Using RTDX with a MATLAB GUI*

Thomas Shen

Based on *Using the Serial Port with a MATLAB GUI*† by

Jason Laska

Matthew Berry

Daniel Sachs

This work is produced by OpenStax-CNX and licensed under the
Creative Commons Attribution License 2.0‡

Abstract

Explains how to send and receive data with the DSP through the USB port using RTDX. Example code is shown in C as well as MATLAB for interfacing on the PC. Some example code for creating a MATLAB GUI is also shown.

1 Introduction

The USB port on the DSP box can also be used to transmit data between the DSP and the PC during real-time operation. Texas Instruments came up with Real Time Data Exchange (RTDX) to allow users real-time bidirectional exchange of data between the target and host. This allows the simulation of data input and output which can be used as feedback from the DSP for a variety of applications. Both input and output data are buffered until read, allowing transmission of larger amounts of data.

2 Setting Up Input/Output Channels

The RTDX works by setting up input and output channels to the DSP. RTDX functionality is supplied through the include file `rtdx.h` so be sure it is included in your `main.c` and any other files that use rtdx functions.

Depending on whether there will be input and/or output from the computer in your project, add input and output channels in the `main.c` file using the commands. These are declared like global variables near the top of the code after the include files. They do NOT go in the main() function or any other function.

```
    RTDX_CreateInputChannel(ichan);
RTDX_CreateOutputChannel(ochan);
```

*Version 1.5: Mar 26, 2008 3:56 pm -0500
†http://cnx.org/content/m12062/1.1/
‡http://creativecommons.org/licenses/by/2.0/

By default, these channels are disabled on the DSP. You may enable them in the main loop so that they will be enabled for the duration of the program. This can be accomplished with the following instructions:

```
    RTDX_enableInput(&ichan);
RTDX_enableOutput(&ochan);
```

In other C files that utilize the declared input and output channels, you will need to declare the input and output channels again with an extern so that your files know what the variables are.

```
    extern RTDX_input_channel ichan;
extern RTDX_output_channel ochan;
```

Lastly, RTDX **MUST** be manually enabled on the DSP boards. Go to `Tools->RTDX->Configuration Control` and select 'Enable RTDX' in the window that opens. Another helpful window for debugging is the 'Channel Viewer Control' accessed through `Tools->RTDX->Configuration Control`. This window displays the number of completed and outstanding transfers between computer and board.

3 Using the DSP to Access the USB Port

The data buffer can be written to in C, but requires the block method of input/output used in Lab 4[1] and Lab 5[2] . The sample-by-sample method used in Labs 1 through 3 will not work with RTDX. RTDX functionality is supplied through the include file `rtdx.h` so be sure it is included in your `main.c` file and any other files that use rtdx functions.

3.1 Using C to Send/Receive

There are several functions for transmitting and receiving data within the C environment:

- `RTDX_readNB()` takes three arguments: the first is a pointer to the input channel, the second is a pointer to the variable in which to store read data, and the third is the number of bytes to read. This is a non-blocking read, meaning if no data is available to be read, it will merely return. However, there will then be an outstanding read and the variable will be updated when some data is finally read. It returns 'RTDX_OK' on success, '0' on failure (the target buffer is full), and 'RTDX_READ_ERROR' when the channel is currently busy reading.
- `RTDX_read()` also takes three inputs like `RTDX_readNB()` and but on successful read, it returns the number of bytes of data actually in the buffer. The difference from `RTDX_readNB()` is it's a blocking read, meaning `RTDX_read()` won't return until something is read. If the channel is busy or not enabled, 'RTDX_READ_ERROR' will be returned.
- `RTDX_write()` takes three arguments: the first is the pointer to the output channel, the second is a pointer to the buffer containing the data to write, and the third is the size of the buffer in bytes. It returns an integer, non-zero on success and '0' on failure.
- `RTDX_sizeofInput()` takes a pointer to an input channel and returns the number of bytes of data actually read from the buffer. It is used in conjunction with `RTDX_readNB()` after a read operation is completed.
- `RTDX_channelBusy()` takes a pointer to an input channel and returns an int indicating the status of the channel. A return of '0' means the channel is not busy while non-zero means the channel is busy. This is usually used in conjunction with `RTDX_readNB()` to check if another read request needs to be issued.

[1] http://cnx.org/content/m13809/latest/
[2] http://cnx.org/content/m13805/latest/

More information about the RTDX module and the commands that can be used with it are in the TMS320 DSP/BIOS User's Guide (spru423)[3] and the TMS320C5000 DSP/BIOS API Reference Guide (spru404)[4] .

Example 1

The following example shows a simple C program that will echo received data back through the output channel. This files assumes that the `main.c` file has declared and enabled the input and output channels. The project file and all the necessary files are available from `v:/ece420/55x/block_rtdx`. MATLAB GUI files that are made to interface with this project are rtdx_text.m[5] and rtdx_echotext.m[6] .

```
    #include "dsk5510_dual3006cfg.h"
#include "dsk5510.h"
#include "swi_process.h"
#include "dsplib.h"
#include "rtdx.h"                        // Include file for rtdx functionality

extern RTDX_input_channel ichan;         //ichan has been declared in main.c
extern RTDX_output_channel ochan;        //ochan has been declared in main.c

int recvd;
int sentNew = 0 ;

// all data processing should be done in SWI_ProcessBuffer

void SWI_ProcessBuffer()
{
static unsigned int mbox_value = 0;
short *psrc, *pdest;

mbox_value |= SWI_getmbox();

// buffers are only processed when both transmit and receive are ready
if((mbox_value & DMA_RECEIVE_DONE) && (mbox_value & DMA_TRANSMIT_DONE)) {
  mbox_value = 0;

// get buffer pointers
psrc = receive_buffer[receive_buffer_to_process_index];
pdest = transmit_buffer[transmit_buffer_to_fill_index];

if (!RTDX_channelBusy(&ichan)) {     // read only when not busy
     RTDX_readNB(&ichan, &recvd, sizeof(recvd));
     sentNew = 1;
}

if (sentNew == 1) {  // echo back when data has been received
RTDX_write(&ochan, &recvd, sizeof(recvd));
sentNew = 0;
}
```

[3] http://focus.ti.com/lit/ug/spru423f/spru423f.pdf
[4] http://focus.ti.com/lit/ug/spru404g/spru404g.pdf
[5] http://cnx.org/content/m14388/latest/rtdx_text.m
[6] http://cnx.org/content/m14388/latest/rtdx_echotext.m

```
receive_buffer_processed = 1; // flag receive buffer as processed
transmit_buffer_filled = 1; // flag output buffer as full
}
}
```

4 Using MATLAB to Access the DSP Board (PC)

MATLAB can be used to access the data coming from the DSP board. A simple typing/echo GUI for the block_rtdx project has been provided. An interface can also be programmed in Visual Basic. The setup and transfer/receive commands for Matlab will be described below.

4.1 Sending Data

Before accessing the DSP board, it must be initialized through MATLAB. This is done with this code:

```
h = actxserver('RTDX');
```

which sets the port with all necessary parameters. The port is still not open for writing. To open the port, specify the name of the channel you would like to open in the command:

```
invoke(h,'Open','ichan','W');
```

To write to the buffer, you invoke h with a 'Write' parameter and the data to send.

```
invoke(h,'Write',int16(v));
```

In this case, v was a char and we wanted to send the ASCII value. (There is a limitation to the ASCII converter in Matlab: it does not take all possible key presses.) Multiple channels can be opened in this manner. When there are multiple channels open, the write will be to the most recently opened channel.

Before finishing a function, or before executing a read, the port should be closed. The port is closed with the command:

```
invoke(h,'Close');
```

4.2 Receiving Data

To read data from the DSP, a read channel must be set up. This is done using the invoke open command with an 'R' parameter.

```
invoke(h,'Open','ochan','R');
```

Reading data from the DSP board is a little more complicated. It seems that the DSP board buffers all the data. To get the latest piece of data, you must first 'Seek' to the current message.

```
[status,nummsgs] = invoke(h,'GetNumMsgs');
status = invoke(h,'Seek',nummsgs);
```

Once at the correct message, the actual reading can be done.

```
[status, values] = invoke(h,'ReadI2');
```

As with writing, the port should be closed after reading.

5 Using MATLAB GUI Features

MATLAB has some nice Graphical User Interface (GUI) features which can be used to control the flow of data to and from the RTDX port. The basic implementation consists of a blank window (figure) which can have different interface elements placed on it. These elements can be sliders, buttons, text boxes, etc...

When an element is accessed (for instance, a slider is moved, or a button is pushed), MATLAB will execute a "callback routine" which is a MATLAB function defined by the user. Desgining these interfaces is simple.

5.1 Creating a User Interface with Sliders

Download These Files

- rtdx_sliders.m[7] - User Interface
- rtdx_wrt_sliders.m[8] - Callback File

Example 2

```
1   % rtdx_sliders - initializes RTDX port and sets up three sliders
2
3   h = actxserver('RTDX');
4
5   % open a blank figure for the slider
6   Fig = figure(1);
7   % open sliders
8   % first slider
9   sld1 = uicontrol(Fig,'units','normal','pos',[.2,.7,.5,.05],...
10       'style','slider','value',4,'max',254,'min',0,'callback','rtdx_wrt_sliders');
11
12   % second slider
13   sld2 = uicontrol(Fig,'units','normal','pos',[.2,.5,.5,.05],...
14       'style','slider','value',4,'max',254,'min',0,'callback','rtdx_wrt_sliders');
15
16   % third slider
17   sld3 = uicontrol(Fig,'units','normal','pos',[.2,.3,.5,.05],...
18       'style','slider','value',4,'max',254,'min',0,'callback','rtdx_wrt_sliders');
```

Lines 9 through the end create the three sliders for the user interface. Several parameters are used to specify the behavior of each slider. The first parameter, Fig, tells the slider to create itself in the window we created in Line 6. The rest of the parameters are property/value pairs:

- `units`: Normal tells Matlab to use positioning relative to the window boundaries.
- `pos`: Tells Matlab where to place the control.
- `style`: Tells Matlab what type of control to place. slider creates a slider control.
- `value`: Tells Matlab the default value for the control.
- `max`: Tells Matlab the maximum value for the control.
- `min`: Tells Matlab the maximum value for the control.
- `callback`: Tells Matlab what script to call when the control is manipulated. rtdx_wrt_sliders is a Matlab file that writes the values of the controls to the RTDX port.

[7] http://cnx.org/content/m14388/latest/rtdx_sliders.m
[8] http://cnx.org/content/m14388/latest/rtdx_wrt_sliders.m

Every time a slider is moved, the rtdx_wrt_sliders.m file is called:

Example 3

```
1   % rtdx_wrt_sliders : writes values of sliders out to rtdx
2
3   % open rtdx port for data transfer
4   status = invoke(h,'Open','ichan','W');
5
6   % send value from sld1
7   v1 = round(get(sld1,'value'));
8   status = invoke(h,'Write',int16(v1));
9
10   % send value from sld2
11   v2 = round(get(sld2,'value'));
12   status = invoke(h,'Write',int16(v2));
13
14   % send value from sld3
15   v3 = round(get(sld3,'value'));
16   status = invoke(h,'Write',int16(v3));
17
18   % send reset pulse
19   status = invoke(h,'Write',int16(2989));
20
21   % close rtdx port
22   status = invoke(h,'Close');
```

Line 7 retrieves the value from the slider using the get function to retrieve the value property. The value is then rounded off to create an integer, and the integer is sent as an 16-bit quantity to the DSP in Line 8. The other two sliders are sent in the same way. Line 19 sends 2989 to the DSP, which can be used to indicate that the three previously-transmitted values represent a complete set of data points. (You can use whatever value you want.) This can be used to prevent the DSP and Matlab from losing synchronization if a transmitted character is not received by the DSP and provides some error detection.

NOTE: Line 22 closes the RTDX port. Make sure you close the port after sending a data block to the DSP.

5.2 Advanced features

The slider example shows some basic features of the gui tools. The handle for the RTDX server is generated into the workspace so that it can be used for writing. But other elements, such as text boxes cannot be dealt with as easily. The Parameters from these can be accessed through their returned handles. Some examples:

Example 4

```
%GUI.m
%****Sample GUI, Text and a Button***

%open a blank figure
Fig = figure(1);
set(Fig,'Name','Test GUI');
```

```
%Space to enter text
ed2 = uicontrol(Fig,'backgroundcolor','white','units','Normalized','pos',[.1,.6,.4,.05],...
                'string','Default Text','style','edit');

%Button
but1 = uicontrol(Fig,'foregroundcolor','blue','units','Normalized','pos',[.1,.4,.5,.1],...
                'string','Press Me!','style','pushbutton','callback','SampleGUI');
```

A Text box is created with default text in it that says: "Defaul Text". A button is also created, which when pressed, will execute the callback function SampleGUI.m

```
%SampleGUI.m

%Get Text
testText = get(ed2,'string')
```

Now testText holds whatever string was entered into the text box. The function `get()` is used to retrieve the data from the 'string; parameter in the ed2 handle. MATLAB `help uicontrol` gives the full list of options for interface elements.