PC/104 Compliant Load Cell Controller 754P-LC3
Desktop (ISA) Load Cell Controller 754W-IN1

Table of Contents

Scanning Devices Inc.

PC/104 Compliant Load Cell Controller 754P-LC3
Software Libraries and Sample Programs 754P-SW3

Part Number 754P-LC3 Includes:

SD814 Module, compliant with PC/104 standards. (Previous revision was SD811)
EPROM programmed with microcode: NJ003.4

Part Number 754P-SW3 Includes:

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

Lancelot and *Loadcell* directories containing source and executable programs
.h header files for C++ sources
.cpp C++ sources
.exe executable programs
.dsk and .prj C++ project files
Basic directory containing QBasic sources (Loadcell.bas)

Lancelot is a graphic implemention requiring a mouse and hardware floating point. In many small system installations, the graphics conflict with floating point processing. Lancelot failed on attempting floating point data handling. Loadcell is the alternative.

Loadcell implements recent enhancements: Access to the sequence control DATACOUNT variable and contol of the A/D converter precision and conversion rate.

Basic was stabilized in January 1998. No further updates are planned.

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.

Utilities for 754P-LC3 are distributed as Util811.cpp in source form. Other .cpp files are sample programs in source form which use these Utilities. The sample programs are integrated into Loadcell.exe and Lancelot.exe, a sample system for demonstrating the 754P-LC3 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-LC3 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-LC3 load cell controller to aid a user in designing and implementing a system to obtain results specific to a user's situation.

The sample programs are 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 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 docutmenation for your program development tools for specifics.

Scanning Devices SD814 module

The SD814 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. **Bus Address**

The module uses shorting plugs/jumpers to allow the user to select an appropriate bus address. Address bits are set(1)/cleared(0) by installing/removing jumpers on Jumper Strip A, Jumpers J1-J8, to establish an 8-bit pattern representing the address.

		Alternate Settings	
Address Line	Corresponding Jumper	220 (SA9, SA5)	330 (SA9,8,5,4)
SA0	Always 0		
SA1	Always 0		
SA2	J1	Removed	Removed
SA3	J2	Removed	Removed
SA4	J3	Removed	Installed
SA5	J4	Installed	Installed
SA6	J5	Removed	Removed
SA7	J6	Removed	Removed
SA8	J7	Removed	Installed
SA9	J8	Installed	Installed

NOTE: *Bus Address*: factory testing and sample software configures the SD814 module at bus address 330. The bus address is configured by installing/removing (8) jumpers on jumper strip A. If the bus address is changed with jumpers, the sample software must also be changed to accommodate the new address. Edit the file UTIL811.H to make the corresponding bus address change to the sample software. This change will require building a new executable module with the modified software.

2. **IRQ selection**.

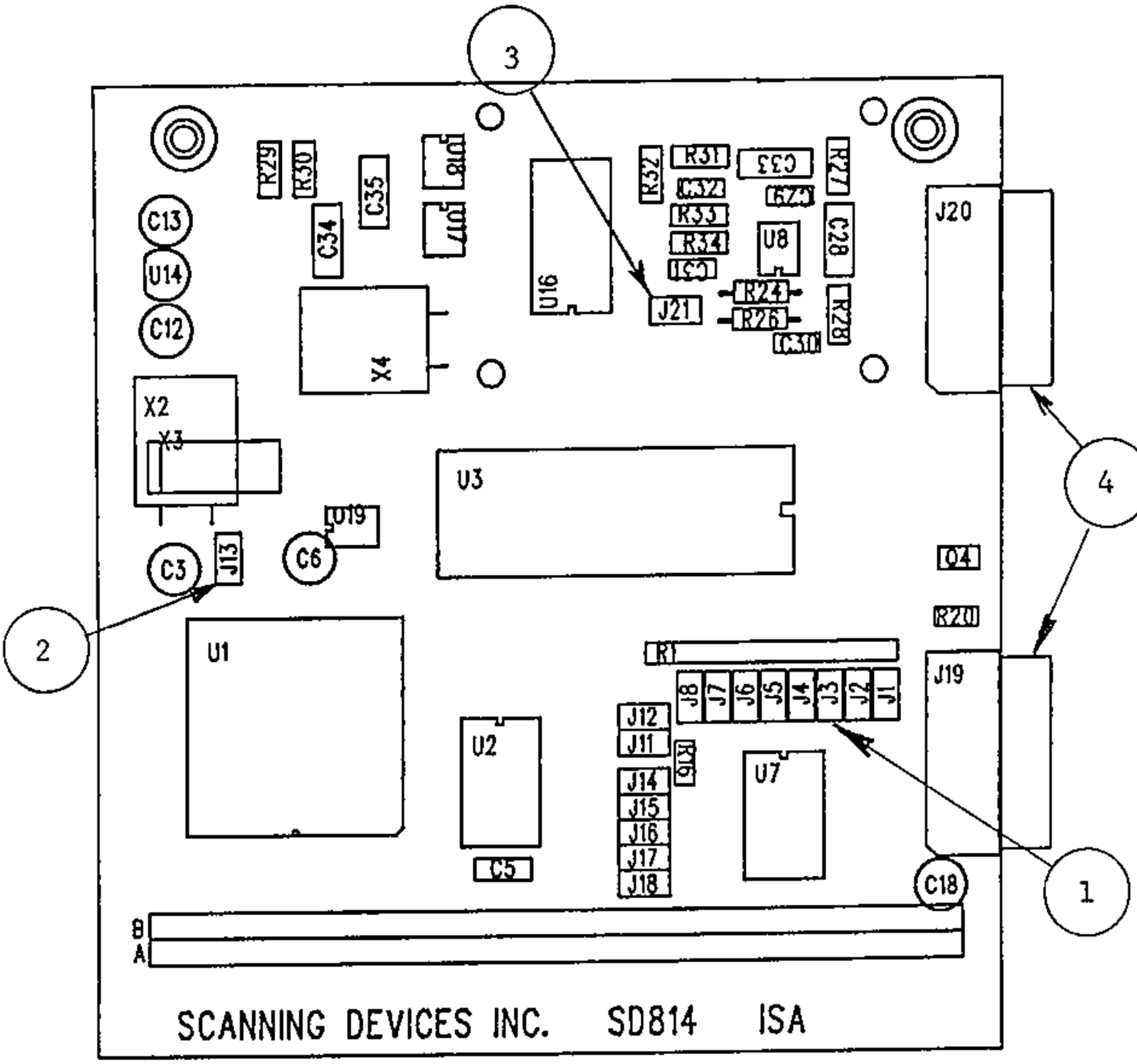
The SD814 module is equipped with hardware for selection of IRQ points (jumper strip B, J11, J12, J14, J15, J16,J17,J18). These are not implement in the 754P-LC3, reserved for future use.

3. **Microcontroller Reset**

The SD811'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 J13. Terminal 1 of jumper J13 is +5 Volts. Therefore, connecting the two terminals of jumper J13, 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-LC3 and the PC.)

An external reset may be valuable, especially during software development and testing. A normally opened push button switch wired to J13 makes a convenient reset button, comparible to the reset button on most PC's.



SD814 Module Setup

1. J1 - J8 Bus Address
2. Microcontroller Reset (J13)
3. Preamplifier Input Gain (J21)
4. Input/Output Connections
   - J20 - Load Cell
   - J19 - Digital Input/Output

4. **Preamplifier Input Gain**

The 754P-LC3 module uses an instrumentation amplifier to condition the 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 J21 to select one of two popular load cell signal ranges.

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 30 millivolt signal at full capacity to 5 volts for the analog-to-digital converter.

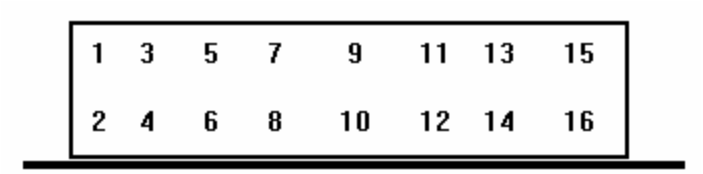
With the jumper installed, the amplfier gain is suitable for load cells with 2 mv/V signals.The instrumentation amplifer converts the load cell's 20 millivolt signal at full capacity to 5 volts for the analog-to-digital converter.

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

5. **I/O Connections**

External signals connect to SD814 via two 16-pin connectors. Connector J20 is the analog connector for the load cell. Connector J19 is the digital connector for digital inputs and outputs.

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



Connector Pin Assignments are:

Load Cell Connection

J20 - Load Cell Connector

Pins	Usage
1 and 2	Signal Ground
3 and 4	Load Cell Excitation Plus (+5 Volts DC)
5 and 6	Load Cell Sense-Plus
7 and 8	Load Cell Sense-Minus
9 and 10	Signal Ground (Load Cell Excitation Minus)
11 and 12	Load Cell Signal-Plus
13 and 14	Load Cell Signal-Minus
15 and 16	Signal Ground

Note:

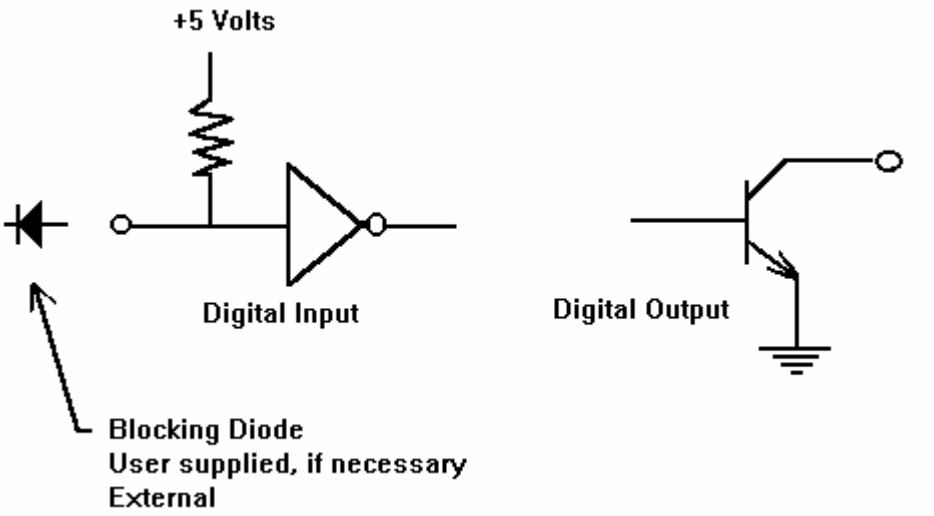
For 6 wire load cells, attach as indicated above.

For 4 wire load cells (no sense wires) two wire jumpers are required:

1. Jumper pins 3 and 4 to pins 5 and 6 (attaches Excitation Plus to Sense Plus)
2. Jumper pins 7 and 8 to pins 9 and 10 (attaches Excitation Minus to Sense Minus)

Digital Input and Output Connection

J19 - Digital Input/Output Connector



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 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	+5 Volts
3	-12 Volts
4	+12 Volts
5	Output_1
6	Output_3
7	Output_0
8	Output_2
9	Signal Ground
10	Signal Ground
11	Input_1
12	Input_0
13	Input_3
14	Input_2
15	Not Used
16	Not Used

Operation of the 754P-LC3

The 754P-LC3 is a micro-controller based peripheral controller with four general functions combined in a PC/104 compliant module

1. Low level signal pre-amplification from millivolts to volts
2. High precision analog-to-digital conversion
3. Computation and storage for calculation of engineering units and comparision to setpoints
4. Data transfer via PC bus.

This combination make the 754P-LC3 ideal for real time applications requiring the combination of bridge-style transducers and PC data processing and storage.

The 754P-LC3 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. The A/D converter generates a new data measurement at a nominal rate of 200 Hz. The configuration of the digital filter can be loaded via a software command from the PC to deliver digital data as generated by the A/D converter or to process that data using a moving average filtering technique.

The on-board microcontroller manages data generated by the A/D converter, 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 Bus Interface

The bus interface consists of a data port and a control status register. The module is addressed

$_{9}$ $_{2}$

when the bus address specified by signal SA - SA is equal to the jumper-defined address

$_{0}$

physically set on the module. Address line SA determines whether the module's data port or

0 0

control status register is addressed. Logic Zero on SA selects the data port. Logic One on SA selects the control status register.

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-LC3 to insure that data properly passes between the two.

Sample PC Program illustrating bus interface operations

The following C++ function, SD811_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 if not, returns timeout error	
Writes a command to the microcontroller requesting data, the symbolic constant SD811_GETDATA	
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 SD811_OK	After third byte, transfers a fourth byte, the symbolic constant SD811_OK
Function return value indicates that data transfer has completed normally or with error	

```
int SD811_GetData(unsigned int *pData)
{
        register int iIndex;
        unsigned int *pWord;
        char szBuffer[3];
        char *pByte;

  if(WaitForIBFClear()!=SD811_OK)
    return(SD811_IBF_TIMEOUT);

  outp(SD811_DATA,SD811_GETDATA);

        for(iIndex=0; iIndex<3; iIndex++)
  {
    if(WaitForOBFSet()!=SD811_OK)
      return(SD811_OBF_TIMEOUT);

                szBuffer[iIndex]=inp(SD811_DATA);
  }

  if(WaitForOBFSet()!=SD811_OK)
    return(SD811_OBF_TIMEOUT);

  if(inp(SD811_DATA)!=SD811_OK)
    return(SD811_ERROR);

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

        return(SD811_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(void)
{
        time_t tStart;

        if((inp(SD811_CSR)&0x01)==0)
                return(SD811_OK);

        for(tStart=time(NULL); (time(NULL)-tStart)<2; )
                if((inp(SD811_CSR)&0x01)==0)
                        return(SD811_OK);

        return(SD811_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(void)
{
        time_t tStart;
        int iData;

        iData=inp(SD811_CSR);

        if((iData&0x02)!=0)
                return(SD811_OK);

        for(tStart=time(NULL); (time(NULL)-tStart)<2; )
        {
                iData=inp(SD811_CSR);

                if((iData&0x02)!=0)
                        return(SD811_OK);
        }

        return(SD811_ERROR);
}
```

NJLink Routine

Transfers data between 754P-LC3 Microcontroller-based load cell interface and PC bus in response to PC commands

The 754P-LC3 accepts a single byte command from PC written to the module base address.

During each measurement cycle, 754P-LC3 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 selection jumpers.

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

NJLink responds by interpreting the instruction and executing a pre-programmed data transfer procedure. PC "read" commands initiate data transfer from 754P-LC3 to PC. PC "write" commands initiate data transfers from PC to 754P-LC3.

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 flag remains set for more than approximately two seconds (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 SD811_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 two seconds (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 SD811_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.

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

GETPARAM - Reads (uploads) preset variables
SETPARAM - Writes (downloads) preset variables
GETMODE - Reads the operating mode
SETMODE - Writes the operating mode
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)
GETIO - Reads status of digital inputs and digital outputs
SETIO - Sets digital outputs
GETFILTER - Reads the digital filter setup parameters
SETFILTER - Writes the digital filter setup parameters
RESET - Initiates a software reset and restart at the microcontroller
GETDATACOUNT - Reads the sequence number of the most recent A/D conversion
GETADREGISTER - Reads the current Analog to Digital setup register
SETADREGISTER - Sets the value of the Analog to Digital register

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:

70 (hexidecimal) 112 (decimal) "p" (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)

OPERATING MODE

GETMODE

Command Character:

61 (hexidecimal) 97 (decimal) "a" (ASCII)

Type:

PC read

Data Transferrred:

Mode variable

Data Format:

Unsigned short integer

One byte is transferred

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

Pneumonic (for reference only)

SETMODE

Command Character:

71 (hexidecimal) 113 (decimal) "q" (ASCII)

Type:

PC write

Data Transferrred:

Mode variable

Data Format:

Unsigned short integer

One byte is transferred.

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-LC3's operating mode.

Pneumonic (for reference only)

CALIBRATION

GETCAL

Command Character:

62 (hexidecimal) 98 (decimal) "b" (ASCII)

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-LC3 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 factor 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:

72 (hexidecimal) 114 (decimal) "r" (ASCII)

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-LC3 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:

65 (hexidecimal) 101 (decimal) "e" (ASCII)

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:

63 (hexidecimal) 99 (decimal) "c" (ASCII)

Type:

PC read

Data Transferrred:

Current value of the floating point Weight variable

Data Format:

Single four-byte floating point variable

754P-LC3 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

Pneumonic (for reference only)

I / O STATES

GETIO

Command Character:

64 (hexidecimal) 100 (decimal) "d" (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-LC3

LSB bit 1	Output 1 1 = output de-energized 0 = 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

Note: Output 1 logic is reversed compared to the other outputs.

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

Pneumonic (for reference only)

I / O STATES

SETIO

Command Character:

74 (hexidecimal) 116 (decimal) "t" (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-LC3

LSB bit 1	Output 1	1 = output de-energized 0 = 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)

Note: Output 1 logic is reversed compared to the other outputs.

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.

Pneumonic (for reference only)

DIGITAL FILTER

SETFILTER

Command Character:

76 (hexidecimal) 118 (decimal) "v" (ASCII)

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.

Pneumonic (for reference only)

DIGITAL FILTER

GETFILTER

Command Character:

66 (hexidecimal) 102 (decimal) "f" (ASCII)

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.

Pneumonic (for reference only)

RESET

RESET

Command Character:

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

Operating Mode Definition and Input Output Usage

754P-LC3 has four standard pre-programmed operating modes. (Modes 1, 3, 4, 5; Mode 2 is not available). Operating modes combine weight measurement with input and output logic. If the operating mode satisfies a user's requrement, it may allow the 754P-LC3 to perform its function independent of the PC.

The module is normally shipped with Mode 3 set in its eeprom. When power is first applied, 754P-LC3 is in operating mode 3.

If one of the pre-programmed operating modes does not satisfy the user's requirement, there are several alternatives:

1. Use Operating Mode 5 (Continuous measurement). Use the PC for input and output logic.

2. Contact Scanning Devices with your requirements. A customized operating mode may be available.

Operating Mode 1 - Check weighing

On power-up, 754P-LC3 retrieves calibaration constants and setpoints from its eeprom and begins measurement based on setpoints and the status of input 1.

The following input and output definitions are used in opeating mode 1:

Input 1 - Item to measure is in place

This may be provided by a current sinking sensor or external switch engergized (input 1 connected to signal ground)

Input 2 - Not used

Input 3 - Not used

Input 4 - Not used

Outputs are all off if no presets are exceeded.

Output 1 - Preset 1 exceeded, Preset 2 not exceeded

Output 2 - Preset 2 exceeded, Preset 3 not exceeded

Output 3 - Preset 3 exceeded, Preset 4 not exceeded

Output 4 - Preset 4 exceeded

Operating Mode 1 uses the following logic:

Step 1: Wait for Input 1 to become active (Item to measure is in place)
Step 2: Take measurement, compare to 4 setpoints, set output according to definitions.
Step 3. Wait for Input 1 to become inactive (Item to measure is removed); go to step 1.

In each step, 754P-LC3 interrogates the PC bus interface. If there are PC commands, it responds to those commands. If the commands cause the calibration constants or setpoints to be changed, they are stored in eeprom. Any measurement operation in process during a change in calibration constant or setpoint is abandoned and requires a change of state on Input 1 to resume. If there are no PC commands or if the commands do not require change in calibration constants or setpoints, operation continues.

In operating mode 1, 754P-LC3 automatically categorizes measurements into one of five categories (four individual outputs plus "all outputs off") without PC intervention. Via commands, PC can monitor the measurement and the result.

Operating Mode 2 - In-motion Checkweighing

This operating mode is no longer available.

Scanning Devices customers have had difficulty with this operating mode because of time dependencies introduced by motion.

If you have an interest in In-motion check weighing, contact Scanning Devices for advice.

Operating Mode 3 - Continuous Measurment and Comparison to Setpoints

On power-up, 754P-LC3 retrieves calibaration constants and setpoints from its eeprom and begins measurements. It measures continuously and asserts its outputs (NPN transistors conducting to ground), Output 1 through Output 4 when the measurement exceeds the corresponding setpoint.

After each measurement, 754P-LC3 interrogates the PC bus interface. If there are PC commands, it responds to those commands. If the commands cause the calibration constants or setpoints to be changed, they are stored in eeprom. Measurement continues with the new parameters.

Operating Mode 3 is designed for general purpose measurement. It allows the module to be tested in its basic form, and with the PC to implement a wide range of measurement and control functions.

Operating Mode 4 - Automatic Fill By Weight

On power-up, 754P-LC3 retrieves calibaration constants and setpoints from its eeprom and begins measurement based on setpoints and the status of input 1.

The following input and output definitions are used in opeating mode 4:

Input 1 - Container to fill in place

This may be provided by a current sinking sensor or external switch engergized (input 1 connected to signal ground)

Input 2 - Not used

Input 3 - Not used

Input 4 - Not used

Output 1 - Tare Complete

Output 2 - Preset 1 exceeded

Output 3 - Preset 2 exceeded

Output 4 - not used

Operating Mode 4 uses the following logic:

Step 1: Wait for Input 1 to become active (Container in place)
Step 2: Take measurement, assign to TARE, set Tare Complete Output (Output 1 energized)
Step 3: Take measurement, compare to Setpoint 1; when measurement exceeds Setpoint 1, set Output 2, go to step 4.
Step 4: Take measurement, compare to Setpoint 2; when measurement exceeds Setpoint 2, set Output 3, go to step 5.
Step 5. Wait for Input 1 to become inactive (Container has been removed). When Input 1 is inactive, goto step 1.

In each step, 754P-LC3 interrogates the PC bus interface. If there are PC commands, it responds to those commands. If the commands cause the calibration constants or setpoints to be changed, they are stored in eeprom. Any fill operation in process during a change in calibration constant or setpoint is abandoned and requires a change of state on Input 1 to resume. If there are no PC commands or if the commands do not require change in calibration constants or setpoints, operation continues.

This operating mode allows automatic fill operation without PC intervention. It also allows PC monitoring of the fill operation.

Operating Mode 5 - Continuous Measurement for Calibration

In this Mode, 754P-LC3 measures with currently stored calibration constants and makes the meaurement available to the PC.

Outputs are not affected by the measurments.

This operating mode allows a user-written PC program to evaluate calibration or perform calibration on the load cell.

This operating mode also allows a user-writen PC program to control the module's outputs.

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.

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 on by the 754P-LC3's microcontroller is expected to achieve identical results 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-LC3'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.

$$DATA = \frac{1}{Filter} \sum_{n=1}^{Filter} data[n]$$

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

The Reference constant is loosly associated with the hardware of the 754P-LC3 and is exptected 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-LC3 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

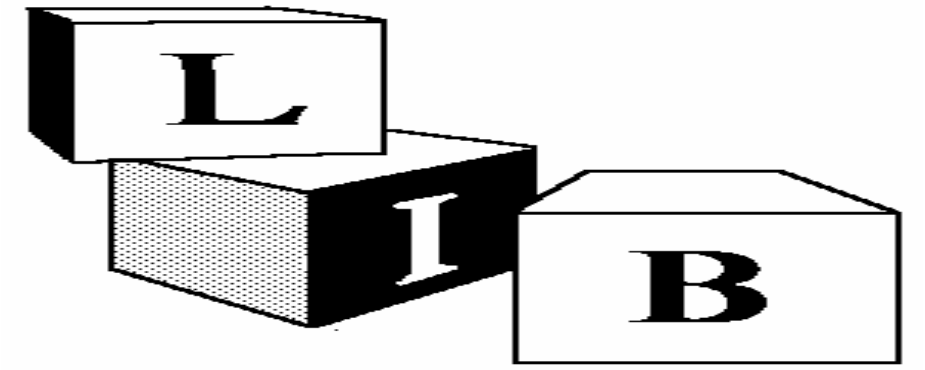
Tare can be updated at any time, calculated as:

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. All four must be transmitted even if only one is changed.

u700 PRODUCTS
documentation



sd811.lib

C routine library for 754P-LC3 PC/104-compatible load cell controller and 754I-SA1 desktop PC (ISA) load cell controller

sd811.lib consists of 12 functions which perform data exchange between the Load Cell Controller module and PC. The library is distributed in source form to encourage integration of these libraries in the customer's application.

The library is distributed as: **UTIL811.CPP** and **UTIL811.H**

The functions are written for Borland C++ Version 3.0. However, they are coded so as 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.

SD811_GetParameters()

Usage:

#include <sd811.h>

int SD811_GetParameters(struct ControlSettings **Setpoints*);

Description:

The SD811_GetParameters() function is used to determine the values of the control setpoints that are currently stored in the 754P-LC3's EEPROM. The ControlSettings structure, which is defined in sd811.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-LC3 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-LC3.

Return Value:

If the function executes properly it will return the value SD811_OK, which is defined in sd811.h. (The symbolic constant SD811_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 sd811.h. The members of the *Setpoints* argument **will not** have valid data (ie. the data will be unchanged).

Return values indicating errors:

SD811_IBF_TIMEOUT -- This indicates that the 754P-LC3 has not processed the last command byte or data byte, and therefore cannot accept a new byte.

SD811_OBF_TIMEOUT -- This indicates that the 754P-LC3 has not placed a data byte into its output register within the timeout period.

SD811_ERROR -- This indicates an invalid data transfer.

Example:

```
#include <sd811.h>
#include <stdio.h>

main()
{
   int ReturnValue;
   struct ControlSettings Point;

/* Read the setpoint values currently stored in the 754P-LC3. */
   ReturnValue=SD811_GetParameters(&Point);

   if(ReturnValue==SD811_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:

SD811_SetParameters()

SD811_SetParameters()

Usage:

#include <sd811.h>

int SD811_SetParameters(struct ControlSettings **Setpoints*);

Description:

The SD811_SetParameters() function is used to load the 754P-LC3's EEPROM with values for its control setpoints. The ControlSettings structure, which is defined in sd811.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-LC3 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-LC3 will be loaded with the values specified by the *Setpoints* argument.

Return Value:

If the function executes properly it will return the value SD811_OK, which is defined in sd811.h. (The symbolic constant SD811_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 sd811.h. None, some, or all of the values specified in the *Setpoints* argument may have been loaded into the 754P-LC3's memory; if an error occurs, you may wish to call SD811_GetParameters() to check the values.

Return values indicating errors:

SD811_IBF_TIMEOUT -- This indicates that the 754P-LC3 has not processed the last command byte or data byte, and therefore cannot accept a new byte.

SD811_OBF_TIMEOUT -- This indicates that the 754P-LC3 has not placed a data byte into its output register within the timeout period.

SD811_ERROR -- This indicates an invalid data transfer.

Example:

```
#include <sd811.h>
#include <stdio.h>

main()
{
   int ReturnValue;
   struct ControlSettings Point;

/* 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-LC3's EEPROM. */
   ReturnValue=SD811_SetParameters(&Point);

   if(ReturnValue==SD811_OK)
      printf("The new values have been downloaded successfully.\n");
   else
      printf("The function returned the error %d.\n",ReturnValue);
}
```

See Also:

SD811_GetParameters()

SD811_GetCalibration()

Usage:

#include <sd811.h>

int SD811_GetCalibration(struct CalibrationSettings *_CalData_);

Description:

The SD811_GetCalibration() function is used to determine the values of the calibration parameters that are currently stored in the 754P-LC3's EEPROM. The CalibrationSettings structure, which is defined in sd811.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-LC3 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-LC3.

Notes:

1. This function receives 15 bytes from the 754P-LC3. 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 SD811_OK, which is defined in sd811.h. (The symbolic constant SD811_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 sd811.h. The members of the *CalData* argument **will not** have valid data (ie. the data will be unchanged).

Return values indicating errors:

SD811_IBF_TIMEOUT -- This indicates that the 754P-LC3 has not processed the last command byte or data byte, and therefore cannot accept a new byte.

SD811_OBF_TIMEOUT -- This indicates that the 754P-LC3 has not placed a data byte into its output register within the timeout period.

SD811_ERROR -- This indicates an invalid data transfer.

Example:

```
#include <sd811.h>
#include <stdio.h>

main()
{
   int ReturnValue;
   struct CalibrationSettings Values;

/* Read the calibration values currently stored in the 754P-LC3. */
   ReturnValue=SD811_GetCalibration(&Values);

   if(ReturnValue==SD811_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:

SD811_SetCalibration()

SD811_SetCalibration()

Usage:

#include <sd811.h>

int SD811_SetCalibration(struct CalibrationSettings *_CalData_);

Description:

The SD811_SetCalibration() function is used to load the 754P-LC3's EEPROM with values for its calibration parameters. The CalibrationSettings structure, which is defined in sd811.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-LC3 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-LC3 will be loaded with the values specified by the *CalData* argument.

Note: This function transmitts 16 bytes to the 754P-LC3. 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 SD811_OK, which is defined in sd811.h. (The symbolic constant SD811_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 sd811.h. None, some, or all of the values specified in the *CalData* argument may have been loaded into the 754P-LC3's memory; if an error occurs, you may wish to call SD811_GetCalibration() to check the values.

Return values indicating errors:

SD811_IBF_TIMEOUT -- This indicates that the 754P-LC3 has not processed the last command byte or data byte, and therefore cannot accept a new byte.

SD811_OBF_TIMEOUT -- This indicates that the 754P-LC3 has not placed a data byte into its output register within the timeout period.

SD811_ERROR -- This indicates an invalid data transfer.

Example:

```
#include <sd811.h>
#include <stdio.h>

main()
{
   int ReturnValue;
   struct CalibrationSettings Values;

/* Set the structure members to the desired values. */
   Values.Scale=0.1;
   Values.K=1.2;
   Values.Tare=2.5e1;
   Values.Ref=30;

/* Load the values into the 754P-LC3's EEPROM. */
   ReturnValue=SD811_SetCalibration(&Values);

   if(ReturnValue==SD811_OK)
      printf("The new values have been downloaded successfully.\n");
   else
      printf("The function returned the error %d.\n",ReturnValue);
}
```

See Also:

SD811_GetCalibration()

SD811_GetMode()

Usage:

#include <sd811.h>

int SD811_GetMode(unsigned int **Mode*);

Description:

The SD811_GetMode() function is used to determine the present operating mode of the 754P-LC3.

After a call to this function, the value of *Mode* will be equal to one of the symbolic constants shown below, which are defined in sd811.h.

Possible *Mode* values:

SD811_CHECKMODE -- This indicates the Check-Weighing Mode.
SD811_MOTIONMODE -- This indicates the In-Motion Check-Weighing Mode.
SD811_CONTMODE -- This indicates the Continuous Measurement Mode.
SD811_FILLMODE -- This indicates the Fill-by-Weight Mode.
SD811_CALMODE -- This indicates the Calibration Mode.

Return Value:

If the function executes properly it will return the value SD811_OK, which is defined in sd811.h. (The symbolic constant SD811_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 sd811.h. The *Mode* argument **may not** have valid data.

Return values indicating errors:

SD811_IBF_TIMEOUT -- This indicates that the 754P-LC3 has not processed the last command byte or data byte, and therefore cannot accept a new byte.

SD811_OBF_TIMEOUT -- This indicates that the 754P-LC3 has not placed a data byte into its output register within the timeout period.

SD811_ERROR -- This indicates an invalid data transfer.

Example:

```
#include <sd811.h>
#include <stdio.h>

main()
{
   int ReturnValue, Mode;

/* Read the current operating mode of the 754P-LC3. */
   ReturnValue=SD811_GetMode(&Mode);

   if(ReturnValue==SD811_OK)
   {
      printf("The present operating mode is: ");

      switch(Mode)
      {
         case SD811_CHECKMODE:
            printf("Check-Weighing Mode.");
            break;
         case SD811_MOTIONMODE:
            printf("In-Motion Check-Weighing Mode.");
            break;
         case SD811_CONTMODE:
            printf("Continuous Measurement Mode.");
            break;
         case SD811_FILLMODE:
            printf("Fill-by-Weight Mode.");
            break;
         case SD811_CALMODE:
            printf("Calibration Mode.");
            break;
         default:
            break;
      }
   }
   else
      printf("The function returned the error %d.\n",ReturnValue);
}
```

See Also:

SD811_SetMode()

SD811_SetMode()

Usage:

#include <sd811.h>

int SD811_SetMode(unsigned int *Mode*);

Description:

The SD811_SetMode() function is used to command the 754P-LC3 to operate in one of its programmed operating modes.

Before calling the function, the value of *Mode* should be set to one of the symbolic constants shown below, which are defined in sd811.h. If the *Mode* value is not recognized by the 754P-LC3, it will default to the Calibration Mode.

Possible operating modes specified by *Mode*:

SD811_CHECKMODE -- This indicates the Check-Weighing Mode.
SD811_MOTIONMODE -- This indicates the In-Motion Check-Weighing Mode.
SD811_CONTMODE -- This indicates the Continuous Measurement Mode.
SD811_FILLMODE -- This indicates the Fill-by-Weight Mode.
SD811_CALMODE -- This indicates the Calibration Mode.

When the function is called, the 754P-LC3 will resume operation in the operating mode that was specified by the value of *Mode.*

Return Value:

If the function executes properly it will return the value SD811_OK, which is defined in sd811.h. (The symbolic constant SD811_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 sd811.h. The hardware **may not** be properly configured to the operating mode specified by *Mode*.

Return values indicating errors:

SD811_IBF_TIMEOUT -- This indicates that the 754P-LC3 has not processed the last command byte or data byte, and therefore cannot accept a new byte.

SD811_OBF_TIMEOUT -- This indicates that the 754P-LC3 has not placed a data byte into its output register within the timeout period.

SD811_ERROR -- This indicates an invalid data transfer.

Example:

```
#include <sd811.h>
#include <stdio.h>

main()
{
   int ReturnValue, Mode;

/* Choose the Continuous Measurement Mode. */
   Mode=SD811_CONTMODE;

/* Command the 754P-LC3 to operate in this mode. */
   ReturnValue=SD811_SetMode(Mode);

   if(ReturnValue==SD811_OK)
      printf("The board is running in Continuous Measurement Mode.\n");
   else
      printf("The function returned the error %d.\n",ReturnValue);
}
```

See Also:

SD811_GetMode()

SD811_GetWeight()

Usage:

#include <sd811.h>

int SD811_GetWeight(float **Weight*);

Description:

The SD811_GetWeight() function is used to read the present scaled weight value that is measured by the 754P-LC3.

After a call to this function, the value of *Weight* will be equal to the last scaled measurement calculated by the 754P-LC3. This value is computed by the 754P-LC3 with the formula:

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

where Data is the raw analog-to-digital conversion result (see SD811_GetData()) and Scale, K, Tare, and Ref are the hardware calibration settings (see SD811_GetCalibration()).

Return Value:

If the function executes properly it will return the value SD811_OK, which is defined in sd811.h. (The symbolic constant SD811_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 sd811.h. The *Weight* argument **will not** have a valid value.

Return values indicating errors:

SD811_IBF_TIMEOUT -- This indicates that the 754P-LC3 has not processed the last command byte or data byte, and therefore cannot accept a new byte.

SD811_OBF_TIMEOUT -- This indicates that the 754P-LC3 has not placed a data byte into its output register within the timeout period.

SD811_ERROR -- This indicates an invalid data transfer.

Example:

```
#include <sd811.h>
#include <stdio.h>

main()
{
   int ReturnValue;
   unsigned int Data;
   float Weight;

/* Choose the Continuous Measurement Mode. */
   if(SD811_SetMode(SD811_CONTMODE)==SD811_OK)
      printf("The board is running in Continuous Measurement Mode.\n");
   else
   {
      printf("Error in setting Mode. Program terminated.\n");
      exit(1);
   }

/* 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=(SD811_GetData(&Data)))==SD811_OK)
         printf("Conversion Result = %04x  ",Data);
      else
         printf("Error %d.\n",ReturnValue);

      if((ReturnValue=(SD811_GetWeight(&Weight)))==SD811_OK)
         printf("Scaled Weight = %f\n",Weight);
      else
         printf("Error %d.\n",ReturnValue);
   }
}
```

See Also:

SD811_GetData(), SD811_GetCalibration(), SD811_SetCalibration()

SD811_GetData()

Usage:

#include <sd811.h>

int SD811_GetData(long **Data*);

Description:

The SD811_GetData() function is used to read the present analog-to-digital conversion value that is measured by the 754P-LC3.

After a call to this function, the value of *Data* will be equal to the last measurement taken by the 754P-LC3. This value is a 24-bit integer in the range [00000000_{16} ,$008fffff_{16}$].

This function transfers three bytes from the 754P-LC3. It then pads the most significant byte with zero.

Return Value:

If the function executes properly it will return the value SD811_OK, which is defined in sd811.h. (The symbolic constant SD811_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 sd811.h. The *Data* argument **will not** have a valid value.

Return values indicating errors:

SD811_IBF_TIMEOUT -- This indicates that the 754P-LC3 has not processed the last command byte or data byte, and therefore cannot accept a new byte.

SD811_OBF_TIMEOUT -- This indicates that the 754P-LC3 has not placed a data byte into its output register within the timeout period.

SD811_ERROR -- This indicates an invalid data transfer.

Example:

```
#include <sd811.h>
#include <stdio.h>

main()
{
   int ReturnValue;
   longint Data;
   float Weight;

/* Choose the Continuous Measurement Mode. */
   if(SD811_SetMode(SD811_CONTMODE)==SD811_OK)
      printf("The board is running in Continuous Measurement Mode.\n");
   else
   {
      printf("Error in setting Mode. Program terminated.\n");
      exit(1);
   }

/* 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=(SD811_GetData(&Data)))==SD811_OK)
         printf("Conversion Result = %08x h ",Data);
      else
         printf("Error %d.\n",ReturnValue);

      if((ReturnValue=(SD811_GetWeight(&Weight)))==SD811_OK)
         printf("Scaled Weight = %f\n",Weight);
      else
         printf("Error %d.\n",ReturnValue);
   }
}
```

See Also:

SD811_GetWeight(), SD811_GetCalibration(), SD811_SetCalibration()

SD811_GetIOStates()

Usage:

#include <sd811.h>

int SD811_GetIOStates(unsigned int **IOBuffer*);

Description:

The SD811_GetIOStates() function is used to read the present states of the input/output pins of the 754P-LC3. This function reads all the input/output pins simultaneously; if you wish to access just one pin, use SD811_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-LC3, 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	Input3	1	0
6	Input2	1	0
5	Input1	1	0
4	Input0	1	0
3	Output3	1	0
2	Output2	1	0
1	Output1	1	0
0	Output0	0	1

Return Value:

If the function executes properly it will return the value SD811_OK, which is defined in sd811.h. (The symbolic constant SD811_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 sd811.h. The *IOStates* argument **may not** have a valid value.

Return values indicating errors:

SD811_IBF_TIMEOUT -- This indicates that the 754P-LC3 has not processed the last command byte or data byte, and therefore cannot accept a new byte.

SD811_OBF_TIMEOUT -- This indicates that the 754P-LC3 has not placed a data byte into its output register within the timeout period.

SD811_ERROR -- This indicates an invalid data transfer.

Example:

```
#include <sd811.h>
#include <stdio.h>

main()
{
   int ReturnValue, Buffer;

/* Choose the Check-Weighing Mode. */
   if(SD811_SetMode(SD811_CHECKMODE)==SD811_OK)
      printf("The board is running in Check-Weighing Mode.\n");
   else
   {
      printf("Error in setting Mode. Program terminated.\n");
      exit(1);
   }

/* Examine the input and output states to see if any inputs are
asserted.*/
   if((ReturnValue=(SD811_GetIOStates(&Buffer)))==SD811_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:

SD811_SetIOStates()

SD811_SetIOStates()

Usage:

#include <sd811.h>

int SD811_SetIOStates(unsigned int *IOBuffer*);

Description:

The SD811_SetIOStates() function is used to set the states of the four output pins of the 754P-LC3. This function modifies all the output pins simultaneously; if you wish to access just one pin, use SD811_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-LC3 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	Output3	1	0
2	Output2	1	0
1	Output1	1	0
0	Output0	0	1

Return Value:

If the function executes properly it will return the value SD811_OK, which is defined in sd811.h. (The symbolic constant SD811_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 sd811.h. The outputs of the 754P-LC3 **may not** have been driven to the states specified in the *IOStates* argument.

Return values indicating errors:

SD811_IBF_TIMEOUT -- This indicates that the 754P-LC3 has not processed the last command byte or data byte, and therefore cannot accept a new byte.

SD811_OBF_TIMEOUT -- This indicates that the 754P-LC3 has not placed a data byte into its output register within the timeout period.

SD811_ERROR -- This indicates an invalid data transfer.

Example:

```
#include <sd811.h>
#include <stdio.h>

main()
{
   int ReturnValue, Buffer;

/* Choose the Check-Weighing Mode. */
   if(SD811_SetMode(SD811_CHECKMODE)==SD811_OK)
      printf("The board is running in Check-Weighing Mode.\n");
   else
   {
      printf("Error in setting Mode. Program terminated.\n");
      exit(1);
   }

/* Set the bit values in Buffer to turn on all the outputs.*/
   Buffer=0x000e;

   if((ReturnValue=(SD811_SetIOStates(Buffer)))==SD811_OK)
      printf("All the outputs have been energized.\n");
   else
      printf("Error %d.\n",ReturnValue);
}
```

See Also:

SD811_GetIOStates()

SD811_GetFilter()

Usage:

#include <sd811.h>

int SD811_GetFilter(unsigned integer **pFilter*);

Description:

The SD811_GetFilter() function is used to determine the value of the digital filter parameter that is currently stored in the 754P-LC3's EEPROM. The Filter variable, which is defined in sd811.h, has the following form:

The 754P-LC3 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-LC3.

Return Value:

If the function executes properly it will return the value SD811_OK, which is defined in sd811.h. (The symbolic constant SD811_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 sd811.h. The *Filter* argument **will not** have valid data (ie. the data will be unchanged).

Return values indicating errors:

SD811_IBF_TIMEOUT -- This indicates that the 754P-LC3 has not processed the last command byte or data byte, and therefore cannot accept a new byte.

SD811_OBF_TIMEOUT -- This indicates that the 754P-LC3 has not placed a data byte into its output register within the timeout period.

SD811_ERROR -- This indicates an invalid data transfer.

Example:

```
#include <sd811.h>
#include <stdio.h>

main()
{
   int ReturnValue;
   int Filter;

/* Read the Filter value currently stored in the 754P-LC3. */
   ReturnValue=SD811_GetFilter(&iFilter);

   if(ReturnValue==SD811_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:

SD811_GetParameters()

SD811_SetFilter()

Usage:

#include <sd811.h>

int SD811_SetFilter(unsigned int *Filter*);

Description:

The SD811_SetFilter() function is used to load the 754P-LC3's EEPROM with values for its filter variable used to define the function of the unit's digital filter.

The 754P-LC3 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-LC3 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-LC3 will be loaded with the values specified by the *Filter* argument.

Return Value:

If the function executes properly it will return the value SD811_OK, which is defined in sd811.h. (The symbolic constant SD811_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 sd811.h. The values specified in the *Filter* variable mayor may not have been loaded into the 754P-LC3's memory; if an error occurs, you may wish to call SD811_GetFilter() to check the values.

Return values indicating errors:

SD811_IBF_TIMEOUT -- This indicates that the 754P-LC3 has not processed the last command byte or data byte, and therefore cannot accept a new byte.

SD811_OBF_TIMEOUT -- This indicates that the 754P-LC3 has not placed a data byte into its output register within the timeout period.

SD811_ERROR -- This indicates an invalid data transfer.

Example:

```
#include <sd811.h>
#include <stdio.h>

main()
{
   int ReturnValue;
   int Filter;

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

/* Load the values into the 754P-LC3's EEPROM. */
   ReturnValue=SD811_SetFilter(Filter);

   if(ReturnValue==SD811_OK)
      printf("The new value has been downloaded successfully.\n");
   else
      printf("The function returned the error %d.\n",ReturnValue);
}
```

See Also:

SD811_GetFilter()

SD811_GetDataCount()

Usage:

#include <sd811.h>

int SD811_GetDataCount(unsigned int **pDataCount*);

Description:

The SD811_GetDataCount() function is used read the 754P-LC3's DATACOUNT variable.

This variable allows the PC to determine when a new measurement result is available, providing a synchronization mechanism between the PC and the module. Reading and monitoring the DataCount allows the PC program to read each measurement once and only once.

The 754P-LC3 module maintains a DATACOUNT variable, incrementing this single byte variable each time a new measurement is available. Each measurement is effectively sequence stamped. The variable 255 rolls over to 0 at the next measurement.

By regularly reading the DataCount variable from the 754P-LC3, the PC can determine if the data available in the module has already been read. If it has not, the PC may issue a GetData or GetWeight command to read the new data

Return Value:

Unlike other functions in the Library, SD811_GetDataCount ***returns only the single-byte DATACOUNT variable, without the terminating SD811_OK.***

Implementation in this fashion allows all 256 possible DATACOUNT values to be valid.

Example:

// Wait for new conversion result, then read it

```
#include <sd811.h>
#include <stdio.h>

main()
{
   int StartData;
   int LastData;
   long Data;
   time_t tStart;

/* Wait for new conversion (but not more than 2 seconds)and then read
it. */
  SD811_GetDataCount(&StartData);
  for(tStart=NULL; time(NULL-tStart)<2)
      {
      SD811_GetDataCount(&LastData);
      if(StartData!=LastData)   //new result is available
            {
            SD811_GetData(&Data);  //read it
            StartData=LastData;    //record the datacount for the
result
            break:
            }
      {
```

SD811_GetADRegister()

Usage:

#include <sd811.h>

int SD811_GetADRegister(long **pADRegister*);

Description:

The SD811_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 sd811.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 SD811_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 SD811_OK, which is defined in sd811.h. (The symbolic constant SD811_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 sd811.h. The *pADRegister* argument **may not** have valid data.

Return values indicating errors:

SD811_IBF_TIMEOUT -- This indicates that the 754P-LC3 has not processed the last command byte or data byte, and therefore cannot accept a new byte.

SD811_OBF_TIMEOUT -- This indicates that the 754P-LC3 has not placed a data byte into its output register within the timeout period.

SD811_ERROR -- This indicates an invalid data transfer.

Example:

```
#include <sd811.h>
#include <stdio.h>

main()
{
   int ReturnValue;
    long pADRegister ;

/* Read the current A/D Register setup value of the 754P-LC3. */
   ReturnValue=SD811_GetADRegister(&pADRegister);

   if(ReturnValue==SD811_OK)
   {
      printf("The present register set for: ");

      switch(pADRegister)
      {
         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",pADRegister");
            break;
      }
   }
   else
      printf("The function returned the error %d.\n",ReturnValue);
}
```

See Also:

SD811_SetADRegister()

SD811_SetADRegister()

Usage:

#include <sd811.h>

int SD811_SetADRegister(long *iADRegister*);

Description:

The SD811_SetADRegister() function is used to load the 754P-LC3'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-LC3 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 SD811_OK, which is defined in sd811.h. (The symbolic constant SD811_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 sd811.h. The values specified in the *Filter* variable mayor may not have been loaded into the 754P-LC3's memory; if an error occurs, you may wish to call SD811_GetADRegister() to check the values.

Return values indicating errors:

SD811_IBF_TIMEOUT -- This indicates that the 754P-LC3 has not processed the last command byte or data byte, and therefore cannot accept a new byte.

SD811_OBF_TIMEOUT -- This indicates that the 754P-LC3 has not placed a data byte into its output register within the timeout period.

SD811_ERROR -- This indicates an invalid data transfer.

Example:

```
#include <sd811.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-LC3's EEPROM. */
   ReturnValue=SD811_SetADRegister(iADRegister);

   if(ReturnValue==SD811_OK)
      printf("The new value has been downloaded successfully.\n");
   else
      printf("The function returned the error %d.\n",ReturnValue);
}
```

See Also:

SD811_GetADRegister()

u800 PRODUCTS
Demonstration System

Project Lancelot

754P-LC3 Demonstration/Test System

The Lancelot project is a Borland C++ test system designed to demonstrate the 754P-LC3 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

IO811.CPP
TEST811.CPP
UTIL811.CPP
VIDEO811.CPP
WEIGH811.CPP
SET811.CPP
MODE811.CPP
CAL811.CPP

Header Files

UTIL811.H

Executable Files

LANCELOT.EXE, LOADCELL.EXE

Borland C++ Project Files

LANCELOT.PRJ

Installation and Operation

Lancelot.exe requires that the load cell controller is installed at bus address 220. Any other bus address requires that the header file be edited to the correct bus address and the system be rebuilt. As shipped from Scanning Devices, bus address jumpers are set at 220.

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 directions on the screen.

Calibration: As shipped from Scanning Devices, the load cell controller is calibrated for a 100 pound, 3 millivolt per volt load cell. Use the calibration menu to re-calibrate the controller to the characteristics of the load cell you are using.

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 modes will suggest the signal processing techniques required in your application.

Reference

Header File UTIL811.h

This file specifies the addresses, data and control characters used in transfers between the 754P-LC3 module and host PC.

```
/*
// c:\u800\dev\ninja\bc\util811.h
//
// Company: Scanning Devices Inc.
// Engineer: Stephen Bourque
// Date: 13 Sep 93
// Project: PARSEVAL 1.1 -- SD811 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>
#include <bios.h>

/*
// Hardware register locations.
*/
#define SD811_BASE_ADDRESS    0x220
#define SD811_CSR             SD811_BASE_ADDRESS+3
#define SD811_DATA            SD811_BASE_ADDRESS+0

/*
// Error codes.
*/
#define SD811_OK              0x00
#define SD811_FAIL            0x01
#define SD811_IBF_TIMEOUT     0x02
#define SD811_OBF_TIMEOUT     0x03
#define SD811_ERROR           0xff

/*
// Operating modes.
*/
#define SD811_CHECKMODE       0x00
#define SD811_MOTIONMODE      0x01
#define SD811_CONTMODE        0x02
#define SD811_FILLMODE        0x03
#define SD811_CALMODE         0x04


/*
// Filter parameters
```

```
*/
#define SD811_FILTER1                                   0x01
#define SD811_FILTER2                                   0x02
#define SD811_FILTER4                                   0x04
#define SD811_FILTER8                                   0x08

/*
//A/D Converter Registers
*/

#define AD7712_160HZ                                0x00208080
#define AD7712_80HZ         0x00208100
#define AD7712_40HZ         0x00208200
#define AD7712_20HZ         0x00208400
#define AD7712_10HZ         0x002087a0
/*
// IO states and selections.
*/
#define SD811_IO_ON         0x00
#define SD811_IO_OFF        0x01
#define SD811_INPUT3        0x80
#define SD811_INPUT2        0x40
#define SD811_INPUT1        0x20
#define SD811_INPUT0        0x10
#define SD811_OUTPUT3       0x08
#define SD811_OUTPUT2       0x04
#define SD811_OUTPUT1       0x02
#define SD811_OUTPUT0       0x01

/*
// Instructions.
*/
#define SD811_RESET         0x50
#define SD811_GETPARAM      0x60
#define SD811_SETPARAM      0x70
#define SD811_GETMODE       0x61
#define SD811_SETMODE       0x71
#define SD811_GETCAL        0x62
#define SD811_SETCAL        0x72
#define SD811_GETWEIGHT     0x63
#define SD811_GETIO         0x64
#define SD811_SETIO         0x74
#define SD811_GETDATA       0x65
#define SD811_HALT          0x51
#define SD811_START         0x52
#define SD811_GETFILTER         0x66
#define SD811_SETFILTER     0x76
#define SD811_GETDATACOUNT          0x67
#define SD811_GETADREGISTER         0x68
#define SD811_SETADREGISTER         0x78

/*
// 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 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
```

```
/*
// Com Port Parameters
*/
// #define COM2 1
// #define SETTINGS (_COM_2400|_COM_CHR7|_COM_STOP1|_COM_EVENPARITY)

/*
// 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];
};



/*
// Function prototypes in c:\u800\dev\ninja\bc\test811.c.
*/
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 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 About(void);
int gscanf(char *szBuffer, int MaxNum);
void KBFlush(void);
```

```
/*
// Function prototypes in c:\u800\dev\ninja\bc\weigh811.c.
*/
int WeighMenu(void);
int WeighContinuousMenu(void);
int WeighSingleMenu(void);
int WeighFileMenu(void);
int WeighDataMenu(void);
int WeighScaleMenu(void);
int FilterData(long *iData);

/*
// Function prototypes in c:\u800\dev\ninja\bc\set811.c.
*/
int SettingsMenu(void);
int SettingsSetpointMenu(int iIndex);
int SettingsViewMenu(void);

/*
// Function prototypes in c:\u800\dev\ninja\bc\util811.cpp
*/
int SD811_Reset(void);
int SD811_GetParameters(struct ControlSettings *Points);
int SD811_SetParameters(struct ControlSettings *Points);
int SD811_GetMode(unsigned int *pMode);
int SD811_GetDataCount(unsigned int *pDataCount);
int SD811_SetMode(unsigned int iMode);
int SD811_SetFilter(unsigned int iFilter);
int SD811_GetFilter(unsigned int *pFilter);
int SD811_GetCalibration(struct CalibrationSettings *Points);
int SD811_SetCalibration(struct CalibrationSettings *Points);
int SD811_GetWeight(float *pWeight);
int SD811_GetIOStates(unsigned int *pIOBuffer);
int SD811_SetIOStates(unsigned int iIOBuffer);
int SD811_GetSingleIOState(unsigned int iSelectPin, unsigned int *pSwitch);
int SD811_SetSingleIOState(unsigned int iSelectPin, unsigned int iSwitch);
int SD811_GetData(long *pData);
int SD811_SetADRegister(long iADRegister);
int SD811_GetADRegister(long *pADRegister);
int WaitForIBFClear(void);
int WaitForOBFSet(void);
int WaitForNewData(void);


/*
// Function prototypes in c:\u800\dev\ninja\bc\mode811.c.
*/
int ModeMenu(void);
int ModeModeXMenu(int iIndex);
int ModeViewMenu(void);

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

```
int IOInputMenu(unsigned int *pIOBuffer);

/*
// Function prototypes in c:\u800\dev\ninja\bc\cal811.c.
*/
int CalibrateMenu(void);
int CalibrateStandardMenu(void);
int CalibrateSpecificationMenu(void);
int CalibrateFactoryMenu(void);
int CalibrateTareMenu(void);
int CalibrateViewMenu(void);
int FilterViewMenu(void);
int FilterXMenu(int iIndex);
int CalibrateFilterMenu();
int ConverterMenu(void);
/*
// 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);
```

754P-LC3 PC/104 Compliant Load Cell Controller
754I-SA1 Desktop PC (ISA) Load Cell Controller

Installation Notes

These notes reflect user experiences, engineering changes and other items which may effect installation and operation of the load cell controller. They are updated periodically posted on the internet at http:/www.tiac.net/users/scanning

1. User experience - Lancelot requires a mouse; systems without a mouse hang when keyboard input is used. Lancelot also requires hardware floating point.

Lancelot uses PC register 33 to de-bounce the mouse click, preventing switch bounce from being interpreted as sequential clicks. Processors with custom bios may use a different register to interface the mouse. Systems without a mouse wait forever as Lancelot waits for the register 33 to clear.

Lancelot graphics conflict with floating point emulation.

Modified lancelot named "loadcell" has been constructed to by-pass these problems, especially in small system environments. Use - "loadcell.exe" software in situations where a mouse is not used, with processors employing custom bios, or processors using floating point emulation.

2. User experience - Lancelot hangs on GetWeight using 8080, won't complete calibration.

Sorry folks, we can go back only so far. 8080 does not support floating point data types without external floating point hardware. 754P-LC3 stores its variables Weight and Presets 1-4 in floating point format (consistent with IEEE float point data types) Calibration constants used to calculate Weight from Data (long integer) are also floating point data types. These calibration constants must originate in the PC.

Without floating point on the PC, the 754P-LC3 can measure and report only integer data and must operate in mode 5 (no comparison to setpoints). Do not attempt to GetWeight, GetParameters or SetParameters.

3. User experience - A Win Systems processor requires only +5 Volt supply and entices users to bypass the +12 and -12 supplies identified in the PC/104 specification. 754P-LC3 requires and uses the +12 supply for biasing of the analog circuitry including circuitry which regulates the load cell excitation and circuitry which powers the instrumentation amplifier used to condition the load cell measurement signal.

Note: Attempting to operate the 754P-LC3 without +12 supply will produce unsatisfactory results and may damage the module's analog circuitry.

4. User experience - Same as #1 for Ampro "Little Board" (model number unknown). User reports that an on-board jumper is required to source +12 volt supply.

5. Engineering Change - 754P-LC3 revision 1 released 11/8/95. All shipments after this date are being made with a revised module. Revisions are summarized here:

Higher precision analog-to-digital converter
Use of a Post conversion digital filter with user-programmable filter parameters
Windows software supported as "WinCell"
Lancelot Software incorporates Filter parameter setting in Calibration Menus
On-board EPROM is labeled NJ003.2 (or later) and is required for this revision.

6. User experience - Bus address 220 is commonly used for PC sound-blaster. Conflict in bus address must be resolved in favor of sound blaster because it is not flexible.

Recommendation - reset the 754P-LC3 bus address to 330; change one parameter in the "UTIL811.H" file from 220 to 330 and rebuild the software.

There is no known use for bus address 330

Contact Scanning Devices for "WinCell" software using bus address 330 if necessary.

7. New Product - 4-channel load cell controller for PC/104 (Scanning Devices Part # 754P-LC4)

Implements four independent load cell input channels. Each channel operates asynchronously. As each conversion/measurement is completed, it is stored in on-board memory is available to the PC. Throughput is increased by a factor of four (Careful! Each channel runs at the same speed as on the single channel module. Multiplexing is NOT used, so that no analog switch settling time is required)

Note 1: The speed of each channel is the same as in the case of the single channel version.
Note 2: The load cells have a common excitation. Excitation voltage is 4.3 volts
Note 3: Only +5 volt supply is required. +12 and -12 are not used.
Note 4: Load Cell connection is by means of 24-pin rather than 16-pin connector. Connector type is the same. Load Cell connector and Digital I/O connector are on opposite edges of the module.
Note 5: Channel specific PC commands address channel within module. The channel address consists of the least significant two bits of the command. Therefore, the 4-channel module is NOT software compatible with the single channel module. The Lancelot software system for the four channel module has been expanded to support multiple modules per system.

If you have written your application "correctly", conversion requires only changes in the header files which define the commands. Contact Scanning Devices for advice.

8. New Product - ISA bus interface (IBM and compatible desktop computers) load cell controller is released. Scanning Devices part number (754I-SA1, fourth character is "Letter I") Same as 754P-LC3 except:

> +5 Volt supply is only power supply required. +12 and -12 volt supplies are not used.
> Connection to load cell and digital input/outputs are by 9-pin D-Sub connectors on module, consistent with desktop PC I/O.
> Bus address is held in software and programmable by the user.
> Windows Software is released as "WinCell"

9. Publication - Circuit Cellar Ink published an article describing Scanning Devices Load Cell controllers in their June 1996 issue. The article was written by David Chanoux. Reprints are available and may be of interest to users.

For a reprint of the article, email to mail@scanningdevices.com

10. Graphic conflict with floating point.

Lancelot doesn't run on systems without floating point hardware.

Borland BGI graphics conflict with the Borland floating point emulator in C++ version 3. After initializing BGI graphics, any attempt to perform floating point operations will cause the system to hang with a floating point exception. Masking the floating point registers simply transfers the problem to the extended memory manager.

Scanning Devices has developed and released "LOADCELL" a text-only version of Lancelot. This system performs the same functions as Lancelot but without the BGI graphics. It is a minimal system designed to demonstrate load cell operations in text mode. It requires keyboard and VGA monitor only. The mouse is not supported.

If you have are running on a system without floating point hardware (no 287, 387 or Wietek chip) use "LOADCELL".

Loadcell began distribution on 10-1-96 with all orders for 754P-SW3. Contact Scanning Devices if you need a copy of "LOADCELL".

11. More bus address conflicts

Some Packard Bell systems include a combined sound blaster and fax modem card occupying two bus addresses: 220 for the sound blaster and 330 as COM4 for the fax modem. The SD816 ISA bus controller uses bus address 330 hard coded in the modules eprom. Scanning Devices Load Cell Software for Windows allows the user to change the bus address but requires an eprom coded with the alternate address. Lancelot and Loadcell systems allow the user to change the bus address in the header file. This procedure requires rebuilding the executable files.

Q: How do I recognize bus address conflicts?

A: First, you will see an error message generated by Wincell, Loadcell or Lancelot to the effect that the module is not responding due to timeout or error. This error may be the result of an address conflict. Second, examine your system to determine which addresses are occupied. You may do this through the Microsoft Diagnostics (enter "\DOS\MSD" on the command line to start Microsoft diagnostics in MS-DOS or open the Control Panel in Windows). These tools will identify MS-DOS and Windows supported devices by their addresses.

If you need an alternate bus address to operate your SD816 Load Cell Controller, contact Scanning Devices for advice.

If you have advice you are willing to post for other users, send email to:
mail@scanningdevices.com

12. Synchronizing PC and 754 module. - 754 software version 3.3 implements a DATACOUNT variable which is updated on completion of an analog to digital conversion. The PC may read the current value of the module's DATACOUNT variable to determine if new data is available. This technique allows the PC to obtain each and every conversion result. See instruction book, page 57-58 for details.

13. Adjusting the module's A/D sampling rate - The Analog-to-Digital converter's precision and update rate may be adjusted to serve a variety of application needs. 754 software version 3.4 implements functions which read and write the converter's setup register to adjust update rate and precision. See the instruction book, page 59-63 for details.

Last Revision 6-16-99