single-ended, 0 = differential.
' When the 1298 is single-ended, the oddSign bit selects the active input
' channel; 0 = channel 0 (pin 2), 1 = channel 1 (pin 3).
' When the 1298 is differential, the oddSign bit selects the polarity
' between the two inputs; 0 = channel 0 is +, 1 = channel 1 is +.
' The msbf bit doesn't matter in this application, and is set to 1
' by the same logical OR that sets the start bit in the ADC subroutine.

ADcnfg  var     nib              ' Stores configuration bits for ADC.
ADres   var     word             ' Variable to hold 12-bit AD result.
ADstb   var     ADcnfg.bit0      ' Start bit for comm with ADC (always 1).
ADmode  var     ADcnfg.bit1      ' Single-ended (1) or differential (0).
ADch    var     ADcnfg.bit2      ' Channel or + selection (0 or 1).
ADmsbf  var     ADcnfg.bit3      ' Output 0s after xfer (doesn't matter).

'======================================================================
'            NJU6355 CLOCK/CALENDAR CONSTANTS AND VARIABLES
'======================================================================
' The NJU6355ED clock/calendar chip maintains a 13-digit BCD account
' of the current year, month, day, day of week, hour, minute, and
' second. The clock subroutines transfer this data to/from a 13-nibble
' array in the BS2's RAM called "DTG" for "date-time group." The
' constants below allow you to refer to the digits by name; e.g.,
' "Y10s" is the tens digit of the year. Note that there's no "am/pm"
' indicator--the NJU6355 uses the 24-hour clock. For instance, 2:00 pm
' is written or read as 14:00 (without the colon, of course).
Y10s    con     1       ' Array position of year 10s digit.
```

```
Y1s     con     0       '  "       "     "  year 1s    "
Mo10s   con     3       '  "       "     "  month 10s  "
Mo1s    con     2       '  "       "     "  month 1s   "
D10s    con     5       '  "       "     "  day 10s    "
D1s     con     4       '  "       "     "  day 1s     "
H10s    con     8       '  "       "     "  hour 10s   "
H1s     con     7       '  "       "     "  hour 1s    "
M10s    con     10      '  "       "     "  minute 10s "
M1s     con     9       '  "       "     "  minute 1s  "
S10s    con     12      '  "       "     "  second 10s "
S1s     con     11      '  "       "     "  second 1s  "
day     con     6       '  "       "     "  day-of-week (1-7) digit.
digit   var     nib     ' Number of 4-bit BCD digits read/written.
DTG     var     nib(13) ' Array to hold "date/time group" BCD digits.

'=====================================================================
'                         DEMONSTRATION PROGRAM
'=====================================================================
' This program implements a simple, user-friendly data logger.
' It can be used in two ways: If a PC running terminal software [9600
' baud, N81, carriage returns (CR) converted to CR plus linefeed,
' no handshaking] is connected, the program displays a menu and prompt
' that allows the user to display the ADC readings; view/set the clock,
' read, dump, or erase memory; or start the logger. If no terminal is
' connected, or the user doesn't press a key within a preset time, the
' program automatically starts logging data. It takes samples from
' both ADC inputs every 60 seconds and records these, along with a
' time-tag and error-detection checksum, into EEPROM.

' Initial setup.
  OUTH = pwrOn          ' Get ready to turn on peripherals.
  DIRH = $FF            ' Set all bits of high port to output.
  DIRL = $FF            ' Set all bits of low port to output.

' setup =============
' When you apply power to the board, the BS2 will send a message at
' 9600 bps through the built-in serial port asking for setup
' instructions. If you don't press a key within timeout seconds, the
' program will begin logging data.
setup:
  serout 16,b96,[CR,CR,"BS2 Data Logger"]         ' Sign-on and choices.
  serout 16,b96,[CR,"(A)DC (T)ime (C)lock (R)ead (D)ump (E)rase (L)og",CR,">"]
  serin 16,b96,timeout,log,[temp]        ' Look for instruction.

' If no instruction arrives within timeout milliseconds, then log data,
' else get choose from setup menu.
  temp = temp & $0DF                     ' Convert to uppercase.
  IF temp = "A" THEN showADC             ' Show current ADC measurement.
  IF temp = "T" THEN time                ' Show the current time/date.
  IF temp = "C" THEN clock               ' Set the clock/calendar.
  IF temp = "R" THEN readDataLog         ' Read data w/checksums.
  IF temp = "D" THEN dumpDataLog         ' Read entire EEPROM.
  IF temp = "E" THEN eraseLog            ' Write 0s to all EEPROM bytes.
  IF temp = "L" THEN log                 ' Start logging.
goto setup                               ' Unrecognized entry; try again.
```

```
' showADC =============
' Take a single-ended measurement from each of the two ADC channels
' and send it out the built-in serial port at 9600 baud. Handy for
' calibrating analog conditioning circuitry.
showADC:
  ADmode = 1                     ' Single-ended measurement.
  for ADch = 0 to 1              ' Both channels, 0 and 1.
    gosub ADread                 ' Read selected channel.
    serout 16,b96,[CR,"Ch",DEC ADch,": ",DEC ADres]      ' Show result.
  next                           ' Next channel.
goto setup

' time =============
' Put the time and date from the NJU clock chip on the screen and
' go back to the setup routine.
time:
  gosub read_clock       ' Update DTG data.
  gosub show_time        ' Display the time.
  gosub show_date        ' Display the date.
goto setup               ' Back to setup.

' clock =============
' Let the user set the current time and date via the set_clock subroutine.
' This code just jury-rigs an IF ... THEN GOSUB ... instruction, which
' PBASIC lacks. When done, goes back to setup.
clock:                   ' Set the internal clock.
  gosub set_clock
goto setup               ' Return to setup for more instructions.

' log =============
' Take a single-ended reading from both ADC channels every minute
' and records the data followed by the time and a checksum byte to
' the EEPROM starting at address 0.
log:
  ADmode = 1             ' Set single-ended mode for ADC.
  EEaddr = 0             ' Start at address 0.
  EEincr = 1             ' Enable auto-increment of EEPROM address.
logLoop:
  if EEaddr <= EEtop then nextAddr       ' If EEaddr > EEtop, EEaddr = 0
  EEaddr = 0
nextAddr:
  EEcksum = 0                    ' Clear the checksum for new data.
  gosub read_Clock               ' Get the current time.
for ADch = 0 to 1                ' Do both channels.
  gosub ADread                   ' Get ADC result.
  EEdata = ADres.highbyte        ' Record the high byte of ADres word.
  gosub EEnableAndWrite          ' Write the data to EEPROM.
  EEdata = ADres.lowbyte         ' Record the low byte of ADres word.
  gosub EEnableAndWrite          ' Write the data to EEPROM.
next
  EEdata.highnib = DTG(H10s)     ' Record hours.
  EEdata.lownib = DTG(H1s)
  gosub EEnableAndWrite
  EEdata.highnib = DTG(M10s)     ' Record minutes.
  EEdata.lownib = DTG(M1s)
  gosub EEnableAndWrite

  EEdata.highnib = DTG(S10s)     ' Record seconds.
  EEdata.lownib = DTG(S1s)
  gosub EEnableAndWrite

  EEdata = EEcksum               ' Store the checksum.
  gosub EEnableAndWrite

' Turn off the peripheral power supply and sleep for approximately
' 60 seconds (according to the BS2's internal timing). The actual
' sleep time will be the nearest multiple of 2.3 seconds (59.8)
' plus or minus about a second for oscillator tolerance.
  high Shtdwn                    ' Turn off peripherals.
  sleep 60                       ' Shut down for ~ 60 seconds.
  low Shtdwn                     ' Turn on power for clock.
goto logLoop                     ' Else get another sample.

' dumpDataLog =============
' Dump the entire contents of the EEPROM(s) as hex digits separated
' by spaces. For compatibility with terminal software and convenient
' reckoning of addresses, this loop presents 16 entries on one line.
' This output mode would be handy for recovering data from EEPROM in
' the event a hardware or software problem caused the checksum method
' to fail. It's also useful for debugging new data formats, since it
' lets you look at all the values in the EEPROM.
dumpDataLog:
  EEaddr = 0                             ' Start at address 0.
  EEincr = 1                             ' Autoincrement EEPROM addresses.
dumpLoop:
  serout 16,b96,[CR,hex4 EEaddr,":  "]   ' Start line "<addr>: "
  for temp = 1 to 16                     ' Put 16 entries across screen.
    if EEaddr = EEmax then setup ' End of memory: exit.
    gosub EEread                         ' Read a byte from EEPROM memory.
    serout 16,b96,[hex2 EEdata," "]
  next                                   ' Show "address: <data data...>"
goto dumpLoop                            ' Do it again.

' eraseLog =============
' Erase (write 0s) to every address in the combined EEPROM storage bank.
' This process takes several minutes to complete, after which the
' "BS2 Data Logger" prompt reappears on the screen. This is a brute-
' force erase routine, in that it does not take advantage of the
' EEPROM page-write capabilities, or the possibility of writing to the
' two EEPROMs simultaneously. These steps might speed the erasure
' process to a minute or less, but at the expense of extra code.
eraseLog:
  EEdata = 0: EEincr = 0                 ' Write 0s; don't increment addr.
  for EEaddr = 0 to (EEmax-1)            ' Write to all EEPROM addresses.
    gosub EEnableAndWrite
  next
goto setup                               ' Done: back to setup menu.
```

```
' readDataLog =============
' Read time-tagged data entries from the EEPROM until an invalid
' checksum, a string of 0s, or the end of memory. Note that the
' string-of-zeros condition will be falsely triggered at midnight,
' January 1, if both ADC readings happen to be zero.
readDataLog:
  EEaddr = 0            ' Start at EEPROM address 0.
  EEincr = 1            ' Enable auto-increment of EEPROM address.
next_entry:
  EEblank = 1           ' Reset 0 flag to test for sequences of 0s.
  EEcksum = 0           ' Reset checksum to test for data validity.
  serout 16,b96,[CR]    ' Start a new line on the terminal screen.
  for ADch = 0 to 1     ' Retrieve each of the recorded ADC channels.
    gosub EEread        ' Read high byte from the EEPROM.
    ADres.highbyte = EEdata     ' Put it into high byte of ADres.
    gosub EEread                ' Repeat this for the low byte.
    ADres.lowbyte = EEdata
    serout 16,b96,[dec ADres,TAB]       ' Display ADres word in decimal..
  next                                  ' ..followed by a tab character.
  gosub EEread                          ' Now get time digits..
  serout 16,b96,[hex2 EEdata,":"]       ' ..show "hh:" (hours)
  gosub EEread
  serout 16,b96,[hex2 EEdata,":"]       ' .."mm:" (minutes)
  gosub EEread
  serout 16,b96,[hex2 EEdata]           ' .."ss" (seconds)
  EEnoCk = 1    ' Now, get checksum. Turn off read checksum calculation.
  gosub EEread  ' And read in the written checksum byte.
  if EEdata <> EEcksum then invalid_sum                 ' Verify checksum.
  if EEcksum = 0 and EEblank = 1 then invalid_blank     ' Check for 0s.
  if EEaddr > EEtop then endOfLog
goto next_entry                         ' Do the next one.

invalid_blank:          ' Display message when blanks encountered.
  serout 16,b96,[CR,"Stopped. End of data.",CR]
goto setup

invalid_sum:            ' Display messgage when checksums don't match.
  serout 16,b96,[CR,"Stopped. Invalid checksum.",CR]
goto setup

endOfLog:               ' Display message when end of memory reached.
  serout 16,b96,[CR,"Stopped. End of EEPROM.",CR]
goto setup

' ==================================================================
'                       EEPROM SUBROUTINES
' ==================================================================
' Depending on the address in EEaddr, select either EEPROM 0 (addresses
' 0-8191) or EEPROM 1 (addresses 8192-16383). Note that this routine
' activates the selected EEPROM, but leaves it to the calling code
' to deactivate it. An additional effect of the read and write
' routines is to add the value of EEdata read or written to a variable
' called EEcksum. This checksum can serve as a marker to separate
' valid data from leftover garbage in the EEPROM. It's not infallible,
' but combined with time/date tagging, it can be very effective.
' pickEE =============
' Based on the address in EEaddr and the size of the installed
' EEPROMs, turn on the correct EEPROM's enable line.
pickEE:
  EEsel = 0                     ' If EEaddr <= EEsize, then EEsel=0..
  IF EEaddr <= EEsize then skip1 ' ..Else EEsel = 1.
  EEsel = 1
skip1:
  low (EE_CS0+EEsel)            ' Activate selected EEPROM.
return                          ' Return to caller.

' EEdisable =============
' Write-protect (disable) the EEPROM.
EEdisable:
  gosub pickEE
  shiftout DATA_,CLK,msbfirst,[WRDI]     ' Send the disable opcode.
  high (EE_CS0+EEsel)                    ' Deactivate the EEPROM.
return

' EEnableAndWrite =============
' Check the EEPROM and wait until it's not busy. Then write enable
' the EEPROM, and write the byte stored in EEdata to the address
' stored in EEaddr. Update the checksum byte by adding EEdata to
' the previous contents of EEcksum. Finally, if EEincr = 1,
' increment EEaddr to point to the next EEPROM address.
EEnableAndWrite:
  gosub Read_stats                       ' Get the status register.
  if EEbusy = 1 then EEnableAndWrite     ' If busy, check again.
  low (EE_CS0+EEsel)                     ' Activate EEPROM.
  shiftout DATA_,CLK,msbfirst,[WREN]     ' Send the enable opcode.
  high (EE_CS0+EEsel)                    ' Deactivate the EEPROM.
  low (EE_CS0+EEsel)                     ' Activate EEPROM.
  shiftout DATA_,CLK,msbfirst,[WriteEE]  ' Send write opcode.
  gosub Send_addr                        ' Send the address.
  shiftout DATA_,CLK,msbfirst,[EEdata]   ' Send the data.
  high (EE_CS0+EEsel)                    ' Deactivate the EEPROM.
  EEcksum = EEcksum + EEdata             ' Update checksum.
return                                   ' Return to program.

' EEread =============
' Read data from the EEPROM. Use the address in EEaddr;
' put data into EEdata. Add the value of EEdata into the byte
' EEcksum as an informal check of the validity of a sequence
' of bytes. Setting the bit EEnoCk turns off checksum calculation
' for one byte read--usually the checksum itself.
' This code also clears the readZero bit whenever it retrieves
' a non-zero byte. This lets other parts of the program recognize
' unbroken sequences of zeros, which would otherwise pass the
' checksum test.
EEread:
  gosub pickEE                           ' Select the EEPROM.
  shiftout DATA_,CLK,msbfirst,[ReadEE]   ' Send the read opcode.
  gosub Send_addr                        ' Send the address.
  shiftin DATA_,CLK,msbpre,[EEdata]      ' Send the data.
  high (EE_CS0+EEsel)                     ' Deactivate the EEPROM.
  if EEnoCk = 1 then noCkSum              ' No cksum if bit=1
  EEcksum = EEcksum + EEdata              ' Update checksum.
noCkSum:
  EEnoCk = 0                              ' Reset no-check bit
  if EEdata = 0 then readZero             ' If data=0, don't clear EEblank.
  EEblank = 0
readZero:
return                                    ' Return to program.

' Send_addr =============
' Send the EEPROM address for a read or write and optionally increment
' the address afterwards for writing sequential addresses.
Send_addr:
  shiftout DATA_,CLK,msbfirst,[(EEaddr & EEsize)\16]
  EEaddr = EEaddr + EEincr                ' Increment address if bit=1.
return                                    ' Return to program.

' Read_stats =============
' Read the EEPROM status register, which contains the write-in-progress
' or "busy" bit. This should be checked before each write to ensure
' that the EEPROM is ready to accept new data.
Read_stats:
  gosub pickEE
  shiftout DATA_,CLK,msbfirst,[RDSR]      ' Send read-stats opcode.
  shiftin DATA_,CLK,msbpre,[EEstats]      ' Get status.
  high (EE_CS0+EEsel)                     ' Deactivate the EEPROM.
return                                    ' Return to program.

' ===================================================================
'                   NJU6355 CLOCK/CALENDAR SUBROUTINES
' ===================================================================

' read_clock =============
' Get the current date/time group from the NJU6355 clock and store
' it in the array DTG(n).
read_clock:
  low NJU_IO                                      ' Set for read.
  high NJU_CE                                     ' Select the chip.
  for digit = 0 to 12                             ' Get 13 digits.
    shiftin DATA_,CLK,lsbpre,[DTG(digit)\4]       ' Shift in a digit.
  next                                            ' Next digit.
  low NJU_CE                                      ' Deselect the chip.
return                                            ' Return to program.

' write_clock =============
' Get the time stored in DTG(n) and write it to the NJU6355 clock.
' Note that the NJU6355 does not allow you to write the seconds digits.
' If clears the seconds digits when written, so if you set it for
' 08:30 (hh:mm), when the write is complete, the NJU6355 starts at
' 08:30:00 (hh:mm:ss).
write_clock:
  high NJU_IO                   ' Set for write.
  high NJU_CE                   ' Select the chip.
  for digit = 0 to 10           ' Write 11 digits.
```

```
    shiftout DATA_,CLK,lsbfirst,[DTG(digit)\4]    ' Shift out a digit.
  next                                            ' Next digit.
  low NJU_CE                                      ' Deselect the chip.
return                                            ' Return to program.

' set_clock =============
' Set the clock/calendar based on data entered by the user at
' 9600 bps through the built-in serial-port connector.
set_clock:
  serout 16,b96,[CR,"Year (YY): "]: gosub get2BCD         ' Get year.
  DTG(Y10s) = temp.highnib:DTG(Y1s) = temp.lownib        ' Store.
  serout 16,b96,[CR,"Month (MM): "]: gosub get2BCD        ' Get month.
  DTG(Mo10s) = temp.highnib:DTG(Mo1s) = temp.lownib      ' Store.
  serout 16,b96,[CR,"Day (DD): "]: gosub get2BCD          ' Get day.
  DTG(D10s) = temp.highnib:DTG(D1s) = temp.lownib        ' Store.
  serout 16,b96,[CR,"DOW (1-7): "]                        ' Get day of wk.
  serin 16,b96,[HEX1 temp]
  DTG(day) = temp.lownib                                  ' Store.
  serout 16,b96,[CR,"Hour (HH): "]: gosub get2BCD         ' Get year.
  DTG(H10s) = temp.highnib:DTG(H1s) = temp.lownib        ' Store.
  serout 16,b96,[CR,"Minute (MM): "]: gosub get2BCD       ' Get month.
  DTG(M10s) = temp.highnib:DTG(M1s) = temp.lownib        ' Store.
  gosub write_clock
return

' show_date =============
' Display the date stored in DTG.
show_date:
  serout 16,b96,[CR,HEX DTG(Mo10s), HEX DTG(Mo1s),"/",HEX DTG(D10s), ~
~       HEX DTG(D1s),"/",HEX DTG(Y10s), HEX DTG(Y1s)]
return

' show_time =============
' Display the time stored in DTG.
show_time:
  serout 16,b96,[CR,HEX DTG(H10s), HEX DTG(H1s),":",HEX DTG(M10s), ~
~       HEX DTG(M1s),":",HEX DTG(S10s), HEX DTG(S1s)]
return

' get2BCD =============
' Get two BCD digits using the HEX2 modifier. Within the range 0-9,
' hex and BCD are the same, so this is useful for setting the clock
' in its BCD format.
get2BCD:
  serin 16,b96,[HEX2 temp]
return
```

```
' =================================================================
'                 LTC1298 12-BIT ADC SUBROUTINE
' =================================================================

' ADread =============
' Configure the ADC for the channel/mode set by the bits of ADcnfg,
' then take an ADC reading, returning the result in the word variable
' ADres. The program should set up the mode bit (ADmode; 1 = single
' ended, 0 = differential) and channel (ADch; 1 = ch1; 0 = ch 0) before
' executing a Gosub to this routine. Note that when the mode is
' differential, the channel selection indicates which channel is
' considered to be the + connection for the measurement.
ADread:
  ADcnfg = ADcnfg | %1001                ' Set start bit and msbf.
  low ADC_CS                             ' Activate the ADC.
  shiftout DATA_,CLK,lsbfirst,[ADcnfg\4] ' Send config bits.
  shiftin DATA_,CLK,msbpost,[ADres\12]   ' Get data bits.
  high ADC_CS                            ' Deactivate the ADC.
return                                   ' Return to program.
```