Developing a VXIbus-Based Pager Receiver Functional Tester

Abstract
To demonstrate the use of a VXIbus System for RF Networking and Communications applications, a demonstration system was produced for a typical application: RF pager testing. The frequency of interest (930MHz) and the digital protocol are typical of many wireless or wired communications systems. The following information is a brief description of how this system was developed and to serve as an example of the flexibility of VXIbus hardware as well as the ease of developing a software solution.

Test Goals
Test goals were aimed at producing a system to test a typical RF product. Since most pagers are one-way devices, they receive but do not transmit, this will be an example of RF receiver testing only. The test can later be extended to test 2-way devices, such as 2-way pagers, cordless phones, RF modems, etc.

A simple strategy was defined to test the pager:

1) Transmit an alpha-numeric message to the pager at varied signal strengths, and
2) Read back the message from the pager to verify correct transmission.

This strategy tests more than just the functionality of the pager's receiver, it also tests the decoder and the serial port (the pager's display is not tested). The performance of the decoder and serial port should not be affected by changes in signal strength. The pager's receiver is very sensitive to signal strength since as the signal strength is reduced a point will be reached when the receiver can no longer acquire the signal. Thus, the sensitivity of the pager's receiver is what is really being tested.

The Pager to Be Tested
A Motorola pager was selected as the pager to be tested. The model to be tested was the "Advisor" pager which uses the POCSAG protocol. Other similar protocols are also used for pagers, such as GSC or FLEX. Minor modifications to this test are easily incorporated to support these and other protocols.

The Advisor has a built-in serial port which allowed the pager to be wired to a VXIbus readback device. This requires a small pod (available from Motorola) which converts the Advisor's serial port output to RS-232 levels.

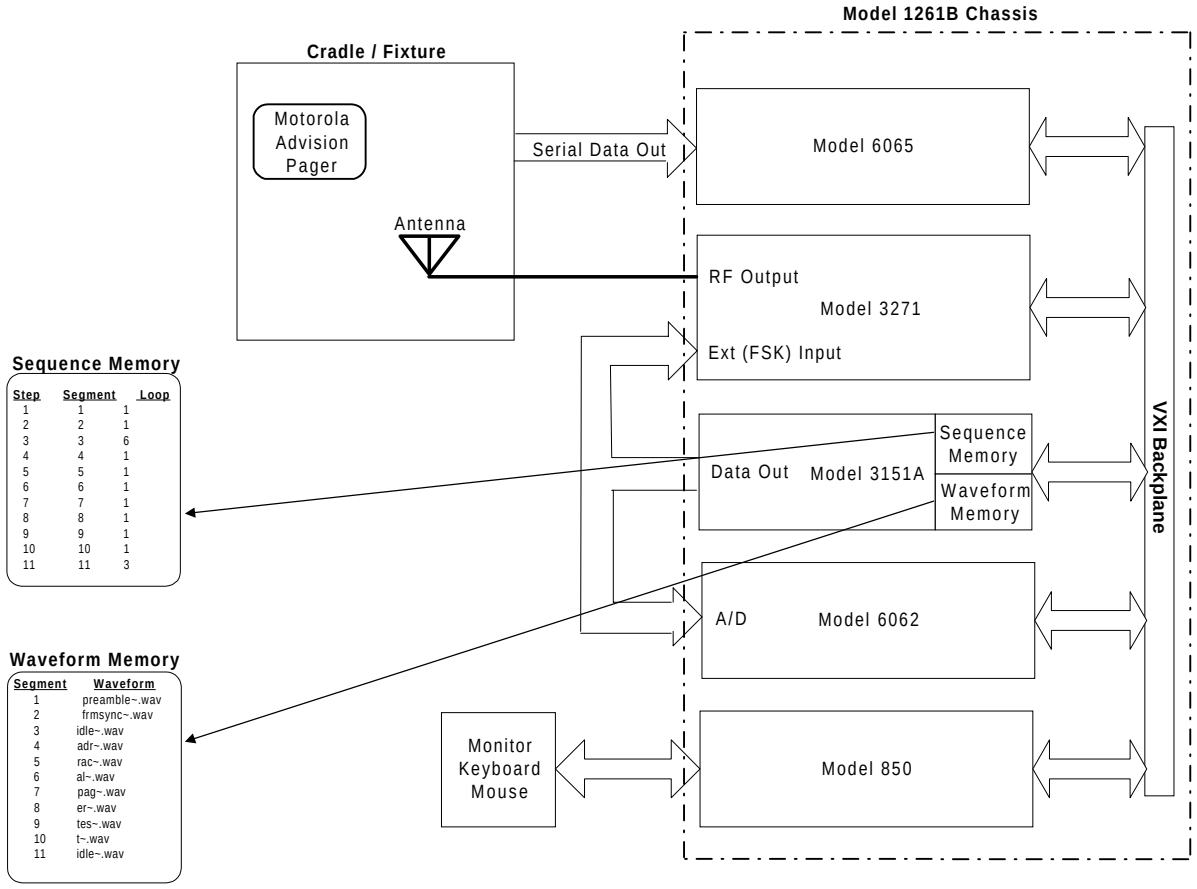
Choosing the Test Equipment
Test equipment was chosen to:

1) produce the RF signals which send messages to pagers,
2) read the page from the pager and display the results,
3) provide a functional test platform that would maximize reliability, speed and repeatability, yet minimize system downtime, and
4) facilitate easy software development.

The RF signal required by the Advisor is an FSK (Frequency Shift Keyed) signal in the 900MHz range. The Racal Instruments 3271 2.4GHz Signal Generator was chosen and the Model 3151A

Arbitrary Waveform Generator was selected to modulate the 3271's output via the external FSK port. The Racal Instruments 6065P Serial Interface module provides the serial readback. A Model 1264C 6-slot mainframe and National Instruments Model 850 embedded controller were also required. In addition, a 900MHz antenna was constructed out of copper clad PCB for signal transmission. Note: Calibrated antennae known as "Radiation Test Fixtures" for the Advisor and other pagers are available from Motorola, but weren't required to demonstrate the concept. The space around the pager and the antenna were shielded to prevent interference from local RF signals.

The equipment was then connected as in the following diagram:



Sending a Page

The POCSAG code information was found in a document from the British Post Office entitled "A Standard Code for Radiopaging." The other documents from Motorola and various web sites condensed this information and changed or omitted required details in the process.

The POCSAG string for this application consists of:
1) A preamble section (576 bit reversals, i.e. 101010, etc.),

2) A frame synchronization codeword (a 32 bit word which begins each “batch” of data). Batches contain 8 frames and each frame contains 2 codewords,
3) A number of “IDLE” codewords (32 bit words that fill unused frames within a batch),
4) A pager address codeword (the “CAP” code from the back of the pager minus the 3 LSBs which indicate the frame [0-7] of the batch where the address codeword must appear for each pager),
5) The actual ASCII data, or message codewords, being sent to the pager (with 7 bit ASCII data, bit-reversed and formed into strings which continue between frames).

Items 2-5 listed above include Error Correcting Codes which are commonly used for digital data transmission. The particular ones that POCSAG protocol uses are called Bose-Chaudhuri-Hocquenenghen (BCH) codes. This required the calculation of these codes (shown below) for all address and data information being sent to the pager.

The codes for the frame synchronization and idle frames are fixed and the BCH codes for these were listed in the documentation and did not require calculation. The codes for the address and message codewords had to be calculated using modulo-2 arithmetic. A sample calculation for the error correcting code for the message string “RAC” is listed below:

The individual bits are treated as modulo-2 exponents; the term on the right is the modulo-2 value of the message (“RAC”) and the term on the left is the BCH “generating polynomial” for a 21-bit message.

$$x^{10}+x^{9}+x^{8}+x^{6}+x^{5}+x^{3}+x^{0}\,\overline{)\,x^{30}+x^{28}+x^{25}+x^{23}+x^{22}+x^{17}+x^{16}+x^{15}+x^{14}+x^{10}}$$

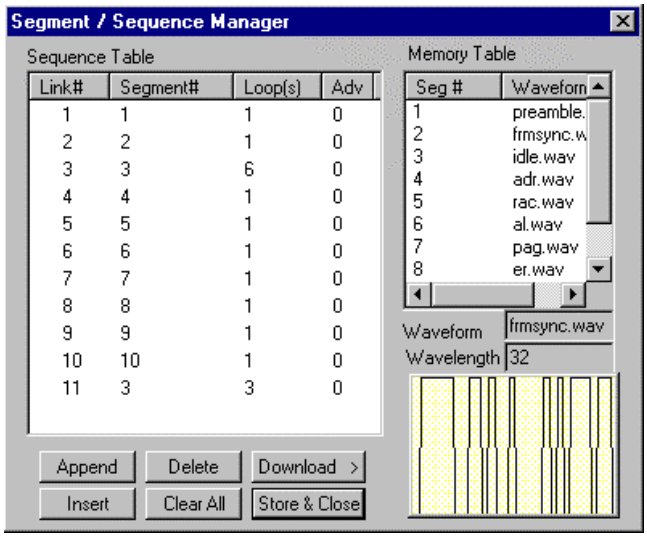
For short hand, just the exponents are included. The entire calculation using modulo-2 long division was done by hand (next time I’ll use Matlab!) and is given below:

$$10+9+8+6+5+3+0\,\overline{)\,30+28+25+23+22+17+16+15+14+10}$$

$$\begin{array}{l}
20+19+18+15+14+13+11+8+0 \\
\hline
30+28+25+23 \qquad\qquad\qquad\qquad +29+26+20 \\
\hline
29+26+22+20+17+16+15+14+10 \\
29 \quad +22 \qquad\qquad\qquad\qquad +28+27+25+24+19 \\
\hline
28+27+26+25+24+20+19+17+16+15+14+10 \\
28+27+26 \quad +24 \qquad\qquad\qquad\qquad +23+21+18 \\
\hline
25+23+21+20+19+18+17+16+15+14+10 \\
25+23+21+20 \quad +18 \qquad +15 \qquad +24 \\
\hline
24+19+17+16+14+10 \\
24+19+17 \quad +14 \quad +23+22+20 \\
\hline
23+22+20+16+10 \\
23+22 \quad +16 \quad +21+19+18+13 \\
\hline
21+20+19+18+13+10 \\
21+20+19 \qquad +17+16+14+11 \\
\hline
18+17+16+14+13+11+10 \\
18+17+16+14+13+11 \quad +8 \\
\hline
10+8 \\
\hline
10+8+9+6+5+3+0
\end{array}$$

$r = 9+6+5+3+0 \rightarrow 1001101001$

Once the correct code strings were computed, they next had to be converted to actual waveforms that could be output by the 3151A Waveform Generator. Since the data is digital, a text editor was used to create lists containing two waveform values, one and zero. A 576 entry list was created to represent the idle codeword, and 32 entry lists were created to represent all of the frame sync, idle, address and message codewords that would be required. WaveCAD software was then used to convert these ASCII files to downloadable .WAV files and to assemble these bits and pieces into a meaningful sequence. The sophisticated sequence definition feature of WaveCAD was used for sequence definition, with an actual sequence looking like:



These data format details are similar to the ones used in other digital data transmission schemes. Therefore the same equipment used can be adapted for use in networking and telecommunications applications (via wire, air and fiber optics) as well as with other custom applications. This is a good example of the flexibility of a VXIbus-based system.

Building a Test Soft Front Panel

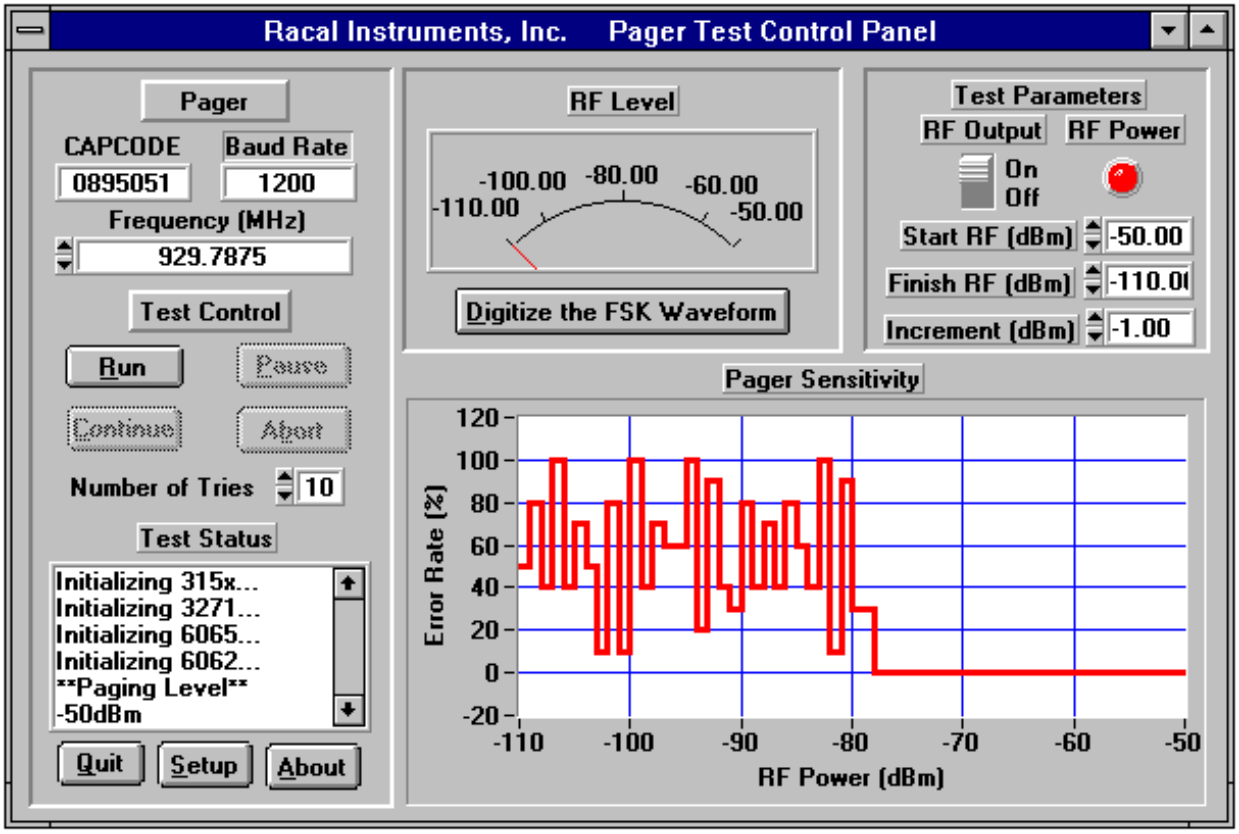
Creation of the test panel went smoothly due to the following 2 factors:
1) LabWindows/CVI environment was used, and
2) The VXI*plug&play* drivers for the instruments all contained LabWindows/CVI instrument drivers.

There are three simple steps to creating this panel:
1) Definition of the panel's user interface,
2) Selection of the required LabWindows/CVI driver commands, and
3) Addition of more "C" code to detail the panel's basic test logic.

The panel's user interface contains five basic sections:
1) A control section that allows the user to control the test and read the test status;
2) A parameter adjustment section that allows the user to adjust parameters such as

beginning RF power level, ending RF power level and step size;
3) An RF Power Meter displaying the current RF power output;
4) A results display section which graphs the test results;
5) A digitizer screen which captures the FSK modulation to allow inspection of the transmitted waveform (this requires the addition of a Racal 6062 Digitizer which is an optional part of this demonstration).

This panel is easily designed using the LabWindows/CVI User Interface Editor and is shown below:



After creation of the test panel in the User Interface Editor, LabWindows/CVI was used to generate a “C” program shell that proved to interface the test panel to the pager test program. The rest of the code was produced using the LabWindows/CVI drivers along with some additional “C” code. A listing of this code is provided as an appendix to this application note.

Conclusion

The most difficult part of developing a functional test system for an RF Pager from scratch was the learning of pager protocols, device ID codes, data encoding, and even

modulation polarity because this information had to be researched and assembled by hand. In contrast, the task of developing hardware and software to perform the test using VXIbus hardware from Racal Instruments and LabWindows/CVI graphical software from National Instruments was straightforward.

Likewise, the test engineer developing a new test system would like to leverage his knowledge of the technology to be tested and test methodologies to minimize the learning curve required to develop the test, like the one that I experienced when I researched the operation of a pager. Using a flexible hardware and software platform to implement the test will further reduce tester development time.

Appendix – Pager Test Soft Front Panel Source Code

```
/* preprocessor directives */
#include <utility.h>
#include <stdio.h>
#include <ansi_c.h>
#include <userint.h>
#include <nivxi.h>
#include "pager.h"
#include "ri6062.h"
#include "ri6065.h"
#include "ri3271.h"
#include "c:\vxipnp\win\include\ri3151.h"

#define  NUMCHARS              38
#define      NUMPOINTS         10000

/* CVI Driver Variables: Required to be static (global) */
static int ctrlPanel, aboutBox, setupBox, digBox;
static int panelHandle, panel2, numTries;
static int arbHandle, sgenHandle, serHandle, digHandle;
static int stsLine=0,pause=0,Abort1=0,cont=0,firstTime=0;
static ViInt16 digDat[NUMPOINTS];
static int plotHandle;
static double oldx1=0, oldx2=100;


/* function prototypes (locals) */
void init_3151(void);
void init_3271(void);
void init_6065(int);
void init_6062(void);
void doPagerTest(void);
void terminateTest(void);
void initTest(void);
int  updateInterface(void);

void main ()
{
        ctrlPanel = LoadPanel (0, "pager.uir", CTRL_PANEL);
        aboutBox = LoadPanel (0, "pager.uir", ABOUT_BOX);
        setupBox = LoadPanel (0,"pager.uir", setupPANEL);
        digBox = LoadPanel (0,"pager.uir", wavePANEL);
        DisplayPanel (ctrlPanel);
        SetCtrlAttribute(ctrlPanel,CTRL_PANEL_RUN_TEST_BUTTON,ATTR_DIMMED,1);
        init_3151();
        init_3271();
```

```
	init_6065(1);
	init_6062();
	SetCtrlAttribute(ctrlPanel,CTRL_PANEL_RUN_TEST_BUTTON,ATTR_DIMMED,0);
	while(1) { RunUserInterface (); };
}

/* init_3151(): initializes 3151 Waveform Generator
   (produces the modulation string)		*/

void init_3151()
{
	SetCtrlVal(ctrlPanel,CTRL_PANEL_STATUSBOX,"Initializing 315x...\n");
	ri3151_init ("VXI::1::INSTR", 1, 1, &arbHandle);
	ri3151_load_wavecad_file (arbHandle,
	"C:\\wcad20\\rac3151\\pagetest.cad");
}

/* init_3271(): initializes 3271 RF Signal Generator
   (transmits FSK to the pager)			*/

void init_3271()
{
	SetCtrlVal(ctrlPanel,CTRL_PANEL_STATUSBOX,"Initializing 3271...\n");
	ri3271_init ("VXI0::2::INSTR", VI_TRUE, VI_TRUE, &sgenHandle );
	ri3271_rfLevel (sgenHandle, -40.0, ri3271_RFUNIT_DBM);
	ri3271_rfFreq (sgenHandle, 929.7875e6);
	ri3271_modMode (sgenHandle, ri3271_MODE_2FSK, ri3271_PULSE_OFF);
	ri3271_modControl (sgenHandle, ri3271_MOD_ON, ri3271_PULSE_OFF);
	ri3271_setFSK (sgenHandle, 4500, ri3271_ON);
	ri3271_modFM (sgenHandle, ri3271_MOD1, 0.0, 1.00, ri3271_MODF_SIN,
				ri3271_SRC_EXTDC, ri3271_ON);
}

/* init_6065(int):initializes 6065 serial module
   (reads back pages from pager)		*/

void init_6065(int first)
{
	if (first) {
		SetCtrlVal(ctrlPanel,CTRL_PANEL_STATUSBOX,"Initializing 6065...\n");
		ri6065_init ("VXI::3::INSTR",0,1, &serHandle);
	InitVXIlibrary();
    }
	ri6065_reset(serHandle);
	ri6065_set_recv_baud_rate (serHandle, RI6065_CHAN_1, RI6065_BAUD_1200);
	ri6065_set_terminator (serHandle, RI6065_CHAN_1,
							RI6065_TERM_CHARACTER, 10);
}

/* init_6062(): initializes 6062 digitizer (digitizes pager signal) */

void init_6062()
{
	SetCtrlVal(ctrlPanel,CTRL_PANEL_STATUSBOX,"Initializing 6062...\n");
	ri6062_init ("vxi::4", 0, 1, &digHandle);
	ri6062_Input_Relays (digHandle, 1, 0, 0, 0, 0, 0, 0, 0);
	ri6062_Trig_valid (digHandle, 0, 0, 1);
	ri6062_Trig_period (digHandle, 1, 100);
	ri6062_Trig_count (digHandle, 1, 1);
	ri6062_mode_choice (digHandle, 1, 1);
	ri6062_Trig_sync_choice (digHandle, "IMM", 1);
	ri6062_Def_Trace (digHandle, 1, NUMPOINTS);
}


/* RunTest: handles the Run Test button on the control panel */
```

```
int RunTest (int panel, int control, int event,
                        void *callbackData, int eventData1, int eventData2)

{
        switch (event) {
                case EVENT_COMMIT:
                        SetCtrlAttribute(ctrlPanel,CTRL_PANEL_RUN_TEST_BUTTON,
                        ATTR_DIMMED,1);
                        SetCtrlAttribute(ctrlPanel,CTRL_PANEL_ABORT_TEST_BUTTON,
                        ATTR_DIMMED,0);
                        SetCtrlAttribute(ctrlPanel,CTRL_PANEL_PAUSE_TEST_BUTTON,
                        ATTR_DIMMED,0);
                        doPagerTest();
                        QuitUserInterface (0);
                        break;
        }
        return 0;
}

/* initTest(): reads parameters from the control panel before the test */

void initTest()
{
        double rfFreq;

        ri3271_rfState (sgenHandle, ri3271_ON);
        ri3151_output (arbHandle, RI3151_OUTPUT_ON);
        SetCtrlVal(ctrlPanel,CTRL_PANEL_RFSWITCH,1);
        SetCtrlVal(ctrlPanel,CTRL_PANEL_LED,1);
        GetCtrlVal(ctrlPanel,CTRL_PANEL_freqNUMERIC,&rfFreq);
        ri3271_rfFreq (sgenHandle, rfFreq*1e6);
}

/* doPagerTest(): controls the sequence of the test, displays results */

void doPagerTest()
{
        float startLevel, endLevel, incrLevel;
        char outbuf[512],inbuf[512],buf[1024],buf2[50];
        long x,numRead,y;
        float results[81];
        int   resultLen,level,i,j;
        static int graph=0;

        initTest();
        GetCtrlVal(ctrlPanel,CTRL_PANEL_STARTLVL,&startLevel);
        GetCtrlVal(ctrlPanel,CTRL_PANEL_FINISHLVL,&endLevel);
        GetCtrlVal(ctrlPanel,CTRL_PANEL_INCRLVL,&incrLevel);
        GetCtrlVal(ctrlPanel,CTRL_PANEL_numTries,&numTries);
        level = startLevel;
        resultLen = (endLevel - startLevel)/incrLevel + 1;
        i=0;j=0;
        while (level >= endLevel)
        {
                sprintf(outbuf,"**Paging Level**\n%idBm\n",level);
                SetCtrlVal(ctrlPanel,CTRL_PANEL_STATUSBOX,outbuf);
                SetCtrlVal(ctrlPanel,CTRL_PANEL_STATUSBOX,
                "**Xmit data**\nRacal Pager Test\n");
                ri3271_rfLevel (sgenHandle, level, ri3271_RFUNIT_DBM);
                SetCtrlVal(ctrlPanel,CTRL_PANEL_LEVELMETER,(float)level);
                results[i] = 0;
                for (j=1;j<=numTries;j++){
                        if (updateInterface()) {terminateTest();return;};
                        Delay (.3);
                        if (updateInterface()) {terminateTest();return;};
```

```
                                ri3151_trigger (arbHandle);
                                if (updateInterface()) {terminateTest();return;};
                                Delay (1.9);            /* to allow pager to receive and print data */
                                if (updateInterface()) {terminateTest();return;};
                                ri6065_read_recv_byte_count (serHandle, RI6065_CHAN_1, &numRead, &x);
                                /* using a string length based error detect scheme */
                                if ((numRead < NUMCHARS) || (numRead > 511)) {
                                        results[i] += (1-numRead/NUMCHARS)*100;
                                        sprintf(outbuf,
                                        "**Rcv data (%i)**\nFAIL\n%i bytes read\n",j,numRead);
                                        SetCtrlVal(ctrlPanel,CTRL_PANEL_STATUSBOX,outbuf);
                                        continue;
                                }
                                if (numRead>NUMCHARS) numRead = numRead-NUMCHARS;
                                results[i] += (1-numRead/NUMCHARS)*100;
                                ri6065_read_recv_data (serHandle, RI6065_CHAN_1, 2577, buf, &numRead);
                                ri6065_read_recv_data (serHandle, RI6065_CHAN_1, 2577, inbuf, &numRead);
                                inbuf[numRead-1]=0;
                                strcpy(buf,inbuf);
                                sprintf(outbuf,"**Rcv data (%i)**\n%s\n",j,buf);
                                SetCtrlVal(ctrlPanel,CTRL_PANEL_STATUSBOX,outbuf);
                                init_6065(0);
                        };
                        if (updateInterface()) {terminateTest();return;};
                        level += incrLevel;
                        results[i]/=numTries;
                        i++;
                }
                if (graph) DeleteGraphPlot(ctrlPanel,CTRL_PANEL_GRAPH,graph,
                                        VAL_DELAYED_DRAW);
                graph = PlotWaveform (ctrlPanel, CTRL_PANEL_GRAPH, results, resultLen,
                                VAL_FLOAT, 10.0, 0.0, startLevel, incrLevel,
                                VAL_FAT_STEP, VAL_SOLID_CIRCLE, VAL_SOLID, 1,
                                VAL_YELLOW);
                terminateTest();
}


/* updateInterface(): handles control panel buttons during the test */

int updateInterface()
{
                int x;

                x=0;
                while(x++<9) {
                        ProcessSystemEvents ();
                        if (pause==1){
                                pause=0;
                                SetCtrlAttribute(ctrlPanel,CTRL_PANEL_CONTINUE_BUTTON,
                                ATTR_DIMMED,0);
                                SetCtrlAttribute(ctrlPanel,CTRL_PANEL_PAUSE_TEST_BUTTON,
                                ATTR_DIMMED,1);
                                SetCtrlAttribute(ctrlPanel,CTRL_PANEL_ABORT_TEST_BUTTON,
                                ATTR_DIMMED,1);
                                while(cont==0){ProcessSystemEvents ();};cont=0;
                                SetCtrlAttribute(ctrlPanel,CTRL_PANEL_CONTINUE_BUTTON,
                                ATTR_DIMMED,1);
                                SetCtrlAttribute(ctrlPanel,CTRL_PANEL_PAUSE_TEST_BUTTON,
                                ATTR_DIMMED,0);
                                SetCtrlAttribute(ctrlPanel,CTRL_PANEL_ABORT_TEST_BUTTON,
                                ATTR_DIMMED,0);
                        } else if (Abort1==1){ Abort1=0;return 1;};
                };
                return 0;
}
```

```
/* terminateTest(): cleans up after the test is complete */

void terminateTest()
{
	ri3271_rfState (sgenHandle, ri3271_OFF);
	ri3151_output (arbHandle, RI3151_OUTPUT_OFF);
	SetCtrlAttribute(ctrlPanel,CTRL_PANEL_RUN_TEST_BUTTON,ATTR_DIMMED,0);
	SetCtrlAttribute(ctrlPanel,CTRL_PANEL_CONTINUE_BUTTON,ATTR_DIMMED,1);
	SetCtrlAttribute(ctrlPanel,CTRL_PANEL_ABORT_TEST_BUTTON,ATTR_DIMMED,1);
	SetCtrlAttribute(ctrlPanel,CTRL_PANEL_PAUSE_TEST_BUTTON,ATTR_DIMMED,1);
	init_6065(0);
}

/* PauseTest, Continue, Abort, Quit, About, AboutOK, rfSwitch:
   these routines handle buttons and switches on the control panel */

int CVICALLBACK PauseTest (int panel, int control, int event,
		void *callbackData, int eventData1, int eventData2)
{
	switch (event) {
		case EVENT_COMMIT:
			pause=1;
			QuitUserInterface (0);
			break;
	}
	return 0;
}

int CVICALLBACK Continue (int panel, int control, int event,
		void *callbackData, int eventData1, int eventData2)
{
	switch (event) {
		case EVENT_COMMIT:
			cont=1;
			QuitUserInterface (0);
			break;
	}
	return 0;
}


int CVICALLBACK Abort (int panel, int control, int event,
		void *callbackData, int eventData1, int eventData2)
{
	switch (event) {
		case EVENT_COMMIT:
			Abort1=1;
			QuitUserInterface (0);
			break;
	}
	return 0;
}

int CVICALLBACK Quit (int panel, int control, int event,
		void *callbackData, int eventData1, int eventData2)
{
	switch (event) {
		case EVENT_COMMIT:
			QuitUserInterface (0);
			exit(0);
			break;
	}
	return 0;
}
```

```
int CVICALLBACK About (int panel, int control, int event,
		void *callbackData, int eventData1, int eventData2)
{
	switch (event) {
		case EVENT_COMMIT:
			DisplayPanel (aboutBox);
			QuitUserInterface (0);
			break;
	}
	return 0;
}

int CVICALLBACK AboutOK (int panel, int control, int event,
		void *callbackData, int eventData1, int eventData2)
{
	switch (event) {
		case EVENT_COMMIT:
			HidePanel(aboutBox);
			QuitUserInterface (0);
			break;
	}
	return 0;
}

int CVICALLBACK rfSwitch(int panel, int control, int event, void *callbackData,
		int eventData1, int eventData2)
{
	int switchState;

	switch (event) {
		case EVENT_COMMIT:
			GetCtrlVal(ctrlPanel,CTRL_PANEL_RFSWITCH,&switchState);
			SetCtrlVal(ctrlPanel,CTRL_PANEL_LED,switchState);
			if (switchState) ri3271_rfState (sgenHandle, ri3271_ON);
			else ri3271_rfState (sgenHandle, ri3271_OFF);
			QuitUserInterface (0);
			break;
	}
	return 0;
}

int CVICALLBACK setup (int panel, int control, int event,
		void *callbackData, int eventData1, int eventData2)
{
	switch (event) {
		case EVENT_COMMIT:
			DisplayPanel (setupBox);
			QuitUserInterface (0);
			break;
	}
	return 0;
}

int CVICALLBACK doneSetup (int panel, int control, int event,
		void *callbackData, int eventData1, int eventData2)
{
	switch (event) {
		case EVENT_COMMIT:
			HidePanel(setupBox);
			QuitUserInterface (0);
			break;
	}
	return 0;
}

int CVICALLBACK digitize (int panel, int control, int event,
```

```
                    void *callbackData, int eventData1, int eventData2)
{
        switch (event) {
                    case EVENT_COMMIT:
                                DisplayPanel (digBox);
                                QuitUserInterface (0);
                                break;
        }
        return 0;
}

int CVICALLBACK quitDigitize (int panel, int control, int event,
                    void *callbackData, int eventData1, int eventData2)
{
        switch (event) {
                    case EVENT_COMMIT:
                                HidePanel (digBox);
                                QuitUserInterface (0);
                                break;
        }
        return 0;
}

int CVICALLBACK doDigitize (int panel, int control, int event,
                    void *callbackData, int eventData1, int eventData2)
{
        int count;

        switch (event) {
                    case EVENT_COMMIT:
                                ri3151_output (arbHandle, RI3151_OUTPUT_ON);
                                Delay(0.6);
                                ri6062_Trig_sync_start (digHandle, 1);
                                ri6062_Trig_sync_init (digHandle, 1);
                                ri3151_trigger (arbHandle);
                                Delay(1);
                                ri6062_Chan_to_array (digHandle, digDat, 1, NUMPOINTS);
                                if (!firstTime)
                                            DeleteGraphPlot (wavePANEL, wavePANEL_digitizerGRAPH, plotHandle,
                                                                                VAL_IMMEDIATE_DRAW);
                                else firstTime = 0;
                                plotHandle = PlotWaveform (wavePANEL, wavePANEL_digitizerGRAPH,
                                digDat, NUMPOINTS, VAL_SHORT_INTEGER, .0003, 0.0, 1.0, 1.0,
                                VAL_THIN_LINE, VAL_NO_POINT, VAL_SOLID, 1, VAL_GREEN);
                                ri3151_output (arbHandle, RI3151_OUTPUT_OFF);
                                break;
        }
        return 0;
}

int CVICALLBACK zoom (int panel, int control, int event,
                    void *callbackData, int eventData1, int eventData2)
{
        double x1,x2,y;
        double delta;
        int    start;

        switch (event) {
                    case EVENT_COMMIT:
                                GetGraphCursor (wavePANEL, wavePANEL_digitizerGRAPH,1,&x1,&y);
                                GetGraphCursor (wavePANEL, wavePANEL_digitizerGRAPH,2,&x2,&y);
                                delta = abs(x2-x1);
                                start = (x2>x1)?x1:x2;
                                SetAxisScalingMode (wavePANEL, wavePANEL_digitizerGRAPH, VAL_XAXIS,
                                                                                VAL_AUTOSCALE, x1, x2);
                                DeleteGraphPlot (wavePANEL, wavePANEL_digitizerGRAPH, plotHandle,
```

```
                                                                VAL_IMMEDIATE_DRAW);
                                plotHandle = PlotWaveform (wavePANEL, wavePANEL_digitizerGRAPH,
                                &digDat[start], delta, VAL_SHORT_INTEGER, .0003, 0.0, x1,
                                1.0, VAL_THIN_LINE, VAL_NO_POINT, VAL_SOLID, 1, VAL_YELLOW);

                                break;
        }
        return 0;
}

int CVICALLBACK resetZoom (int panel, int control, int event,
                        void *callbackData, int eventData1, int eventData2)
{
        switch (event) {
                case EVENT_COMMIT:
                        DeleteGraphPlot (wavePANEL, wavePANEL_digitizerGRAPH, plotHandle,
                                                                VAL_IMMEDIATE_DRAW);
                        plotHandle = PlotWaveform (wavePANEL, wavePANEL_digitizerGRAPH,
                        digDat, NUMPOINTS, VAL_SHORT_INTEGER, .0003, 0.0, 0, 1.0,
                        VAL_THIN_LINE, VAL_NO_POINT, VAL_SOLID, 1, VAL_GREEN);

                        break;
        }
        return 0;
}
```