The DDC112’s Test Mode

By Jim Todsen

This application bulletin covers the DDC112’s test mode. It elaborates on the explanation given in the data sheet and is written with the purpose of helping you use the test mode to its fullest capability. It does assume a basic understanding of the DDC112’s operation. For a good introduction to the DDC112, see the DDC112’s data sheet.

The organization of this application bulletin is as follows: First, an overview presents the basic operation of the test mode. Next are explanations of single packet and multiple packet test signals. Noise and linearity performance of the test mode are then covered followed by some special considerations for using the test mode. Finally, a program for investigating test mode performance is discussed with the listing for the program given afterwards.

OVERVIEW

During normal operation, a sensor connected to each DDC112 input supplies signal current which is measured by DDC112 and converted to a 20-bit digital word. To help with debug during system development and to provide a good system-level diagnostic check, the DDC112 has a test mode which is enabled by the TEST pin. While in test mode, the inputs are disconnected and a test signal is measured instead. Figure 1 shows a simplified block diagram of the front end of the DDC112 including test mode circuitry.

You enter test mode by holding TEST HIGH while CONV toggles. Likewise, you exit test mode by holding TEST LOW while CONV toggles. The integration timing control for test mode is identical to that of normal operation: CONV still sets the side (A or B) and length of integration. As in normal operation, there is a continuous and non-continuous mode in test mode determined by the length of TINT (see Application Bulletin AB-131 for more information on the continuous and non-continuous modes).

C_TEST in Figure 1 supplies the test signal to the front-end integrators. Switches S_A and S_B, which in normal operation steer the sensor’s current to the A or B integrator, are opened and S_AT, S_BT and S_VT are used instead (these switches are only used in test mode). C_TEST, previously charged to V_REF by S_VT, is dumped to the appropriate side by either S_AT or S_BT depending on CONV. After the dumped charge from C_TEST is integrated by C_INT, S_AT (or S_BT) opens and C_TEST recharges to V_REF via S_VT. C_TEST is now ready for another charge dump. When the integration period is over, the ADC measures the signal integrated onto C_INT (the total test signal dumped by C_TEST) just as in normal operation. S_GND shorts the DDC112’s inputs to ground during test mode to prevent any current from the sensor building a charge on the DDC’s inputs.

C_TEST can be dumped once or multiple times onto the integrator during an integration period. The next two sections explain the timing necessary to control the number of dumps. Note that the magnitude of the test signal is not a direct function of the integration time T_INT, but rather the number of charge packet dumps. If the number of charge packets dumped remains constant, changing T_INT will not affect the test mode data.

FIGURE 1. Simplified Block Diagram of the DDC112’s Front End.
SINGLE PACKET

When TEST is held HIGH, a single packet of charge from \( C_{\text{TEST}} \) is dumped onto the integrator at the beginning of every test mode integration. Figure 2 shows the timing diagram and specifications for this case. The Integrator Status trace highlights the fact that the test mode functions the same as normal operation with respect to integration, measurement, auto-zero, reset, etc. The bottom trace of Figure 2 shows the output voltage on the side A integrator. CONV triggers the sequence that controls \( S_{\text{AT}}, S_{\text{BT}}, \) and \( S_{\text{VT}} \). A short time after CONV toggles, \( S_{\text{VT}} \) opens and switch \( S_{\text{AT}} \) (or \( S_{\text{BT}} \)) dumps \( C_{\text{TEST}} \)’s charge onto the appropriate integrator side. After the charge transfers from \( C_{\text{TEST}} \) to \( C_{\text{INT}} \), switch \( S_{\text{AT}} \) (or \( S_{\text{BT}} \)) opens and \( C_{\text{TEST}} \) recharges to \( V_{\text{REF}} \) using \( S_{\text{VT}} \). The test circuitry then waits until the next integration begins before dumping \( C_{\text{TEST}} \) to the other side. This cycle continues until TEST is held LOW while CONV toggles. At that point, the test mode is over and normal operation resumes.

The size of each charge packet is \((V_{\text{REF}})(C_{\text{TEST}})\), approximately 13pC. If, for example, the range is set to 250pC, a single packet test signal equals approximately 5.2% of full scale. The absolute value of the charge packet can vary by 20% or more due to fabrication process variations. As previously mentioned, the readings during single packet test mode are independent of \( T_{\text{INT}} \) since the test mode supplies a fixed amount of charge (≈13pC) each integration.

The A and B sides of a channel use the same \( C_{\text{TEST}} \). Due to the nature of the switching arrangement, there is a small imbalance in the charge injection between sides A and B during test mode. This imbalance results in slightly different effective sizes for the side A and B test packets. Typical mismatch between side A and B charge packets is ≈0.2pC.

### SYMBOL DESCRIPTION MIN TYP MAX UNITS

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>DESCRIPTION</th>
<th>MIN</th>
<th>TYP</th>
<th>MAX</th>
<th>UNITS</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_{1} )</td>
<td>Setup Time for Test Mode Enable</td>
<td></td>
<td>100</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>( t_{2} )</td>
<td>Setup Time for Test Mode Disable</td>
<td></td>
<td>100</td>
<td></td>
<td>ns</td>
</tr>
</tbody>
</table>


The information provided herein is believed to be reliable; however, BURR-BROWN assumes no responsibility for inaccuracies or omissions. BURR-BROWN assumes no responsibility for the use of this information, and all use of such information shall be entirely at the user’s own risk. Prices and specifications are subject to change without notice. No patent rights or licenses to any of the circuits described herein are implied or granted to any third party. BURR-BROWN does not authorize or warrant any BURR-BROWN product for use in life support devices and/or systems.
MULTIPLE PACKET

As described above, a packet of charge is dumped onto the integrator at the beginning of every integration during the test mode. Additional packets can be dumped creating a larger test signal. Multiple packets are dumped onto the integrator by toggling TEST during the integration as illustrated in Figure 3. The rising edge of TEST during a test mode integration triggers the dumping of an additional packet. Multiple toggles on TEST result in multiple packets being dumped. In order to stay in test mode, TEST must be HIGH when CONV toggles at the end of the integration.

Figure 3 shows the relevant timing specifications for multiple packet dumps. The specifications limit the maximum number of multiple packet dumps for a given \( T_{INT} \). Since the rising edges of TEST must be at least 5.4µs apart, the maximum number of dumps is \( \frac{T_{INT}}{5.4 \mu s} \) (round down when this number is a non-integer). This limitation is not a concern during the continuous mode. Even for \( T_{INT} = 500 \mu s \), over 90 dumps are allowed which corresponds to more than 1200pC of test signal—easily enough to reach full scale on any range. It is only for the short \( T_{INT} \)'s of the noncontinuous mode where you need to watch the limit.

NOISE

The noise of the test mode signal is very low. In fact, the noise performance in the test mode is usually better than what can be achieved in normal operation, even with very low sensor capacitance. Since the DDC112’s inputs are disconnected during test mode, any coupling or interference on them (e.g., 60Hz pickup) will not affect the readings. This feature can be helpful in isolating the source of interference during system evaluation. Table I lists typical noise performance for the different internal ranges while in test mode with a single packet dump. The noise increases slightly for multiple packet dumps, particularly at the lower ranges.

<table>
<thead>
<tr>
<th>RANGE (Full-Scale Charge)</th>
<th>TYPICAL NOISE (rms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (50pC)</td>
<td>4.7ppm</td>
</tr>
<tr>
<td>2 (100pC)</td>
<td>3.6ppm</td>
</tr>
<tr>
<td>3 (150pC)</td>
<td>3.2ppm</td>
</tr>
<tr>
<td>4 (200pC)</td>
<td>3.1ppm</td>
</tr>
<tr>
<td>5 (250pC)</td>
<td>3.0ppm</td>
</tr>
<tr>
<td>6 (300pC)</td>
<td>3.0ppm</td>
</tr>
<tr>
<td>7 (350pC)</td>
<td>2.9ppm</td>
</tr>
</tbody>
</table>

TABLE I. Typical Noise Performance in Test Mode.

---

LINEARITY

As with the noise performance, the linearity of the test signal is very good. Measurements made in the test mode show that the linearity of the test mode is usually limited by the performance of the DDC112 itself. That is, the linearity of the test mode is typically only slightly worse than the linearity measured in normal operation with a linear input signal. Figure 4 shows a comparison of integral non-linearity (INL) for normal operation of the DDC112 with a very linear input signal and for the test mode. INL is defined here as the deviation of the data from a line connected between the data’s end points. Notice in Figure 4 how the test mode INL is only slightly worse than that of the DDC112 in normal operation. This indicates that the test mode signal is very linear and the linearity of the test mode is limited by the DDC112’s inherent linearity.

FIGURE 4. Normal Mode and Test Mode INL (end-point line fit).

SPECIAL CONSIDERATIONS

For the very best performance in test mode, T_INT might need to be an integer multiple of 3 CLK periods, depending on your PC board layout. For example, if CLK = 10MHz, T_INT might need to be a integer multiple of 300ns. If T_INT is not an integer multiple of 3 CLK periods, there can be excessive noise in the test mode. The reason for this is explained in Application Bulletin, AB-131. Basically, the internal slow clock (discussed in AB-131) is always left running while in test mode, even when T_INT > 4794 CLK periods.

To prevent requiring T_INT to be an integer multiple of 3 CLK periods, pay close attention to the PC board layout. Place the bypass capacitors for V_REF and the power supplies as close to the pins as possible. Use a good analog ground plane with the analog pins on the DDC112 connecting directly to it. See the Evaluation Fixture layout for an example. In this layout, the test mode noise shows no need for T_INT to be an integer multiple of 3 CLK periods. That is, the same noise performance is achieved with T_INT = 500µs, 501µs, 502µs, etc.

EXAMPLE PROGRAM

To help investigate the test mode, a Pascal program similar to the one in Application Bulletin, AB-125 is given in Listing 1. This program runs under DOS and controls the Evaluation Fixture hardware. Using the program, you can change the number of data points averaged per reading, full-scale range, T_INT, and the number of test mode packets. With these variables set, you can then retrieve a reading. The displayed average for the retrieved reading is normalized to full scale = 1.0 and the rms noise is given in ppm.

A routine to measure the linearity of the test mode signal is also included in the program. The program determines the size of each packet by making an initial measurement with a single packet test signal. It then calculates the number of packets needed to reach full scale. A loop increments the packets up to this number while taking data at each step. Afterwards, an end-point line is connected between the first point and the final point. The program stores the linearity data in a file and reports the maximum difference between the data points and the end-point line as the INL.
Listing 1

*******************************************************************************
(*
(*                      BURR-BROWN CORPORATION *)
(*
(*                    Application Bulletin AB-135 *)
(*
*******************************************************************************
(*
(*                      DDC112 Test Mode Program *)
(*
(* Written in Turbo Pascal 7.0 *)
(*
(* This program is designed to illustrate use of the DDC112's test mode. This program requires the DDC112 to be connected to the Evaluation Fixture DEM-DDC112U-C. See AB-125 for more info on the Evaluation Fixture. *)
(*
(* The user can program the # of averages/reading, range, tint and # of test packets from the menu. A single reading can be retrieved or a complete measurement of the DDC112's linearity can be taken. The results of the linearity measurement are saved to a file. *)
(*
(* Created by Jim Todsen 5/20/98 *)
(*
*******************************************************************************
(*{$R-,S-} *)
{$A+,B-,E+,F+,G-,I+,L-,N+,O+,R+,S+,V-,X+}

program DDC112TestMode;
uses  Dos,CRT,Strings;

var
  Range : longint;
  Tint : real;
  Nt : longint;
  Tpck : longint;
  PCPort : longint;
  pPrintPort : ^word;
  readdr, wraddr, strbaddr : word;
  DataCkCode : word;
  SysCkCode : word;
  Ctrl1Code, Ctrl2Code : word;
  DXmitDelayCode : word;
  UserDDCCode : word;
  Yave, Yrms : Real4by1;

procedure badBeep;
begin
  sound(100);
  delay(200);
  nosound;
end;

procedure goodBeep;
begin
  sound(300);
  delay(50);
  nosound;
end;
procedure xilinxWrite(XilinxAddr, XilinxData : Byte);
{Write data to a specified address on the DDC112 Evaluation Fixture}
begin
  PORT[wraddr] := $7F AND XilinxAddr;
  Delay(2);
  PORT[strbaddr] := 1;
  Delay(1);
  PORT[strbaddr] := 0;
  Delay(1);
  PORT[wraddr] := $80 OR XilinxData;
  Delay(2);
  PORT[strbaddr] := 1;
  Delay(1);
  PORT[strbaddr] := 0;
  Delay(1);
end;

procedure setRange(newRange: word);
{Set the DDC112 range}
begin
  writeln(' setting Range = ',newRange);
  Ctrl1Code := Ctrl1Code AND $78;
  Ctrl1Code := Ctrl1Code OR newRange;
  xilinxWrite(6,Ctrl1Code);
  delay(1000);  {wait for settling after change}
end;

procedure setTint(var newTint: real) ;
{Set the DDC112 Integration Time}
var
  IntCountCode: longint;
begin
  if newTint > 1500000 then newTint:= 1500000;
  IntCountCode := round(newTint*10)-1;            {10 MHz CLK}
  writeln(' setting Tint = ',newTint:7:1);
  xilinxWrite(7,(Ctrl2Code AND $7E));             {hold conv}
  Delay(1000);       {wait to insure CONV is stopped}
  xilinxWrite(5,(IntCountCode AND $7F));
  xilinxWrite(4,((IntCountCode SHR 7)  AND $7F));
  xilinxWrite(3,((IntCountCode SHR 14) AND $7F));
  xilinxWrite(2,((IntCountCode SHR 21) AND $07));
  xilinxWrite(7,(Ctrl2Code OR $1)); {release conv}
end;

procedure setTstPckts(NumPckt: longint);
{Set the # of Test Packets dumped to the DDC112 each integration}
begin
  if NumPckt > 0 then
    begin
      Ctrl1Code := Ctrl1Code OR $8; {Test ON}
      xilinxWrite(6,Ctrl1Code);
      if NumPckt = 1 then xilinxWrite(11,127) {all 1’s = 1 packet}
      else xilinxWrite(11,(NumPckt-2));
    end
  else
    begin
      Ctrl1Code:= Ctrl1Code AND $F7; {Test OFF}
      xilinxWrite(6, (Ctrl1Code));
    end;
  Delay(1000); {settling time after change}
end;
procedure setPCport(PCport: word);
{*******************************************************************************}
{Set the address of the PC's parallel port used for the Evaluation Fixture}
begin
  if PCPort = 1 then pPrintPort := Ptr($40,$08)
  else pPrintPort := Ptr($40,$0A);
  wraddr:= pPrintPort^;
  readdr:= wraddr + 1;
  strbaddr:= wraddr + 2;
  PORT[strbaddr]:= 0;  {init Strobe at 0 (really, 1 out)}
end;
{*******************************************************************************}
procedure xilinxRefresh;
{*******************************************************************************}
{Refresh the Xilinx FPGAs on the Evaluation Fixture}
begin
  PORT[strbaddr] := 0;    {init port strobe high}
  Delay(1);
  xilinxWrite(0, DataCkCode);
  xilinxWrite(1, SysCkCode);
  xilinxWrite(6, Ctrl1Code);
  xilinxWrite(7, Ctrl2Code);
  xilinxWrite(8, ((DXmitDelayCode AND $180) SHR 7));
  xilinxWrite(9, ((DXmitDelayCode AND $7F)));
  xilinxWrite(10, (UserDDCCode SHL 1));
  setRange(Range);
  setTint(Tint);
  setTstPckts(TPck);
  goodBeep;
end;
{*******************************************************************************}
procedure getInt(descrip: string; Min, Max: longint;
var Value: longint; var Err: boolean);
{*******************************************************************************}
{Get an INTEGER value from the user}
var
  userinput : string[25];
  check : integer;
  temp : longint;
begin
  Err:= true;
  ClrScr;
  writeln(' The current ',descrip,' is: ',Value);
  writeln;
  writeln('  Enter the new ',descrip,':
  readln(userinput);
  if userinput = 'x' then exit;
  if userinput = 'X' then exit;
  if userinput = '' then exit;
  val(userinput,temp,check);
  if check <> 0 then
    begin
      badBeep;
      writeln(' *** Invalid ',descrip);
      writeln;
      delay(1000);
      exit;
    end
  else
    begin
      Value := temp;
      Err := false;
    end;
end;
procedure getReal(descrip: string; Min, Max: longint; 
    var Value: real; var Err: boolean);
{Get a REAL value from the user}
var 
    userinput : string[25];
    check : integer;
    temp : real;

begin 
    Err := true;
    ClrScr;
    writeln;
    writeln(' The current ',descrip,' is: ',Value:9:3);
    writeln;
    writeln('  Enter the new ',descrip,':');
    readln(userinput);
    if userinput = 'x' then exit;
    if userinput = 'X' then exit;
    if userinput = ''  then exit;
    val(userinput,temp,check);
    if check <> 0 then 
        begin
            badBeep;
            writeln(' *** Invalid ',descrip);
            writeln;
            delay(500);
            exit;
        end
    else 
        begin
            if (temp < min) or (temp > max) then 
                begin
                    badBeep;
                    writeln;
                    writeln(' *** Invalid ',descrip);
                    writeln;
                    delay(1000);
                    exit;
                end
            else 
                begin
                    Value := temp;
                    Err := false;
                end;
        end;
end;

procedure readData(var DRdError: Boolean);
{Readback "Nt" points from Evaluation Fixture; find Yave, Yrms, return status}
var 
    FNum, N : longint; { final number }
    BusyBit3, Loop : integer; { data ready bit }
    Nib, D, SideRd : byte;
    PortDataRd : byte;
    Ysum, Ysumsq : array[1..4] of Extended;
    temp : extended;
    iRd, Trd, K, Count, DataTime, I, J : word;
    FScale,DelTime : real;
    FirstTime : boolean;
    timereq, SampleTime : real;
    numTints : word;

begin 
    write(' filling Demo Board memory...  ');
    FirstTime := true;    {flag to check for side A or B on first data point}
    if Tint < 479.4 then  {ncont mode}
        begin
            timereq := 1501*6*1/10 - Tint; {1501=min # of slowclks to finish ncont meas}
            numTints := trunc(timereq / Tint)+1;  {# of tints skipped}
            if ((numTints MOD 2) <> 0) then numTints := numTints+1; {numTints must be even}
            SampleTime := Tint*(2*numTints);
            end  {cont}
    else SampleTime := Tint*2;
I := 0;
while I < round(Nt*1.4 + 50) do  {extra margin for safety}
begin
  delay(round(sampleTime/1e3));
  if KeyPressed AND (ReadKey = Chr(27)) then
    begin
      goodBeep;
      I := round(Nt*1.4 + 50);
    end;
  I := I + 1;
end;

xilinxWrite(6, Ctrl1Code AND $5F);         {enable readback}

{wait for PC Interface Board Data Valid}
K := 0;
DRdError := false;
BusyBit3 := (PORT[readdr] AND $08) SHR 3;  {initialize bit 3}
While (BusyBit3 <> 1) AND (K <= 20000) do
begin
  BusyBit3 := (PORT[readdr] AND $08) SHR 3;
  inc(K);
  if (K mod 40) = 0 then delay(1);
  if K = 20000 then DRdError := true;
end;
if DRdError = true then
begin
  badBeep;
  clrscr;
  writeln;
  writeln('**** Data Retrieval Error ****');
  writeln;
  writeln('Check the following:');
  writeln;
  writeln('DDC112 securely in socket');
  writeln('power supplies at 5V');
  writeln('cables properly connected');
  writeln('demo board "refreshed"');
  writeln('PC port set correctly');
  writeln;
  writeln('******************************');
  writeln;
  writeln('Hit ENTER to continue');
  readln;
  exit;
end
else
begin
  FScale := $FFFFF-$1000; {fullscale range = all ones - offset}
  for D := 1 to 4 do
begin
    Ysum[D] := 0;
    Ysumsq[D] := 0;
  end;
  writeln('retrieving data...');
  for N := Nt-1 downto 0 do
begin
    for D := 1 to 4 do
begin
      TRd := N + Nt*(D-1);
      Fnum := 0;
      PortDataRd := $6F;{decrement Xilinx RAM memory address}
      PORT[wraddr] := PortDataRd;
      PORT[strbaddr] := 1;
      PORT[strbaddr] := 0;
      PORT[strbaddr] := 1;
      PORT[strbaddr] := 0;
      {prepare for RAM memory address}
      for iRd := 1 to 6 do {get 6 nibbles}
      begin
        PortDataRd := PortDataRd - $10;{MSB, MSB-1,...,LSB nibble}
        PORT[wraddr] := PortDataRd;
        PORT[strbaddr] := 1;
        PORT[strbaddr] := 0;
      end;
      PORT[resaddr] AND $08 SHR 3;  {init bit 3}
While (BusyBit3 <> 1) AND (K <= 20000) do
begin
BusyBit3 := (PORT[readdr] AND $08) SHR 3;
inc(K);
if (K mod 40) = 0 then delay(1);
if K = 20000 then
begin
DRdError := true;
N := 0;
end;
nib := ((PORT[readdr] XOR $80) AND $F0) SHR 4;
{XOR inverts bit 7 (-busy), AND F0 gets rid of 4 lower bits,
shift right puts the data bits in lower nibble}
if iRd = 1 then
SideRd := ((nib AND $4) Xor $4) SHR 2;  {bit3 lo ==> sideA}
Fnum := Fnum SHL 4;
FNum := Fnum OR nib;  {Add nibble to final number}
end;
if FirstTime then
begin
if SideRd = 0 then FirstTime := false {first point is correct side}
else
begin
FirstTime := false;
for count := 1 to ((UserDDCCode-1)*2 + 1) do {read & throw away pts}
begin
PortDataRd := $6F;  {decrement Xilinx RAM Memory Address}
PORT[wraddr] := PortDataRd;
PORT[strbaddr] := 1;
PORT[strbaddr] := 0;
PORT[strbaddr] := 1;
PORT[strbaddr] := 0;
{prepare for RAM memory address (PortDataRd := $6F;)}
for iRd := 1 to 6 do   {get 6 nibbles}
begin
PortDataRd := PortDataRd - $10;  {MSB, MSB-1,...,LSB nibble}
PORT[wraddr] := PortDataRd;
PORT[strbaddr] := 1;
PORT[strbaddr] := 0;
{wait For PC Interface Board DValid}
K := 0;
BusyBit3 := (PORT[readdr] AND $08) SHR 3;  {init bit 3}
While (BusyBit3 <> 1) AND (K <= 20000) do
begin
BusyBit3 := (PORT[readdr] AND $08) SHR 3;
inc(K);
if (K mod 40) = 0 then delay(1);
if K = 20000 then
begin
DRdError := true;
N := 0;
end;
end;
end;
end;
Fnum := 0;
PortDataRd := $6F;  {decrement Xilinx RAM Memory Address}
PORT[wraddr] := PortDataRd;
PORT[strbaddr] := 1;
PORT[strbaddr] := 0;
PORT[strbaddr] := 1;
PORT[strbaddr] := 0;
{prepare for RAM memory address (PortDataRd := $6F;)}
for iRd := 1 to 6 do   {get 6 nibbles}
begin
PortDataRd := PortDataRd - $10;  {MSB, MSB-1,...,LSB nibble}
PORT[wraddr] := PortDataRd;
PORT[strbaddr] := 1;
PORT[strbaddr] := 0;
{wait for PC Interface Board Data Valid}
K := 0;
BusyBit3 := (PORT[readdr] AND $08) SHR 3;  {init bit 3}
While (BusyBit3 <> 1) AND (K <= 20000) do
begin
  BusyBit3 := (PORT[readdr] AND $08) SHR 3;
  inc(K);
  if (K mod 40) = 0 then delay(1);
  if K = 20000 then
  begin
    DRdError := true;
    N := 0;
  end;
end;

nib := (PORT[readdr] XOR $80) AND $F0) SHR 4;
{XOR inverts bit 7 (-busy), AND F0 gets rid of 4 lower bits, shift right puts the data bits in lower nibble}
if iRd = 1 then
  SideRd := ((nib AND $4) XOR $4) SHR 2; {bit3 lo ==> sideA}
  Fnum := Fnum SHR 4;
  FNum := Fnum OR nib; {add nibble to final number}
end;

Fnum := (FNum AND $0000FFFFF); {set bits 31-21 := 0}
Fnum := Fnum - $1000; {subtract offset}
Ysum[D] := Ysum[D] + Fnum;
Fnum := Fnum SHL 4;
FNum := Fnum OR nib; {add nibble to final number}

nib := ((PORT[readdr] XOR $80) AND $F0) SHR 4;
{XOR inverts bit 7 (-busy), AND F0 gets rid of 4 lower bits, shift right puts the data bits in lower nibble}
if iRd = 1 then
  SideRd := ((nib AND $4) XOR $4) SHR 2; {bit3 lo ==> sideA}
  Fnum := Fnum SHR 4;
  FNum := Fnum OR nib; {add nibble to final number}
end;

for D := 1 to 4 do
begin
  Yave[D] := Ysum[D]/(Nt*FScale);
  temp := NT*Ysumsq[D]-(Ysum[D]*Ysum[D]);
  Yrms[D] := Sqrt(abs(temp)/(Nt*(Nt-1)))/Fscale;
  Yrms[D] := Yrms[D] * 1e6; {convert to ppm}
end;

for D := 1 to 4 do
begin
  writeln(' # of pts = ',Nt,' Range = ',Range,' Tint = ',Tint:8:1, ' Test pkts = ',TPck);
  writeln('        average    rms noise');
  writeln(' 1A: ',Yave[1]:11:7,'  ',Yrms[1]:6:2,' ppm');
  writeln(' 2A: ',Yave[2]:11:7,'  ',Yrms[2]:6:2,' ppm');
  writeln(' 1B: ',Yave[3]:11:7,'  ',Yrms[3]:6:2,' ppm');
  writeln(' 2B: ',Yave[4]:11:7,'  ',Yrms[4]:6:2,' ppm');
end;

xilinxWrite(6,Ctrl1Code OR $20); {enable write to RAM}
end;

*****************************************************************************
procedure retrieveData(var err: boolean);
*****************************************************************************
{Call readData then display results}
begin
  readData(err);
  if err = false then
  begin
    ClrScr;
    writeln('average rms noise');
    writeln('1A: ',Yave[1]:11:7,' ',Yrms[1]:6:2,' ppm');
    writeln('2A: ',Yave[2]:11:7,' ',Yrms[2]:6:2,' ppm');
    writeln('1B: ',Yave[3]:11:7,' ',Yrms[3]:6:2,' ppm');
    writeln('2B: ',Yave[4]:11:7,' ',Yrms[4]:6:2,' ppm');
  end;
end;

*****************************************************************************
procedure TestModeLinearity;
*****************************************************************************
{Measure linearity of the DDC112’s Test Mode}
{Fit data with endpoint line and report max deviation from that line.}
var
  n,i : integer;
  err : boolean;
  userName : string[8];
  fileName : string[12];
  F1 : text;
  tpacket : real;
  steps : word;
  Tdat : array[1..4,0..100] of real; {holds data for lin fit}
  m,b : real;
  INLpt, INLmax : real;
  INL : array[1..4] of real;
begin
ClrScr;
if Nt < 10 then
begin
writeLn;
writeLn('***    The number of points / reading is low ***');
writeLn;
writeLn('     Increase the number for better results   ');
badBeep;
gInt('# of points',1,10000,Nt,err);
end;

writeLn;
writeLn('Enter a file name for Test Mode Linearity data (max 8 characters)');
readln(userName);
writeLn;
fileName := userName + '.DAT';
assign(F1,fileName);
delay(1);
{$I-} rewrite(F1); {$I+}
if ioresult <> 0 then
begin
writeLn('');
writeLn(' *** File I/O Problem, try a new name...');
delay(400);
badBeep;
exit;
end;

setTstPckts(1);              {set test pckts = 1 & collect 1st data}
writeLn(' collecting test data 1');
readData(err);
if err = false then
begin
for n := 1 to 4 do Tdat[n,1] := Yave[n];
packet := (Yave[1])*100;  {calc packet size}
writeLn;
writeLn('*** Each test packet is ',packet:4:1,' % of full scale ***');
steps := trunc(1/Yave[1]); {calc # of steps for fullscale}
if steps > (trunc(Tint/5.4)) then
begin
steps := trunc(Tint/5.4);
writeLn;
writeLn('***********************************************');
writeLn('Test Mode will not be able to reach full scale');
writeLn('with current Tint.  Increasing Tint will allow');
writeLn('more test packets dumps during the integration.‘);
writeLn('***********************************************');
badBeep;
end;
end
else exit;

for i := 2 to steps do {collect rest of data}
begin
writeLn;
writeLn(' collecting test data ',i,' out of ',steps);
setTstPckts(i);
readData(err);
end;

setTstPckts(Tpck);          {restore user Test Package setting}
ClrScr;
writeLn;
writeLn('   Linearity Results: Endpoint Fit');
writeLn('   (max deviation from endpoint line)');
writeLn;
for n := 1 to 4 do           {fit linearity}
begin
    b := Tdat[n,1];         {y = mx + b}
    m := (Tdat[n,steps] - Tdat[n,1])/(steps-1);
    INLmax := 0;
    for i := 2 to (steps-1) do {INL for i=1 & steps =0 for endpoint fit}
begin
        INLpt := Abs(Tdat[n,i] - (m*(i-1) + b));
        if INLpt > INLmax then INLmax := INLpt;
    end;
    INL[n] := 1e6* INLmax ;    {convert to ppm}
end;
case n of
    1: writeln('  INL for 1a = ',INL[1]:6:1,' ppm');
    2: writeln('  INL for 2a = ',INL[2]:6:1,' ppm');
    3: writeln('  INL for 1b = ',INL[3]:6:1,' ppm');
    4: writeln('  INL for 2b = ',INL[4]:6:1,' ppm');
end;
end;
writeln(F1,'        Burr-Brown Corporation     ');
writeln(F1,'      Application Bulletin AB-135   ');
writeln(F1,'DDC112 Test Mode linearity measurement');
writeln(F1);
writeln(F1,'# of averages/readings = ',Nt);
writeln(F1,'Range = ',Range);
writeln(F1,'Tint = ',Tint:7:1);
writeln(F1);
writeln(F1,'# Test');
writeln(F1,'Packets     1a ave      2a ave      1b ave      2b ave');
for i := 1 to steps do
    writeln(F1,i:3,'     ',Tdat[1,i]:11:6,' ',Tdat[2,i]:11:6,' ',Tdat[3,i]:11:6,' ',Tdat[4,i]:11:6);
writeln(F1);
writeln(F1,' Endpoint Fit');
writeln(F1,' INL(ppm): ',INL[1]:8:1, INL[2]:12:1, INL[3]:12:1, INL[4]:12:1);
close(F1);
goodBeep;
writeln('Hit ENTER to continue...');
readln;
end;
{***************************************************************************}
{***************************************************************************}
{Main}
{***************************************************************************}
{***************************************************************************}
var
    userinput : string[10];
    goodbye,done : boolean;
    err : boolean;
    choice : word;
    check : integer;
    minTint : real;

begin
    Range := 5;
    Tint := 500;
    Nt := 500;
    TPck := 1;
    PCPort := 1;
    DataCkCode := 2;
    SysCkCode := 0;
    DXmitDelayCode := 0;
    UserDDCCode := 1;
    Ctrl1Code := $20; {initially Read, internal DCLK}
    Ctrl2Code := 1; {demo bd CONV}
    pPrintPort := Ptr($40,$08); {default: PC port=1}
    wraddr := pPrintPort^;
    readdr := wraddr + 1;
    strbaddr := wraddr + 2;
    PORT[strbaddr] := 0;  {init Strobe at 0 (really, 1 out)}
goodbye := false;
while (goodbye <> true) do
begin
ClrScr;
writeln;
xilinxRefresh;
writeln('----------------------------------------- - ');
writeln('  |        Burr-Brown Corporation |');
writeln('  |      Application Bulletin AB1XX |');
writeln('  | |');
writeln('  | DDC112 Test Mode Evaluation Program |');
writeln('  ———————————————————');
writeln;
writeint('# points/reading = ',Nt);
writeln('              Range = ',Range);
writeint('               Tint = ',Tint:7:1);
writeint('# test packets = ',TPck);
writeint;
writeint;
writeint('  ===========================  ');
writeint('  Choose one of the following  ');
writeint('  ===========================  ');
writeint;
writeint('1) Refresh Demo Board');
writeint;
writeint('2) Set # points averaged/reading');
writeint('3) Set Range');
writeint('4) Set Tint');
writeint('5) Set # of Test Mode packets');
writeint('6) Set PC port');
writeint('7) Take single reading');
writeint('8) Measure Linearity of Test Mode');
writeint('9) exit program');
writeint;
readln(userinput);
val(userinput,choice,check);
if check <> 0 then badBeep
else
begin
if (choice < 0) or (choice > 9) then badBeep
else
  case choice of
1: begin
  writeln;
  writeln('Refreshing Demo Board...');
xilinxRefresh;
end;
2: getInt('# of points',1,10000,Nt,err);
3: begin
  getInt('Range',0,7,Range,err);
  if err = false then setRange(Range);
end;
4: begin
  getReal('Tint',0,1500000,Tint,err);
  if err = false then setTint(Tint);
end;
5: begin
  getInt('Test Mode Packets',0,128,TPck,err);
  if err = false then
    begin
      if TPck > Trunc(Tint/5.4) then   {allow 5.4us per packet}
        begin
          writeln('*****************************************************');
          writeln('  # of Test Packets is too high for current Tint');
          writeln('  Tint must be > ',minTint:7:1, ' us for ',TPck,' packets.');
          writeln('  Setting # of Test Packets to ',TPck);
          writeln('*****************************************************');
          writeln;
          writeln('     Hit ENTER to continue...');
          BadBeep;
        end;
      setTstPckts(TPck);
    end;
  end;

6: begin
  getInt('PC port',1,2,PCport,err);
  setPCport(PCport);
  end;

7: begin
  done := false;
  while done = false do
    begin
      retrieveData(err);
      if err = true then done := true
    else
      begin
        writeln;
        writeln('     enter "r" to repeat measurement');
        readln(userinput);
        if (userinput <> 'r') AND (userinput <> 'R')
          then done := true;
      end;
  end;

8: TestModeLinearity;

9: goodbye := true;
end;
end;
IMPORTANT NOTICE

Texas Instruments and its subsidiaries (TI) reserve the right to make changes to their products or to discontinue any product or service without notice, and advise customers to obtain the latest version of relevant information to verify, before placing orders, that information being relied on is current and complete. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgment, including those pertaining to warranty, patent infringement, and limitation of liability.

TI warrants performance of its semiconductor products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Customers are responsible for their applications using TI components.

In order to minimize risks associated with the customer’s applications, adequate design and operating safeguards must be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance or customer product design. TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used. TI's publication of information regarding any third party's products or services does not constitute TI's approval, warranty or endorsement thereof.

Copyright © 2000, Texas Instruments Incorporated