Serial Communication: Sending Data

Overview:

External mode in Simulink allows data to be sent and received from the target board. This communication protocol has significant overhead which limits how fast data can be transferred. With an Arduino Mega sending one or two channels of information in a simple loop the maximum sampling rate is around 30Hz. To obtain information at a faster rate the data must be send directly from the target without using external mode.

This lab explores sending data with Simulink using both built in blocks and Sfunction and with the arduino IDE. Sending floats is accomplished with a SFunction using a union to create the bytestream.

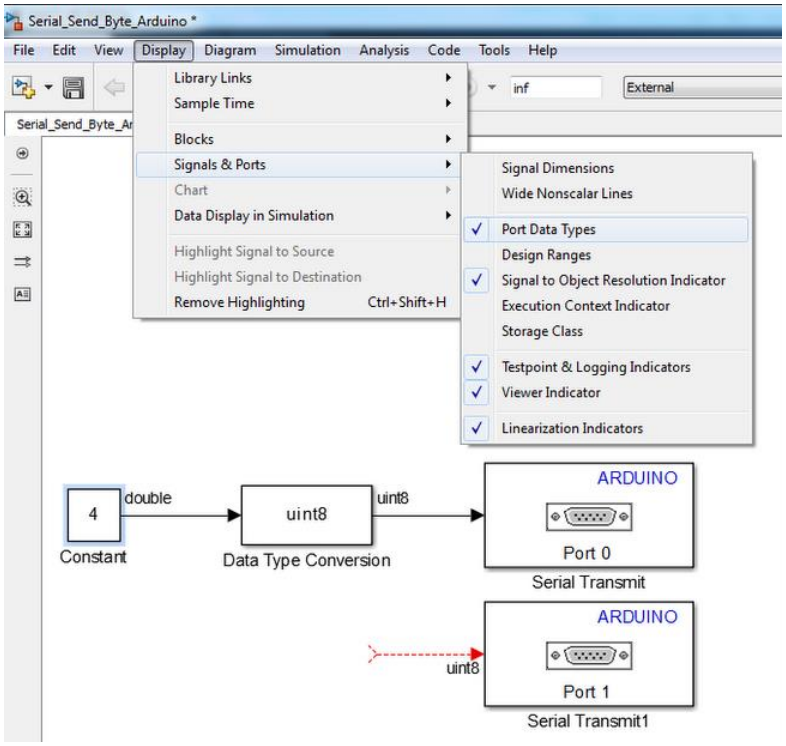
Part 1: Sending 8 bit data with Simulink

Objectives:

- Send 8 bit data over the serial link using the supported Simulink blocks

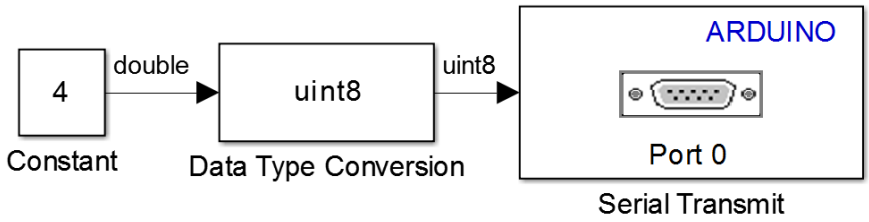
Simulink Model:

- The Simulink Arduino library block "Serial Transmit" can be used to send single bytes at a time. To see this select "Display – Signals & Ports – Port Data Types" from the Display menu: (press control+D to update the diagram)



The input data type for a Serial Transmit is uint8. This means that data must be converted to unit8 before sending across the serial line with this block.

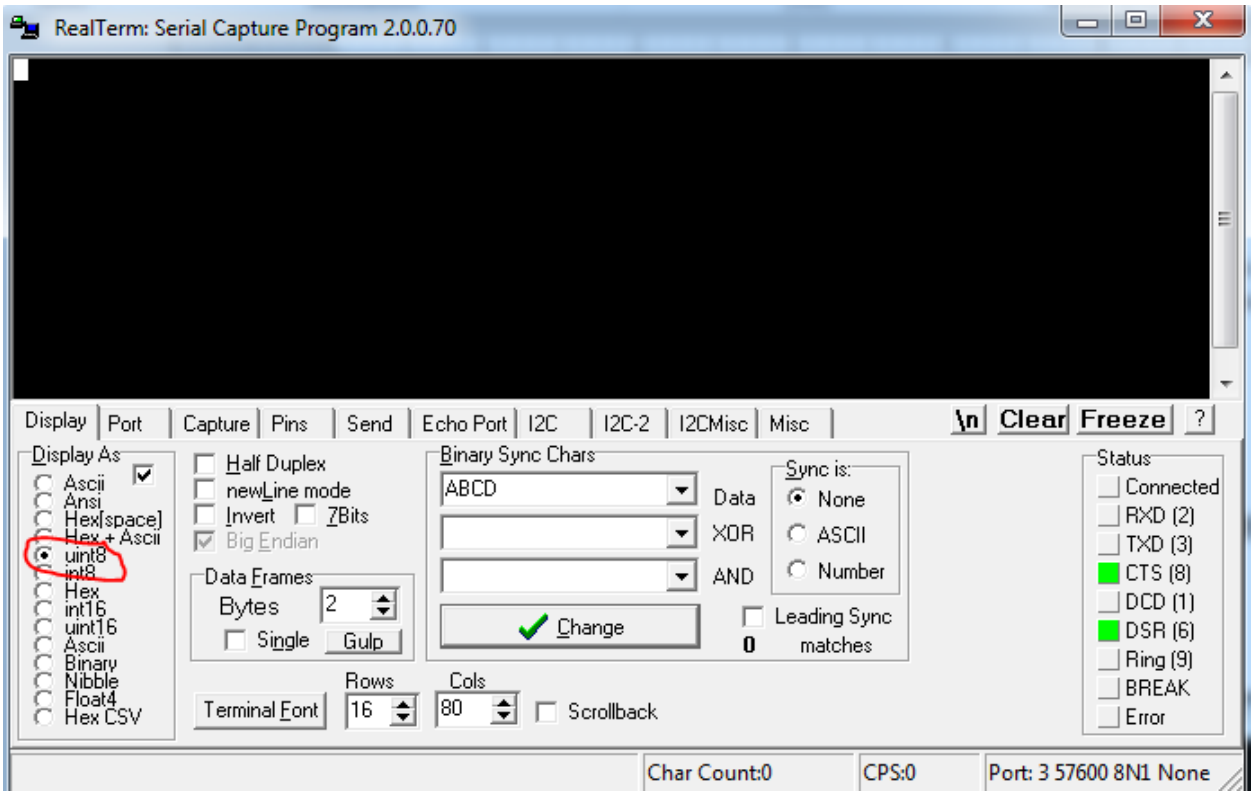
- Build and run the following Simulink diagram.
  - Make sure external mode is NOT enabled. If you try to download and run the code with external mode enabled it will give you an error
  - The baud rates for the ports are set in the Configuration Parameters in the Run On Target Hardware tab – by default they are 9600
  - Set the sample time of the simulation to .1 second – this means it will send the number 4 in binary every .1 seconds through port 0 (which is the USB cable)



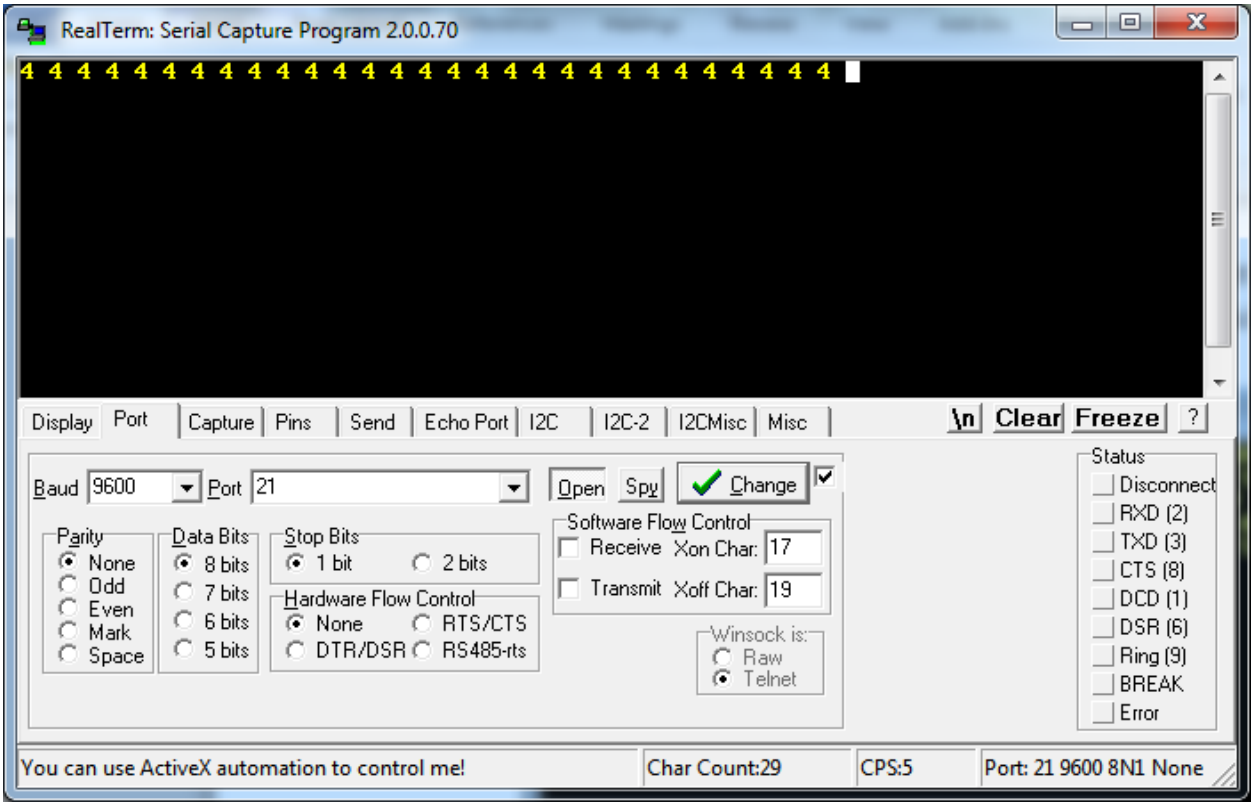
  - After the code is downloaded you should see the TX LED blinking indicating data is being sent across the serial line

Reading the Data with RealTerm:

To obtain this data RealTerm is used – although any serial terminal program will work. After installing RealTerm open it in administrator mode.

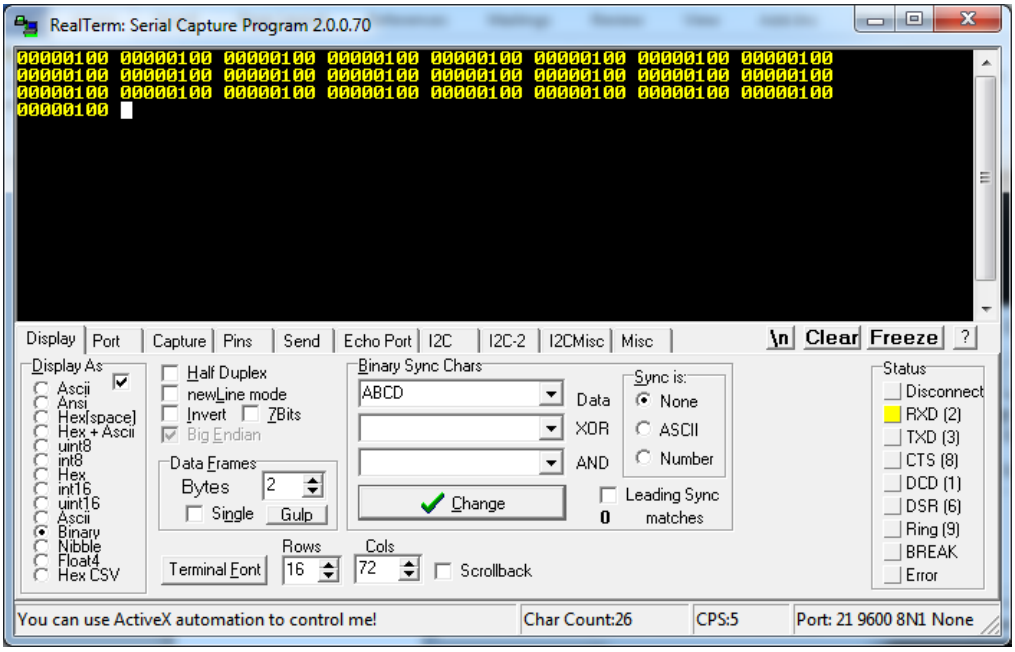


- On the left select “uint8”. This indicates to realterm that the data type to expect is uint8 and it will display the results appropriately.
- Click the “port” tab, change the baud rate to 9600
- select the serial port of your hardware
- If the Open tab is depressed (as in the figure below) click it once to “close” the port, and again to “open” it. It should then start displaying data:



The number 4 is displayed, but there is no indication as to what the actual binary data was received. To see this

- go back to the Display tab and select Binary



Now the actual binary representation of the data is seen: 00000100 is the number 4.

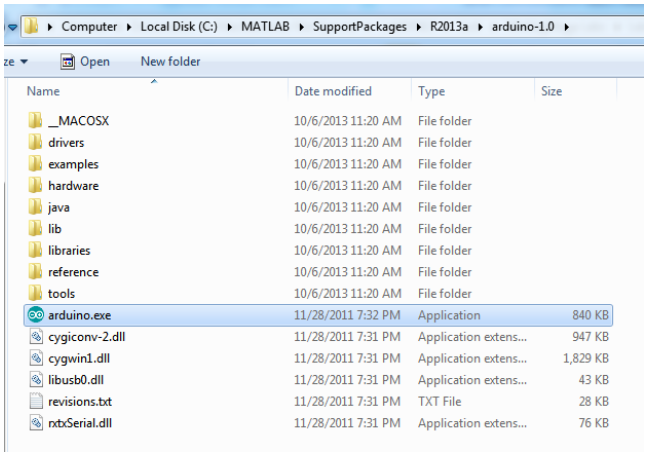
Part 2: Sending data with the Arduino IDE

Objectives:

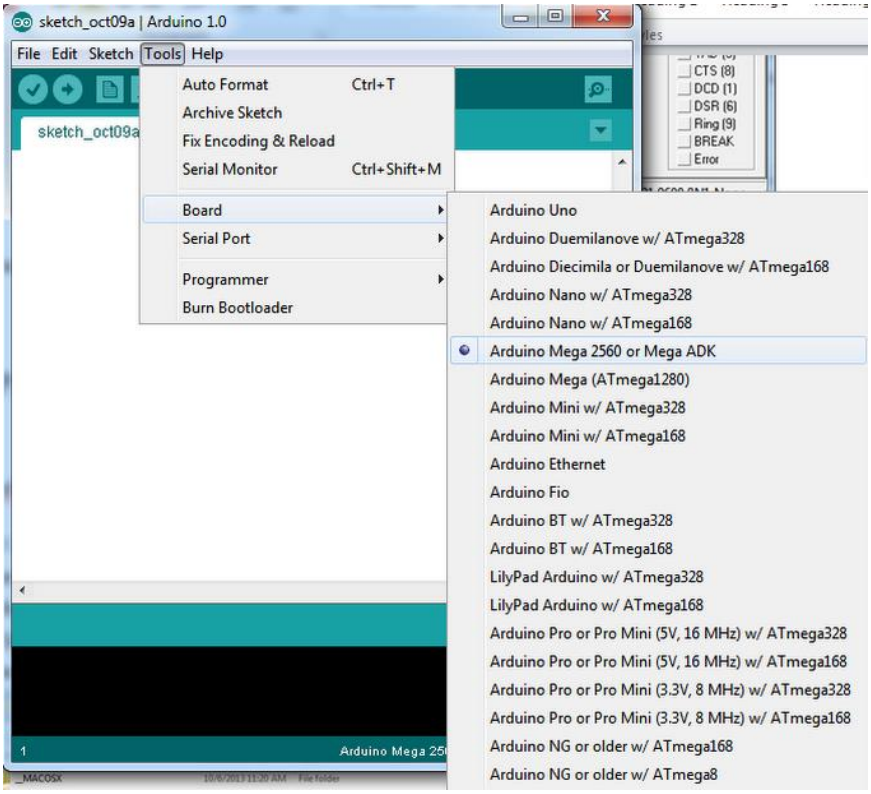
- Use the Arduino IDE to send ascii encoded data and binary data

Arduino Code:

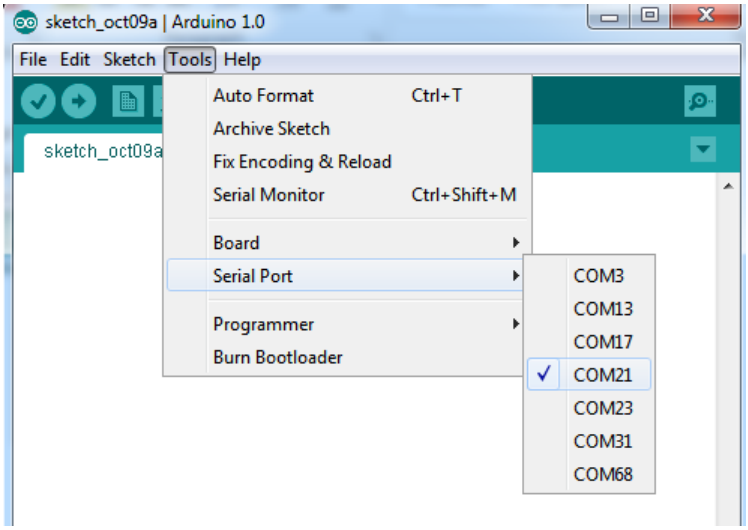
- Matlab uses the Arduino IDE and Arduino libraries. When it installs the Arduino Simulink blocks it installs the Arduino IDE typically in this location
    - C:\MATLAB\SupportPackages\R2013a\arduino-1.0
    - The Arduino IDE can be opened from here:



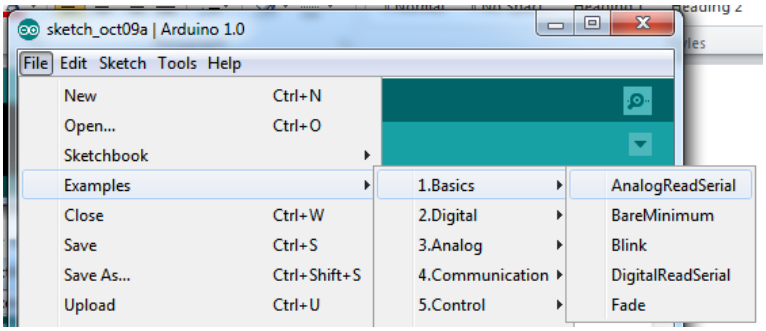
- Select your target board:



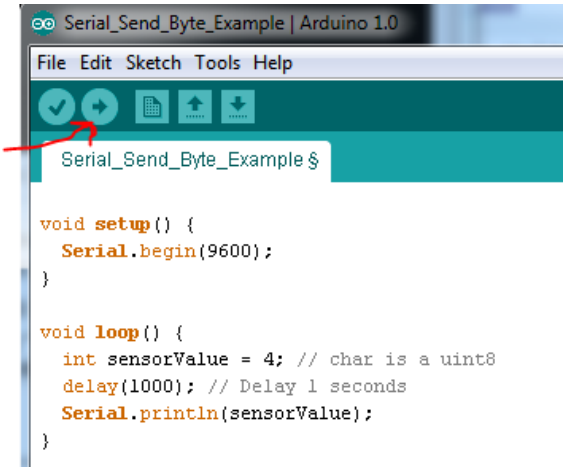
- Select the serial port:



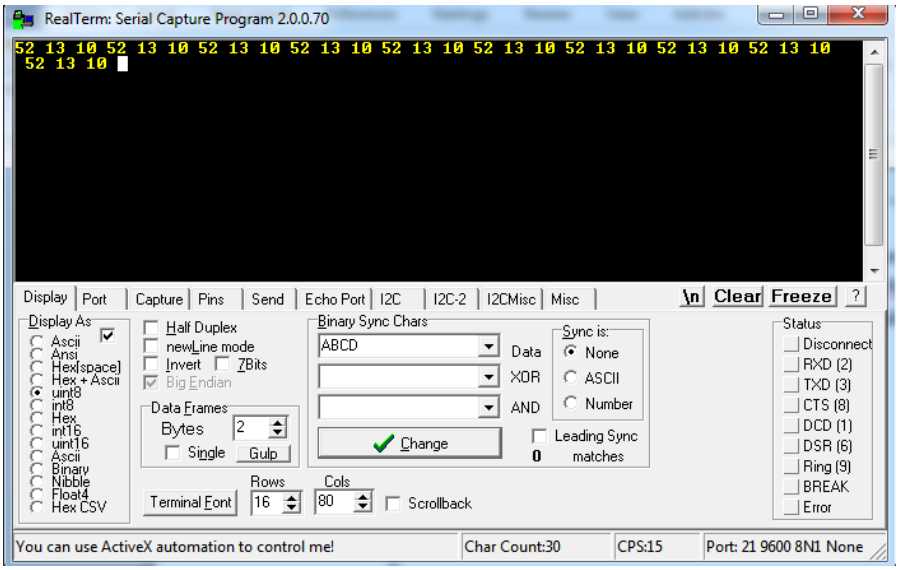
- Open a basic serial example:



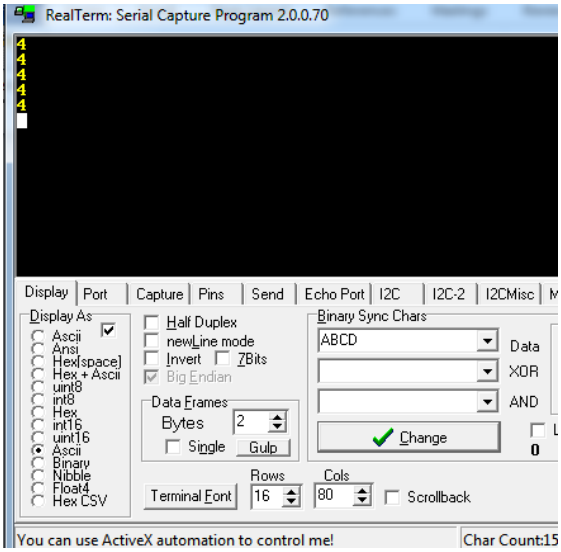
- Modify the code to send only the number 4:
  - Download the code to the board with the Right arrow
  - Note – you must make sure the serial port from RealTerm is closed – it cannot download code when the serial port is open in another application



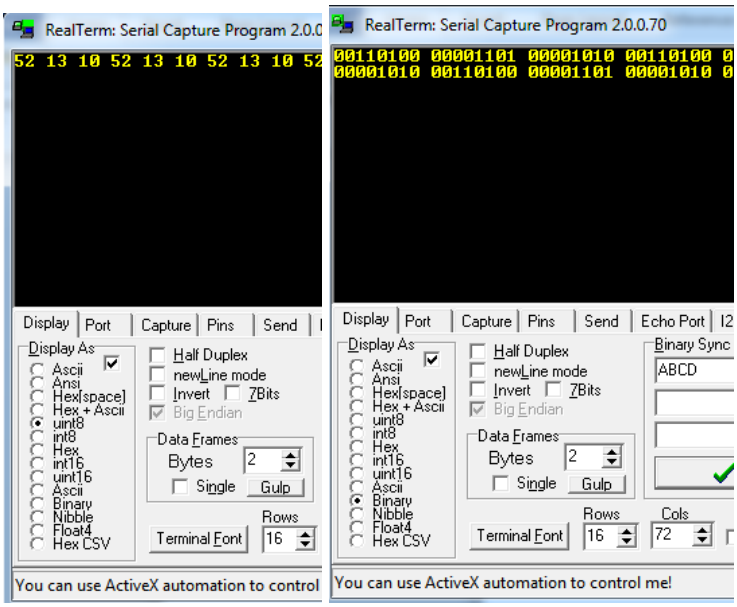
  - Open the terminal program to view



The data is not as expected. Now click on Ascii for the Display as format



The display shows 4. When the 'Serial.println()' command is used the data is encoded to human readable ASCII characters. The data (4) is encoded to the character '4' which is represented in decimal as 52 (101010). In addition each 4 is written on a new line which is a carriage return followed by a new line 13 (00001101) and 10 (00001010):



To write the data in binary (that is 4 as 0000100) use the Serial.Write command:

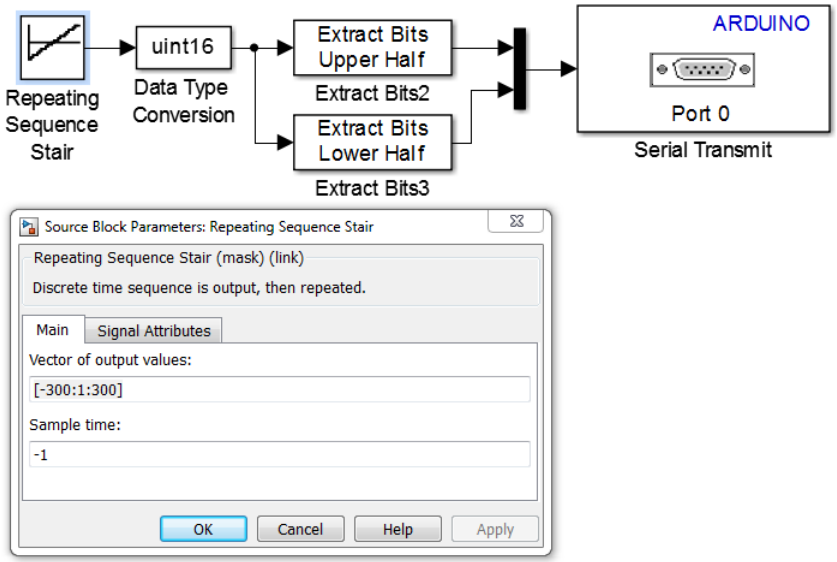
```
% if data looks our of sync skip a byte
```

Objectives:

- Use Simulink to send data with more than 8 bits

Simulink Diagram:

The supported Arduino Simulink block only supports uint8 data types. To send data that is greater than 1 byte the data must be converted to a series of bytes (each uint8).

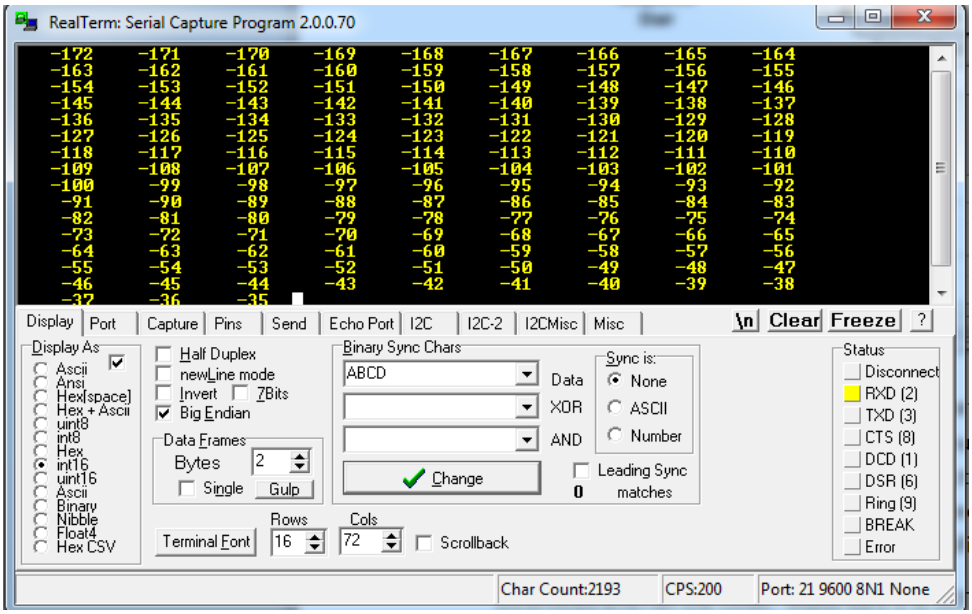


- [-300:1:300] creates a series of numbers that starts at -300 that increases by 1 all the way to 300
- The data is converted to a uint16 (even though there are negative numbers)
- uint16 is two bytes – the high byte is extracted and put first in the resulting vector, then the low byte is extracted and put second and this vector, or byte stream, consists of two uint8 bytes sent to the serial port. Since the most significant byte is first, this is known as big endian format

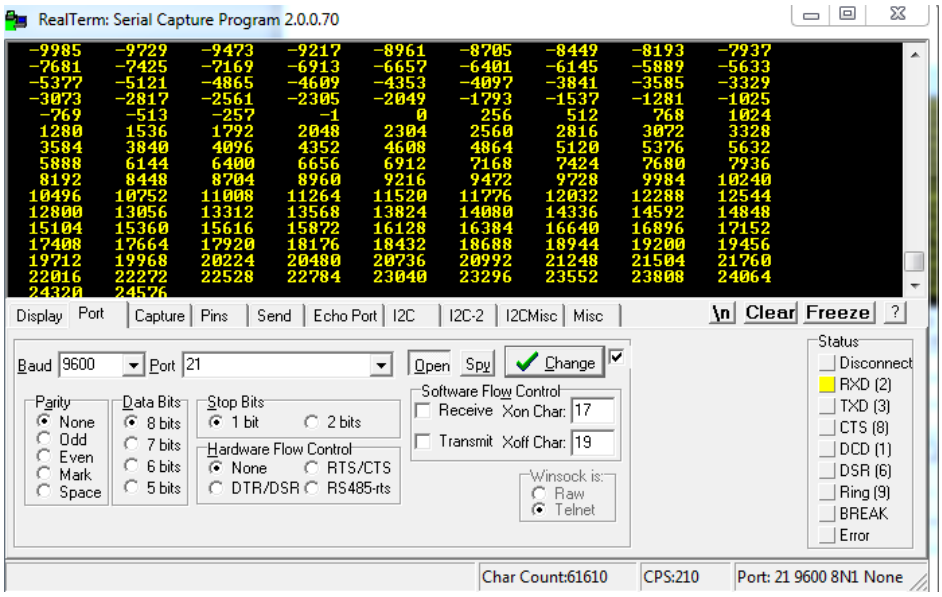
Download and run the code. Make sure it is NOT in external mode. After the code is downloaded it will be constantly sending data to the USB serial port.

Writing the data to a file with RealTerm:

- Open RealTerm, set the baud rate and port, and connect
- Select “Display As” and choose uint16. Notice the the negative numbers are not displayed
- Chose int16, notice the negative numbers are now displayed



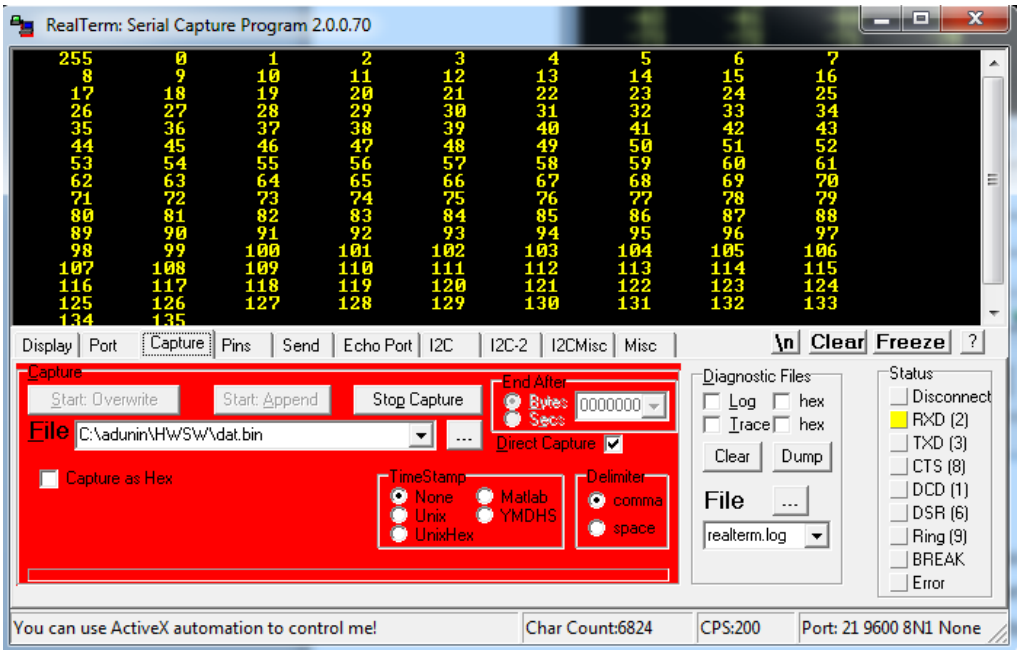
- **Data Out of sync:**
  - Repeatedly open and close the serial port by clicking on the “open” tab until you notice the data is not correct (numbers are greater than 300):



  - Sometimes when it opens the serial port it will not get the very first byte of data and the byte packet will be out of sync.
    - In this case open and close it until it is in sync or by resetting the hardware device
    - There are other ways to address this such as “processing” the data to skip the first byte in matlab, or ensuring the first byte is not skipped

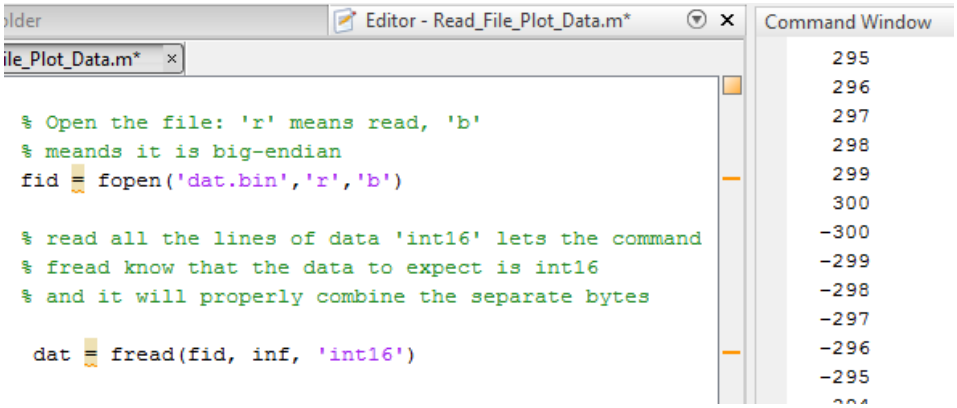
- Click the “Capture” tab

- Enter the location and filename you want to store the data (your current Matlab directory). Since the data is in binary the filename should end in .bin to reflect this
- Click the "Start Overwrite" button to begin writing data to the file



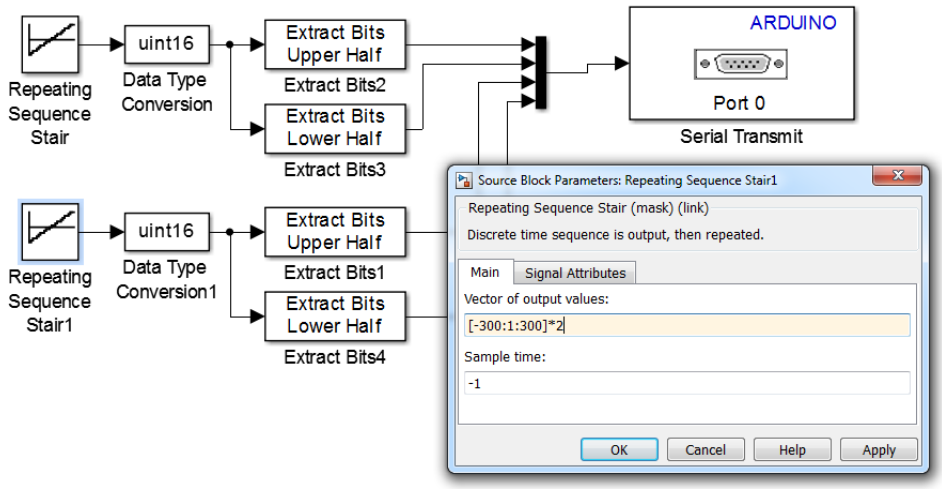
Reading the data file with Matlab:

Write the following M-file to open the file and read the bytes in the appropriate format:



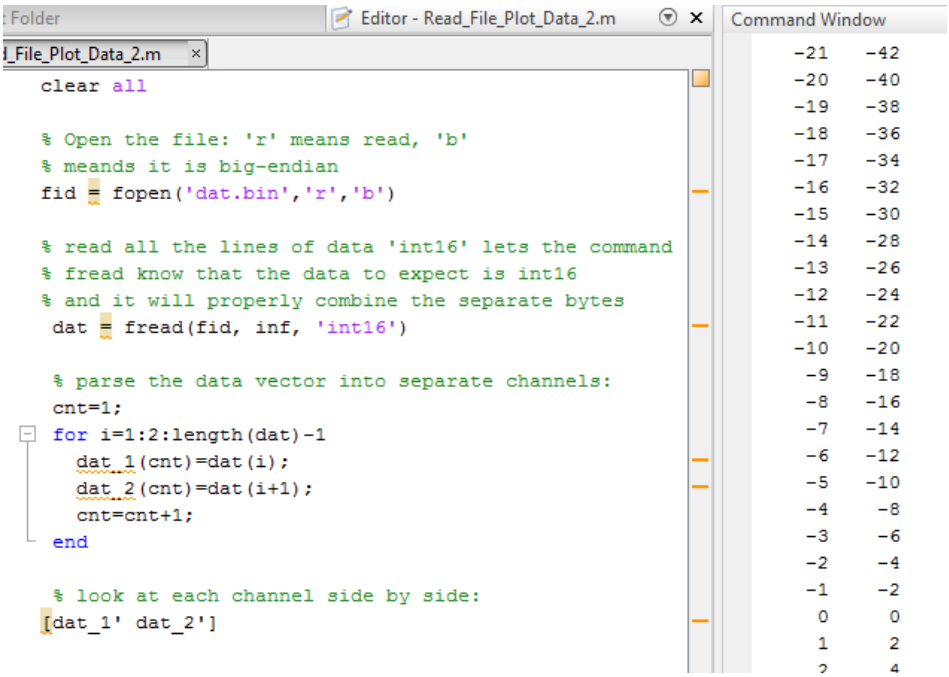
Reading Two Channels of Data:

- Simulink diagram:



The second data channel we scale by 2: [-300:1:300]*2

- Open RealTerm, connect, verify the data is “in sync”, log the data to a file, then run the following script file to “parse” the data



As seen the separate data channels have been recovered

This general procedure can be used to continuously read the file as it is being written to plot the data in real time.

Part 4: Sending an int16 and a Float with a Sfunction using a Union

A union is a data type where the variables share the same memory location. This can be used to get access to each byte of a particular data type:

- Define the union "bint" which stores an interger "i" and two bytes "b[2]":

```
union{
    unsigned char b[2];
    int i;
} bint;
```

- Assign to the memory location the value of the input integer:

```
// Write integer data to memory:
bint.i=in1[0];
```

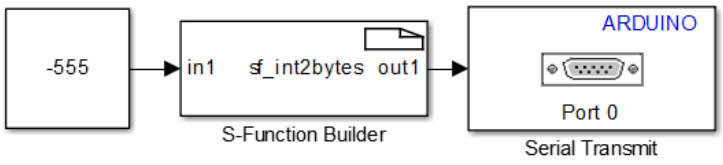
- Extract each byte by accessing each byte:

```
// Assign those bytes in big-endian
// to the output vector
out1[0]=bint.b[1];
out1[1]=bint.b[0];
```

The union easily assigns to the output vector each byte of the input data. This same method can be applied to a float (single) except union will consist of 4 bytes and a float.

Simulink send and M-file read: int16

Use the following Simulink model and Sfunction to send an integer and the m-file to reconstruct the data:

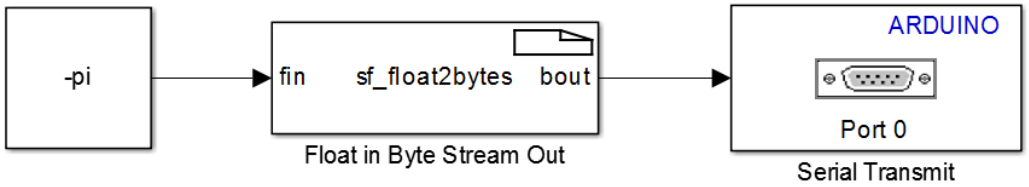


```
% Open the file: 'r' means read, 'b'
% means it is big-endian
fid = fopen('dat.bin','r', 'b')
% read all the lines of data 'int16' lets the command
% fread know that the data to expect is int16
% and it will properly combine the separate bytes

% if data looks our of sync skip a byte
% fread(fid,1, 'char');

% read all the data:
dat = fread(fid, inf, 'int16')
```

```
-555
-555
-555
-555
-555
-555
-555
-555
-555
-555
-555
-555
```

If you look inside the SFunction you will see the code discussed.

Simulink send and M-file read: Float (Single)




```
% Open the file: 'r' means read, 'b'
% means it is big-endian
% fid = fopen('dat.bin','r','b')
fid = fopen('dat.bin','r', 'b')
% read all the lines of data 'int16' lets the command
% fread know that the data to expect is int16
% and it will properly combine the separate bytes
dat = fread(fid, inf, 'single')
```

```
-3.1416
-3.1416
-3.1416
-3.1416
-3.1416
-3.1416
-3.1416
-3.1416
```