DSP II: ELEC 4523

Real-Time Data Exchange

Objectives

- In this laboratory you will become familiar with Real-Time Data Exchange (RTDX) and its use with Visual Basic (VB) to transfer data between the target DSP and the host computer.

Reading

- SPRU423 TMS320 DSP/BIOS Users Guide: Real-Time Data Exchange (section)
- Code Composer Studio Online Help: RTDX
- Code Composer Studio Online Tutorials: RTDX
- Websites dedicated to Visual Basic (VB)
    - http://www.vbsquare.com/
    - http://www.vbtutor.net/
    - http://www.vbexplorer.com/VBExplorer/VBExplorer.asp
    - http://webspace.dialnet.com/paul_pbcoms/vb/tutor.html

Lab Module Prerequisites

You should go through one of the tutorials available on the internet or read an introduction to VB book before starting this lab.

Description

RTDX

RTDX allows DSP programs to transfer data between the DSP device and a host computer without affecting the DSP application. There are components on the DSP device (target) and the host computer that allow this communication to take place. This section will describe the RTDX on the target and the next section will describe the component on the host that is used with VB. RTDX is very flexible and has many features, but only a few will be described here.

Figure 1 shows the data flow between the target and the host. When transferring data from the target to the host an output channel is first opened on the host. Data is written to this channel which causes the data to be written to a buffer set up by the RTDX target library. This buffer is then sent to the host by the RTDX library over the JTAB interface. On the host computer the data is received and either written to a buffer in memory or written to a log file. A host application can then use the data in the buffer using the RTDX host library calls.

To send data from the host to the target, the target must open an input channel. The target requests data from the input channel and the RTDX target library sends a request over the JTAG interface to the host library. Once the host application has written data to the host buffer the RTDX host library will send data to the target over the JTAG interface.

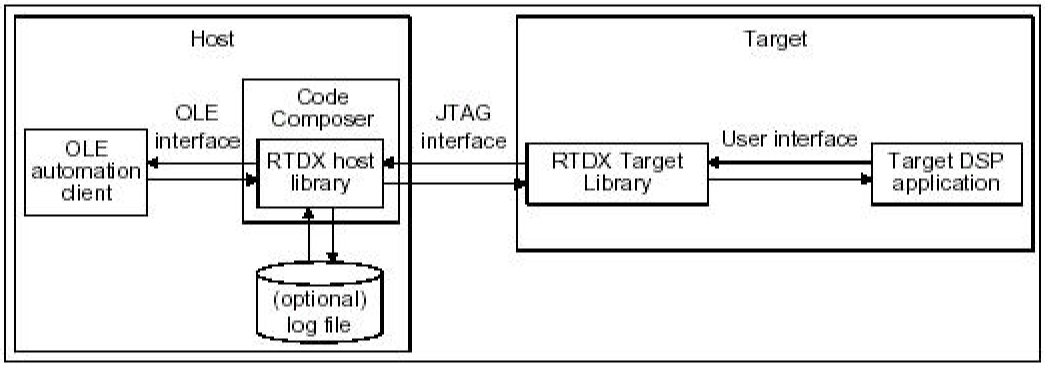

Figure 1: RTDX Data Flow between Host and Target, taken from *SPRU423 TMS320 DSP/BIOS Users Guide*

There are two modes for RTDX to receive data from the target application:

- Continuous mode - In this mode the data is written to a buffer only and continuously written to the target application. Use this mode if you want to continuously send and receive data or you don't need to save data to a log file.
- Non-continuous mode - In this mode data is written to the log file on the host. This should be used to write a finite amount of data to the log file.

To configure RTDX open the RTDX Configuration Tool by selecting Tools->RTDX->Configuration Control. This will open a window that shows the current settings for RTDX. **NOTE: RTDX will not work unless it is enabled by putting a check mark next to Enable RTDX. This must be done before running any RTDX application.** To change the settings make sure the RTDX is disabled, right click on the window and select Property Page. Here the mode can be selected and the size of the buffers determined.

The steps that need to take place in the target application to send or receive data from the host are given below.

- Include the RTDX header file `rtdx.h`.
- Declare a **global** input or output channel for receiving or sending data. The channels are data structures that are declared with the following macros. The channel name is input to the macro and can be any name.

```
RTDX_CreateInputChannel( name );
RTDX_CreateOutputChannel( name );
```

- Initialize the target. There are a couple of things that must be initialized on the target. If using the DSP/BIOS configuration tool these will both occur automatically. However, the second item will only occur after the `main` function has exited.
  - One thing that must be done is the target application must periodically call the function `RTDX_poll` to keep the data flowing between the host and target. If the configuration tool is used it sets this up for you. The HWI `HWI_RESERVED1` is used to call the function `RTDX_poll` on a periodic basis.
  - On the 'C6x RTDX uses interrupts. Therefore the NMI and global interrupts must be enabled.
- Enable the channel for reading or writing using one of the following functions.

```
RTDX_enableInput
RTDX_enableOutput
```

- Read or write the RTDX channel using one of the following function. The function `RTDX_read` is a blocking read function and the function `RTDX_readNB` is a non-blocking read function. The `RTDX_read` function actually waits in a loop for the read to be completed. It will not cause a task to enter the `TSK_BLOCKED` state.

```
RTDX_read
RTDX_readNB
RTDX_write
```

- Disable the channel after use. The functions to disable either an input or output channel are given below.

```
RTDX_disableInput
RTDX_disableOutput
```

This process is shown in the following code section where an SWI is set up to output data to the host. In this example a single integer is sent to the host one time. Examine the code section an compare to the steps outlined above. This example assumes that the configuration tool is used to set up DSP/BIOS and therefore the target does not need to be initialized.

```
/* create a global output channel */
RTDX_CreateOutputChannel( ochan );

void funSWI()
{
   int data = 10;  /* data to send to host */
   int status;

   /* enable the output channel */
   RTDX_enableOutput( &ochan );

   /* send the data to the host */
   status = RTDX_write( &ochan, &data, sizeof(data) );

   /* check the status of the write operation */
   if ( status == 0 ) {
      puts("ERROR: RTDX_write failed!\n" );
      exit(-1 );
   }

   /* disable the channel after use */
   RTDX_disableOutput( &ochan );
}
```

The next section of code shows a similar SWI set up to input an integer from the host. Notice that the structure of the code is the same as the above section of code.

```
/* create a global input channel */
RTDX_CreateInputChannel( ichan );

void funSWI()
{
   int data;
   int status;

   /* enable the input channel */
   RTDX_enableInput( &ichan );

   /* send the data to the host */
   status = RTDX_read( &ichan, &data, sizeof(data) );

   /* check the status of the read operation */
   if ( status != sizeof(data) ) {
      puts("ERROR: RTDX_read failed!\n" );
      exit(-1 );
   }

   /* use the value of the read data here */

   /* disable the channel after use */
   RTDX_disableInput( &ichan );
}
```

There are a few more functions in the RTDX module.

- `RTDX_sizeofinput` is designed to be used in conjunction with `RTDX_readNB` after a read operation has completed. The function returns the number of sizeof units actually read from the specified data channel.
- `RTDX_channelBusy` is also designed to be used in conjunction with `RTDX_readNB`. The return value indicates whether the specified data channel is currently in use or not.
- `RTDX_isInputEnabled` and `RTDX_isOutputEnabled` test whether a channel has been enabled.

Visual Basic

This laboratory will not show you how to program in VB. There are many tutorials on the internet and many books available to learn VB. It is assumed that you have gone through a tutorial on using VB before starting this lab. Descriptions of VB were written for Visual Basic 6.0.

VB Host Application Basics

The steps that need to take place in the host application to send or receive data from the target are given below.

- Define and include the RTDX return code constants. Some functions return status information and the following codes can be used to evaluate that information.

```
Const Success = &H0           ′ Method call is valid
Const Failure = &H80004005       ′ Method call failed
Const ENoDataAvailable = &H8003001E    ′ No data was available.
                     ′ However, more data may be
                     ′ available in the future
Const EEndOfLogFile = &H80030002    ′ No data was available.
                     ′ The end of the log file has
                     ′ been reached.
```

- Declare a variable of type Object.

```
Dim rtdx As Object
```

- Create an instance of the RTDX COM object.

```
Set rtdx = CreateObject("RTDX")
```

- Open a channel for reading or writing where `rtdx` is the object that was opened.

```
status = rtdx.Open("ochan", "R")
status = rtdx.Open("ichan", "W")
```

- Read or write data from the channel.

- `ReadSAI1`, `ReadSAI2`, `ReadSAI4`, `ReadSAF4`, `ReadSAF8`, `ReadSAI2V`, `ReadSAI4V`, `Read` - Read a message and put the result in a VB SAFEARRAY
- `ReadI1`, `ReadI2`, `ReadI4`, `ReadF4`, `ReadF8` - read integer or floating-point data from the data channel.
- Write - Write data from a SAFEARRAY to the host.
- `WriteI1`, `WriteI2`, `WriteI4`, `WriteF4`, `WriteF8` - Write data to the target.

- Close the channel from reading or writing where `rtdx` is the object that was opened.

```
status = rtdx.Close()
```

- Release the reference to the RTDX COM object where `rtdx` is the object that was opened.

```
Set rtdx = Nothing
```

The following code is a host application that inputs one integer at a time. In the loop the channel is read and the status returned is checked to determine the result of the read.

```
Const Success = &H0                   ’ Method call is valid
Const Failure = &H80004005            ’ Method call failed
Const ENoDataAvailable = &H8003001E   ’ No data was available.
                                      ’ However, more data may be
                                      ’ available in the future.
Const EEndOfLogFile = &H80030002      ’ No data was available
                                      ’ The end of the log file has
                                      ’ been reached.

Sub main()
   Dim rtdx As Object   ’COM object
   Dim data As Long
   Dim status As Long

   ’If an error occurs then go to the following function
   On Error GoTo Error_Handler
   ’Create the COM object
   Set rtdx = CreateObject("RTDX")
   ’Open the COM object
   status = rtdx.Open("ochan", "R")

   ’Check the status of the open function
   If status <> Success Then
      Debug.Print "Opening of channel ochan failed"
      GoTo Error_Handler
   End If

   Do
      ’Read an integer from the RTDX input channel
      status = rtdx.ReadI4(data)

      ’Check the status of the read
      Select Case (status)
         Case Success
               Debug.Print "Value " & data & " was received from the target"
         Case ENoDataAvailable
         Case EEndOfLogFile
            Debug.Print "End of log file has been detected"
         Case Failure
            Debug.Print "ReadI4 returned failure"
            Exit Do
         Case Else
            Debug.Print "Unknown return code"
            Exit Do
      End Select
   Loop Until status = EEndOfLogFile

   ’close the input channel
   status = rtdx.Close()
   ’Release the reference to the RTDX COM object
   Set rtdx = Nothing
Exit Sub

Error_Handler:
   Debug.Print "Error in COM method call"
   Set rtdx = Nothing
End Sub
```

VB Application Development

Visual Basic is designed to develop event driven applications. Events occur when something is triggered by the user’s actions, by messages from the system or other applications, or even from the application itself. Different code is executed depending on the event that occurs. Your application will have to handle events such as someone clicking on a button. We will use event driven applications in this laboratory.

In an event driven VB project the main module you will use will be form modules. Form modules (.frm file name extension) can contain textual descriptions of the form and its controls, including their property settings. They can also contain form-level declarations of constants, variables, and external procedures; event procedures; and general procedures.

Laboratory

Part 1

- In this part you will generate a simple VB application without interfacing to RTDX. This will get you familiar with generating a VB application. The application will have a button that when pressed will write some text to a text box.
- Start VB and select Standard EXE. This will open a new project with a default form.
- Save your project. This will save two files. Call them vb1.frm and vb1.vbp.
- Click on the button . This is used to add a command button. On the form click and drag to add a command button.
- With the button selected change the following properties in the properties view:
  - (Name): Print
  - Caption: Print
- Now add a text box by clicking on the button and then clicking and dragging on the form.
- With the text box selected change the following properties in the properties view:
  - (Name): TextPrint
  - Text: Default Text

- Now double click on the button. This will bring up a code window that will have the subroutine that handles the event of clicking on the button. It is called `Print_Click` since the object name is Print and it handles a click. In this window the upper left pull down menu selects different objects and the upper right pull down menu selects the code that handles different events.
- In the function `Print_Click` add code that will change the text of the text box. This is done by the following code.

```
TextPrint.Text = "Printed Text"
```

- Note that the object name, TextPrint, is followed by the property to be changed, Text.
- Run the VB application and test the functionality.
- Try changing the other event handlers to change the text to read something different.

Part 2

- In this part you will be creating a VB application that communicates with a target application through the RTDX. The VB application will have two buttons that when pressed send different numbers to the target. The target application will print the value received to a LOG object.
- First create a VB application.
- Start VB and select Standard EXE. This will open a new project with a default form.
- Save your project. This will save two files. Call them vb2.frm and vb2.vbp.
- Add two command buttons to your form with the following characteristics:
  - (Name): Command1
  - Caption: 1
  - (Name): Command2
  - Caption: 2
- In the Form code in the (General):(Declarations) section add code for the status constants and other variables. The code should be:

```
Option Explicit            ' Require all variables to be explicitly declared

'---------------------------------------------------------------------------
' RTDX Return Status
'---------------------------------------------------------------------------
Const Success = &H0                     ' Method call is valid
Const Failure = &H80004005              ' Method call failed
Const ENoDataAvailable = &H8003001E     ' No data currently available.
Const EEndOfLogFile = &H80030002        ' End of transmission

'---------------------------------------------------------------------------
' Variable Declarations
'---------------------------------------------------------------------------
Dim rtdx As Object                      ' Holds the rtdx object
Dim bufferstate As Long                 ' Holds the number of bytes xmitted
                                        ' or pending xmission
Dim status As Long                      ' RTDX Function call return status
```

- The first function to be executed when a project is started is Form:Load. In this function put the code for initializing the RTDX channel. Call the channel HtoTchan for Host to Target channel.

```
'If an error occurs then go to the following function
On Error GoTo Error_Handler
```

```
   ’Create the COM object
   Set rtdx = CreateObject("RTDX")
   ’Open the COM object
   status = rtdx.Open("HtoTchan", "R")

   ’Check the status of the open function
   If status <> Success Then
      Debug.Print "Opening of channel HtoTchan failed"
      GoTo Error_Handler
   End If
Error_Handler:                          ’ All errors should be handled here
    Debug.Print "Error in Form_Load"
    Set rtdx = Nothing
    End                                 ’ Force program termination
End Sub
```

- One of the last function to execute the Form:Unload function. In this function put the code for cleaning things up.

```
status = rtdx.Close()                    ’ Close rtdx
Set rtdx = Nothing                       ’ Free memory reserved for rtdx obj
```

- In the Command1:Click function put the code to be executed when the button is clicked. When the button is clicked it should send a 4 byte integer number 1 to the target indicating that button 1 was clicked. The code for doing this is:

```
Dim Data As Long

Data = 1
status = rtdx.WriteI4(Data, bufferstate)

If status = Success Then
   Debug.Print "Value " & Data & " was sent to the target"
Else
   Debug.Print "WriteI4 failed"
   End If
```

- In the Command2:Click function put the code to be executed when the button is clicked. When the button is clicked it should send a 4 byte integer number 2 to the target indicating that button 1 was clicked. Change the above code to send a 2 to the target.
- Now the target code needs to be generated.
- Create a new project called `rtdxlab`.
- Create a new DSP/BIOS Configuration file and use the `C6xxx.cdb` template for use with the simulator.
- Save the file as `rtdxlab.cdb` and add it in your project. Also add the `rtdxlabcfg.cmd` file.
- If using the simulator then change the RTDX interface to Simulator by right clicking on Input/Output->RTDX and bringing up the properties. Change the RTDX mode to Simulator. If you do not do this then when you load your program you will see the error "RTDX application does not match emulation protocol." If you are loading onto an EVM or DSK you shouldn’t need to change this.
- Create a LOG object by right clicking on Instrumentation->LOG and selecting Insert LOG. Change the name to `trace`. Set its properties to have a length of 512 and be a circular buffer.
- Create a task called `monitorTSK` that calls the function `funmonitorTSK`.

- Create a `main.c` file and include a `main` function that prints to `trace` a message saying the run is starting. Include this file in your project.
- Create a global input channel called `HtoTchan`.

```
RTDX_CreateInputChannel( HtoTchan );
```

- In the `main.c` file make a function for your TSK, `funmonitorTSK`. Add to the function code to
  - Enable the input channel
  - Add an infinite loop that uses `RTDX_read` to read from the channel and then print the result if the status is the size of the data (int).
  - The code should look like:

```
int data;
int status;

/* enable the input channel */
RTDX_enableInput( &HtoTchan );

/* receive an integer from the host */
while(1)
{
   status = RTDX_read( &HtoTchan, &data, sizeof(data) );


   if ( status != sizeof(data) ) {
      LOG_printf(&trace,"ERROR: RTDX_read failed!\n" );
      exit( -1 );
   } else
      LOG_printf(&trace,"Value sent = %d",data);

}
```

- Build and load your project.
- Use the LOG manager to examine the processing.
- Enable RTDX by opening the Configuration window with Tools->RTDX->Configuration Control and checking the Enable RTDX box.
- Run the target application. It is important that the target application be started before the VB application. If the VB application is started first the channel may not get initialized.
- Run the VB application.
- Click on the buttons a few times. You will probably see that nothing seems to happen. However, if you halt the processor the LOG window should get updated with values that you clicked. The reason for this is the `RTDX_read` function waits in a loop to get the data. Therefore the idle task does not get to run and send over the `LOG_print` data.

Part 3

- In this part you will modify the code from the previous part to use `RTDX_readNB` and cause the task to sleep if the data isn't ready. This will allow the idle task (or other tasks) to run.
- Copy the `main.`c file from above to a new file `mainNB.c`. Remove the `main.c` from your project and add `mainNB.c`.
- Change the code in the function `funmonitorTSK` to the following:

```
int data;
int status;
int busystatus=0;

/* enable the Host to Target channel */
RTDX_enableInput( &HtoTchan );

/* receive an integer from the host */
while(1)
{
   /* check to see if the channel is busy before the read */
   if (busystatus == 0)
   {
      /* Print the data if something was actually read */
      if (RTDX_sizeofInput(&HtoTchan) == sizeof(data))
      {
         LOG_printf(&trace,"Value sent = %d",data);
      }
      status = RTDX_readNB( &HtoTchan, &data, sizeof(data) );
   }

   /* get the status of the channel */
   busystatus = RTDX_channelBusy(&HtoTchan);

   /* If the channel is busy then sleep for a time */
   /* This is done so that other tasks will have time to run */
   if (busystatus == 1)TSK_sleep(1);

}
```

- This code first checks to see the status of the RTDX read. If it is not busy then the `RTDX_sizeofInput` function is used to see if or how much data was read. If the correct amount was read then the value is printed. Following this the channel is read again. After the channel is read the busystatus is checked. If the status is busy then the task will sleep for a short time. This time is set very short so that the function will not take too long on the simulator.
- Build and load your project.
- Use the LOG manager to examine the processing.
- Enable RTDX by opening the Configuration window with Tools->RTDX->Configuration Control and checking the Enable RTDX box.
- Run the target application.
- Run the VB application.
- Click on the buttons a few times. This time you should see that the button value is printed. If you are using the simulator the delay may be significant since it is much slower than a real processor.

Part 4

- In this part you will modify an existing program that filters real-time data from the CODEC. The application will switch between two different filters based on the button clicked.
- This needs to be completed.