XAPP 149 (v1.0) June 6, 2001

Designing an Oscilloscope with the Insight Springboard Kit

Summary

An oscilloscope is a data aquisition device frequently used to measure and display voltage at a particular source. The Handspring Visor™ line of personal computers is an ideal candidate for such an application because it has a built-in LCD screen, a Motorola Dragon Ball™ processor, and includes a Springboard™ expansion slot. The LCD screen can easily be used for displaying the voltage versus time waveform, the processor can be used for collection of data, and the Springboard module can contain the necessary hardware for the oscilloscope implementation. This application note will discuss the design of a simple oscilloscope for the Handspring Visor using a Xilinx CoolRunner™ CPLD to interface a Texas Instruments ADS7870 Data Acquisition System to the Handspring Springboard expansion slot.

This Oscilloscope design is an extension of XAPP146, "Designing an Eight Channel Digital Volt Meter with the Insight Springboard Kit." Instead of displaying the voltage as numerical text, this design will display the voltage for a single channel as a graph versus time. Note that both the Digital Volt Meter design and the Oscilloscope design build upon the TI ADS7870 Data Acquisition System Interface described in XAPP355. This Application Note will not discuss the ADS7870 Interface in detail. Readers should familiarize themselves wiith XAPP355 before proceeding.

All related source code will also be provided for download. To obtain the VHDL code described in this document, go to section "VHDL Code Download" on page 10 for instructions

Overview

This application note presents a straightforward oscilloscope design. Once the oscilloscope application is launched, the Visor's LCD screen will automatically display and update the voltage waveform associated with the signal connected to analog input channel 0 of of the ADS7870. Thirty-eight points will be continuously plotted and erased. The signal will be sampled and stored in SRAM at a rate of 52 ksps (kilo samples/second). This means that the time between two consecutive plotted points will be 19.2 uS. Changes in frequency of the analog input will be instantaneously detected and displayed on the Visor's LCD screen.

The ADS7870 has been configured for single ended operation, which means that the analog input signal can vary from 0V to a maximum of 2.5V. If a wider input voltage is desired, external biasing can be introduced. Also, if the voltage source in question has differential outputs, the ADS7870 can be configured accordingly. However, these two options will not be discussed in this Application Note.

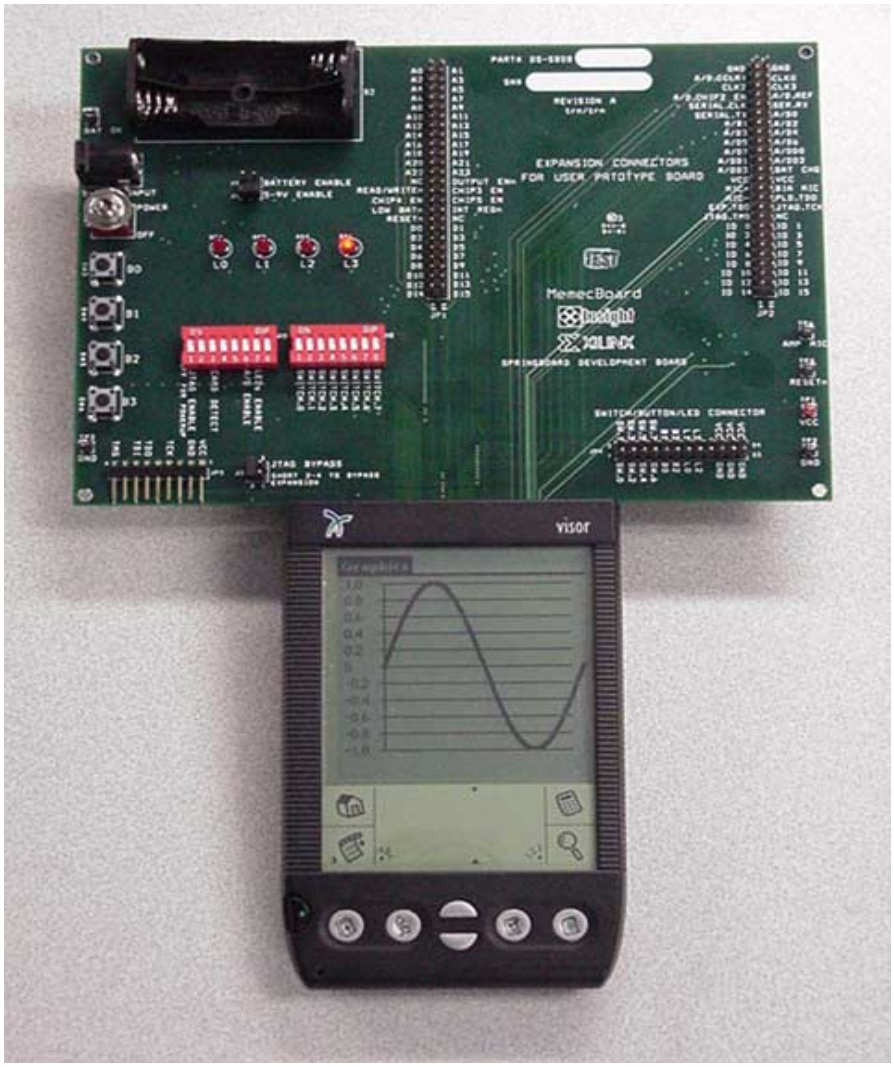

Figure 1: **Oscilloscope Application**

Operational Flow

Figure 2 shows the operational flow of the Digital Volt Meter design. The Handspring application enters an infinite loop in which it first issues a command to the Xilinx CoolRunner CPLD. Upon receiving this specific data and address value, the CoolRunner CPLD commands the ADS7870 to begin converting. Immediately after the ADS7870 is ordered to convert, the first analog input channel, LN0, is sampled thirty-eight times, with each result written to SRAM locations 1-38, respectively. When all results have been written to SRAM, the CoolRunner CPLD allows the Visor to read the contents in SRAM, calculate the voltage, and display the voltage as a graphed time series across the LCD screen.

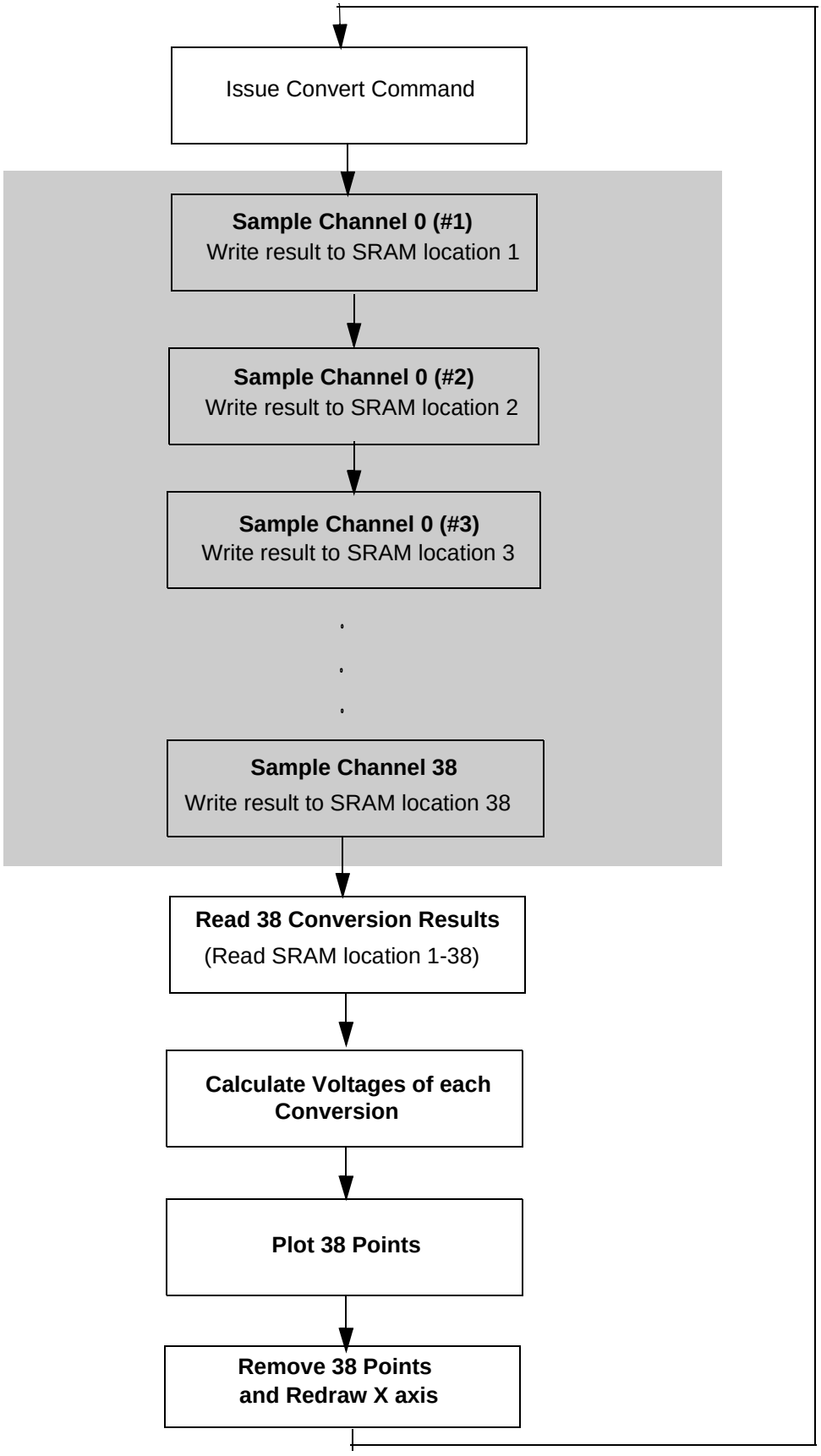

Figure 2: **Operational Flow of a Digital Volt Meter Design**

Notice that the shaded area in Figure 2 represents tasks which are done in hardware. The boxes outside of the shaded area represent tasks done by software (Handspring). In other words, the data aquisition process is done completely in hardware, while data gathering is done by the Handspring Visor.

This is a classic scheme, as many processors are not fast enough handle high speed data transfers. This is especially true in the case of the Handspring Visor, which utilizes a Motorola Dragon Ball processor running at 16 MHz (33 MHz on the premium models). For very fast transactions, such as handling data from a high speed A/D converter, dedicated external hardware is often used to handle the bus transactions. The data can then be stored in memory (SRAM) for the slower processor to read.

In this case, the Xilinx CoolRunner CPLD's combination of high speed and low power make it an ideal candidate for high speed data manipulation.

VHDL Interface

XAPP355 provides and explains the Texas Instruments ADS7870 Data Aquisition System interface. The VHDL code presented in XAPP355 is intended to be a "building block" for future designs. A detailed understanding of the VHDL code is not needed. Rather, the designer needs only to focus on the details of the ADS7870. If certain aspects of the ADS7870 need to be adjusted, the "constants" section of the VHDL code can then be modified to accordingly.

This reference design shows how to customize the original code presented in XAPP355. Slight changes have been made to the "constants" section. Figure 3 below shows the portions of the "constants" section that have been modified.

As shown, only the first analog input channel of the ADS7870 (LN0) has been configured for single-ended operation. The locations in SRAM that will store the conversion results have also been defined. Table 1 shows the conversion result address map.

```
-- *************** DIRECT MODE CONVERSION SINGLE ENDED CHANNEL 0 ********************
constant DM_SNG_LN0_EN : BOOLEAN := TRUE;
constant DM_SNG_LN0 : STD_LOGIC_VECTOR(7 downto 0) := "10001000";
constant SRAM_OFFSET0 : STD_LOGIC_VECTOR (22 downto 0) := "00000000000000000000000";
constant SRAM_HIGH0   : STD_LOGIC_VECTOR (22 downto 0) := "00000000000000000100110";

-- *************** DIRECT MODE CONVERSION SINGLE ENDED CHANNEL 1 ********************
constant DM_SNG_LN1_EN : BOOLEAN := FALSE;
constant DM_SNG_LN1    : STD_LOGIC_VECTOR(7 downto 0) := "10001001";
constant SRAM_OFFSET1  : STD_LOGIC_VECTOR (22 downto 0) := "00000000000000000001000";
constant SRAM_HIGH1    : STD_LOGIC_VECTOR (22 downto 0) := "00000000000000000001111";

-- *************** DIRECT MODE CONVERSION SINGLE ENDED CHANNEL 2 ********************
constant DM_SNG_LN2_EN : BOOLEAN := FALSE;
constant DM_SNG_LN2    : STD_LOGIC_VECTOR(7 downto 0) := "10001010";
constant SRAM_OFFSET2  : STD_LOGIC_VECTOR (22 downto 0) := "00000000000000000010000";
constant SRAM_HIGH2    : STD_LOGIC_VECTOR (22 downto 0) := "00000000000000000010111";

-- *************** DIRECT MODE CONVERSION SINGLE ENDED CHANNEL 3 ********************
constant DM_SNG_LN3_EN : BOOLEAN := FALSE;
constant DM_SNG_LN3    : STD_LOGIC_VECTOR(7 downto 0) := "10001011";
constant SRAM_OFFSET3  : STD_LOGIC_VECTOR (22 downto 0) := "00000000000000000011000";
constant SRAM_HIGH3    : STD_LOGIC_VECTOR (22 downto 0) := "00000000000000000011111";

-- *************** DIRECT MODE CONVERSION SINGLE ENDED CHANNEL 4 ********************
constant DM_SNG_LN4_EN : BOOLEAN := FALSE;
constant DM_SNG_LN4    : STD_LOGIC_VECTOR(7 downto 0) := "10001100";
constant SRAM_OFFSET4  : STD_LOGIC_VECTOR (22 downto 0) := "00000000000000000100000";
constant SRAM_HIGH4    : STD_LOGIC_VECTOR (22 downto 0) := "00000000000000000100111";

-- *************** DIRECT MODE CONVERSION SINGLE ENDED CHANNEL 5 ********************
constant DM_SNG_LN5_EN : BOOLEAN := FALSE;
constant DM_SNG_LN5    : STD_LOGIC_VECTOR(7 downto 0) := "10001101";
constant SRAM_OFFSET5  : STD_LOGIC_VECTOR (22 downto 0) := "00000000000000000101000";
constant SRAM_HIGH5    : STD_LOGIC_VECTOR (22 downto 0) := "00000000000000000101111";

-- *************** DIRECT MODE CONVERSION SINGLE ENDED CHANNEL 6 ********************
constant DM_SNG_LN6_EN : BOOLEAN := FALSE;
constant DM_SNG_LN6    : STD_LOGIC_VECTOR(7 downto 0) := "10001110";
constant SRAM_OFFSET6  : STD_LOGIC_VECTOR (22 downto 0) := "00000000000000000110000";
constant SRAM_HIGH6   : STD_LOGIC_VECTOR (22 downto 0) := "00000000000000000110111";

-- *************** DIRECT MODE CONVERSION SINGLE ENDED CHANNEL 7 ********************
constant DM_SNG_LN7_EN : BOOLEAN := FALSE;
constant DM_SNG_LN7    : STD_LOGIC_VECTOR(7 downto 0) := "10001111";
constant SRAM_OFFSET7  : STD_LOGIC_VECTOR (22 downto 0) := "00000000000000000111000";
constant SRAM_HIGH7    : STD_LOGIC_VECTOR (22 downto 0) := "00000000000000000111111";
```

Figure 3: **Modified "Constants" Section**

Table 1: **SRAM Locations**

Channel	Sample #	SRAM Location (A17..A0)	SRAM Location (Decimal)
0	1	000000000000000001	1
	2	000000000000000010	2
	3	000000000000000011	3
	⋮	⋮	⋮
	38	000000000000100110	38

PocketC Code

The PocketC source code is shown in Appendix A. As stated in XAPP147, the Xilinx Native Library, "IOLib.prc", which defines the functions "IORead" and "IOWrite", is needed in order for PocketC to access the Springboard IO. IOLib.prc must also be installed on the Handspring Visor in order for this DVM Application to work.

The PocketC code is simple because the ADS7870 is interfaced through hardware. The program is comprised of an infinite loop during which two major tasks are done:

1) Initiating a new conversion

2) Retrieving, computing and displaying results stored in SRAM

Note that the code for this oscilloscope implementation is very similar to the code described in XAPP146, "Designing an 8 Channel Digital Volt Meter for the Insight Springboard Kit". Please refer to XAPP146 for a more detailed explanations of how the software computes voltage from samples stored in SRAM.

Initiating a New Conversion

The CoolRunner CPLD will only begin a new conversion process upon receiving a Springboard address of 0x2900003E and a Springboard data value of 0xFFFF followed by an address of 0x2900003E and a data value of 0x0000.

The following two PocketC commands accomplish this:

```
IOWrite(0x2900003e,0xffff);

IOWrite(0x2900003e,0x0000);
```

Retrieving, Computing and Displaying Results

Immediately after initiating a new conversion, the software retrieves and computes the 38 samples which are stored in SRAM. Notice that a wait state is not needed between the time when a new conversion is initiated, and the time when the results are retrieved. The hardware

will have completed its entire chain of events well before the software executes its next line of code.

The PocketC code that is used to retrieve, compute and display the results is shown below:

```
//Plot 38 points:
for(n=0;(n<38);n++){
addr=addr+2;
result=IORead(addr);   //Retrieve conversion result from SRAM
result=result>>4;      //Shift
volt=result;
volt=(volt/2047)*2.5; //Compute Voltage
array_y[array_cnt]=format(volt,2);  //Store in array
array_cnt=array_cnt+1;//increment pointer
plotpoint(format(volt,2),n);        //Plot the computed voltage
}
```

As shown above, each sample must go through three steps. First, the sample is read from SRAM. Next, a voltage is computed from this sample. The voltage is stored as an element in an array, and finally, this voltage is plotted to the screen via the plotpoint() function.

It is necessary to store the 38 computed voltages in an array in order to remove these samples from the screen. The following routine removes the previously plotted points:

```
//Then "Unplot the 38 points:
b=0;
for(j=0;(j<38);j++){
unplot(array_y[j], b);//Unplot
b=b+1;
}
```

After the points are removed, the cycle starts again. A new conversion is initiated, and results are first displayed, then removed.

Conclusion

Many of the useful features found on common oscilloscopes--like triggering, time-scaling, and attenuation--have not been implemented in this oscilloscope design, but can be easily implemented in software.

However, the design presented here can be used by many other Springboard applications which require an analog signal to be displayed on the screen. An example of such an application is a spectrum analyzer, which would read a microphone output, perform some digital signal processing in software, and display a person's speech pattern across the Visor screen.

The uses for the design provided here are far-reaching. Instead of providing a full oscilloscope implmentation, this Application Note demonstrates how to properly use the ADS7870 VHDL Interface as described in XAPP355. Both this note and XAPP146 show that the ADS7870 interface is easy to use and fully customizable.

Appendix A: PocketC Code

```
//Oscilloscope

@cid "OSC1";
@ver "1.0";
@name "Oscilloscope";
@dbname "Oscilloscope";
@licon1 "xilinx.bmp";
@sicon1 "small1.bmp";

library "IOLib"  //Xilinx Native Library, Defines IOWrite, IORead.


int ytop,xlft;
float gRange,lrange;


//********** Drawaxis Function *********************
//
// Purpose:  This function draws the X axis
//************************************************

drawaxis(float hrange, float lr){
int y,x,n;
float r;
  ytop=20;
  xlft=30;
  lrange=lr;
  gRange=hrange-lrange;
  r=gRange/10;
  line(1,xlft,ytop,xlft,ytop+120);
  line(1,xlft,ytop+120, xlft+120,ytop+120);
  textattr(0,1,0);
  for (n=0;n<11;n++){
   y=(120-(n*12))+ytop-5;
   x=(n*12)+xlft-2;
   text(xlft-25,y,format((n*r)+lrange, 1));
   line(1,xlft-3,y+5,xlft+120,y+5);
   text(x,142,n);
   }
}

//********** Plotpoint Function *********************
```

```
//
// Purpose:  This function plots each point of the graph
//*************************************************

plotpoint(float val, int item){
int x,y;
y=(int) (120-(((val-lrange)/gRange)*120)+ ytop);
x=(item*3)+xlft;
rect(1,x-1,y-1,x+2,y+2,1);
}

//********** Unplot Function ********************
//
// Purpose:  This function removes the previously plotted points
//*************************************************

unplot(float val, int item){
int x,y;
y=(int) (120-(((val-lrange)/gRange)*120)+ ytop);
x=(item*3)+xlft;
rect(0,x-1,y-1,x+2,y+2,1);
}


//********** Main Function *********************
//
// Purpose:  Main
//*************************************************

main(){

int result, addr, n, b, j;
float volt;
float array_y[38];
int array_cnt;
graph_on();
clearg();
rect(0,0,0,165,165,1);

while(1){

//Tell CoolRunner CPLD to do a new conversion.
//38 Conversion results will be written to SRAM.
```

```
IOWrite(0x2900003e,0xffff);
IOWrite(0x2900003e,0x0000);


//Draw (or Redraw) the X, Y Axis'
drawaxis(2.5,0);

addr=0x29000002; //start addr1
array_cnt=0;  //reset array to 0

//Plot 38 points:
for(n=0;(n<38);n++){
addr=addr+2;
result=IORead(addr);   //Retrieve conversion result from SRAM
result=result>>4;      //Shift
volt=result;
volt=(volt/2047)*2.5; //Compute Voltage
array_y[array_cnt]=format(volt,2);  //Store in array
array_cnt=array_cnt+1;//increment pointer
plotpoint(format(volt,2),n);        //Plot the computed voltage
}

//Then "Unplot the 38 points:
b=0;
for(j=0;(j<38);j++){
unplot(array_y[j], b);//Unplot
b=b+1;
}
}
}
```

References

Tektronix: XYZs of Oscilloscopes

http://www.tek.com/Measurement/cgi-bin/framed.pl?Document=/Measurement/App_Notes/XYZs/&FrameSet=oscilloscopes

VHDL Code Download

VHDL source code and test benches are available for this design. THE DESIGN IS PROVIDEDTO YOU "AS IS". XILINX MAKES AND YOU RECEIVE NO WARRANTIES OR CONDITIONS, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, AND XILINX SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR A PARTICULAR PURPOSE. While this design has been verified on hardware, it should be used only as an example design, not as a fully functional core. XILINX does not warrant the performance, functionality, or operation of this Design will

meet your requirements, or that the operation of the Design will be uninterrupted or error free, or that defects in the Design will be corrected. Furthermore, XILINX does not warrant or make any representations regarding use or the results of the use of the Design in terms of correctness, accuracy, reliability or otherwise.

XAPP149 - http://www.xilinx.com/products/xaw/coolvhdlq.htm

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
06/07/01	1.0	Initial Xilinx release.