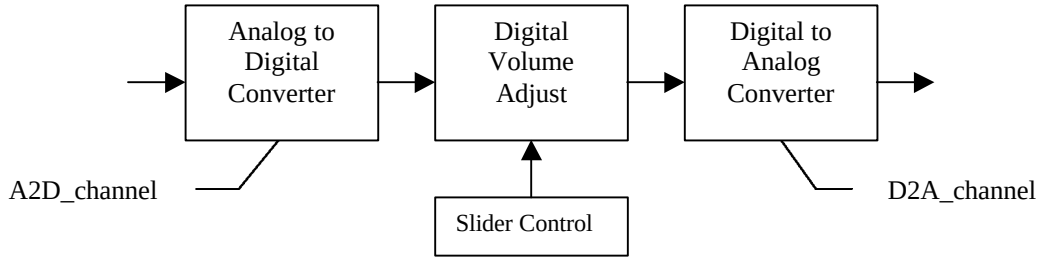
The SignalProg Test Display Example

SignalProg is an example of a testing tool implemented in Visual Basic. SignalProg implements the software equivalent of a signal generator and an oscilloscope. It is used in combination with RTDX test points to allow signal injection and testing of a DSP Algorithm. This example is constructed for use with the hostio.c tutorial example. The hostio.c example mocks a DSP algorithm whose functionality is illustrated in the diagram below.



The data processing part of the example executes on the DSP. The slider control that is used to adjust the volume is a Visual Basic application. Both the A2D and D2A converters contain RTDX calls that act as trace points. The A2D converter contains an RTDX input channel named A2D_channel. The D2A converter contains an RTDX output channel named D2A_channel. Both of these channels are disabled in the application. The SignalProg testing tool enables these channels when needed for testing purposes.

The SignalProg testing tool consists of three Active-X controls:

- A chart which displays the generated signal supplied as an input
- A chart which displays the output of the DSP function
- A button to turn the tester on and off

The major functionality of the testing tool is encapsulated in four routines.

The Test_ON Subroutine

The Test_ON subroutine is called on alternate presses of the tester button. It creates two RTDX channel objects: one for data sent to the DSP, and one for data received from the DSP. It uses the Open function to connect these channel objects to A2D_channel and D2A _channel, respectively. It then enables each of these channels using EnableChannel to begin the flow of data to and from the DSP.

It is important to remember that the tester is being used with a *running* DSP algorithm. As soon as the D2A_channel is enabled, the DSP will begin sending large amounts of result data to the tester. For this reason, the Test_ON subroutine enables the A2D_channel to the DSP first. The DSP algorithm is using a blocking read (RTDX_read) in the A2D converter. Enabling the A2D_channel first insures that the DSP algorithm will receive test data before sending results on the D2A_channel. Had the D2A_channel been enabled first, the DSP algorithm would have sent large amounts of result data before any test input was received.

Test_ON also clears the chart controls and starts the timer that is used to drive the data display functions. For user convenience, the Test_ON subroutine re-labels the push button “Test OFF”.

Creating the SignalProg Form

SignalProg is a Visual Basic program containing four controls:

1. A timer named tmr_MethodDispatch
2. A CommandButton named cmd_Toggle
3. An MSCHART named TransmittedSignal
4. An MSCHART named ReceivedSignal

To create the SignalProg program, create a new project of kind “standard exe”. Drag to resize the form to a convenient size.

Select a timer control from the Controls toolbar, then click on the form and drag to place it. The position of the timer control does not matter. Change the name property to tmr_MethodDispatch. Set Enabled to False and Interval to 1.

Select the CommandButton, then click on the form where you want it and drag to set its size. Change its name to cmd_Toggle.

The chart control used in the SignalProg example is not on the Components toolbar. Select the Project menu. From this menu select the Components item. Scroll down the Components list to the Microsoft Chart Control item. Check it and click OK. The chart control is added to the Controls toolbar.

Select the chart control, then click on the form and drag to set the size of the first control. Change its name to TransmittedSignal. Alter the ColumCount property to 1, the RowCount property to 64, and the ChartType to 3. Repeat the process to form the ReceivedSignal control.

By default the chart controls are set up to automatically scale the vertical axis to accommodate the data range. If you would prefer a fixed scale, you can adjust this feature by clicking on the Custom property.

By selecting the View menu and the Code menu item, you can now add code to your new project. The sections below describe the code supplied with the SignalProg example display.

The Test_OFF Subroutine

The Test_OFF subroutine is called on alternate pushes of the tester button. It disables the timer used to drive the display functions and changes the button label back to “Test ON”. It then disables both RTDX channels using the DisableChannel function and then closes both channels using the Close function.

The Transmit_Signal Subroutine

The Transmit_Signal subroutine sends generated test data to the DSP while displaying this data on the transmitted signal chart. In this example, the generated data is a fixed-point sine wave. The samples array is filled with 64 fixed point signal values. As each value is generated, it is plotted on the TransmittedSignal chart using the Row and Data properties. When the data has all been generated, it is sent to the DSP using the Write function.

The Receive_Signal Subroutine

The Receive_Signal subroutine reads test results from the DSP and displays this data on the received signal chart. The ReadSAI4 returns an array of 4-byte integers. The length of this array is determined by the RTDX_write operation on the DSP. When the array has been read, its contents are plotted on the received signal chart using the Row and Data properties.

The cmd_Toggle_Click Subroutine

The cmd_Toggle_Click subroutine is called whenever the test button is pressed. It alternately calls Test_ON or Test_OFF, depending upon the current state of the button.

The Form_Load Subroutine

The Form_Load subroutine is called when the test application is started. It sets the size of the charts and clears them.

The Form_Resize Subroutine

The Form_Resize subroutine is called when the user changes the size of the tester window. It adjusts the properties of the TransmittedSignal, ReceivedSignal, and cmd_Toggle controls to fill the window.

The tmr_MethodDispatch_Timer Subroutine

The tmr_MethodDispatch_Timer subroutine is called each time the timer ticks. It first calls Transmit_Signal to generate and send a test signal, and then Receive_Signal to read and display the result. Use of the timer to drive this process is a convenient way to distribute execution among the various window functions to be performed. If the tester simply called Transmit_Signal and Receive_Signal continuously in a loop, presses of the test button by the user would be ignored.