PC/104 Compliant Load Cell Controller 754P-LC4

Table of Contents

Scanning Devices Inc.

PC/104 Compliant Load Cell Controller 754P-LC4 Software Libraries and Sample Programs 754P-SW4

Part Number 754P-LC4 Includes:

SD815 Module, compliant with PC/104 standards. (Previous revisons were SD811 and SD814)
EPROM programmed with microcode: NJ004

Part Number 754P-SW4 Includes:

3.5" Floppy disk in MS-DOS format with two directories

754P_LC4 - Contains source and executable programs

.cpp files are C++ sources
.exe files are executable programs
.dsk files and .prj files are C++ project files

LIB_REF - Contains documentation for the module and libraries
Distribution includes Adobe Acrobat (.pdf) and Microsoft Word (.doc) format

Source programs are written and copyrighted by Scanning Devices Inc. for use with Borland International Incorporated C++ Version 3.0 or compatible, Microsoft Corporation MS-DOS Version 5.0 or later. Neither Borland C++ nor MicroSoft MS-DOS are distributed with these software programs.

Utilities for 754P-LC4 are distributed as Util815.cpp in source form. Documentation may make reference to these utilities as SD811.h or SD815.h Other .cpp files are sample programs in source form which use these Utilities. The sample programs are integrated into Project LANCELOT, Version 2, lancekey.exe and lancelot.exe, a sample system for demonstrating the 754P-LC4 load cell controller with a load cell or other comparible signal source and suitable configured DOS compatible computer system.

The sample programs have two purposes:

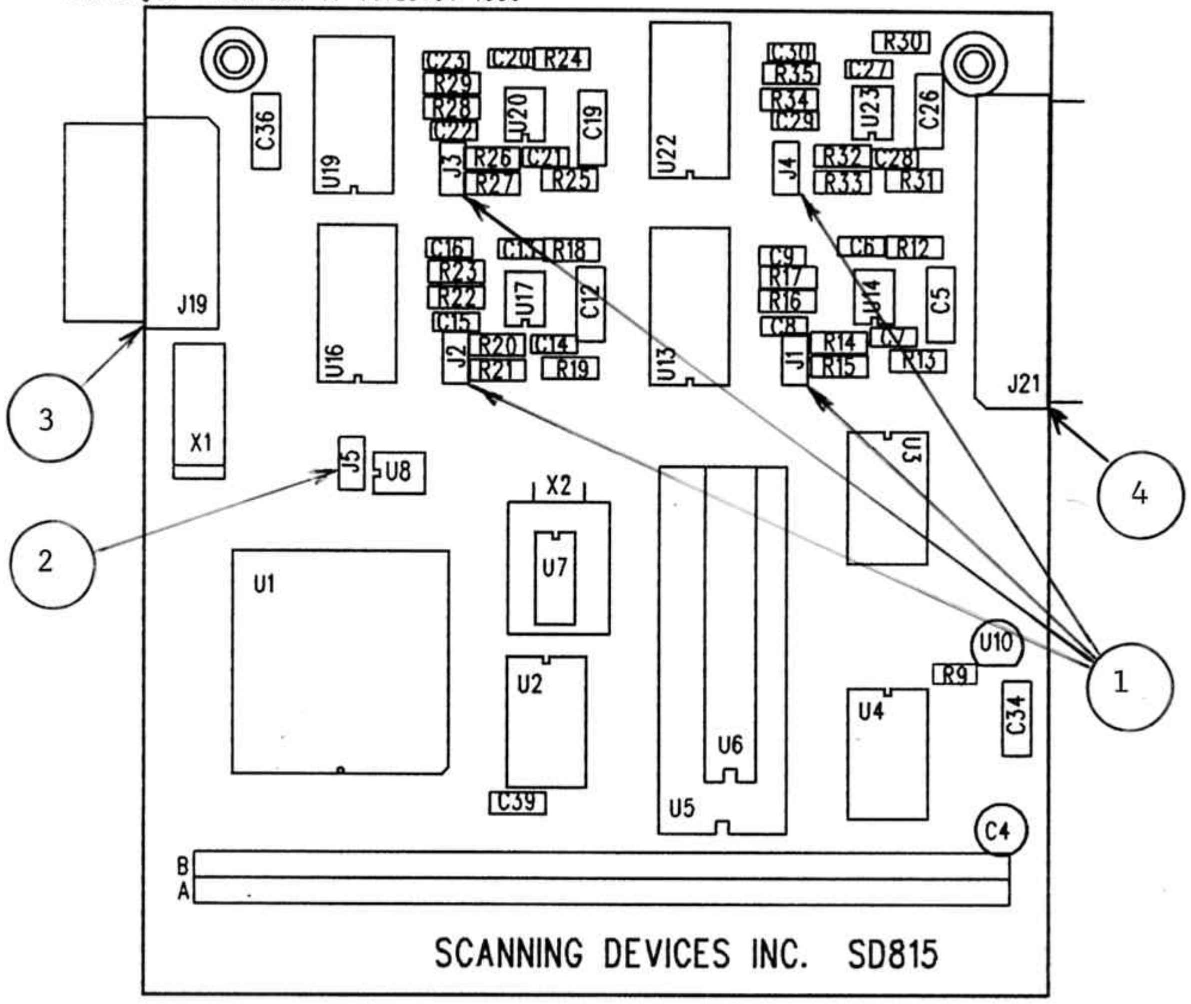
1. To provide a means of demonstrating the functions of the 754P-LC4 load cell controller so that a user can verify its proper operation and determine its suitability for a specific use.

2. To provide examples of application programming methods and techniques for to the 754P-LC4 load cell controller to aid a user in designing and implementing a system to obtain results specific to a user's situation.

The sample program is not intended to be used for purposes other than those listed above. Additional functions, specific to a user's application, such as such as file management, user interface, error detection and response must be added before a system is complete.

The sample programs are distributed in source form so that they may be integrated into a user's program development environment. Some conversion may be required if the program development environment is not equipped with the prerequisite software listed above. Consult the doctumenation for your program development tools for specifics.

sd815.job - Mon Jun 17 14:29:34 1996



"component" side

SD815 Module Setup

1. J1 - J4 Preamplifier gain selection
   Jumper for 2 mV/V Load Cells
   No Jumper for 3 mV/V Load Cells
2. J5 Microcontroller Reset
3. J19 Digital Input/Output Connector
4. J21 Load Cell Connector

Scanning Devices SD815 module

The SD815 module complies with PC/104 Specification Version 1.0, March 1992. It is an 8-bit stackthrough module as described in specification 2.2.1

The stackthrough connector J1/P1 provides electrical signals consistent with an 8-bit slot on an IBM or compatible ISA bus.

See the module layout drawing, Figure 1, page 3, for reference to the following notes:

1. **Microcontroller Reset**

The SD815's on-board microcontroller is equipped with a user-accessible reset input. The microcontroller is reset on a high logic level (+5 Volts) at terminal 2 of jumper J5. Terminal 1 of jumper J5 is +5 Volts. Therefore, connecting the two terminals of jumper J5, holds the microcontroller reset; disconnecting the two terminals starts the microcontroller in a known state.

The microcontroller is reset automatically on ***power up***. The microcontroller is ***not*** affected by the PC's reset and is ***not*** reset by the RESETDRV signal on the PC/104 bus. (This allows independent operation of the 754P-LC4 and the PC.)

An external reset may be valuable, especially during software development and testing. A normally opened push button switch wired to J5 makes a convenient reset button, comparible to the reset button on most PC's.

2. **Preamplifier Input Gain**

The 754P-LC4 module uses an instrumentation amplifier to condition each channel's load cell signal for input to the analog-to-digital converter. The gain of the instrumentation amplifier can be selected via installation/removal of jumper J1 - J4, one jumper for each channel to select one of two popular load cell signal ranges. Separate jumpers allow the gain of each channel to be set indepently. The jumper's number on Figure 1 corresponds to the channel number.

With the jumper removed, the amplifier gain is suitable for load cells with 3 mv/V signals. The instrumentation amplifer converts the load cell's 15 millivolt signal at full capacity to 5 volts for the analog-to-digital converter.

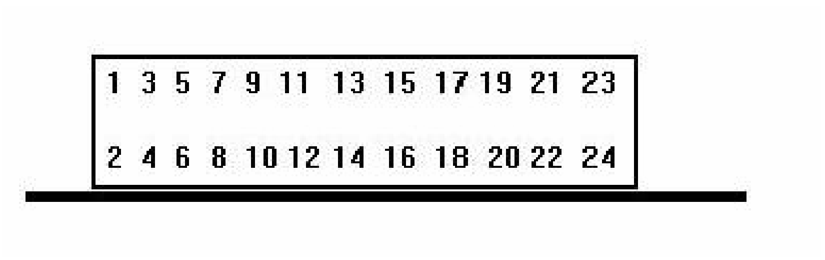
With the jumper installed, the amplfier gain is suitable for load cells with 2 mv/V signals.The instrumentation amplifer converts the load cell's 10 millivolt signal at full capacity to 5 volts for the analog-to-digital converter.

Note: the selection of gain will depend on the load cell signal specification *and* the weighing range relative to the load cell's capacity.

3. **I/O Connections**

External signals connect to SD815 via one 16-pin connector (digital input and outputs) and one 24-pin connector (load cell signals).

Pin numbering is: (As viewed from the male connector and the component side of the circuit board):



Connector Pin Assignments are:

Load Cell Connection

J21 - Load Cell Connector

	Pin Numbers			
	Channel 1	Channel 2	Channel 3	Channel 4
Sense +	1	7	13	19
Excitation +	2	8	14	20
Signal +	3	9	15	21
Signal -	4	10	16	22
Sense -	5	11	17	23
Ground	6	12	18	24

Note:

For 6 wire load cells, attach as indicated above.

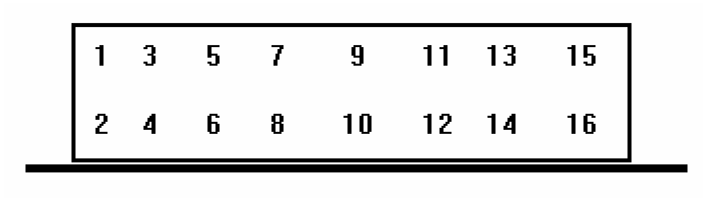
For 4 wire load cells (no sense wires) one wire jumpers is required for each channel:

Attach Excitation Plus to Sense Plus

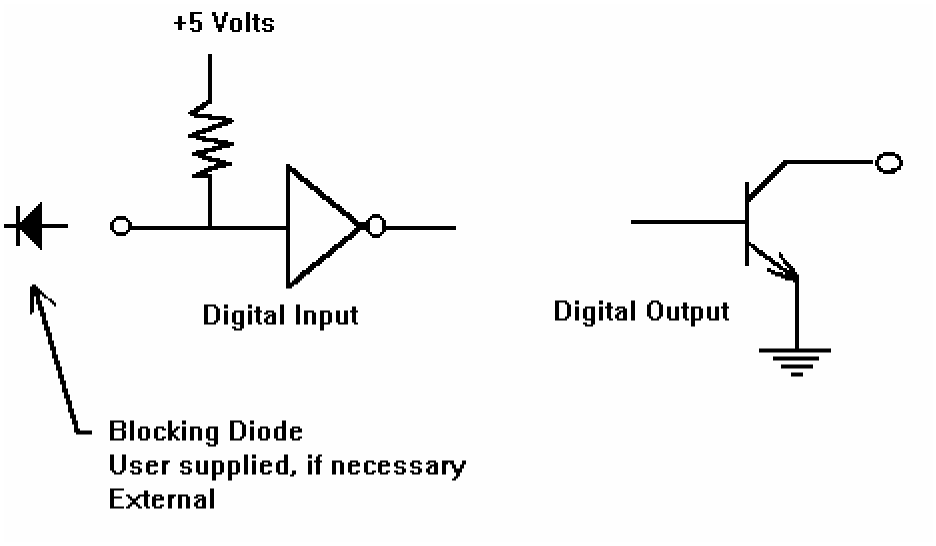
Jumper Pins 1-2, 7-8, 13-14, and 19-20

Digital Input and Output Connection

J19 - Digital Input/Output Connector



Digital Input and Output Circuitry



Digital Inputs

Digital inputs are logic gate inputs with a nominal 10 Kohm pull up resistor to +5 volts. These are designed for current sinking sensors or other current sinking devices. Sinking current (1/2 milliamp) from the input to signal ground asserts the input. ***Do not apply external signals of more than 5 volts. Applying more than 5 volts to the digital input will cause immediate failure.*** Use a blocking diode for connection to sensors which switch voltage instead of current.

Digital Outputs

Digital outputs are uncommitted NPN transistor collectors conducting to signal ground. Transitors are rated at 50 milliamps conducting, 30 volts non-conducting. ***Do not connect the digital ouputs to a voltage source without providing a load. Exceeding the 50 milliamp specification will cause immediate failure.***

Pins	Usage
1 and 2	Not Used
3 and 4	Not Used
5	Output_2
6	Output_4
7	Output_1
8	Output_3
9	Signal Ground
10	Signal Ground
11	Input_2
12	Input_1
13	Input_4
14	Input_3
15 and 16	+5 Volts

Operation of the 754P-LC4

The 754P-LC4 is a micro-controller based peripheral controller with five general functions combined in a PC/104 compliant module

1. Load Cell excitation supply (4 volts DC) sourced from the PC +5 volt supply
2. Four Low level signal pre-amplifiers from millivolts to volts
3. Four High precision analog-to-digital converters
4. Computation and storage for calculation of engineering units and comparision to setpoints
5. Data transfer via PC bus.

This combination make the 754P-LC4 ideal for real time applications requiring the combination of bridge-style transducers and PC data processing and storage.

Each load cell channel on the 754P-LC4 uses a 24-bit sigma-delta analog to digital converter and a post conversion digital filter to generate digital data from millivolt level transducer signals. Each converter operates independently, generating a new data measurement at a nominal rate of 200 Hz. The configuration of each digital filter can be loaded via a software command from the PC to deliver digital data as generated by its A/D converter to process that data using a moving average filtering technique.

The on-board microcontroller manages data generated by the A/D converters, monitors the PC bus for commands and data at its bus address, reads digital input and generates digital outputs.

The microcontroller processes a polling list of functions requiring service. When a service is initated (A/D conversion, bus transfer, setting outputs, etc), it is allowed to run to completion before the next service is initiated. Services have been designed to be completed quickly to minimize waiting.

PC Control of Module Operation

The module's functional polling list is controlled by two on board variables which may be read or written by the PC, allowing the PC to control module operations.

The ***Status*** variable specifies if a load cell channel is included in the polling list. If the corresponding bit in the *Status* variable is set, the channel is included in the polling list and measurements are made on that channel. If the corresponding bit in the *Status* variable is clear, the channel is not included in the polling list and no measurements are made.

The ***Mode*** variable specifies if a digital output is controlled (energized or de-energized) by the comparison between measured weight and stored setpoint. If the corresponding bit in the *Mode* variable is set, (and the channel is active as specified by the *Status* variable) then a digital output is associated with the load cell channel and is set or cleared depending on the result of the comparision between the measurement made on that channel and the associated preset value. If the corresponding bit in the *Mode* variable is clear, then the digital output is independent of the measured weight and may be controlled by the PC.

See the GetCStatus, SetCStatus, GetMode, SetMode commands in the LINK ROUTINE section and in the Lancelot Library

With the *Mode* variable clear or the *Status* clear for a specific channel, the associated digital output is under PC control. The PC can set or clear the digitial output.

See the SetIO and GetIO commands in the LINK ROUTINE section.

PC Bus Interface

The bus interface consists of a data port and a control status register. The module is addressed when the bus address specified by signal SA_9 – SA_2 is equal to the register address stored in the module. Address line SA_0 determines whether the module's data port or control status register is addressed. Logic Zero on SA_0 selects the data port. Logic One on SA_0 selects the control status register.

The bus address is loaded from the module's eeprom on power up and can be modified under program control by the PC. When modified, the revised bus address is stored in on-board eeprom and used until altered again by the PC. This feature facilitates installation of multiple modules on a single PC bus, with automatic re-configuration under software control.

The data port initiates an 8-bit bi-directional transfer to or from the PC bus in response to the IOW (I/O Write) or IOR (I/O Read) strobe signal.

The control status register transfers an 8-bit byte in response to IOW or IOR, although only certain data is meaningful. The CSR contains the status of the port's input and output buffers, providing indicators that data is available at the port for reading or that the port is available for writing.

CSR.0 - Input Buffer Full Flag

The IBF bit is set to logical 1 when data is loaded into the module's input buffer under control of of IOW (data written to the module). The IBF bit is cleared when the module's processor transfers the data from the input buffer.

When CSR.0 is logical 1, a write to the module will over-write unread data. Before writing to the module, the PC program should interrogate the IBF flag at CSR.0 to insure that buffer space is available to accept the write.

CSR.1 - Output Buffer Full Flag

The OBF flag is set to logical 1 when the module's processor writes data to the module's output buffer. The OBF bit is cleared by the trailing edge of the IOR signal on a bus read from the data port address.

When CSR.1 is logical 0, there is no new data available at the module address. Before reading from the module, the PC program should interrogate the OBF flag at CSR.1 to insure that data at the port is current.

These two flags provide a means of handshaking between the PC and the 754P-LC4 to insure that data properly passes between the two.

Sample PC Program illustrating bus interface operations

The following C++ function, SD815_GetData() illustrates the handshaking between the PC and the load cell controller.   The C++ function calls two additional functions, described below, which test the port control status register (CSR). The microcontroller's response is shown on the right.

C++	Microcontroller
Check that Input Buffer Full Flag is zero at the specified bus addresss if not, returns timeout error	
Writes a command to the microcontroller requesting data, the symbolic constant SD815_GETDATA and the specified channel	
The write operation sets the input buffer full flag	Microcontroller detects that the input buffer full is set and reads the instruction, clearing input buffer full
Wait for input buffer full flag to clear	
Sets up a loop to read three bytes	Interprets the instruction, sets up a loop to write three bytes. Checks output buffer full and if clear, moves first byte to dataport, setting output buffer full.
Waits for output buffer full returns timeout error if not set	
Reads byte to szBuffer, read clears the output buffer full flag	Waits until the output buffer full flag is cleared returns a timeout error if not cleared
Waits for output buffer full for next byte	When cleared, moves the next byte to dataport
Repeat for third byte	Repeats for third byte
Reads fourth byte SD815_OK	After third byte, transfers a fourth byte, the symbolic constant SD815_OK
Function return value indicates that data transfer has completed normally or with error	

```
int SD815_GetData(unsigned int *pData, int Bus_Address, int iChannel)
{
        register int iIndex;
        unsigned int *pWord;
        char szBuffer[3];
        char *pByte;

  if(WaitForIBFClear(Bus_Address)!=SD815_OK)
    return(SD815_IBF_TIMEOUT);

  outp(Bus_Address,SD815_GETDATA+iChannel);

        for(iIndex=0; iIndex<3; iIndex++)
  {
    if(WaitForOBFSet(Bus_Address)!=SD815_OK)
      return(SD815_OBF_TIMEOUT);

                szBuffer[iIndex]=inp(Bus_Address);
  }

  if(WaitForOBFSet(Bus_Address)!=SD815_OK)
    return(SD815_OBF_TIMEOUT);

  if(inp(Bus_Address)!=SD815_OK)
    return(SD815_ERROR);

        pByte=&szBuffer[0];
        pWord=(unsigned int *)pByte;
        *pData=*pWord;

        return(SD815_OK);
}
```

The following C++ function tests the CSR Input Buffer Full Flag, returning a clear indicator or an error on timeout beyond 2 seconds. This function is called while waiting for the Microcontroller to accept data written to the dataport address. It is also used to prevents a subsequent write from over-writing data not yet moved from the dataport by the microcontroller.

```
int WaitForIBFClear(int Bus_ Address)
{
        time_t tStart;

        if((inp(Bus_Address+CSR)&0x01)==0)
                return(SD815_OK);

        for(tStart=time(NULL); (time(NULL)-tStart)<2; )
                if((inp(Bus_Address+CSR)&0x01)==0)
                        return(SD815_OK);

        return(SD815_ERROR);
}
```

The following C++ function tests the CSR Output Buffer Full Flag, returning a set indicator or an error on timeout beyond 2 seconds. This function is called while waiting for the Microcontroller to make data available for reading at the dataport address. It is also used to prevent re-reading data which has already been read from the dataport address.

```
int WaitForOBFSet(int Bus_Address)
{
        time_t tStart;
        int iData;

        iData=inp(Bus_Address+CSR);

        if((iData&0x02)!=0)
                return(SD815_OK);

        for(tStart=time(NULL); (time(NULL)-tStart)<2; )
        {
                iData=inp(Bus_Address+CSR);

                if((iData&0x02)!=0)
                        return(SD815_OK);
        }

        return(SD815_ERROR);
}
```

Link Routine

The Microcontroller's Software Link Module controls data transfers between 754P-LC4 load cell interface and the PC bus in response to PC commands

The 754P-LC4 accepts a single byte command from PC written to the module bus base address.

During each measurement cycle, 754P-LC4 interrogates the input buffer full flag. This flag is set by a PC write to the bus base address set in the on-board address register latch.

Whenever the microcontroller finds the input buffer full flag set, it calls the Link subroutine to interpret and process the PC command.

Link responds by interpreting the instruction and executing a pre-programmed data transfer procedure.   PC "read" commands initiate data transfer from 754P-LC4 to PC. PC "write" commands initiate data transfers from PC to 754P-LC4.

PC "read" commands

Data transfers - Transfers are sequences of single byte transfers.   The output buffer full flag is used as the synchronizing device between the PC and microcontroller in the following manner:

When the microcontroller moves a byte to the dataport, the output buffer full flag is set. This flag indicates that data at the dataport is valid and should be read by the PC. The microcontroller waits until the output buffer full flag is cleared by the PC's reading the dataport. When the flag is cleared, the microcontroller moves the next byte in the transfer to the dataport, which sets the output buffer full flag. This sequence continues for as many bytes as are needed to complete the transfer.

Timeout - If the output buffer full flag remains set for more than approximately 750 microseconds (meaning the PC has not read data presented at the dataport, the microcontroller times out and abandons the command processing. The microcontroller resets and restarts.

Data Format - Format depends on the data type being transferrred. The data type is consistend with C++ data types; that is, a floating point variable is transferred as four bytes, least significant byte first. A short integer is transferred as a single byte.

Termination - After the requested data is transferred, the microcontroller writes a terminating byte SD815_OK="0" as an indicator of successful completion. If the PC does not receive the terminating byte, an error condition can be inferred and error recover (retry, etc) should be undertaken.

PC "write" commands

Data transfers - Transfers are sequences of single byte transfers.   The input buffer full flag is used as the synchronizing device between the PC and microcontroller in the following manner:

When the PC writes a byte to the dataport, the input buffer full flag is set. This flag indicates that new data is available at the dataport and should be read by the Microcontroller. When the microcontroller reads the byte from the dataport, the input buffer full flag is cleared. The microcontroller processes each byte as received and waits for the input buffer full flag to be set by the next PC write.   When the input buffer full flag is cleared, the PC moves the next byte in the transfer to the dataport, which sets the input buffer full flag again. This sequence continues for as many bytes as are needed to complete the transfer.

Timeout - If the input buffer full flag remains clear for more than approximately 750 microseconds (meaning the PC has not written data expected at the dataport), the microcontroller times out and abandons the command processing. The microcontroller resets and restarts.

Data Format - Format depends on the data type being transferrred. The data type is consistend with C++ data types; that is, a floating point variable is transferred as four bytes, least significant byte first. A short integer is transferred as a single byte.

Termination - After the requested data is transferred, the microcontroller writes a terminating byte SD815_OK="0" to the dataport as an indicator of successful completion. If the PC does not receive the terminating byte, an error condition can be inferred and error recover (retry, etc) should be undertaken.

Link Commands

The following Link commands have been implemented. Specifications for these commands are listed on the following pages.

Channel Specific Commands

GETCAL - Reads the calibration settings used to compute weight
SETCAL - Writes the calibration settings used to compute weight
GETWEIGHT - Reads the most recently computed weight measurement
GETDATA - Reads the most recently computed load cell measurement (filtered raw data)
GETFILTER - Reads the digital filter setup parameters
SETFILTER - Writes the digital filter setup parameters
GETCSTATUS - Reads the activated/de-activated status of each channel
SETCSTATUS - Writes the channel status byte, used to activate/de-activate each channel
GETMODE - Reads the mode byte, indicates the source of digital output control
SETMODE - Writes the output control byte, used to select output control

Module Specific Commands

GETIO - Reads status of digital inputs and digital outputs
SETIO - Sets digital outputs
GETPARAM - Reads (uploads) preset variables
SETPARAM - Writes (downloads) preset variables
GETMODULEID - Reads the module's serial number
SETBUSADDRESS - Alters the modules's bus address
RESET - Initiates a software reset and restart at the microcontroller

For channel specific commands (those which read or write data for a specific load cell channel), the channel identifier is specified by the least two significant bits of the command. For example: GETCAL refers to channel 1 data, GETCAL+1 refers to channel 2 data, GETCAL+2 refers to channel 3 data and GETCAL+3 refers to channel 4 data.

For module level commands (those which read or write one data set for the moduel), the least significant two bits of the command are assumed to be zero.

Specific Commands

PARAMETERS

Pneumonic (for reference only)

GETPARAM

Command Character:

60 (hexidecimal) 96 (decimal) " ' " (Accent) ASCII

Type:

PC read

Data Transferrred:

Four preset variables in sequence: PS1, PS2, PS3, PS4

Data Format:

Floating point, each variable is transferred as 4 bytes, least significant byte first

After transferring 16 bytes, a 17th byte = 0 (hexidecimal) is transferred to indicated end of transfer

Pneumonic (for reference only)

SETPARAM

Command Character:

64 (hexidecimal) 100 (decimal) "d" (ASCII)

Type:

PC write

Data Transferrred:

Four preset variables in sequence PS1, PS2, PS3, PS4

Data Format:

Floating point, each variable is transferred as 4 bytes, least significant byte first

After receiving 16 bytes, one byte = 0 (hexidecimal) is transferred from the microcontroller to PC to indicated end of transfer

After the transfer, the four preset variables are stored in on-board eeprom and retained until altered by a subsequent SETPARM command.

Pneumonic (for reference only)

GETCAL

Command Character:

68 (hexidecimal) 104 (decimal) "h" (ASCII) (references channel 1)

Type:

PC read

Data Transferrred:

Four calibration coefficients used to calculate weight from raw data.

These coefficients are described by the C++ structure

```
struct CalibrationSettings
{
        float Scale;
        float K;
        unsigned long Tare;
        unsigned long Ref;
};
```

754P-LC4 calculates weight from raw data (counts) using the formula

    Weight = Scale * K * (Data - Zero)

where Data is the measurment produced by the analog-to-digital conversion of the load cell signal and Zero = Tare - Ref

K and Ref are set by factory calibration of the analog circuitry and are modified by field calibration

Data Format:

As described in the C++ structure, Scale and K are 4-byte floating point variables transferred Least significant byte first. Tare and Ref are unsigned long integers (4-bytes) transferred least significant byte first.

Execution of the GETCAL command results in the transfer of 16 data bytes in the following sequence:

Scale
K
Tare
Ref

After transferring 16 bytes, a 17th byte = 0 (hexidecimal) is transferred to indicated end of transfer

Pneumonic (for reference only)

CALIBRATION

SETCAL

Command Character:

6C (hexidecimal) 108 (decimal) "l" (ASCII) (references channel 1)

Type:

PC write

Data Transferrred:

Four calibration coefficients used to calculate weight from raw data.

These coefficients are described by the C++ structure

```
struct CalibrationSettings
{
        float Scale;
        float K;
        unsigned long Tare;
        unsigned long Ref;
};
```

754P-LC4 calculates weight from raw data (counts) using the formula

Weight = Scale * K * (Data - Zero)

where Data is the measurment produced by the analog-to-digital conversion of the load cell signal and Zero = Tare - Ref

Data Format:

As described in the C++ structure, Scale and K are 4-byte floating point variables transferred Least significant byte first. Tare and Ref are unsigned long integers (4-bytes) transferred least significant byte first.

The GETCAL command expects the transfer of 16 data bytes in the following sequence:
Scale
K
Tare
Ref

After receiving 16 bytes, one byte = 0 (hexidecimal) is transferred by the microcontroller to indicated end of transfer

After the transfer, the four preset variables are stored in on-board eeprom and retained until altered by a subsequent SETCAL command.

The transferred values of these variables are used in the next weight calculation.

Pneumonic (for reference only)

GET DATA / WEIGHT

GETDATA

Command Character:

74 (hexidecimal) 116 (decimal) "t" (ASCII) (References channel 1)

Type:

PC read

Data Transferrred:

24 bit result of analog-to-digital conversion of load cell signal, modified by digital filter

Data Format:

three binary bytes are transfered, least significant byte first

After transfering three bytes, one byte = 0 (hexidecimal) is transfered to indicate end of transfer.

Pneumonic (for reference only)

GETWEIGHT

Command Character:

70 (hexidecimal) 112 (decimal) "p" (ASCII) (References channel 1)

Type:

PC read

Data Transferrred:

Current value of the floating point Weight variable

Data Format:

Single four-byte floating point variable

754P-LC4 calculates weight from raw data (counts) using the formula

Weight = Scale * K * (Data - Zero)

This weight is compared to setpoint generated by the SETPARAM command and used to set outputs according to Mode logic.

Four bytes are transferred, least significant byte first.

After transmitting 4 bytes, one byte = 0 (hexidecimal) is transferred by the microcontroller to indicated end of transfer

I / O STATES

Pneumonic (for reference only)

GETIO

Command Character:

4C (hexidecimal) 76 (decimal) "L" (ASCII)

Type:

PC read

Data Transferrred:

One byte containing 4 input and 4 output states

Data Format:

Single unsigned character

Bits in this character represent the state of the four input and 4 outputs on the 754P-LC4

LSB bit 1	Output 1	0 = output de-energized	1 = output energized
bit 2	Output 2	0 = output de-energized	1 = output energized
bit 3	Output 3	0 = output de-energized	1 = output energized
bit 4	Output 4	0 = output de-energized	1 = output energized
bit 5	Input 1	1 = input closed, low;	0 = input open, high
bit 6	Input 2	1 = input closed, low;	0 = input open, high
bit 7	Input 3	1 = input closed, low;	0 = input open, high
MSBbit 8	Input 4	1 = input closed, low;	0 = input open, high

After transmitting one byte, one byte = 0 (hexidecimal) is transferred by the microcontroller to indicated end of transfer

I / O STATES

Pneumonic (for reference only)

SETIO

Command Character:

50 (hexidecimal) 80 (decimal) "P" (ASCII)

Type:

PC write

Data Transferrred:

One byte containing 4 input and 4 output states

Data Format:

Single unsigned character

Bits in this character represent the state of the four input and 4 outputs on the 754P-LC4

LSB bit 1	Output 1	0 = output de-energized 1 = output energized
bit 2	Output 2	0 = output de-energized 1 = output energized
bit 3	Output 3	0 = output de-energized 1 = output energized
bit 4	Output 4	0 = output de-energized 1 = output energized
bit 5	Input 1	x (don't care)
bit 6	Input 2	x (don't care)
bit 7	Input 3	x (don't care)
MSBbit 8	Input 4	x (don't care)

After receiving one byte, one byte = 0 (hexidecimal) is transferred by the microcontroller to indicated end of transfer.

After the transfer, the output pins are set with the data transferred in bits 1-4. The inputs are set and allowed to assume the state dictated by the signals (if any) attached to the inputs.

Note: The result of this transfer is ***not stored in eeprom.*** The outputs are not retained through a power down or reset.

DIGITAL FILTER

Pneumonic (for reference only)

SETFILTER

Command Character:

7C (hexidecimal) 125 (decimal) "|" (ASCII Vertical Line) (References channel 1)

Type:

PC write

Data Transferrred:

one byte containing the digital filter setup variable

Data Format:

unsigned integer representing the filter selected

Valid values for this integer are:

- 1 - no filtering, "data" is the A/D conversion result.
- 2 - "data" is the moving average of two A/D samples
- 4 - "data" is the moving average of four A/D samples
- 8 - "data" is the moving average of eight A/D samples

Any other value is considered an error. If any value other than these four are received, the microcontroller sets filter = 1.

After receiving one byte, one byte = 0 (hexidecimal) is transmitted to indicate end of transfer.

After the end of transfer, the filter parameter is stored in eeprom and retained until changed by subsequent SETFILTER commands.

After storing the filter variable in eeprom, the new value is immediately used in processing the next Analog-to-Digital measurement.

Note: when the filter variable is changed, the digital filter is purged. Depending on the value of the filter variable selected, several measurements may be required to reload the digital filter. No new "data" variable is produced until after the digital filter is reloaded.

The GETDATA and GETWEIGHT commands transfer the current "data" or "weight" variable. A GETDATA immediately after a SETFILTER may obtain "data" which reflect measurements generated by the previous filter variable. Allow sufficient time for the digital filter reload before processing new data.

DIGITAL FILTER

Pneumonic (for reference only)

GETFILTER

Command Character:

78 (hexidecimal) 120 (decimal) "x" (ASCII) (References channel 1)

Type:

PC read

Data Transferrred:

one byte containing the digital filter setup variable

Data Format:

unsigned integer representing the filter selected

After one bye is transferred, a second byte = 0 (hexidecimal) is transmitted to indicate end of transfer.

CHANNEL STATUS

Pneumonic (for reference only)

GETCHANNELSTATUS

Command Character:

54 (hexidecimal) 84 (decimal) "T" (ASCII)

Type:

PC read

Data Transferrred:

One byte containing status of 4 channels

Data Format:

Single unsigned character

Bits in this character represent the state of the four load cell channels on the 754P-LC4

LSB bit 1	Output 1	0 = channel 1 not active	1 = channel 1 active
bit 2	Output 1	0 = channel 2 not active	1 = channel 2 active
bit 3	Output 1	0 = channel 3 not active	1 = channel 3 active
bit 4	Output 4	0 = channel 4 not active	1 = channel 4 active
bit 5	x - don't care		
bit 6	x - don't care		
bit 7	x - don't care		
bit 8	x - don't care		

After one bye is transferred, a second byte = 0 (hexidecimal) is transmitted to indicate end of transfer.

CHANNEL STATUS

Pneumonic (for reference only)

SETCHANNELSTATUS

Command Character:

58 (hexidecimal) 88 (decimal) "X" (ASCII)

Type:

PC write

Data Transferrred:

One byte containing the status of the four load cell channels

Data Format:

Single haracter

Bits in this character represent the state of the four load cell channels on the 754P-LC4

LSB bit 1	Output 1	0 = channel 1 not active	1 = channel 1 active
bit 2	Output 1	0 = channel 2 not active	1 = channel 2 active
bit 3	Output 1	0 = channel 3 not active	1 = channel 3 active
bit 4	Output 4	0 = channel 4 not active	1 = channel 4 active

At the end of the transfer, each channel status is updated and stored in on-board eeprom.

After one bye is received, a single byte = 0 (hexidecimal) is transmitted to indicate end of transfer.

OPERATING MODE

Pneumonic (for reference only)

GETMODE

Command Character:

80 (hexidecimal) 128 (decimal) "NULL" (ASCII)

Type:

PC read

Data Transferrred:

Mode variable

Data Format:

Single character

One byte is transferred

Bits in this character represent the state of output control in each of the four load cell channels on the 754P-LC4

Note: Output control can be enabled only if the status of a channel is Active.

LSB bit 1	Output 1	0 = independent of weight
		1 = output 1 controlled by weight and preset, on if weight=>preset 1
bit 2	Output 2	0 = independent of weight
		1 = output 2 controlled by weight and preset, on if weight=>preset 2
bit 3	Output 3	0 = independent of weight
		1 = output 3 controlled by weight and preset, on if weight=>preset 3
bit 4	Output 4	0 = independent of weight
		1 = output 4 controlled by weight and preset, on if weight=>preset 4

After transferring the bytes, a second byte = 0 (hexidecimal) is transferred to indicated end of transfer

OPERATING MODE

Pneumonic (for reference only)

SETMODE

Command Character:

84 (hexidecimal) 132 (decimal) "EOT" (ASCII)

Type:

PC write

Data Transferrred:

Mode variable

Data Format:

Single Character

One byte is transferred.

Bits in this character represent the state of output control in each of the four load cell channels on the 754P-LC4.

Note: Channel status takes precendent over Operating Mode. Output control can be enabled only if the status of the specified channel is Active.   If the channel status is de-active, no measurement is made and no change is made to the output even if the Mode bit for that channel is set.

LSB bit 1	Output 1	0 = independent of weight
		1 = output 1 controlled by weight and preset, on if weight=>preset 1
bit 2	Output 2	0 = independent of weight
		1 = output 2 controlled by weight and preset, on if weight=>preset 2
bit 3	Output 3	0 = independent of weight
		1 = output 3 controlled by weight and preset, on if weight=>preset 3
bit 4	Output 4	0 = independent of weight
		1 = output 4 controlled by weight and preset, on if weight=>preset 4

After the byte is transferred, the microcontroller transfers one byte = 0 (hexidecimal) to indicate successful transfer.

After receiving the byte, the microcontroller writes the mode variable to on-board eeprom and resets the 754P-LC4's operating mode.

MODULE ID

Pneumonic (for reference only)

GETID

Command Character:

44 (hexidecimal) 68 (decimal) "D" (ASCII)

Type:

PC read

Data Transferrred:

4 bytes containing the module serial number, least significant byte first

Data Format:

long integer representing the module serial number

After four bytes are transferred, a fifth byte = 0 (hexidecimal) is transmitted to indicate end of transfer.

Pneumonic (for reference only)

SETBUSADDRESS

Command Character:

48 (hexidecimal) 72 (decimal) "H" (ASCII)

Type:

PC write

Data Transferrred:

2 bytes containing the new bus address, least significant byte first

Data Format:

short integer representing the bus address

After two bytes are transferred, a single byte = 0 (hexidecimal) is transmitted to indicate end of transfer.

After the end-of-transfer is completed, the new bus address is stored in the module's eeprom and loaded into the modules address register latch.

Note: This instruction sequence should be executed from the PC referencing the ***old bus address***. After successful completion, the ***new bus address*** should be used to reference the module.

Note: After changing the bus address, make sure the PC system can address the bus; ie, a means of the PC's knowing the bus address is required.

Pneumonic (for reference only)

DATACOUNT

GETDATACOUNT

Command Character:

88 (hexidecimal) 136 (decimal) (ASCII “backspace”) (References channel 1)

Type:

PC read

Data Transferrred:

One byte containing the datacount variable for the specified channel. The DataCount variable is used to synchronize operations between the 754P-LC4 and the PC. The variable is incremented in the 754P-LC4 each time a new A/D conversion result becomes available on each channel. The value 255 increments to 0.

Data Format:

unsigned integer representing the current datacount variable for the specified channel.

The operation consists of a one-byte transfer. The SD815_OK variable typically transferred at the end of an operation is ***not*** transferred, allowing all 256 possible values of DataCount to be valid.

Pneumonic (for reference only)

GETADREGISTER

Command Character:

8C (hexidecimal) 140 (decimal) "FF" (ASCII Form Feed Character) (References channel 1)

Type:

PC read

Data Transferrred:

Four bytes containing the Analog-to-Digital setup register variable, least significant byte first, followed by the terminating byte. This variable is used to control the converter's precision and sampling rate.

Data Format:

Long integer representing the current Analog-to-Digital converter register setup variable for the specified channel.

Pneumonic (for reference only)

SETADREGISTER

Command Character:

90 (hexidecimal) 14 (decimal) ("non-printing" ASCII) (References channel 1)

Type:

PC write

Data Transferrred:

Four bytes are transferred, least significant byte first. The least significant three bytes represent the Analog-to-Digital converter register for the selected channel. This value controls the converter's precision and sampling rate.

After transmitting 4 bytes, one byte = 0 (hexidecimal) is transferred by the microcontroller to PC to indicate successful transfer.

Pneumonic (for reference only)

RESET

Command Character:

40 (hexidecimal) 64 (decimal) "@" (ASCII)

Type:

PC write

Data Transferrred:

none

Data Format:

none

After receiving the RESET command, the microcontroller processes a software reset and restarts at a point equivalent to a power-up or hardware reset. All variables are initialized, all outputs are placed in their reset state.

This instruction is a failure recovery method in the case where the controller is not responding to the PC or where the PC and microcontroller have lost properly sequenced communication.

Load Cell Signals and Calibration

This section describes the load cell signal, input pre-amplification, analog to digital conversion and the process of computing the weight. Calibration parameters are described and specified. Each channel operates with separate and independent calibration parameters. The following discussion describes a single channel. Operations of the other three channels is identical.

Weight is computed from the load cell signal according to the equation:

Weight = Scale * K * (Data - Zero) were Zero = Tare + Ref

Microcontroller math is based on floating point data types with 24 bit mantissa and 7bit exponent. This is consistent with C++ math implemented on PC. The translation from raw data to weight by the 754P-LC4's microcontroller is expected to achieve the identical result as if the raw data was transferred to the PC and the calculation done there, as long as the calibration constants are the same.

The choice of where to do the math depends only on the application. 754P-LC4's data transfer capabilities allow either raw data or scaled weight or both to be transferred to PC.

Note however, setpoints are compared to calculated weight, not raw data.

Raw Data - to - Computed Weight

Step 1. "Data" is the 24 bit result of the analog-to-digital conversion of the load cell signal. It is in the range of {00 00 00 (hexidecimal) to ff ff ff (hexidecimal). After each conversion, this variable is sign extended to 32 bits, making it the equivalent of a long integer. For all conceivable results, "data" is positive, so sign extension appends a leading zero byte to the variable. This is done for convenience in math and has no effect on precison.

Step 2. "Data" is processed by a digital filter based on the filter setup parameter stored in on-board eeprom. The resulting DATA is a moving average of "Filter" samples, where "Filter" can be 1, 2, 4 or 8. Filtering beyond 8 samples is best done in the PC.

$$\text{DATA} = \frac{1}{\text{Filter}} \sum_{n=1}^{\text{Filter}} \text{data}(n)$$

Step 3. (Data - Zero) is computed. Zero is the sum of a Reference constant and an operational Tare.

The Reference constant is loosely associated with the hardware of the 754P-LC4 and is expected to be approximately 00 80 00 00 (hexidecimal), typically the mid-point of the five volt load cell excitation). This reflects the offset built into hardware. The offset serves to keep the analog circuitry operating in its linear region and allows both positive and negative forces to be measured.

"Tare" is a variable which allows a software or operational zeroing of the measurement. When the system is tared, the value of the variable "Tare" is computed so that (Data - Zero) = 0.

The combination of the Reference constant and Tare provide flexibility to tare at any time via PC command without recalibrating.

Step 4. (Data - Zero) is converted to floating point. The result retains the 24 bit precision of the measurement.

Step 5. (Data - Zero) is multiplied by K and the result mulitplied by Scale.

The factor "Scale" is loaded at the time of manufacture and is used to calibrate the 754P-LC4 with a load cell specification. The factor "Scale" may be changed by field calibration.

The factor "K" is a calibration constant which converts the measurement into engineering units. It may be changed by field calibration as well.

The result of the multiplication is the floating point variable "Weight".

This 5-step process is repeated with each analog-to-digital conversion, with "DATA" and "Weight" variables updated.

Calculating Calibration Constants:

To compute the calibration constants, take measurements with two weights (one may be zero)

Then: K = (Weight2 - Weight1) / (Data2 - Data1)

Scale = 1 / K

If Data1 is zero:

Tare = Data1 - Ref

Tare can be updated at any time, calculated as:

Tare = Data - Ref

The calibration procedure calculation be done in the PC, extracting filtered raw data, computing and down-loading the calibration constants.

Note The instruction SETCAL loads all four calibration constants for the channel specified. All four must be transmitted even if only one is changed.

Project Lancelot

754P-LC4 Demonstration/Test System

The Lancelot project is a Borland C++ test system designed to demonstrate the 754P-LC4 Load Cell Controller. It allows the user to exercise all functions of the load cell controller from a suitably equipped DOS-based PC with keyboard/display/mouse.

The Lancelot system consists of the following files:

Source Files

IO815.CPP
TEST815.CPP
UTIL815.CPP
VIDEO815.CPP
WEIGH815.CPP
SET815.CPP
CAL815.CPP
STAT815.CPP
CONFIG815.CPP
MODE815.CPP

Header Files

UTIL815.H

Executable Files

LANCELOT.EXE
LANCEKEY.EXE

Borland C++ Project Files

LANCELOT.PRJ

Installation and Operation

As shipped, Lancelot.exe assumes that the load cell controller is installed at bus address 220, which is the factory default. If the controller address has been changed, read commands will terminate in timeout errors with no data exchanged. The configuration section of the Lancelot System, (main menu, item #6) will allow the load cell controller an dLancelot to synchronize on a bus address of the user's choice.

If multiple load cell controllers are to be installed, insure that no two are at the same bus address. Install one and move it to an address other than 220 before installing the next.

If bus address 220 is not available or not desirable, the Lancelot system must be rebuilt to change the default address. Edit the header file to change the value of the Macro BUS_ADDRESS to the desired default address.

Installation

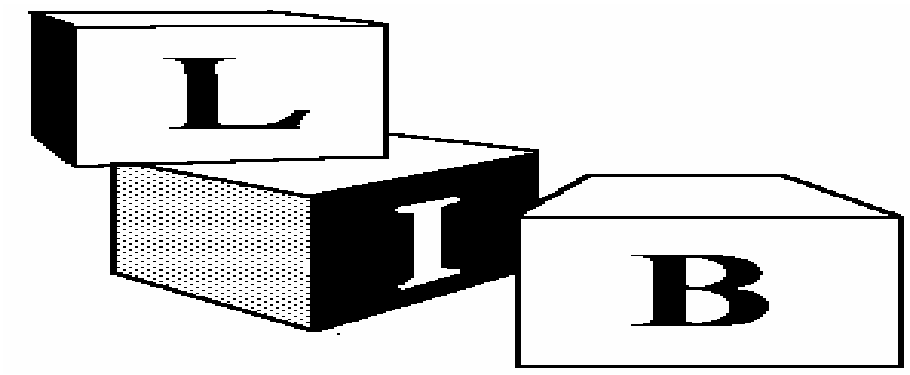
Copy the system files to an active directory and start the system with the command LANCELOT.

Use a mouse or keyboard to select menu options from the display. Follow direction on the screen.

Calibration: As shipped from Scanning Devices, each channel of the load cell controller is calibrated for a 100 pound, 2 mV/V load cell. Use the calibration menu to re-calibrate the controller to the characteristics of the load cell in use.

After calibration, weighing can begin. Lancelot provides several weighing modes which will demonstrate the capabilities of the controller and the PC software. Comparing results of different digital filter settings will suggest the signal processing techniques required in your application.

u700 PRODUCTS
documentation



sd815.lib

C routine library for 754P-LC4 PC/104-compatible 4-channel load cell controller

sd815.lib consists of 19 functions which perform data exchange between the 754P-LC4 Load Cell Controller and a PC. The library is distributed in source form to encourage integration of these libraries in the customer's application.

The library is distributed as: **UTIL815.CPP** and **UTIL815.H**

The functions are written for Borland C++ Version 3.0. However, they are coded to be readable by a knowledgeable programmer. The intent is to describe and specify the operation of the Load Cell Controller to allow the programmer to build his own application using an alternate programming language.

Functions are documented in the following pages.

SD815_GetParameters()

Usage:

#include <sd815.h>

int SD815_GetParameters(int *Bus_Address*, struct ControlSettings **Setpoints*);

Description:

The SD815_GetParameters() function is used to determine the values of the control setpoints that are currently stored in the 754P-LC4's EEPROM. The ControlSettings structure, which is defined in sd815.h, has the following form:

```
struct ControlSettings{
   float PSet1;
   float PSet2;
   float PSet3;
   float PSet4;
   };
```

The four setpoints, PSet1 through PSet4, are used by the 754P-LC4 in its measurement comparisons to determine the states of its four outputs.

After a call to this function, the members of the *Setpoints* argument will have the current values that are stored in the 754P-LC4.

Return Value:

If the function executes properly it will return the value SD815_OK, which is defined in sd815.h. (The symbolic constant SD815_OK has the value 0.)

If the function **does not** execute properly it will return one of the non-zero values shown below, which are defined in sd815.h. The members of the *Setpoints* argument **will not** have valid data (ie. the data will be unchanged).

Return values indicating errors:

SD815_IBF_TIMEOUT -- This indicates that the 754P-LC4 has not processed the last command byte or data byte, and therefore cannot accept a new byte.

SD815_OBF_TIMEOUT -- This indicates that the 754P-LC4 has not placed a data byte into its output register within the timeout period.

SD815_ERROR -- This indicates an invalid data transfer.

Example:

```
#include <sd815.h>
#include <stdio.h>

main()
{
   int ReturnValue;
   struct ControlSettings Point;
   int Bus_Address=0x220;

/* Read the setpoint values currently stored in the 754P-LC4. */
   ReturnValue=SD815_GetParameters(Bus_Address, &Point);

   if(ReturnValue==SD815_OK)
   {
      printf("The present control settings are:\n");
      printf("Setpoint #1=%f\n",Point.PSet1);
      printf("Setpoint #2=%f\n",Point.PSet2);
      printf("Setpoint #3=%f\n",Point.PSet3);
      printf("Setpoint #4=%f\n",Point.PSet4);
   }
   else
      printf("The function returned the error %d.\n",ReturnValue);
}
```

See Also:

SD815_SetParameters()

SD815_SetParameters()

Usage:

#include <sd815.h>

int SD815_SetParameters(int *Bus_Address*, struct ControlSettings **Setpoints*);

Description:

The SD815_SetParameters() function is used to load the 754P-LC4's EEPROM with values for its control setpoints. The ControlSettings structure, which is defined in sd815.h, has the following form:

```
struct ControlSettings{
  float PSet1;
  float PSet2;
  float PSet3;
  float PSet4;
  };
```

The four setpoints, PSet1 through PSet4, are used by the 754P-LC4 in its measurement comparisons to determine the states of its four outputs.

Before calling this function, the members of *Setpoints* must be assigned the appropriate values. When the function is called, the EEPROM on the 754P-LC4 will be loaded with the values specified by the *Setpoints* argument.

Return Value:

If the function executes properly it will return the value SD815_OK, which is defined in sd815.h. (The symbolic constant SD815_OK has the value 0.)

If the function **does not** execute properly it will return one of the non-zero values shown below, which are defined in sd815.h. None, some, or all of the values specified in the *Setpoints* argument may have been loaded into the 754P-LC4's memory; if an error occurs, you may wish to call SD815_GetParameters() to check the values.

Return values indicating errors:

SD815_IBF_TIMEOUT -- This indicates that the 754P-LC4 has not processed the last command byte or data byte, and therefore cannot accept a new byte.

SD815_OBF_TIMEOUT -- This indicates that the 754P-LC4 has not placed a data byte into its output register within the timeout period.

SD815_ERROR -- This indicates an invalid data transfer.

Example:

```
#include <sd815.h>
#include <stdio.h>

main()
{
   int ReturnValue;
   struct ControlSettings Point;
   int Bus_Address=0x220;
/* Set the structure members to the desired values. */
   Point.PSet1=0.1;
   Point.PSet2=1.2;
   Point.PSet3=2.5e1;
   Point.PSet4=30;

/* Load the values into the 754P-LC4's EEPROM. */
   ReturnValue=SD815_SetParameters(Bus_Address,&Point);

   if(ReturnValue==SD815_OK)
      printf("The new values have been downloaded successfully.\n");
   else
      printf("The function returned the error %d.\n",ReturnValue);
}
```

See Also:

SD815_GetParameters()

SD815_GetCalibration()

Usage:

#include <sd815.h>

int SD815_GetCalibration(int *Bus_Address*, struct CalibrationSettings **CalData,* int *iChannel*);

Description:

The SD815_GetCalibration() function is used to determine the values of the calibration parameters for the specified channel that are currently stored in the 754P-LC4's EEPROM. The CalibrationSettings structure, which is defined in sd815.h, has the following form:

```
struct CalibrationSettings{
   float Scale;
   float K;
   long int Tare;
   long int Ref;
   };
```

The four members of the *CalData* argument are used by the 754P-LC4 in its calculation of the measured weight.

After a call to this function, the members of the *CalData* argument will have the current values that are stored in the 754P-LC4.

Notes:

1. This function receives 15 bytes from the 754P-LC4. The long int Ref is padded with zero as its most significant byte, reflecting the 24 bit limiation of the data measurement.

Return Value:

If the function executes properly it will return the value SD815_OK, which is defined in sd815.h. (The symbolic constant SD815_OK has the value 0.)

If the function **does not** execute properly it will return one of the non-zero values shown below, which are defined in sd815.h. The members of the *CalData* argument **will not** have valid data (ie. the data will be unchanged).

Return values indicating errors:

SD815_IBF_TIMEOUT -- This indicates that the 754P-LC4 has not processed the last command byte or data byte, and therefore cannot accept a new byte.

SD815_OBF_TIMEOUT -- This indicates that the 754P-LC4 has not placed a data byte into its output register within the timeout period.

SD815_ERROR -- This indicates an invalid data transfer.

Example:

```
#include <sd815.h>
#include <stdio.h>

main()
{
   int ReturnValue;
   struct CalibrationSettings Values;
   int iChannel=0; // pointer to channel 1
   int Bus_Address=0x220;
/* Read the calibration values currently stored in the 754P-LC4. */
   ReturnValue=SD815_GetCalibration(Bus_Address,&Values,iChannel);

   if(ReturnValue==SD815_OK)
   {
      printf("The present calibration settings are:\n");
      printf("Setpoint #1=%f\n",Values.Scale);
      printf("Setpoint #2=%f\n",Values.K);
      printf("Setpoint #3=%u\n",Values.Tare);
      printf("Setpoint #4=%u\n",Values.Ref);
   }
   else
      printf("The function returned the error %d.\n",ReturnValue);
}
```

See Also:

SD815_SetCalibration()

SD815_SetCalibration()

Usage:

#include <sd815.h>

int SD815_SetCalibration(int *Bus_Address*, struct CalibrationSettings **CalData,* int *iChannel*);

Description:

The SD815_SetCalibration() function is used to load the 754P-LC4's EEPROM with values for its calibration parameters for the specified channel. The CalibrationSettings structure, which is defined in sd815.h, has the following form:

```
struct CalibrationSettings{
  float Scale;
  float K;
  long int Tare;
  long int Ref;
  };
```

The four members of the *CalData* argument are used by the 754P-LC4 in its calculation of the measured weight.

Before calling this function, the members of *CalData* must be assigned the appropriate values. When the function is called, the EEPROM on the 754P-LC4 will be loaded with the values specified by the *CalData* argument.

Note: This function transmitts 16 bytes to the 754P-LC4. However, the most significant byte of the long int Ref is disregarded on receipt at the module and only the three low order bytes are stored. This precision is consistent with the 24 bit limitation of the data measurment.

Return Value:

If the function executes properly it will return the value SD815_OK, which is defined in sd815.h. (The symbolic constant SD815_OK has the value 0.)

If the function **does not** execute properly it will return one of the non-zero values shown below, which are defined in sd815.h. None, some, or all of the values specified in the *CalData* argument may have been loaded into the 754P-LC4's memory; if an error occurs, you may wish to call SD815_GetCalibration() to check the values.

Return values indicating errors:

SD815_IBF_TIMEOUT -- This indicates that the 754P-LC4 has not processed the last command byte or data byte, and therefore cannot accept a new byte.

SD815_OBF_TIMEOUT -- This indicates that the 754P-LC4 has not placed a data byte into its output register within the timeout period.

SD815_ERROR -- This indicates an invalid data transfer.

Example:

```
#include <sd815.h>
#include <stdio.h>

main()
{
   int ReturnValue;
   struct CalibrationSettings Values;
   int Bus_Address=0x220;
   int iChannel=0;  //pointer to channel 1

/* Set the structure members to the desired values. */
   Values.Scale=2.5e1;
   Values.K=1.2;
   Values.Tare=0.2;
   Values.Ref=30;

/* Load the values into the 754P-LC4's EEPROM. */
   ReturnValue=SD815_SetCalibration(Bus_Adr,&Values,iChannel);

   if(ReturnValue==SD815_OK)
      printf("The new values have been downloaded successfully.\n");
   else
      printf("The function returned the error %d.\n",ReturnValue);
}
```

See Also:

SD815_GetCalibration()

SD815_GetCStatus()

Usage:

#include <sd815.h>

int SD815_GetCStatus(int *Bus_Address*, unsigned int **Status*);

Description:

The SD815_GetCStatus() function is used to determine the present channel status of the 754P-LC4.

After a call to this function, the value of *Status* will be equal to one of the symbolic constants shown below, which are defined in sd815.h.

LSB = *Status.0* = 0 for channel 1 not active, 1 for channel 1 active
Status.1 = 0 for channel 2 not active, 1 for channel 2 active
*Status.*2 = 0 for channel 3 not active, 1 for channel 3 active
*Status.*3 = 0 for channel 4 not active, 1 for channel 4 active

Return Value:

If the function executes properly it will return the value SD815_OK, which is defined in sd815.h. (The symbolic constant SD815_OK has the value 0.)

If the function **does not** execute properly it will return one of the non-zero values shown below, which are defined in sd815.h. The *Status* argument **may not** have valid data.

Return values indicating errors:

SD815_IBF_TIMEOUT -- This indicates that the 754P-LC4 has not processed the last command byte or data byte, and therefore cannot accept a new byte.

SD815_OBF_TIMEOUT -- This indicates that the 754P-LC4 has not placed a data byte into its output register within the timeout period.

SD815_ERROR -- This indicates an invalid data transfer.

Example:

```
#include <sd815.h>
#include <stdio.h>

main()
{
   int ReturnValue, Status;
   int Bus_Address=0x220;

/* Read the current channel status of the 754P-LC4. */
   ReturnValue=SD815_GetCStatus(Bus_Address,&Status);

   if(ReturnValue==SD815_OK)
   {
      printf("The present module channel status is: ");
      if ((*Status&SD815_CHANNEL1)==0)
            printf("Channel #1 is OFF");
      else
            printf("Channel #1 is Active");

      if ((*Status&SD815_CHANNEL2)==0)
            printf("Channel #2 is OFF");
      else
            printf("Channel #2 is Active");

      if ((*Status&SD815_CHANNEL3)==0)
            printf("Channel #3 is OFF");
      else
            printf("Channel #3 is Active");

      if ((*Status&SD815_CHANNEL4)==0)
            printf("Channel #4 is OFF");
      else
            printf("Channel #4 is Active");
   }
   else
      printf("The function returned the error %d.\n",ReturnValue);
}
```

See Also:

SD815_SetCStatus()

SD815_SetCStatus()

Usage:

#include <sd815.h>

int SD815_SetMode(int *Bus_Address*, unsigned int *Status*);

Description:

The SD815_SetCStatus() function is used to command the 754P-LC4 to activate or de-activate one or more of its four load cell channels..

Before calling the function, the value of *Status* should be set to represent the desired channel configuration. A channel is active if its corresponding bit in *Status* is set (=1) and not active if its corresponding bit in *Status* is clear (=0).

When the function is called, the 754P-LC4 will resume operation with the four load cell channesl activated or deactived consistent with the bit settings in the least significant four bits specified by the value of *Status*. (See SD815_GetCStatus().)

Return Value:

If the function executes properly it will return the value SD815_OK, which is defined in sd815.h. (The symbolic constant SD815_OK has the value 0.)

If the function **does not** execute properly it will return one of the non-zero values shown below, which are defined in sd815.h. The hardware **may not** be properly configured to the reflect the channels status specified by *Status*.

Return values indicating errors:

SD815_IBF_TIMEOUT -- This indicates that the 754P-LC4 has not processed the last command byte or data byte, and therefore cannot accept a new byte.

SD815_OBF_TIMEOUT -- This indicates that the 754P-LC4 has not placed a data byte into its output register within the timeout period.

SD815_ERROR -- This indicates an invalid data transfer.

Example:

```
#include <sd815.h>
#include <stdio.h>

main()
{
   int ReturnValue, Status;
   int Bus_Address=0x220;

/* Activate Channel 2, independent of its prior activity */
   ReturnValue=SD815_GetCStatus(Bus_Address,&Status);
   if(ReturnValue==SD815_OK)
      {
      *Status|=SD815_CHANNEL2;
      ReturnValue=SD815_SetCStatus(Bus_Address,*Status);
      }
   if(ReturnValue==SD815_OK)
      printf("Channel 2 has been activated.\n");
   else
      printf("The function returned the error %d.\n",ReturnValue);
}
```

See Also:

SD815_GetCStatus()

SD815_GetWeight()

Usage:

#include <sd815.h>

int SD815_GetWeight(int *Bus_Address*, float **Weight,* int *iChannel*);

Description:

The SD815_GetWeight() function is used to read the present scaled weight value that is measured by the 754P-LC4.

After a call to this function, the value of *Weight* will be equal to the last scaled measurement on load cell channel *iChannel* calculated by the 754P-LC4. This value is computed by the 754P-LC4 with the formula:

Weight = Scale * K * (Data - Ref - Tare)

where Data is the raw analog-to-digital conversion result (see SD815_GetData()) and Scale, K, Tare, and Ref are the hardware calibration settings for the specified channel (see SD815_GetCalibration()).

Return Value:

If the function executes properly it will return the value SD815_OK, which is defined in sd815.h. (The symbolic constant SD815_OK has the value 0.)

If the function **does not** execute properly it will return one of the non-zero values shown below, which are defined in sd815.h. The *Weight* argument **will not** have a valid value.

Return values indicating errors:

SD815_IBF_TIMEOUT -- This indicates that the 754P-LC4 has not processed the last command byte or data byte, and therefore cannot accept a new byte.

SD815_OBF_TIMEOUT -- This indicates that the 754P-LC4 has not placed a data byte into its output register within the timeout period.

SD815_ERROR -- This indicates an invalid data transfer.

Example:

```
#include <sd815.h>
#include <stdio.h>

main()
{
   int ReturnValue;
   unsigned int Data;
   float Weight;
   int iChannel=0;
   int Bus_Address-0x220;

/* Take a measurement on every keyboard hit until <tab> is pressed,
displaying both the raw analog-to-digital conversion result and the
scaled Weight value.*/
   printf("Press any key to perform a measurement.\n");
   printf("Press <tab> to quit.\n");

   while(getch()!='\t')
   {

if((ReturnValue=(SD815_GetData(Bus_Address,&Data,iChannel)))==SD815_OK)
         printf("Conversion Result = %04x  ",Data);
      else
         printf("Error %d.\n",ReturnValue);


if((ReturnValue=(SD815_GetWeight(Bus_Address,&Weight,iChannel)))==SD815
_OK)
         printf("Scaled Weight = %f\n",Weight);
      else
         printf("Error %d.\n",ReturnValue);
   }
}
```

See Also:

SD815_GetData(), SD815_GetCalibration(), SD815_SetCalibration()

SD815_GetData()

Usage:

#include <sd815.h>

int SD815_GetData(int *Bus_Address*, long **Data*, int *iChannel*);

Description:

The SD815_GetData() function is used to read the present analog-to-digital conversion value for the specified channel that is measured by the 754P-LC4.

After a call to this function, the value of *Data* will be equal to the last measurement taken by the 754P-LC4. This value is a 24-bit integer in the range [00000000_{16}, $0008ffff_{16}$].

This function transfers three bytes from the 754P-LC4. It then pads the most significant byte with zero.

Return Value:

If the function executes properly it will return the value SD815_OK, which is defined in sd815.h. (The symbolic constant SD815_OK has the value 0.)

If the function **does not** execute properly it will return one of the non-zero values shown below, which are defined in sd815.h. The *Data* argument **will not** have a valid value.

Return values indicating errors:

SD815_IBF_TIMEOUT -- This indicates that the 754P-LC4 has not processed the last command byte or data byte, and therefore cannot accept a new byte.

SD815_OBF_TIMEOUT -- This indicates that the 754P-LC4 has not placed a data byte into its output register within the timeout period.

SD815_ERROR -- This indicates an invalid data transfer.

Example:

```
#include <sd815.h>
#include <stdio.h>

main()
{
   int ReturnValue;
   longint Data;
   float Weight;
   int iChannel=0;
   int Bus_Address=0x220;

/* Take a measurement on every keyboard hit until <tab> is pressed,
displaying both the raw analog-to-digital conversion result and the
scaled Weight value.*/
   printf("Press any key to perform a measurement.\n");
   printf("Press <tab> to quit.\n");

   while(getch()!='\t')
   {

if((ReturnValue=(SD815_GetData(Bus_Address,&Data,iChannel)))==SD815_OK)
         printf("Conversion Result = %08x h ",Data);
      else
         printf("Error %d.\n",ReturnValue);


if((ReturnValue=(SD815_GetWeight(Bus_Address,&Weight,ichannel)))==SD815
_OK)
         printf("Scaled Weight = %f\n",Weight);
      else
         printf("Error %d.\n",ReturnValue);
   }
}
```

See Also:

SD815_GetWeight(), SD815_GetCalibration(), SD815_SetCalibration()

SD815_GetIOStates()

Usage:

#include <sd815.h>

int SD815_GetIOStates(int *Bus_Address*, unsigned int **IOBuffer*);

Description:

The SD815_GetIOStates() function is used to read the present states of the input/output pins of the 754P-LC4. This function reads all the input/output pins simultaneously; if you wish to access just one pin, use SD815_GetSingleIOState().

After a call to this function, each bit in the low byte of *IOStates* represents the present state of one of the input or output pins of the 754P-LC4, as shown below. The high byte of *IOStates* is undefined.

IOStates

Bit(s)	Name	"Low"(energized) State	"High" (de-energized) State
15 - 8	Undefined	X	X
7	Input4	1	0
6	Input3	1	0
5	Input2	1	0
4	Input1	1	0
3	Output4	1	0
2	Output3	1	0
1	Output2	1	0
0	Output1	1	0

Return Value:

If the function executes properly it will return the value SD815_OK, which is defined in sd815.h. (The symbolic constant SD815_OK has the value 0.)

If the function **does not** execute properly it will return one of the non-zero values shown below, which are defined in sd815.h. The *IOStates* argument **may not** have a valid value.

Return values indicating errors:

SD815_IBF_TIMEOUT -- This indicates that the 754P-LC4 has not processed the last command byte or data byte, and therefore cannot accept a new byte.

SD815_OBF_TIMEOUT -- This indicates that the 754P-LC4 has not placed a data byte into its output register within the timeout period.

SD815_ERROR -- This indicates an invalid data transfer.

Example:

```
#include <sd815.h>
#include <stdio.h>

main()
{
   int ReturnValue, Buffer;

/* Examine the input and output states to see if any inputs are
asserted.*/
   if((ReturnValue=(SD815_GetIOStates(Bus_Address,&Buffer)))==SD815_OK)
   {
      if((Buffer&0x00f0)==0)
         printf("None of the inputs have been asserted.\n");
      else
         printf("At least one input has been pulled low.\n");
   }
   else
      printf("Error %d.\n",ReturnValue);
}
```

See Also:

SD815_SetIOStates()

SD815_SetIOStates()

Usage:

#include <sd815.h>

int SD815_SetIOStates(int *Bus_Address*, unsigned int *IOBuffer*);

Description:

The SD815_SetIOStates() function is used to set the states of the four output pins of the 754P-LC4. This function modifies all the output pins simultaneously; if you wish to access just one pin, use SD815_SetSingleIOState().

Before calling this function, the four least significant bits of *IOStates* must be set to the appropriate value. The four outputs of the 754P-LC4 will be driven according to the bit values specified by *IOStates*, as shown below. Only the low nibble of IOStates must hold significant data; the high byte and the high nibble of the low byte of *IOStates* is undefined.

IOStates

Bit(s)	Name	"Low"(energized) State	"High" (de-energized) State
15 - 4	Undefined	X	X
3	Output4	0	1
2	Output3	0	1
1	Output2	0	1
0	Output1	0	1

Return Value:

If the function executes properly it will return the value SD815_OK, which is defined in sd815.h. (The symbolic constant SD815_OK has the value 0.)

If the function **does not** execute properly it will return one of the non-zero values shown below, which are defined in sd815.h. The outputs of the 754P-LC4 **may not** have been driven to the states specified in the *IOStates* argument.

Return values indicating errors:

SD815_IBF_TIMEOUT -- This indicates that the 754P-LC4 has not processed the last command byte or data byte, and therefore cannot accept a new byte.

SD815_OBF_TIMEOUT -- This indicates that the 754P-LC4 has not placed a data byte into its output register within the timeout period.

SD815_ERROR -- This indicates an invalid data transfer.

Example:

```
#include <sd815.h>
#include <stdio.h>

main()
{
   int ReturnValue, Buffer;
   int Bus_Address=0x220;


/* Set the bit values in Buffer to turn on (energize) all the
outputs.*/
   Buffer=0x0000;

   if((ReturnValue=(SD815_SetIOStates(Bus_Address,Buffer)))==SD815_OK)
      printf("All the outputs have been energized.\n");
   else
      printf("Error %d.\n",ReturnValue);
}
```

See Also:

SD815_GetIOStates()

SD815_GetFilter()

Usage:

#include <sd815.h>

int SD815_GetFilter(int *Bus_Address*, unsigned integer **pFilter,* int *iChannel*);

Description:

The SD815_GetFilter() function is used to determine the value of the digital filter parameter that is currently stored in the 754P-LC4's EEPROM for the specified channel. The Filter variable, which is defined in sd815.h, has the following form:

The 754P-LC4 passes its load cell measurement through a digital filter to provide a moving average of the raw data.

As each measurement is completed, the new raw data sample is added to a data array, replacing the oldest raw data sample. The average of the array is computed and becomes the new filtered data.

Four digital filter configurations are available and selected by the value of the filter parameter: The four configurations vary the number of samples retained and included in the moving average.

Filter = 1

No Filtering, raw data is immediately available; prior measurments have no effect on the current measurement.

Filter = 2

Moving average of two data samples. Current and most recent data samples have equal weight in computation of the current measurment.

Filter =4

Moving average of four data samples. Four samples have equal weight in computation of the current measurement.

Filter = 8

Moving average of eight data samples. The eight samples have equal weight in computation of the current measurement. This value of Filter provides the most history or memory of the past data signal.

After a call to this function, the *Filter* argument will have the current values that are stored in the 754P-LC4.

Return Value:

If the function executes properly it will return the value SD815_OK, which is defined in sd815.h. (The symbolic constant SD815_OK has the value 0.)

If the function **does not** execute properly it will return one of the non-zero values shown below, which are defined in sd815.h. The *Filter* argument **will not** have valid data (ie. the data will be unchanged).

Return values indicating errors:

SD815_IBF_TIMEOUT -- This indicates that the 754P-LC4 has not processed the last command byte or data byte, and therefore cannot accept a new byte.

SD815_OBF_TIMEOUT -- This indicates that the 754P-LC4 has not placed a data byte into its output register within the timeout period.

SD815_ERROR -- This indicates an invalid data transfer.

Example:

```
#include <sd815.h>
#include <stdio.h>

main()
{
   int ReturnValue;
   int Filter;
   int Bus_Address=0x220;
   int iChannel=0;
/* Read the Filter value currently stored in the 754P-LC4. */
   ReturnValue=SD815_GetFilter(Bus_Address,&iFilter, iChannel);

   if(ReturnValue==SD815_OK)
   {
      printf("The present filter is:\n");
      printf("Filter #1=%d\n",iFilter);
   }
   else
      printf("The function returned the error %d.\n",ReturnValue);
}
```

See Also:

SD815_GetParameters()

SD815_SetFilter()

Usage:

#include <sd815.h>

int SD815_SetFilter(int *Bus_Address*, unsigned int *Filter,* int *iChannel*);

Description:

The SD815_SetFilter() function is used to load the 754P-LC4's EEPROM with values for its filter variable used to define the function of the unit's digital filter.

The 754P-LC4 passes its load cell measurement through a digital filter to provide a moving average of the raw data.

As each measurement is completed, the new raw data sample is added to a data array, replacing the oldest raw data sample. The average of the array is computed and becomes the new filtered data.

Four digital filter configurations are available and selected by the value of the filter parameter: The four configurations vary the number of samples retained and included in the moving average.

Filter = 1

No Filtering, raw data is immediately available; prior measurments have no effect on the current measurement.

Filter = 2

Moving average of two data samples. Current and most recent data samples have equal weight in computation of the current measurment.

Filter =4

Moving average of four data samples. Four samples have equal weight in computation of the current measurement.

Filter = 8

Moving average of eight data samples. The eight samples have equal weight in computation of the current measurement. This value of Filter provides the most history or memory of the past data signal.

Before calling this function, the integer variable *Filter* must be assigned the appropriate value. Only values 1, 2, 4, or 8 will be accepted by the 754P-LC4 module. If an value other than one of these four is passed to the module, the EEPROM will be loaded with the value 1 (default, no filtering).

When the function is called, the EEPROM on the 754P-LC4 will be loaded with the values specified by the *Filter* argument.

Return Value:

If the function executes properly it will return the value SD815_OK, which is defined in sd815.h. (The symbolic constant SD815_OK has the value 0.)

If the function **does not** execute properly it will return one of the non-zero values shown below, which are defined in sd815.h. The values specified in the *Filter* variable mayor may not have been loaded into the 754P-LC4's memory; if an error occurs, you may wish to call SD815_GetFilter() to check the values.

Return values indicating errors:

SD815_IBF_TIMEOUT -- This indicates that the 754P-LC4 has not processed the last command byte or data byte, and therefore cannot accept a new byte.

SD815_OBF_TIMEOUT -- This indicates that the 754P-LC4 has not placed a data byte into its output register within the timeout period.

SD815_ERROR -- This indicates an invalid data transfer.

Example:

```
#include <sd815.h>
#include <stdio.h>

main()
{
   int ReturnValue;
   int Filter;
   int Bus_Address=0x220;
   int iChannel=0;

/* Set the variable to the desired value. */
   Filter=1;

/* Load the values into the 754P-LC4's EEPROM. */
   ReturnValue=SD815_SetFilter(Bus_Address,Filter,iChannel);

   if(ReturnValue==SD815_OK)
      printf("The new value has been downloaded successfully.\n");
   else
      printf("The function returned the error %d.\n",ReturnValue);
}
```

See Also:

SD815_GetFilter()

SD815_GetMode()

Usage:

#include <SD815.h>

int SD815_GetMode(int Bus_Address, unsigned int **Mode*);

Description:

The *Mode* variable controls the comparison of measured weight with preset value and the setting/clearing of a digital output. Each of the least significant four bits of the *Mode* variable corresponds to a channel, a preset variable and a digital output.

The SD815_GetMode() function is used to read the current value of the *Mode* variable and to determine the present operating mode of each channel of the 754P-LC4.

If the bit corresponding to the specified channel in the *Mode* variable is set (1), then on each measurement made on that channel, the weight is compared to the preset and the corresponding digital output set or cleared depending on the result of the comparison. If the measurement result is greater than the preset value, the digital output is energized; if the measurement result is less than or equal to the preset value, the digital output is de-energized.

If the bit in the *Mode* variable is clear (0), then the comparison is by-passed and the digital output is not changed.

The *Mode* variable is similar to the *Status* variable which is used to activate/deactivate each channel.

The *Status* variable takes priority over the *Mode* variable. If the *Status* variable indicates that a specific channel is not active (*Status* bit cleared (0) for the specified channel), then no measurement is made on that channel, the *Mode* variable is not tested, the comparison is not made and the digital output is not changed.

After a call to this function, the least significant 4 bits of *Mode* will be set/cleared to reflect the operations of the module as described below:

LSB = *Mode.0* =	0	for channel 1 has no effect on output 1
	1	for channel 1 output 1 depends on measurement 1 and preset 1
Mode.1 =	0	for channel 2 has no effect on output 2
	1	for channel 2 output 2 depends on measurement 2 and preset 2
*Mode.*2 =	0	for channel 3 has no effect on output 3
	1	for channel 3 output 3 depends on measurement 3 and preset 3
*Mode.*3 =	0	for channel 4 has no effect on output 4
	1	for channel 4 output 4 depends on measurement 4 and preset 4

Return Value:

If the function executes properly it will return the value SD815_OK, which is defined in SD815.h. (The symbolic constant SD815_OK has the value 0.)

If the function **does not** execute properly it will return one of the non-zero values shown below, which are defined in SD815.h. The *Mode* argument **may not** have valid data.

Return values indicating errors:

SD815_IBF_TIMEOUT -- This indicates that the 754P-LC4 has not processed the last command byte or data byte, and therefore cannot accept a new byte.

SD815_OBF_TIMEOUT -- This indicates that the 754P-LC4 has not placed a data byte into its output register within the timeout period.

SD815_ERROR -- This indicates an invalid data transfer.

Example:

```
#include <SD815.h>
#include <stdio.h>

main()
{
   int ReturnValue, Mode;
   int Bus_Address=0x220;

/* Read the current operating mode of the 754P-LC4. */
   ReturnValue=SD815_GetMode(Bus_Address,&Mode);

   if(ReturnValue==SD815_OK)
   {
      printf("The present operating mode is: ");
   if(ReturnValue==SD815_OK)
   {
      printf("The present module channel status is: ");
      if ((*Mode&SD815_CHANNEL1)==0)
            printf("Channel #1 Output 1 is independent of measurment");
      else
            printf("Channel #1 is Using Preset 1 to control Output 1");

      if ((*Status&SD815_CHANNEL2)==0)
        printf("Channel #2 Output 2 is independent of measurement");
      else
        printf("Channel #2 is Using Preset 2 to control Output 2");

      if ((*Status&SD815_CHANNEL3)==0)
        printf("Channel #3 Output 3 is independent of measurement");
      else
            printf("Channel #3 is using Preset 3 to control Output 3");

      if ((*Status&SD815_CHANNEL4)==0)
        printf("Channel #4 Output 4 is independent of measurement");
      else
            printf("Channel #4 is using Preset 4 to control Output 4");
   }
   else
      printf("The function returned the error %d.\n",ReturnValue);
   }
   else
      printf("The function returned the error %d.\n",ReturnValue);
}
```

See Also:

SD815_GetCStatus, SD815_SetCstatus, SD815_SetMode()

SD815_SetMode()

Usage:

#include <SD815.h>

int SD815_SetMode(int Bus_Address, unsigned int *Mode*);

Description:

The *Mode* variable controls the comparison of measured weight with preset value and the setting/clearing of a digital output. Each of the least significant four bits of the *Mode* variable corresponds to a channel, a preset variable and a digital output.

The SD815_SetMode() function is used to command the 754P-LC4 to compare measured weight with preset values and set or clear digital outputs depending on the result of the comparision.

Before calling the function, the value of *Mode* should be set to indicate which channels should control digital outputs by comparing measured weight to setpoints.

The bit pattern in the *Mode* variable corresponds the channels as described below:

```
LSB = Mode.0 = 0 for channel 1 has no effect on output 1
               1 for channel 1 output 1 depends on measurement 1 and preset 1
      Mode.1 = 0 for channel 2 has no effect on output 2
               1 for channel 2 output 2 depends on measurement 2 and preset 2
      Mode.2 = 0 for channel 3 has no effect on output 3
               1 for channel 3 output 3 depends on measurement 3 and preset 3
      Mode.3 = 0 for channel 4 has no effect on output 4
               1 for channel 4 output 4 depends on measurement 4 and preset 4
```

The other 4 bits in the *Mode* variable are not used.

When the function is called, the 754P-LC4 will resume operation in the operating mode that was specified by the value of *Mode.*

The *Status* variable takes priority over the *Mode* variable. If the *Status* variable indicates that a specific channel is not active (*Status* bit cleared (0) for the specified channel), then no measurement is made on that channel, the *Mode* variable is not tested, the comparison is not made and the digital output is not changed.

If the *Mode* variable was changed from 0 to 1 by a call to SD815_SetMode(), the digital output will assume its correct state after the next measurement which produces valid data (assuming that the *Status* variable has activated this channel). This may require filling the on-board digital filter if filtering was selected on the specified channel.

If the *Mode* variable was changed from 1 to 0 by a call to SD815_SetMode(), the digital output will remain in the state set by the last comparison. To avoid leaving a digital output in an unknown or undesirable state, it is good practice to follow an SD815_SetMode() command with a command to set the digital output to a known state.

Return Value:

If the function executes properly it will return the value SD815_OK, which is defined in SD815.h. (The symbolic constant SD815_OK has the value 0.)

If the function **does not** execute properly it will return one of the non-zero values shown below, which are defined in SD815.h. The hardware **may not** be properly configured to the operating mode specified by *Mode*.

Return values indicating errors:

SD815_IBF_TIMEOUT -- This indicates that the 754P-LC4 has not processed the last command byte or data byte, and therefore cannot accept a new byte.

SD815_OBF_TIMEOUT -- This indicates that the 754P-LC4 has not placed a data byte into its output register within the timeout period.

SD815_ERROR -- This indicates an invalid data transfer.

Example:

```
#include <SD815.h>
#include <stdio.h>

main()
{
   int ReturnValue, Mode;
   int Bus_Address=0x220;

/* Control Output 2 from Channel 2, independent of its prior activity
*/

   ReturnValue=SD815_GetMode(Bus_Address,&Mode);

   if(ReturnValue==SD815_OK)
      {
      *Mode|=SD815_CHANNEL2;
      ReturnValue=SD815_SetMode(Bus_Address,*Mode);
      }

   if(ReturnValue==SD815_OK)
      printf("The board is running with output 2 dependent on
                  measurement 2.\n");
   else
      printf("The function returned the error %d.\n",ReturnValue);
}
```

See Also:

SD815_GetMode(), SD815_GetCStatus(), SD815_SetCStatus()

SD815_GetID()

Usage:

#include <sd815.h>

int SD815_GetID(int *Bus_Address*, long **Module_ID*);

Description:

The SD815_GetID() function is used to read modules serial number stored in eeprom. The Module Serial number is a unique long integer assigned to the module at the time of manufacture. The eeprom stores the serial number and an additional bit to indicate that the serial number is valid.

The serial number can be used to identify the module's position in a system with regard to its bus address and other modules. The SD815_GetID() function can be a convenient system test; ie, successful completion of this function tests the module and the system bus without changing any operating parameters or variable in the system.

Return Value:

If the function executes properly it will return the value SD815_OK, which is defined in sd815.h. (The symbolic constant SD815_OK has the value 0.)

If the function **does not** execute properly it will return one of the non-zero values shown below, which are defined in sd815.h. The *Data* argument **will not** have a valid value.

Return values indicating errors:

SD815_IBF_TIMEOUT -- This indicates that the 754P-LC4 has not processed the last command byte or data byte, and therefore cannot accept a new byte.

SD815_OBF_TIMEOUT -- This indicates that the 754P-LC4 has not placed a data byte into its output register within the timeout period.

SD815_ERROR -- This indicates an invalid data transfer.

Example:

```
#include <sd815.h>
#include <stdio.h>

main()
{
   int ReturnValue;
   longint Module_ID;
   int Bus_Address=0x220;

if((ReturnValue=(SD815_GetID(Bus_Address,&Module_ID)))==SD815_OK)
         printf("Module Serial Number = %08x h ",Module_ID);
      else
         printf("Error %d.\n",ReturnValue);

}
```

See Also:

None

SD815_SetBusAddress()

Usage:

#include <sd815.h>

int SD815_SetBusAddress(int *Old_Bus_Address*, int *New_Bus_Address*);

Description:

The SD815_SetBusAddress() function is used to load the 754P-LC4's EEPROM and bus address latch with a base address value used to decode PC bus addressing.

Warning!

Before calling this function, the integer variable *New_Bus_Address* must be assigned an appropriate value. PC addressing conventions must be observed carefully. It is possible to assign a bus address which cannot be accessed by the PC, making the 754P-LC4 unreadable and effectively inoperable.

The 754P-LC4 makes no attempt to check or verify the validity of the bus address transmitted, as that depends on the configuration of the system on which the 754P-LC4 is installed..

The 754P-LC4 accepts the data, loads the address register latch and waits for commands transmitted to it at *New_Bus_Address*. Commands transmitted to *Old_Bus_Address* will immediately have no effect .

The default bus address for Lancelot Systems is 220 hexidecimal.

By convention, the least significant 2 bits must be zero. Each module requires an address space of 4 addresses.

Also by convention, the maximum value for *New_Bus_Address* is 3FC hexidecimal

Return Value:

If the function executes properly it will return the value SD815_OK, which is defined in sd815.h. (The symbolic constant SD815_OK has the value 0.)

If the function **does not** execute properly it will return one of the non-zero values shown below, which are defined in sd815.h. The values specified in the *New_Bus_Address* variable mayor may not have been loaded into the 754P-LC4's memory; if an error occurs, you may wish to call SD815_GetID() to check the location of the module.

Return values indicating errors:

SD815_IBF_TIMEOUT -- This indicates that the 754P-LC4 has not processed the last command byte or data byte, and therefore cannot accept a new byte.

SD815_OBF_TIMEOUT -- This indicates that the 754P-LC4 has not placed a data byte into its output register within the timeout period.

SD815_ERROR -- This indicates an invalid data transfer.

Example:

```
#include <sd815.h>
#include <stdio.h>

main()
{
   int ReturnValue;
   int Old_Bus_Address=0x220;
   int New_Bus_Address=0x230;


/* Load the values into the 754P-LC4's EEPROM. */
   ReturnValue=SD815_SetBusAddress(Old_Bus_Address,New_Bus_Address);

   if(ReturnValue==SD815_OK)
      printf("The new bus address has been loaded successfully.\n");
   else
      printf("The function returned the error %d.\n",ReturnValue);
}
```

See Also:

SD815_GetID()

SD815_GetDataCount()

Usage:

#include <sd815.h>

int SD815_GetDataCount(int *Bus_Address*, unsigned int **pDataCount*, int *iChannel*);

Description:

The SD815_GetDataCount () function is used to synchronize operations 754P-LC4 and the PC. A single byte datacount variable is maintained for each module channel's A/D converter. The datacount variable is incremented in the 754P-LC4 each time a new A/D conversion result becomes available on each channel. The value 255 is incremented to 0.

Warning!

The operation consists of a one byte transfer. The SD815_OK variable typically transferred at the end of an operation is ***not*** transferred, allowing all 256 possible values of DataCount to be valid.

Return Value:

The function fills in the contents of the pointer pDataCount. It always returns 0.

Example:

This example uses GetDataCount to return the next data sequence number or 0. Since 0 is a possible data sequence number, the programmer should keep track of data sequence numbers so that a return of 0 could be interpreted correctly. A typical program sequence might be: GetDataCount, compare with last read, if unequal (new data present) GetData; if equal call this function to wait for new data. Timeout is 2 seconds coded in for statement. It may be changed to suit the applicaton.

```
int SD815_WaitForNewData(int Bus_Adr, int iChannel)
{
      unsigned int StartData;
      unsigned int LastData;
      time_t tStart;

      SD815_GetDataCount(Bus_Adr,&LastData,iChannel);
      for(tStart=time(NULL); (time(NULL)-tStart)<2; )
      {
            SD815_GetDataCount(Bus_Adr,&StartData,iChannel);
            if(StartData!=LastData)
            {
                  return(StartData);
            }
      }
            return(0);
}
```

SD815_GetADRegister()

Usage:

#include <sd815.h>

int SD815_GetADRegister((int *Bus_Address*, long **pADRegister*, int *iChannel*);

Description:

The SD815_GetADRegister() function is used to determine the present setup value of the 754's Analog to Digital Converter. The setup value determines the converters precision and conversion rate..

After a call to this function, the value of *pADRegister* will be equal to one of the symbolic constants shown below, which are defined in sd815.h.

Possible *pADRegister* values:

AD7712_160HZ	0x00208080 - approximately 16 bits precision
AD7712_80HZ	0x00208100 - approximately 18 bits precision
AD7712_40HZ	0x00208200 - approximately 20 bits precision
AD7712_20HZ	0x00208400 - approximately 21 bits precision
AD7712_10HZ	0x002087a0 - approximately 22 bits precision

(Other values are possible if you set non-standard values using SD815_SetADRegister(), see SetADRegister function for details of selecting the value of the *pADRegister* variable.

Return Value:

If the function executes properly it will return the value SD815_OK, which is defined in sd815.h. (The symbolic constant SD815_OK has the value 0.)

If the function **does not** execute properly it will return one of the non-zero values shown below, which are defined in sd815.h. The *pADRegister* argument **may not** have valid data.

Return values indicating errors:

SD815_IBF_TIMEOUT -- This indicates that the 754P-LC4 has not processed the last command byte or data byte, and therefore cannot accept a new byte.

SD815_OBF_TIMEOUT -- This indicates that the 754P-LC4 has not placed a data byte into its output register within the timeout period.

SD815_ERROR -- This indicates an invalid data transfer.

Example:

```
#include <sd815.h>
#include <stdio.h>

main()
{
   int ReturnValue;
   long iADRegister;
/* Read the current A/D Register value of the 754P-LC4, channel 1. */
   ReturnValue=SD815_GetADRegister(SD815_BASE_ADDRESS, &iADRegister,
            SD815_Channel_1);

   if(ReturnValue==SD815_OK)
   {
      printf("The present register set for: ");

      switch(iADRegister)
      {
         case AD7712_160HZ:
            printf("160 Hz conversion rate.");
            break;
         case AD7712_80HZ:
            printf("80 Hz conversion rate.");
            break;
         case AD7712_40HZ:
            printf("40 Hz conversion rate.");
            break;
         case AD7712_20HZ:
            printf("20 Hz conversion rate.");
            break;
         case AD7712_10HZ :
            printf("10 Hz conversion rate.");
            break;
         default:
            printf("A unique value: %lh",iADRegister");
            break;
      }
   }
   else
      printf("The function returned the error %d.\n",ReturnValue);
}
```

See Also:

SD815_SetADRegister()

SD815_SetADRegister()

Usage:

#include <sd815.h>

int SD815_SetADRegister(long *iADRegister*);

Description:

The SD815_SetADRegister() function is used to load the 754P-LC4's EEPROM and Analog to Digital Converter with values for the Converter's internal filters. These values determine the Converter's conversion rate and the precision of its digital result.

Before calling this function, the integer variable *iADRegister* must be assigned an appropriate value.

How to determine an appropriate value:

The 754's Analog-to-Digital converter uses a Delta-Sigma conversion technique which allows a speed-precision trade-off to be selected. The speed is selected by the least significant 12 bits in the converter's 24 bit setup register. The five selections below provide a range of values. The standard factory setting is 20 Hz, which produces 22-bit conversions.

```
#define AD7712_160HZ     0x00208080 - Approximately 16 bits precison
#define AD7712_80HZ      0x00208100 - Approximately 18 bits precison
#define AD7712_40HZ      0x00208200 - Approximately 20 bits precision
#define AD7712_20HZ      0x00208400 - Approximately 21 bits precison
#define AD7712_10HZ      0x002087a0 - Approximately 22 bits precison
```

Things to note:

1. The most significant byte (2 hexidecimal digits) are dummies and represented by 0x00. They are passed to the 754 but immediately discarded.
2. The next three hexidecimal digits 0x--208--- must be 208. Don't change them. You CAN, but then the converter won't run. 208 defines the hardware implementation to the converter.
3. The last three hexidecimal digits 0x00208xxx represent the 12 bits which select the speed and precision. The minimum value is 080 and selects the maximum speed. A value less than 080 results in a divide by zero operation and causes the converter to crash. The maximum value is 7a0 and selects the maximum precision. A value greater than 7a0 produces an overflow which causes the converter to stop.
The 754 does no checking for range, but assumes that the PC knows what it is doing when it sends a value.
4. Within the range, precision and the update period (inverse of speed) are linearly related to the decimal equivalent of the 12 bit value. If you use Scanning Devices software you will be presented with five choices covering the range. IF you need a specific value, keep it between 080 and 7a0.

When the function is called, the EEPROM on the 754P-LC4 will be loaded with the values specified by the *iADRegister* argument. The Analog to Digital converter register will also be loaded. After purging filters, the converter will operate at the new rate and precision.

Return Value:

If the function executes properly it will return the value SD815_OK, which is defined in sd815.h. (The symbolic constant SD815_OK has the value 0.)

If the function **does not** execute properly it will return one of the non-zero values shown below, which are defined in sd815.h. The values specified in the *Filter* variable mayor may not have been loaded into the 754P-LC4's memory; if an error occurs, you may wish to call SD815_GetADRegister() to check the values.

Return values indicating errors:

SD815_IBF_TIMEOUT -- This indicates that the 754P-LC4 has not processed the last command byte or data byte, and therefore cannot accept a new byte.

SD815_OBF_TIMEOUT -- This indicates that the 754P-LC4 has not placed a data byte into its output register within the timeout period.

SD815_ERROR -- This indicates an invalid data transfer.

Example:

```
#include <sd815.h>
#include <stdio.h>

main()
{
   int ReturnValue;
   long iADRegister;

/* Set the variable to the desired value. */
   iADRegister=AD7712_40HZ;

/* Load the values into the 754P-LC4's EEPROM. */
   ReturnValue=SD815_SetADRegister(SD815_BASE_ADDRESS, iADRegister,
            SD815_Channel_1);

   if(ReturnValue==SD815_OK)
      printf("The new value has been downloaded successfully.\n");
   else
      printf("The function returned the error %d.\n",ReturnValue);
}
```

See Also:

SD815_GetADRegister()

Reference

Header File UTIL815.h

This file specifies the addresses, data and control characters used in transfers between the 754P-LC4 module and the PC. It makes extensive use of compiler directive "define" in order to document the sample programs.

```
/*
// c:\u800\dev\ninja\bc\util815.h
//
// Company: Scanning Devices Inc.
// Engineer: Stephen Bourque
// Date: 13 Sep 93
// Project: PARSEVAL 1.1 -- SD815 Utility Program
*/
#include <dos.h>
#include <conio.h>
#include <graphics.h>
#include <io.h>
#include <malloc.h>
#include <math.h>
#include <process.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

/*
// Hardware register locations.
*/
#define SD815_BASE_ADDRESS    0x220
#define SD815_CSR             SD815_BASE_ADDRESS+3
#define SD815_DATA            SD815_BASE_ADDRESS+0
#define CSR                                                  3

/*
// Error codes.
*/
#define SD815_OK              0x00
#define SD815_FAIL            0x01
#define SD815_IBF_TIMEOUT     0x02
#define SD815_OBF_TIMEOUT     0x03
#define SD815_ERROR           0xff

/*
// Operating modes.
// Used only with SD815
*/
#define SD815_CHECKMODE       0x00
#define SD815_MOTIONMODE      0x01
#define SD815_CONTMODE        0x02
#define SD815_FILLMODE        0x03
#define SD815_CALMODE         0x04
/*
//A/D Converter Registers
*/
```

```
#define AD7712_620HZ          0x00208020
#define AD7712_410HZ          0x00208030
#define AD7712_320HZ          0x00208040
#define AD7712_160HZ          0x00208080
#define AD7712_80HZ           0x00208100
#define AD7712_40HZ           0x00208200
#define AD7712_20HZ           0x00208400
#define AD7712_10HZ           0x00208700
#define AD7712_STD            0x00200000


/*
// Filter parameters
*/
#define     SD815_FILTER1           0x01
#define     SD815_FILTER2           0x02
#define     SD815_FILTER4           0x04
#define     SD815_FILTER8           0x08
#define     SD815_FILTER16          0x10
#define     SD815_FILTER32          0x20
#define     SD815_FILTER64          0x40
/*
// IO states and selections.
*/
#define SD815_IO_ON           0x00
#define SD815_IO_OFF          0x01
#define SD815_INPUT3          0x80
#define SD815_INPUT2          0x40
#define SD815_INPUT1          0x20
#define SD815_INPUT0          0x10
#define SD815_OUTPUT3         0x08
#define SD815_OUTPUT2         0x04
#define SD815_OUTPUT1         0x02
#define SD815_OUTPUT0         0x01
#define SD815_CHANNEL4        0x08
#define SD815_CHANNEL3        0x04
#define SD815_CHANNEL2        0x02
#define SD815_CHANNEL1        0x01


/*
// Instructions.
*/
#define SD815_RESET           0x40
#define SD815_GETID           0x44
#define SD815_SETBUSADR       0x48
#define SD815_GETIO           0x4C
#define SD815_SETIO           0x50
#define SD815_GETCSTATUS      0x54
#define SD815_SETCSTATUS      0x58
#define SD815_SETID           0x5C
#define SD815_GETPARAM        0x60
#define SD815_SETPARAM        0x64
#define SD815_GETCAL          0x68
#define SD815_SETCAL          0x6C
#define SD815_GETWEIGHT       0x70
#define SD815_GETDATA         0x74
#define SD815_GETFILTER       0x78
```

```
#define SD815_SETFILTER       0x7C
#define SD815_GETMODE         0x80
#define SD815_SETMODE         0x84
#define SD815_HALT            0x38
#define SD815_START           0x3C
#define SD815_GETDATACOUNT    0x88
#define SD815_GETADREGISTER   0x8C
#define SD815_SETADREGISTER   0x90



/*
//Instruction modifiers
//Channel Indexes/Pointers
*/
#define     SD815_Channel_1                     0x00
#define     SD815_Channel_2                     0x01
#define     SD815_Channel_3                     0x02
#define     SD815_Channel_4                     0x03

/*
// Weighing constants.
*/
#define     N           10

/*
// Colors.
*/
#define VGA_BACKGROUND        0
#define VGA_BLACK             1
#define VGA_GRAY4             2
#define VGA_GRAY3             3
#define VGA_GRAY2             4
#define VGA_GRAY1             5
#define VGA_WHITE                         7
#define VGA_NOTUSED           20
#define VGA_VIOLET            56
#define VGA_BLUE              57
#define VGA_GRAYBLUE          58
#define VGA_GRAYGREEN         59
#define VGA_DARKGREEN         60
#define VGA_DARKRED           61
#define VGA_ORANGE            62
#define VGA_YELLOW            63

/*
// Graphics constants.
*/
#define     LINELENGTH             80
#define     NUM_ITEMS_1            7
#define     NUM_ITEMS_1_1          6
#define     NUM_ITEMS_1_2          6
#define     NUM_ITEMS_1_3          7
#define     NUM_ITEMS_1_4          6
#define     NUM_ITEMS_1_4_X   3
#define     NUM_ITEMS_1_5          6
#define NUM_ITEMS_1_5_2 5
#define NUM_ITEMS_1_6   4
```

```
#define     MAIN_FONT               SMALL_FONT
#define     MAIN_SIZE               0

/*
// Dimensions.
*/
#define CHARWIDTH 8
#define CHARHEIGHT 16
#define TITLE_X1 5
#define TITLE_Y1 3
#define TITLE_WIDTH 70
#define TITLE_HEIGHT 3
#define MENU_X1 5
#define MENU_Y1 8
#define MENU_WIDTH 70
#define MENU_HEIGHT 20
#define YESNO_X1 30
#define YESNO_Y1 20
#define YESNO_WIDTH 20
#define YESNO_HEIGHT 2
#define DIALOG_X1 10
#define DIALOG_Y1 19
#define DIALOG_WIDTH 60
#define DIALOG_HEIGHT 5


/*
// Data structures.
*/
struct ControlSettings
{
      float PSet1;
      float PSet2;
      float PSet3;
      float PSet4;
};

struct CalibrationSettings
{
      float Scale;
      float K;
      unsigned long Tare;
      unsigned long Ref;
};

struct filespec_
{
      char drive[3];
      char path[50];
      char file[10];
      char ext[5];
};

//struct Bus_Address
//{
//    int Data;
//    int CSR;
//};
```

```
/*
// Function prototypes in c:\u800\dev\ninja\bc\test811.c.
*/
/* Functions not used in text mode
void SetupScreen(class Window *Main, char szText[LINELENGTH+1]);
void SetupButton(class Window *Main, class Button *Select, int iIndex,
char szText[LINELENGTH+1]);
void SetupDialog(class Window *Dialog, char szText[LINELENGTH+1]);
void SetupMini1(class Window *Main, char szText[LINELENGTH+1]);
void SetupMini2(class Window *Main, char szText[LINELENGTH+1]);
void SetupMini3(class Window *Main, char szText[LINELENGTH+1]);
void SetupReadout(class Window *Main, char szText[LINELENGTH+1]);
void SetupReadout1(class Window *Main, char szText[LINELENGTH+1]);
void SetupReadout2(class Window *Main, char szText[LINELENGTH+1]);
void SetupReadout3(class Window *Main, char szText[LINELENGTH+1]);
void SetupReadout4(class Window *Main, char szText[LINELENGTH+1]);
void SetupTitle(class Window *Main);
void FillTitle(class Window *Main);
void SetupDialogButton(class Window *Main, class Button *Select, char
szText[LINELENGTH+1]);
int ReportError(int iReturnVal);
int gscanf(char *szBuffer, int MaxNum);
*/
int About(void);
void KBFlush(void);
void WaitForKBHit(void);

/*
// Function prototypes in c:\u800\dev\ninja\bc\weigh811.c.
*/
int WeighMenu(int Bus_Address);
int WeighContinuousMenu(int Bus_Address);
int WeighSingleMenu(int Bus_Address);
int WeighFileMenu(int Bus_Address);
int WeighDataMenu(int Bus_Address, int iChannel);
int WeighScaleMenu(int Bus_Address);
int FilterData(int Bus_Address,long *iData, int iChannel);
int SelectChannel(void);
/*
// Function prototypes in c:\u800\dev\ninja\bc\set811.c.
*/
int SettingsMenu(int Bus_Address);
int SettingsSetpointMenu(int Bus_Address,int iIndex);
int SettingsViewMenu(int Bus_Address);

/*
// Function prototypes in c:\u800\dev\ninja\bc\util811.cpp
*/
int SD815_Reset(void);
int SD815_GetParameters(int Bus_Address,struct ControlSettings
*Points);
int SD815_SetParameters(int Bus_Address,struct ControlSettings
*Points);
int SD815_GetMode(int Bus_Address,unsigned int *pMode);
int SD815_SetMode(int Bus_Address,unsigned int iMode);
int SD815_SetFilter(int Bus_Address,unsigned int iFilter, int
iChannel);
int SD815_GetFilter(int Bus_Address,unsigned int *pFilter, int
iChannel);
int SD815_GetCalibration(int Bus_Address,struct CalibrationSettings
*Points, int iChannel);
int SD815_SetCalibration(int Bus_Address,struct CalibrationSettings
*Points, int iChannel);
int SD815_GetWeight(int Bus_Address,float *pWeight, int Channel);
int SD815_GetIOStates(int Bus_Address,unsigned int *pIOBuffer);
int SD815_SetIOStates(int Bus_Address,unsigned int iIOBuffer);
int SD815_GetSingleIOState(int Bus_Address,unsigned int iSelectPin,
unsigned int *pSwitch);
int SD815_SetSingleIOState(int Bus_Address,unsigned int iSelectPin,
unsigned int iSwitch);
int SD815_GetData(int Bus_Address,long *pData, int iChannel);
int WaitForIBFClear(int Bus_Adr);
int WaitForOBFSet(int Bus_Adr);
int SD815_SetCStatus(int Bus_Address,unsigned int iIOBuffer);
int SD815_GetCStatus(int Bus_Address,unsigned int *pIOBuffer);
int SD815_GetID(int Bus_Address,long *pData);
int SD815_SetID(int Bus_Address,unsigned long lData);
int SD815_SetBusAddress(int Old_Bus_Adr, int New_Bus_Adr);
int SD815_GetADRegister(int Bus_Adr, long *pADRegister, int iChannel);
int SD815_SetADRegister(int Bus_Adr, long iADRegister, int iChannel);
int SD815_GetDataCount(int Bus_Adr, unsigned int *pDataCount, int
iChannel);
int SD815_WaitForNewData(int Bus_Adr, int iChannel);

/*
// Function prototypes in \u700\dev\sd815\config815.cpp
*/

int ConfigureMenu(int BusAddress);
int FindModules(void);
int NewBusAddress(int OldAddress);
int SelectModuleID(void);
int SelectNewAddress(int OldAddress);
int SetModuleID(int BusAddress);



/*
// Function prototypes in c:\u800\dev\ninja\bc\mode811.c.
*/
int ModeMenu(int Bus_Address);
int ChannelModeMenu(int Bus_Address, int iChoose, unsigned int
*pIOBuffer);
int ChannelModeDisplay(int Bus_Address, unsigned int *pIOBuffer);


/*
// Function prototypes in c:\u800\dev\ninja\bc\io811.c.
*/
int IOMenu(int Bus_Address);
int IOOutputMenu(int Bus_Address,int iChoose, unsigned int *pIOBuffer);
int IOInputMenu(int Bus_Address,unsigned int *pIOBuffer);
/*
```

```
// Function prototypes in c:\u700\dev\sd815\bc\stat815.c.
*/
int StatusMenu(int Bus_Address);
int ChannelStatusMenu(int Bus_Address, int iChoose, unsigned int
*pIOBuffer);
int ChannelStatusDisplay(int Bus_Address, unsigned int *pIOBuffer);


/*
// Function prototypes in c:\u800\dev\ninja\bc\cal811.c.
*/
int CalibrateMenu(int Bus_Address,int iChannel);
int CalibrateStandardMenu(int Bus_Address,int iChannel);
int CalibrateSpecificationMenu(int Bus_Address,int iChannel);
int CalibrateFactoryMenu(int Bus_Address,int iChannel);
int CalibrateTareMenu(int Bus_Address,int iChannel);
int CalibrateViewMenu(int Bus_Address,int iChannel);
int FilterViewMenu(int Bus_Address,int iChannel);
int FilterXMenu(int Bus_Address,int iIndex,int iChanel);
int CalibrateFilterMenu(int Bus_Address,int iChannel);
int SelectCalibrateMenu(int Bus_Address);
int ConverterMenu(int Bus_Adr,int iChannel);
int ConverterViewMenu(int Bus_Adr,int iChannel);
int ConverterXMenu(int Bus_Adr,long iADRegister, int iChannel);


/*
// Function prototypes in c:\u800\dev\ninja\bc\video811.c.
*/
int InitializeGraphics(void);

class Window
{
protected:
      char szName[LINELENGTH+1];
      int iTextSize;
      int iColorText;
      int iColorBackground;
      int iColorOutline;
      int iColorHighlight;

public:
      int xLeft;
      int yTop;
      int xWidth;
      int yHeight;
      int xRes;
      int yRes;

public:
      void SetParameters(int xIRes, int yIRes, int xILeft, int yITop,
int xIWidth, int yIHeight, char *pName, int iITextSize);
      void SetColors(int iCText, int iCBackground, int iCOutline, int
iCHighlight);
      void Clear();
      void SetRelativeCoordinates();
      void Window::PrintLine(int iLineNumber, char
szLine[LINELENGTH+1]);
```

```
      void Window::PrintLine(char szLine[LINELENGTH+1]);
      void Window::EraseLine(int iLineNumber);
      void Window::EraseLine(void);
};


class Button:public Window
{
public:
      void Clear();
      void Press();
};


class Board:public Window
{
      int iBoard;

public:
      void Clear();
      void SetID(int iBoardID);
      void DrawEPROM(int iColor);
      void DrawEEPROM(int iColor);
      void Draw80451(int iColor);
      int DrawRAM(int iColor);
      int DrawDisplay(int iColor);
      int DrawIO(int iColor);
};

int CheckMouse(class Button *Select, int iNumOfSelects);
void DebounceClick(void);
int WaitForClick(class Button *Select);
void RedrawCursor(void);
void UndrawCursor(void);
void ClickButton(class Button *Select, int iButtonNumber);
int PaintFloor(void);
void Sword(int x, int y);
```