BURR-BROWN® APPLICATION BULLETIN
Mailing Address: PO Box 11400, Tucson, AZ 85734 · Street Address: 6730 S. Tucson Blvd., Tucson, AZ 85706 · Tel: (520
Telex: 066-6491 · FAX (520) 889-1510 · Product Info: (800) 548-6132 · Internet: www.burr-brown.com/ · FAXLine: (80

ACCESSING THE ADS1210 DEMO BOARD WITH YOUR PC

by Miaochen Wu

The ADS1210 demo board along with the PC software is a complete data acquisition system using the ADS1210 24-bit delta-sigma analog-to-digital converter. The demo board has two microcontrollers and 32K on board memory. It is accessed using a PC program through the PC parallel port. This application note explains the hardware design and the software design of the demo board. The hardware design discusses the system block diagram and the circuit diagram. The software design discusses the data structure and the core procedures for the PC programming. An example of PC Pascal program is provided to illustrate how to access the ADS1210 demo board. With the understanding of the demo board design, a new PC program can be implemented to do the data acquisition using the ADS1210 demo board.

SYSTEM OVERVIEW

The ADS1210 demo board consists of four major devices. Figure 1 is the system block diagram. The device U4 is considered as a central control device. It gets instruction from the PC software, controls the operation of the ADS1210, and transfers the data converted from the ADS1210 to the device U5. The device U5 is mainly for the 32K memory management as U4 is usually heavily loaded by the PC or the ADS1210. The PC software provided with the ADS1210 demo board is mainly designed for customer to evaluate the performance of the ADS1210. However, you can write your own PC program to access the ADS1210 demo board.

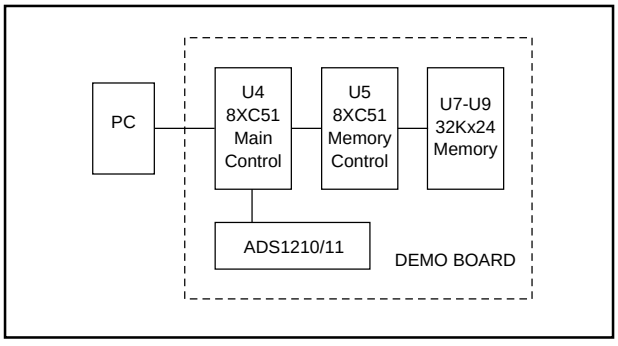

FIGURE 1. System Block Diagram.

CIRCUIT DIAGRAM

The ADS1210 demo board circuit diagram consists three sections: the PC interface section, the memory management section and the ADS1210 device section. Figure 2 shows the circuit diagram of PC interface section. The parallel port of the PC is connected to the demo board by the connector J1. Table I shows the pin connection and the pin function of the connector J1.

PC PORT PIN NO.	BUS NAME	FUNCTION
1	_STB	The PC interrupt signal to the device U4
2 - 9	DB0 - DB7	The PC data or instruction output to the ADS1210 demo board
10	DATA3	The data input from the ADS1210 demo board
11	_BUSY	The busy signal input from the ADS1210 demo board
12	DATA2	The data input from the ADS1210 demo board
13	DATA1	The data input from the ADS1210 demo board
14	AUTOFEED	The PC interrupt signal to the device U5
15	DATA0	The data input from the ADS1210 demo board
16 - 18	NC	Not connected
19 - 25	GND	Ground

TABLE I. The Pin Connection and the Function of the Connector J1.

The PC sends two kinds of information to the ADS1210 demo board. The data is the information that PC sends to or retrieves from the demo board. The instruction is the information that PC sends to the demo board. The demo board does some operations when an instruction is received.

The PC sends the signal _STB to interrupt the device U4 when the PC wants to access to the device U4. If the device U4 is busy, the signal _BUSY is sent to the PC to indicate the busy status of the device.

The PC sends the data or instruction through the data bus DB0 - DB7, and retrieves the data or instruction through the data bus DATA3 - DATA0. The data sent by the PC to the demo board is always in one byte (8-bit) package. However, the data retrieved by the PC from the demo board is always in one nibble (4-bit) package.

Figure 3 shows the circuit diagram of the memory management section. The PC sends the signal AUTOFEED to interrupt the device U5 when the PC wants to retrieve the data from the 32K memory. The data in the 32K memory is the output data from the ADS1210. The device U5 accesses the 32K memory with an incremental address. The PC retrieves data of the 32K memory with a decrement address. Figure 4 illustrates the memory accessing direction used by the device U5 and the PC.

The circuit diagram of the ADS1210 device section is not shown here. Please refer to the ADS1210 demo board data sheet.

Printed in U.S.A. March, 1997

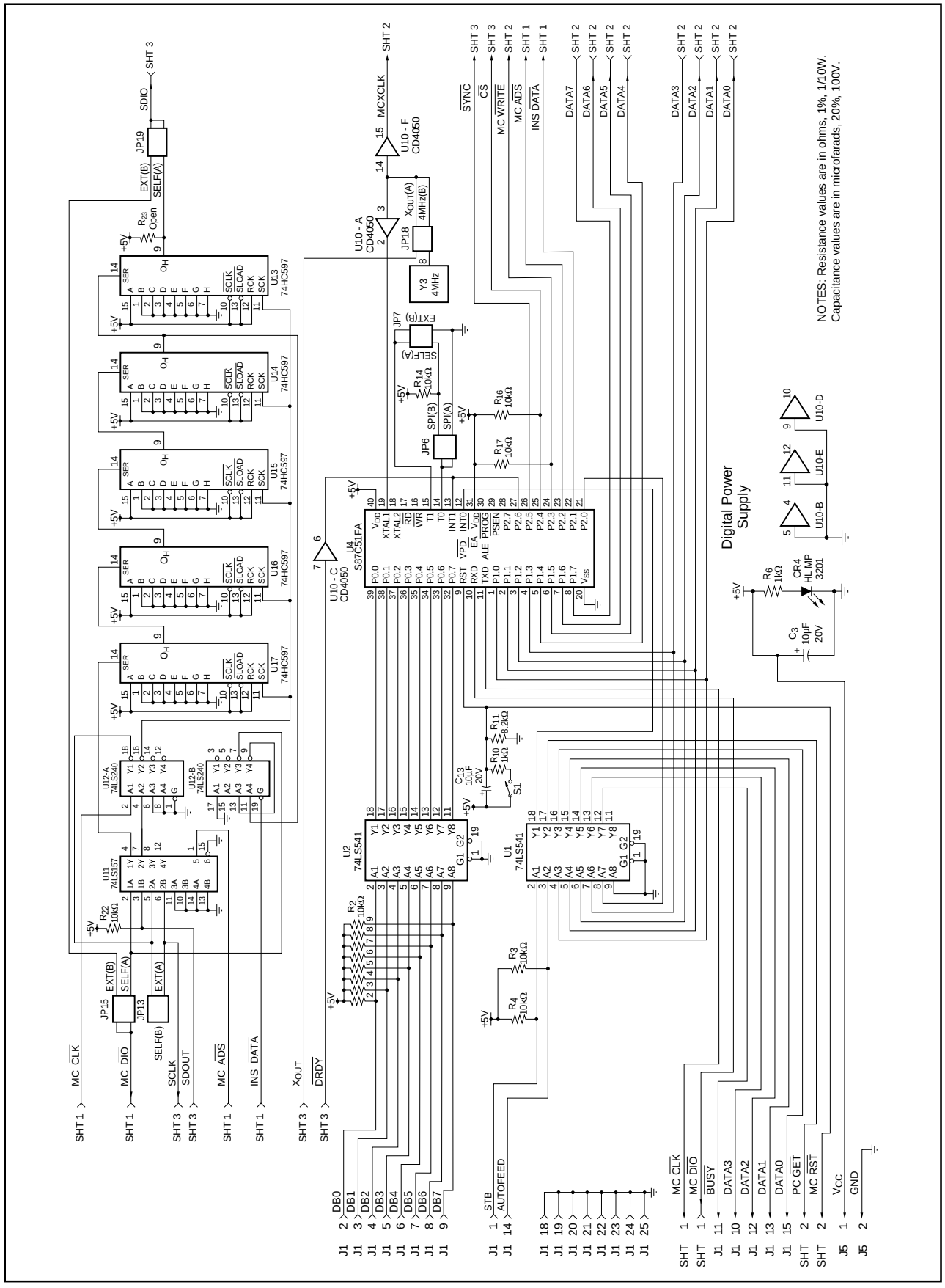

FIGURE 2. Circuit Diagram of PC Interface Section (SHT 1).

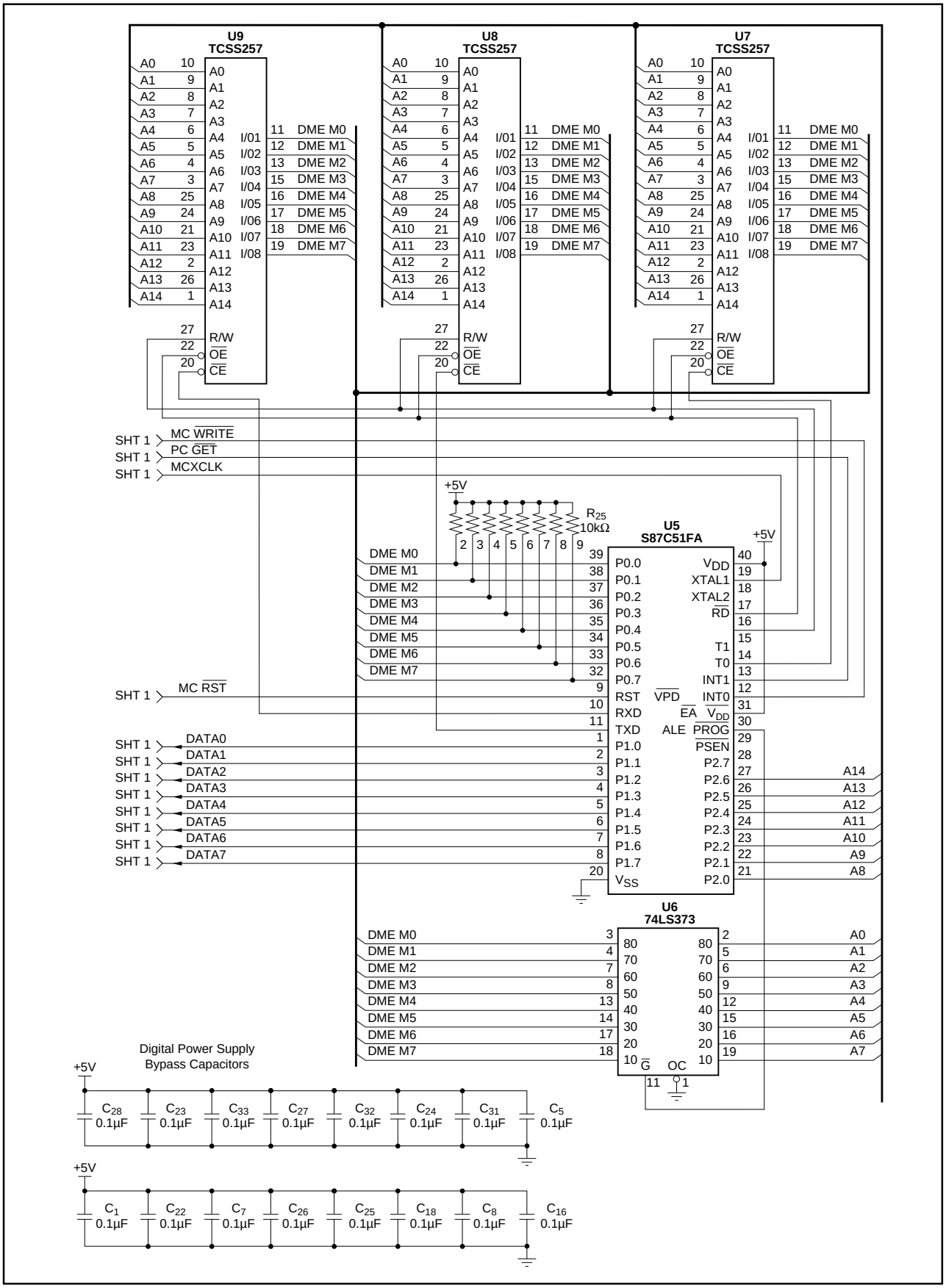

FIGURE 3. Circuit Diagram of Memory Management Section (SHT 2).

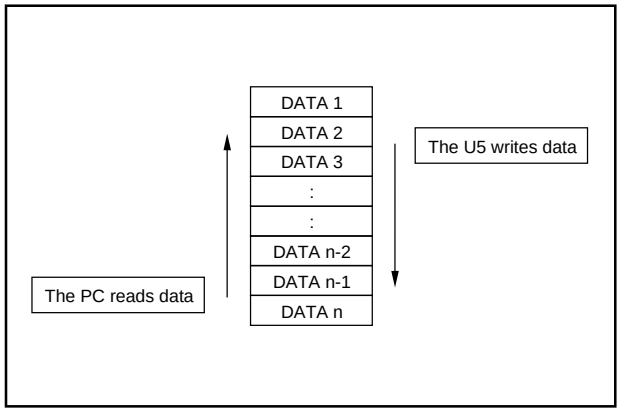

FIGURE 4. The Memory Accessing for the PC and the Device U5.

PC PROGRAMMING

Data Structure

The ADS1210 has five internal registers. They are the 24-bit Data Output Register (DOR), 8-bit Instruction Register (INSR), 32-bit Command Register (CMR), 24-bit Offset Calibration Register (OCR) and 24-bit Full-Scale Calibration Register (FCR). To access the content of the ADS1210 internal registers, the address in Table II is used by the PC program to access the ADS1210 internal registers and some registers of the microcontroller U4. There address are not necessary to be the same address defined in the ADS1210 product data sheet because this address is primary used between the PC program and the microcontroller U4.

REGISTER	ADDRESS	DESCRIPTION
DOR2	00H	The ADS1210 data register byte 2 (MSByte)
DOR1	01H	The ADS1210 data register byte 1
DOR0	02H	The ADS1210 data register byte 0 (LSByte)
INSR	03H	The ADS1210 instruction register
CMR3	04H	The ADS1210 command register byte 3 (MSByte)
CMR2	05H	The ADS1210 command register byte 2
CMR1	06H	The ADS1210 command register byte 1
CMR0	07H	The ADS1210 command register byte 0 (LSByte)
OCR2	08H	The ADS1210 offset calibration register byte 2 (MSByte)
OCR1	09H	The ADS1210 offset calibration register byte 1
OCR0	0AH	The ADS1210 offset calibration register byte 0 (LSByte)
ADS	0BH	The device ID register in the microcontroller
FCR2	0CH	The ADS1210 full scale calibration register byte 2 (MSByte)
FCR1	0DH	The ADS1210 full scale calibration register byte 1
FCR0	0EH	The ADS1210 full scale calibration register byte 0 (LSByte)

TABLE II. The Address Definition.

All the information to be sent to the ADS1210 internal registers must be sent to the above address from the PC program.

Instruction

There are eight operations defined between the PC program and the microcontroller U4. These operations are coded. These codes are the instruction sent by the PC program to the device U4. The device U4 performs the operation requested by the PC program.

The instruction format is defined in the following.

c3	c2	c1	c0	a3	a2	a1	a0
d7	d6	d5	d4	d3	d2	d1	d0

The c3c2c1c0 is the instruction code. The a3a2a1a0 is the address defined in Table II. The d7d6d5d3d2d1d0 is the data to be sent from the PC program to the demo board.

Instruction: MCwrite51
Code: 0000_a3a2a1a0
d7d6d5d4d3d2d1d0
Bytes: 2
Operation: Write the one byte data (d7-d0) to the U4 into the address a3a2a1a0.

Instruction: MCretrieve
Code: 0010_xxxx
Bytes: 1
Operation: Start retrieving data from the 32K memory into the PC.

Instruction: MCwrite1210
Code: 0100_xxxx
Bytes: 1
Operation: Copy the data in the address a3a2a1a0 of the device U4 into the ADS1210 register.

Instruction: MCread1210
Code: 0110_xxxx
Bytes: 1
Operation: Read the ADS1210 register defined in the INSR into the address a3a2a1a0 in the microcontroller U4.

Instruction: MCreadback
Code: 1000_xxxx
Bytes: 1
Operation: Read back data from the microcontroller U4 with the address a3a2a1a0.

Instruction: MCconvert
Code: 1010_xxxx
Bytes: 1
Operation: Setup the ADS1210 in the data conversion mode.

Instruction: MCsync
Code: 1100_xxxx
Bytes: 1
Operation: Toggle the ADS1210 _SYNC pin or reset the ADS1210.

Instruction: MCpcget1
Code: 1110_xxxx
Bytes: 1
Operation: Finish the data retrieving from the demo board.

When the PC sends instruction to the ADS1210 demo board, the instruction is always sent with the MSB (Most Significant Bit) first. However, when the PC writes data to the demo board or reads data from the demo board, the data is always with the LSB (Least Significant Bit) first.

PC Programming

After understanding the data structure and the instruction discussed above, here comes a collect of core statements in Pascal language to access the ADS1210 demo board. The procedure MC51write is provided in the section of an example of PC programming. It performs the basic writing function to the demo board. The XAddr_REG is the address of the register REG. For example, the address of CMR3 is 04H from Table II. Therefore, XAddr_CMR3 = $04.

The procedure revbit(revin,revcode) performs the MSB first to LSB first conversion as the data has to be in the LSB first format. The data revin is converted to the LSB first data. The variable revcode carries the converted result back.

Writing four bytes Command Register, CMR=42200146H

```
{ Writing the CMR3 byte first }
MC51write(MCwrite51 OR XAddr_CMR3);  { Issue a writing instruction }
revbit($42,revcode);    { convert $42 to the LSB first format, revcode = $42 }
MC51write(revcode);  { write the CMR3 data into the demo board }

{ Writing the CMR2 byte }
MC51write(MCwrite51 OR XAddr_CMR2);  { Issue a writing instruction }
revbit($20,revcode);    { convert $20 to the LSB first format, revcode = $04 }
MC51write(revcode);  { write the CMR2 data into the demo board }

{ Writing the CMR1 byte }
MC51write(MCwrite51 OR XAddr_CMR1);  { Issue a writing instruction }
revbit($01,revcode);    { convert $01 to the LSB first format, revcode = $80 }
MC51write(revcode);  { write the CMR1 data into the demo board }

{ Writing the CMR0 byte }
MC51write(MCwrite51 OR XAddr_CMR0);  { Issue a writing instruction }
revbit($46,revcode);    { convert $46 to the LSB first format, revcode = $62 }
MC51write(revcode);  { write the CMR0 data into the demo board }

{ Writing the INSR byte }
MC51write(MCwrite51 OR XAddr_INSR);  { Issue a writing instruction }
MC51write($64); { INSR=64H }
```

After the above operation, the CMR data and the INSR data are sent to the device U4, but not to the ADS1210 internal Command Register yet. A MCwrite1210 instruction has to be sent. This instruction requests the microcontroller to send the data into the ADS1210 internal register.

```
{ Programming ADS1210 }
MC51write(MCwrite1210);  { Issue an instruction to programming ADS1210 }
```

Writing Offset/Full Scale Calibration Register

The writing sequence and the statements are very similar to what has been presented in Writing four bytes Command Register. Please refer to the procedure SetOCR and procedure SetFCR in the section of an example of PC programming.

Reading four bytes Command Register

```
{ Writing the INSR first }
MC51write((MCwrite51 OR XAddr_INSR)); { Issue a writing instruction }
MC51write($E4);  { Write INSR = E4H means reading four bytes CMR }
MC51write(MCread1210);    { Issue an instruction to reading ADS1210 }
delay(100);
ReadBack := True;           { Set ReadBack flag to be True }
ADSRBack;                   { Call the Readback procedure }
```

Reading Offset/Full Scale Calibration Register

The reading operations are very similar to Reading four bytes Command Register. Please refer to the procedure SetOCR and procedure SetFCR in the section of an example of PC programming.

Retrieving Data

If the PC program does not issue new instruction to the demo board, the demo board switches to the data acquisition mode automatically. The data converted from the ADS1210 is continuously collected by the demo board. The procedure retrieve performs the data retrieving operation from the demo board 32K memory to the PC.

To retrieve data, the following statements can be utilized.

```
GetTime(Hour, Minute, Second, Sec100);      { recording the present time }
OldReadTime := 3600*Hour + 60*Minute + Second + Sec100 div 100;
                    { Convert the time to seconds }
ConvertTime :=      0.01;       { give the ADS1210 conversion time in
                                  second }
                                { ADS1210 Data Rate = 100Hz }
Nt := 256;                  { Number of Data point }
Nbyte2x := 6;               { Six nibbles equal to three bytes }
ReadData(Nt, ConvertTime, OldReadTime, RdError);
                            { Call procedure ReadData to do data retrieving }
if(RdError = True) then     { RdError indicates a communication problem }
      writeln('Data Retrieval Error.');
```

Synchronize and Reset the ADS1210

The _SYNC pin of ADS1210 can be toggled to perform synchronization function. (Please refer to the ADS1210 product data sheet). Also, the ADS1210 can be reset using the serial data clock SCLK if the serial data clock is provided externally.

The following statements perform a synchronization operation if the serial data clock SCLK of the ADS1210 is provided by the ADS1210 itself. The statements perform a serial reset operation if the serial data clock SCLK is provided externally.

```
MC51write((MCwrite51 OR XAddr_ADS)); { Issue a writing instruction }
MC51write(ADSID);    { Write ADSID into the demo board. }
                     { ADSID=00H for ADS1210/11 }
                     { ADSID=01H for ADS1212/13 }
MC51write(MCsync);     { Issue an instruction to do sync or reset }
```

AN EXAMPLE OF PC PROGRAMMING

An example of PC programming to access the ADS1210 demo board is provided in this section. The const paragraph defines the code of the eight operations between the PC program and the demo board. The variable XAddr_XXX defines the address of the registers defined in Table II. The following is a list of the procedures and the functions.

procedure MC51write(MC51code : byte);

This procedure writes the code of MC51code into the demo board.

procedure revbit(revin : byte; var revcode : byte);

This procedure converts the input byte revin from the MSB first format to the LSB first format. The variable revcode returns the converted data.

procedure Renew;

This procedure performs the renew function that writes four byte CMR data with the corresponding INSR data , three bytes OCR data with the corresponding INSR data and the FCR data with the corresponding INSR data. Three reading back operations are followed to read back CMR, OCR and FCR.

procedure SetINSR;

This procedure sends the INSR data into the demo board.

procedure SetCMR;

This procedure writes the CMR data into the demo board. A reading back operation is followed after the writing.

procedure SetOCR;

This procedure writes the OCR data into the demo board. A reading back operation is followed after the writing

procedure SetFCR;

This procedure writes the FCR data into the demo board. A reading back operation is followed after the writing

procedure ADSsync;

This procedure performs the synchronization or serial reset function.

procedure ADSRback;

This procedure performs the data reading back function from the demo board to PC.

procedure Retrieve;

This procedure performs the data retrieving function from the 32K memory into the PC.

procedure ReadSetup(var SURdError : Boolean; ADSSetupCode : longint);

This procedure is called by the procedure ADSRback. It is the core for reading back the ADS1210 registers.

procedure ReadData(Nt : longint; ConvertTime : Real; var OldReadTime : Real; var DRdError : Boolean);
This procedure is called by the procedure retrieve. It is the core for reading back the data from the 32K memory.

procedure PCPort;

This procedure initializes the PC parallel port.

Below is an example of the PC program in Pascal.

```
program AP10PC;
uses  CRT, DOS;
const
  MCwrite51          = $00;
  MCretrieve         = $20;
  MCwrite1210        = $40;
  MCread1210         = $60;
  MCreadback         = $80;
  MCconvert          = $A0;
  MCsync             = $C0;
  MCpcgetl           = $E0;
var
  strbaddr           : Word;
  wraddr             : Word;
  rdaddr             : Word;
  pPrintPort         : ^word;
  RADSSetup          : Longint;
  INSRegCode         : Byte;
  CMRegCode          : Longint;
  OCRegCode          : Longint;
  FCRegCode          : Longint;
  XAddr_DOR2         : Byte;
  XAddr_DOR1         : Byte;
  XAddr_DOR0         : Byte;
  XAddr_INSR         : Byte;
  XAddr_CMR3         : Byte;
  XAddr_CMR2         : Byte;
  XAddr_CMR1         : Byte;
  XAddr_CMR0         : Byte;
  XAddr_OCR2         : Byte;
  XAddr_OCR1         : Byte;
  XAddr_OCR0         : Byte;
  XAddr_FCR2         : Byte;
  XAddr_FCR1         : Byte;
  XAddr_FCR0         : Byte;
  XAddr_ADS          : Byte;
  ADSID              : Byte;
  XAddr_READADD0     : Byte;
  XAddr_READADD1     : Byte;
  XAddr_READADD2     : Byte;
  Nbyte2x            : Byte;
  INSRaddr           : Byte;
  Readback           : Boolean;

procedure ADSRback; forward;
procedure ReadSetup(var SURdError : Boolean; ADSSetupCode : longint);
          forward;
procedure ReadData(Nt : longint; ConvertTime : Real;
          var OldReadTime : Real; var DRdError : Boolean); forward;
procedure MC51write(MC51code : byte);
var
      busy           : Byte;
      K              : Word;
begin
      { busy=H in board, busy7=L in code }
      K :=0;
  busy := PORT[rdaddr] AND $80;
  while (busy <> $80) AND (K<=80000) do
      begin
      busy := PORT[rdaddr] AND $80;
      inc(K);
      end;
  PORT[wraddr] := MC51code;
  delay(1);
  PORT[strbaddr] := 1;
  delay(0);
  PORT[strbaddr] := 0;
  delay(1);
  if K >= 80000 then
          begin
          sound(1000);
          delay(200);
          nosound;
          writeln;
          writeln('Program halted as System Setup Errors detected.');
          writeln('Possible causes include: Power supply off, Bad cables,
          Program halted as System Setup Errors detected.');
          writeln('Wrong PC Parallel Port, ADS1210 not installed.');
          sound(1000);
          delay(300);
          nosound;
          halt;
          end;
end;
procedure revbit(revin : byte; var revcode : byte);
var
      i    : word;
      j    : word;
```

```
begin
  revcode := 0;
  j := 1;
  for i := 0 to 7 do
        begin
        revcode := revcode SHL 1;
        if revin AND j > 0 then
        revcode := revcode + 1;
        j := j SHL 1;
        end;
end;
procedure Renew;
type
        NewCType    = array[1..3] of Byte;
var
      CMRbyte       : Byte;
      OCRbyte       : Byte;
      FCRbyte       : Byte;
      revcode       : Byte;
      CC            : Byte;
      NewCode       : NewCType;
begin
        {Write CMR byte 3}
        CMRbyte := ( CMRegCode SHR 24 ) AND $FF;
        MC51write((MCwrite51 OR XAddr_CMR3));
        revbit(CMRbyte,revcode);
        MC51write(revcode);
        {Write CMR byte 2}
        CMRbyte := ( CMRegCode SHR 16 ) AND $FF;
        MC51write((MCwrite51   OR XAddr_CMR2));
        revbit(CMRbyte,revcode);
        MC51write(revcode);
        {Write CMR byte 1}
        CMRbyte := ( CMRegCode SHR 8 ) AND $FF;
        MC51write((MCwrite51 OR XAddr_CMR1));
        revbit(CMRbyte,revcode);
        MC51write(revcode);
        {Write CMR byte 0}
        CMRbyte := CMRegCode AND $FF;
        MC51write((MCwrite51 OR XAddr_CMR0));
        revbit(CMRbyte,revcode);
        MC51write(revcode);
        {Write INSR}
        delay(100);
        MC51write((MCwrite51 OR XAddr_INSR));
        MC51write($64);
        {Programming ADS1210}
        MC51write(MCwrite1210);
        {Write OCR}
        delay(100);
        OCRbyte := ( OCRegCode SHR 16 ) AND $FF;
        MC51write((MCwrite51 OR XAddr_OCR2));
        revbit(OCRbyte,revcode);
        MC51write(revcode);
        OCRbyte := ( OCRegCode SHR 8 ) AND $FF;
        MC51write((MCwrite51 OR XAddr_OCR1));
        revbit(OCRbyte,revcode);
        MC51write(revcode);
        OCRbyte := OCRegCode AND $FF;
        MC51write((MCwrite51 OR XAddr_OCR0));
        revbit(OCRbyte,revcode);
        MC51write(revcode);
        delay(100);
        MC51write((MCwrite51 OR XAddr_INSR));
        MC51write($48);
        MC51write(MCwrite1210);
        {Write FCR}
        delay(100);
        FCRbyte := ( FCRegCode SHR 16 ) AND $FF;
        MC51write((MCwrite51 OR XAddr_FCR2));
        revbit(FCRbyte,revcode);
        MC51write(revcode);
        FCRbyte := ( FCRegCode SHR 8 ) AND $FF;
        MC51write((MCwrite51 OR XAddr_FCR1));
        revbit(FCRbyte,revcode);
        MC51write(revcode);
        FCRbyte := FCRegCode AND $FF;
        MC51write((MCwrite51 OR XAddr_FCR0));
        revbit(FCRbyte,revcode);
        MC51write(revcode);
        delay(100);
        MC51write((MCwrite51 OR XAddr_INSR));
        MC51write($4C);
        MC51write(MCwrite1210);
        {Read back ADS1210}
        NewCode[1] := $E4;
        NewCode[2] := $C8;
        NewCode[3] := $CC;
        for CC := 1 to 3 do
                begin
                delay(100);
                INSRegCode := NewCode[CC];
        Nbyte2x := (INSRegCode AND $60) SHR 5;
        Nbyte2x := Nbyte2x + 1;
        Nbyte2x := Nbyte2x SHL 1;
        INSRaddr := INSRegCode AND $1F;
        MC51write((MCwrite51 OR XAddr_INSR));
        MC51write(INSRegCode);
        MC51write(MCread1210);
                delay(100);
                ReadBack := True;
                ADSRBack;
                end;
end;
procedure SetINSR;
var
        I               : Byte;
begin
  Nbyte2x := (INSRegCode AND $60) SHR 5;
  Nbyte2x := Nbyte2x + 1;
  Nbyte2x := Nbyte2x SHL 1;
INSRaddr := INSRegCode AND $1F;
MC51write((MCwrite51 OR XAddr_INSR));
MC51write(INSRegCode);
  I := INSRegCode AND $80;
  case I of
        $00:  begin
if ((INSRaddr = $00) OR (INSRaddr = $01) OR (INSRaddr = $02)) then
writeln('Don''t write data register. ')
                                else
                                        MC51write(MCwrite1210);
                                end;
        $80:  begin
                                MC51write(MCread1210);
                                ReadBack := True;
                                ADSRBack;
                                end;
  end;
end;
procedure SetCMR;
var
  CMRbyte     : Byte;
  revcode     : Byte;
begin
                        {CMR Byte 3}
CMRbyte := ( CMRegCode SHR 24 ) AND $FF;
MC51write((MCwrite51 OR XAddr_CMR3));
revbit(CMRbyte,revcode);
MC51write(revcode);
                        {CMR Byte 2}
CMRbyte := ( CMRegCode SHR 16 ) AND $FF;
MC51write((MCwrite51 OR XAddr_CMR2));
revbit(CMRbyte,revcode);
MC51write(revcode);
                        {CMR Byte 1}
CMRbyte := ( CMRegCode SHR 8 ) AND $FF;
MC51write((MCwrite51 OR XAddr_CMR1));
revbit(CMRbyte,revcode);
MC51write(revcode);
                {CMR Byte 0}
CMRbyte := CMRegCode AND $FF;
MC51write((MCwrite51 OR XAddr_CMR0));
revbit(CMRbyte,revcode);
MC51write(revcode);
```

```
            { Setup for reading back }
            INSRegCode := $64;
Nbyte2x := (INSRegCode AND $60) SHR 5;
Nbyte2x := Nbyte2x + 1;
      Nbyte2x := Nbyte2x SHL 1;
INSRaddr := INSRegCode AND $1F;
Delay(10);
MC51write((MCwrite51 OR XAddr_INSR));
MC51write(INSRegCode);
]MC51write(MCwrite1210);
      delay(100);
INSRegCode := $E4;
MC51write((MCwrite51 OR XAddr_INSR));
MC51write(INSRegCode);
MC51write(MCread1210);
      delay(100);
      ReadBack := True;
      ADSRback;
end;
procedure SetOCR;
var
  OCRByte     : Byte;
  revcode     : Byte;
begin
      {OCR Byte 3}
OCRbyte := ( OCRegCode SHR 16 ) AND $FF;
MC51write((MCwrite51 OR XAddr_OCR2));
revbit(OCRbyte,revcode);
MC51write(revcode);
      {OCR byte 2}
OCRbyte := ( OCRegCode SHR 8 ) AND $FF;
MC51write((MCwrite51 OR XAddr_OCR1));
revbit(OCRbyte,revcode);
MC51write(revcode);
      {OCR byte 0}
OCRbyte := OCRegCode AND $FF;
MC51write((MCwrite51 OR XAddr_OCR0));
revbit(OCRbyte,revcode);
MC51write(revcode);
      {Setup for reading back}
      INSRegCode := $48;
Nbyte2x := (INSRegCode AND $60) SHR 5;
Nbyte2x := Nbyte2x + 1;
      Nbyte2x := Nbyte2x SHL 1;
INSRaddr := INSRegCode AND $1F;
Delay(10);
MC51write((MCwrite51 OR XAddr_INSR));
MC51write(INSRegCode);
MC51write(MCwrite1210);
      delay(100);
      INSRegCode := $C8;
MC51write((MCwrite51 OR XAddr_INSR));
MC51write(INSRegCode);
MC51write(MCread1210);
      delay(100);
      ReadBack := True;
      ADSRBack;
end;
procedure SetFCR;
var
  FCRbyte     : Byte;
  revcode     : Byte;
begin
      {FCR Byte 2}
FCRbyte := ( FCRegCode SHR 16 ) AND $FF;
MC51write((MCwrite51 OR XAddr_FCR2));
revbit(FCRbyte,revcode);
MC51write(revcode);
      {FCR byte 1}
FCRbyte := ( FCRegCode SHR 8 ) AND $FF;
MC51write((MCwrite51 OR XAddr_FCR1));
revbit(FCRbyte,revcode);
MC51write(revcode);
      {FCR byte 0}
FCRbyte := FCRegCode AND $FF;
MC51write((MCwrite51 OR XAddr_FCR0)); r
revbit(FCRbyte,revcode);
MC51write(revcode);
      {Setup for reading back}
      INSRegCode := $4C;
Nbyte2x := (INSRegCode AND $60) SHR 5;
Nbyte2x := Nbyte2x + 1;
      Nbyte2x := Nbyte2x SHL 1;
INSRaddr := INSRegCode AND $1F;
Delay(10);
MC51write((MCwrite51 OR XAddr_INSR));
MC51write(INSRegCode);
MC51write(MCwrite1210);
      delay(100);
INSRegCode :=$CC;
MC51write((MCwrite51 OR XAddr_INSR));
MC51write(INSRegCode);
MC51write(MCread1210);
      delay(100);
      ReadBack := True;
      ADSRBack;
end;
procedure ADSsync;
begin
MC51write((MCwrite51 OR XAddr_ADS));
MC51write(ADSID);
MC51write(MCsync);
end;
procedure ADSRback;
var
  NRbackBit           : Byte;
  Dbyte3              : Byte;
  Dbyte2              : Byte;
  Dbyte1              : Byte;
  Dbyte0              : Byte;
  coderev             : Byte;
  RdError             : Boolean;
  codetype            : string[4];
begin
  If (ReadBack = True) then
      begin
case INSRaddr of
$04,$05,$06,$07:      begin
      codetype := 'CMR';
      ReadSetup(RdError,CMRegCode);
      end;
$08,$09,$0A:                  begin
      codetype := 'OCR';
      ReadSetup(RdError,OCRegCode);
      end;
$0C,$0D,$0E:          begin
            codetype := 'FCR';
            ReadSetup(RdError,FCRegCode);
            end;
            else
                  RdError := True;
        end;
      end;

 if (RdError = True) then
    writeln('Setup Readback Error.')
 else
     writeln('The ', codetype, ' readback is  ', RADSSetup);
end;
procedure Retrieve;
var
  Hour                : Word;
  Minute              : Word;
  Second              : Word;
  Sec100              : Word;
  Nt                  : longint;
  RdError             : Boolean;
  OldReadTime         : Real;
  ConvertTime         : Real;
begin
  GetTime(Hour, Minute, Second, Sec100);
  OldReadTime := 3600*Hour + 60*Minute + Second + Sec100 div 100;
  ConvertTime :=      0.01;  {ADS1210 Data Rate = 100Hz}
  Nt := 16; { Number of Data point }
```

```
  Nbyte2x := 6;
  ReadData(Nt, ConvertTime, OldReadTime, RdError);
  if(RdError = True) then
      writeln('Data Retrieval Error.');
end;
{Procedure ReadSetup reads back ADS1210 data }
procedure ReadSetup(var SURdError : Boolean; ADSSetupCode : longint);
var
      FNum          : Longint;
      readbkloop    : Longint;
      BusyBit       : Byte;   { data ready bit }
      nib           : Byte;   { nibble from MC51 }
      iRd           : Word;
      K             : Word;
      coderev       : Byte;
begin
  SURdError := False;
  Fnum := $FFFFFFFF;
  if SURdError = False then
          begin
          readbkloop := 0;
WHILE ((FNUM AND $FFFFFFFF) <> (ADSSetupCode AND $FFFFFFFF))
AND (readbkloop < 1) DO
            BEGIN
            readbkloop := readbkloop + 1;
            Fnum := 0;
            for iRd := 1 to NByte2x do
                    begin
                    K :=0;
                    BusyBit := PORT[rdaddr] AND $80;
                    while (BusyBit <> $80) AND (K <= 200000)  do
                            begin
                            BusyBit := PORT[rdaddr] AND $80;
                            inc(K);
                            if (K mod 40) = 0 then delay(1);
                            if K = 200000 then SURdError := True;
                            end;
                    PORT[wraddr] := $80;
                    delay(1);
                    PORT[strbaddr] := 1;
                    delay(0);
                    PORT[strbaddr] := 0;
                    delay(1);
nib := (PORT[rdaddr] AND $78) SHL 1;
revbit(nib,coderev);
                    Fnum := Fnum SHL 4;
                    FNum := Fnum OR coderev;
                    end; { end of NByte2x nibbles}
            RADSSetup := FNum;
            END;
          end;
  { end read setup code }
  K :=0;
  BusyBit := PORT[rdaddr] AND $80;
  While (BusyBit <> $80) AND (K <= 200000) do
          begin
          BusyBit := PORT[rdaddr] AND $80;
          inc(K);
          If (K mod 80) = 0 then delay(1);
          if K = 200000 then SURdError := True;
          end;
  PORT[wraddr] := $E0;
  delay(1);
  PORT[strbaddr] := 1;
  delay(0);
  PORT[strbaddr] := 0;
  delay(1);
end;
procedure ReadData(Nt : longint; ConvertTime : Real; var OldReadTime : Real;
      var DRdError : Boolean);
{ Needs 32 bit CMR code from main program. This procedure checks data
valid
until ready, read the word 4 bits at a time, MSB first, reassemble it,
determines data
format,(unipolar 2's complement MSByte), returns a final 24bit number
(Fnum) in
the proper format}
var
  FNum          : Longint;
  readbkloop    : Longint;
  J             : Longint;
  BusyBit       : Byte;   { data ready bit }
  nib           : Byte;   { nibble from MC51 }
  Hour          : Word;
  Minute        : Word;
  Second        : Word;
  Sec100        : Word;
  RdError       : Boolean;
  iRd           : Word;
  K             : Word;
  normdata      : Real;
  FScale        : Real;
  NewReadTime   : Real;
  DelTime       : Real;
  DataTime      : Real;
  coderev       : Byte;
  Dbyte2        : Byte;
  Dbyte1        : Byte;
  Dbyte0        : Byte;
begin
  DataTime := (Nt+100) * ConvertTime ;
  DelTime := 0;
  while DelTime < DataTime do
      begin
      GetTime(Hour, Minute, Second, Sec100);
      NewReadTime := 3600*Hour + 60*Minute + Second + Sec100/100;
      DelTime := NewReadTime - OldReadTime;
if KeyPressed AND (ReadKey = chr(27)) then DelTime := DataTime;
end;
  FScale := 16777216;
  {Send MCODE=MCretrive to evaluation board}
  DRdError :=  False;
  K := 0;
  Busybit := PORT[rdaddr] AND $80;
  while (Busybit <> $80) AND (K <= 200000)  do
      begin
      Busybit := PORT[rdaddr] AND $80;
      inc(K);
      if (K mod 80) = 0 then delay(1);
      if K = 200000 then DRdError := True;
      end;
  PORT[wraddr] := $20;
  delay(1);
  PORT[strbaddr] := 1;
  delay(0);
  PORT[strbaddr] := 0;
  delay(1);
  {Wait for evaluation board data valid}
  K := 0;
  BusyBit := PORT[rdaddr] AND $80;
      While (BusyBit <> $80) AND (K <= 200000) do
      begin
      BusyBit := PORT[rdaddr] AND $80;
      inc(K);
      if (K mod 80) = 0 then delay(1);
      if K = 200000 then DRdError := True;
      end;
  if DRdError = False then
      begin
      for J := Nt-1 downto 0 do
          begin
          Fnum := 0;
            for iRd := 1 to NByte2x do
            begin
            PORT[strbaddr] := $02;
            delay(0);
            PORT[strbaddr] := 0;
            delay(1);
nib := (PORT[rdaddr] AND $78) SHL 1;
revbit(nib,coderev);
              Fnum := Fnum SHL 4;
              FNum := Fnum OR coderev;
              end;
```

```
{  In the case of MSByte first, MSB first and 2's complement }
            if (Fnum AND $800000) > 1 then
                        Fnum := Fnum OR $FF000000
            else
                        Fnum := Fnum AND $7FFFFF;
            normdata :=  Fnum / FScale;
writeln('The Data [',J,'] = ', 10*normdata, '  Volts');
end;
      end;
  K :=0;
  BusyBit := PORT[rdaddr] AND $80;
      While (BusyBit <> $80) AND (K <= 200000) do
      begin
      BusyBit := PORT[rdaddr] AND $80;
      inc(K);
      if (K mod 80) = 0 then delay(1);
      if K = 200000 then DRdError := True;
      end;
  PORT[wraddr] := $E0;
  delay(1);
  PORT[strbaddr] := 1;
  delay(0);
  PORT[strbaddr] := 0;
  delay(1);
  GetTime(Hour, Minute, Second, Sec100);
  OldReadTime := 3600*Hour + 60*Minute + Second + Sec100/100;
end;
procedure PCPort;
begin
  {Get LPT1 or LPT2 Address from BIOS Memory}
  {LPT1}
  pPrintPort := Ptr($40,$08);
  {LPT2}
  {pPrintPort := Ptr($40,$0A);}
  wraddr := pPrintPort^;
  rdaddr := wraddr + 1;
  strbaddr := wraddr + 2;
end;
{********     Main Program    ********}

begin
{  ADS1210 Configuration }
    {INSR = 01100100}
    INSRegCode := $64;
    {CMR = 01000010_00100000_00000001_01000110}
      CMRegCode := $42200146;
{OCR = 00000000_00000000_00000000}
OCRegCode := $000000;
{FCR = 10100011_01110101_10010100}
FCRegCode := $A37594;

{Address Definition}
  XAddr_DOR2 := $00;
  XAddr_DOR1 := $01;
  XAddr_DOR0 := $02;
  XAddr_INSR  := $03;
  XAddr_CMR3 := $04;
  XAddr_CMR2 := $05;
  XAddr_CMR1 := $06;
  XAddr_CMR0 := $07;
  XAddr_OCR2 := $08;
  XAddr_OCR1 := $09;
  XAddr_OCR0 := $0A;
  XAddr_FCR2  := $0C;
  XAddr_FCR1  := $0D;
  XAddr_FCR0  := $0E;
  XAddr_ADS   := $0B;

{ADSID = $00 for ADS1210/11 and $01 for ADS1212/13 }
     ADSID      := $00;

  Nbyte2x    := 6;
  ReadBack   := False;

{Run procedures}
  writeln('AP10PC');
  PCPort;
  delay(1000);
  Renew;
  delay(1000);
  Retrieve;
end.
```

SUMMARY

This application note explains the hardware design and the software design of the demo board. The hardware design discusses the system block diagram and the circuit diagram. The software design discusses the data structure and the core procedures for the PC programming. An example of PC Pascal program is provided to illustrate how to access the ADS1210 demo board. With the understanding of the demo board design, a new PC program can be implemented to do the data acquisition using the ADS1210 demo board.