RENSSELAER MECHATRONICS

Communication: Sending Data

Contents

Overview:

“Monitor and Tune” mode in Simulink allows data to be sent and received from the target board (historically called external mode). This communication protocol has overhead which limits how fast data can be transferred. For example an Arduino Mega in external mode the maximum sampling rate is around 30Hz with 2015a or earlier and up to 1Khz with later versions of MATLAB. To obtain information at a faster rate the data can be send directly from the target without using “Monitor and Tune”.

This lab explores sending data with Simulink using both built in blocks, system object blocks and with the Arduino IDE. It will also review different data types

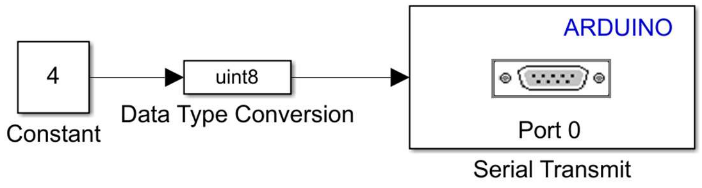
Part 1: Sending 8 bit data with Simulink

Objectives:

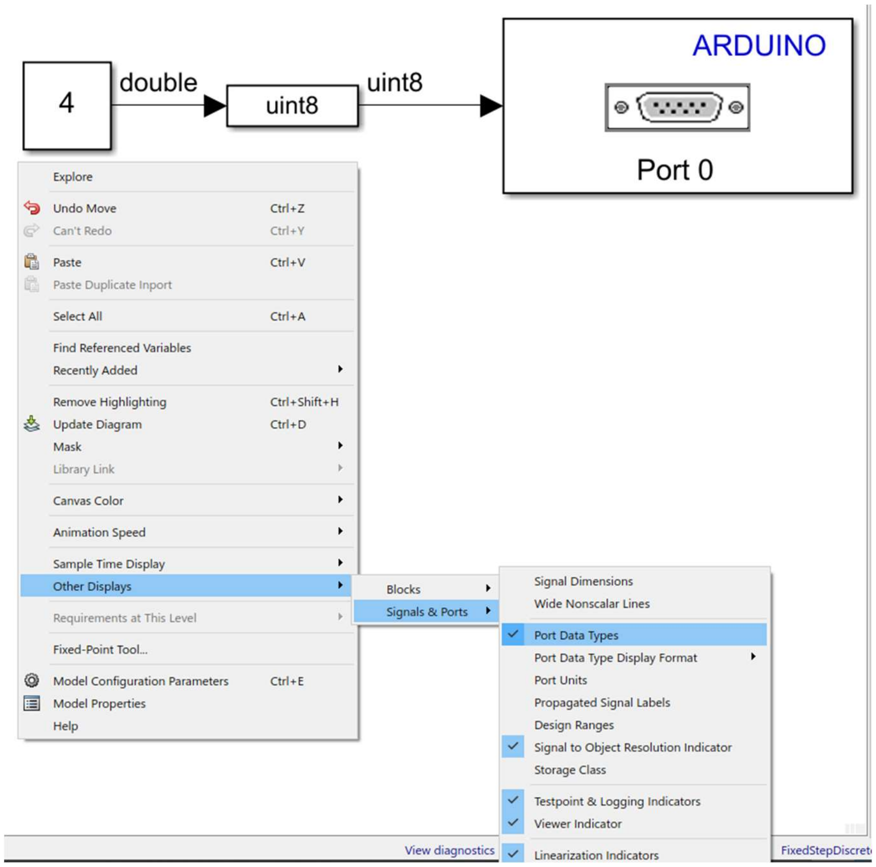
- Send 8 bit data over the serial link using the supported Simulink blocks

Simulink Model:

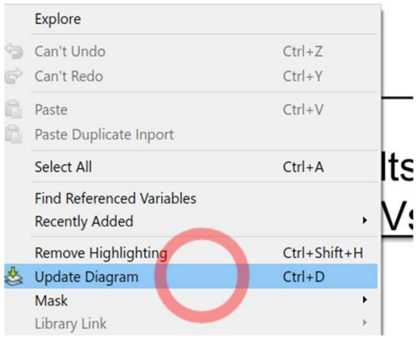
- Build the Simulink diagram form a “Constant”, “Data Type Conversion” and a “Serial Transmit” block. Be sure to use the Demo Simulink diagram and create this new one from that one.



- The Simulink Arduino library block “Serial Transmit” can be used to send single bytes at a time. To see this right-click in the main windows and select “Other Displays – Signals & Ports – Port Data Types”:

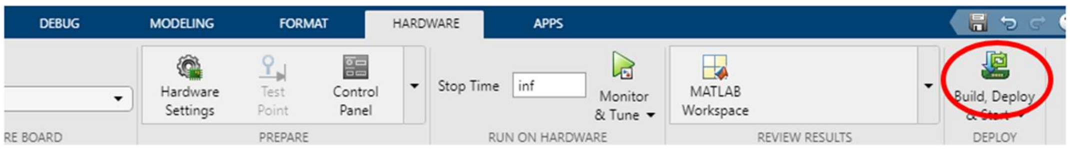


- Press “Control+D” to update the diagram see the data types, right-click in the diagram to Update the Diagram:

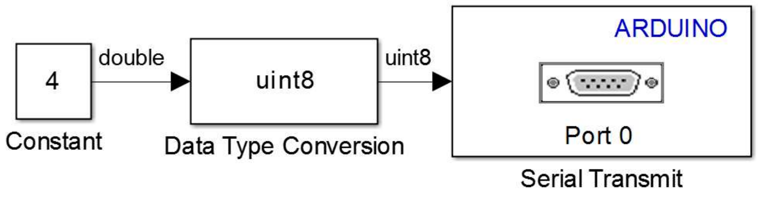


The input data type for a Serial Transmit is uint8 (a single byte, 8 bits, of data). This means that data must be converted to unit8 before sending across the serial line with this block.

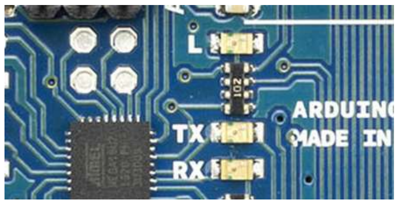
- Build and run the following Simulink diagram.
    - Click the 'Build, Deploy & Start' button:



  - The baud rates for the ports are set in the Configuration Parameters in the Run On Target Hardware tab – by default they are 921600
  - 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)

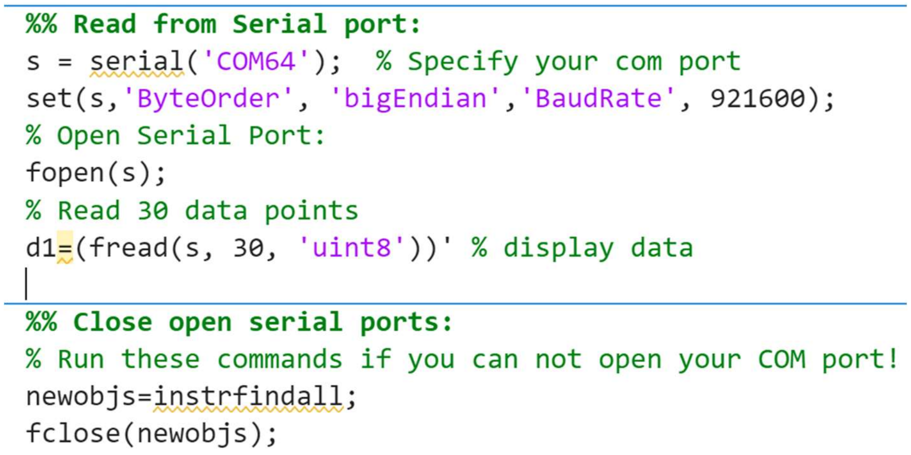


  - After the code is downloaded you should see the TX or RX LED blinking, or remain lit, indicating data is being sent across the serial line:



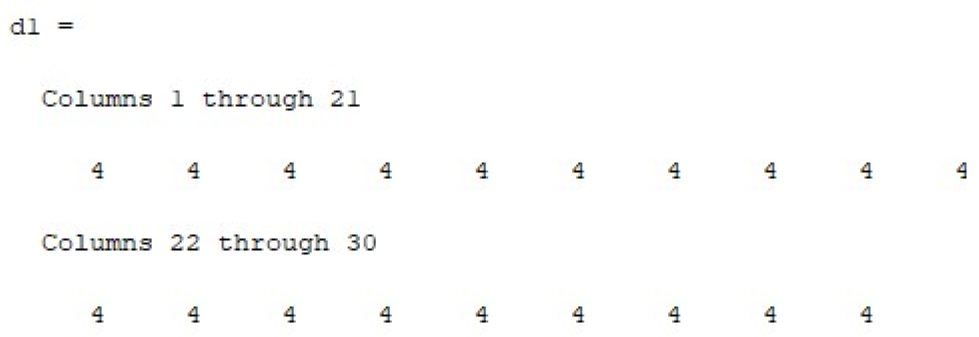
Reading the Data with MATLAB:

Create an m-file with the following code. Use the COM port of your device. This code will open the com port and store the results in the variable d1:



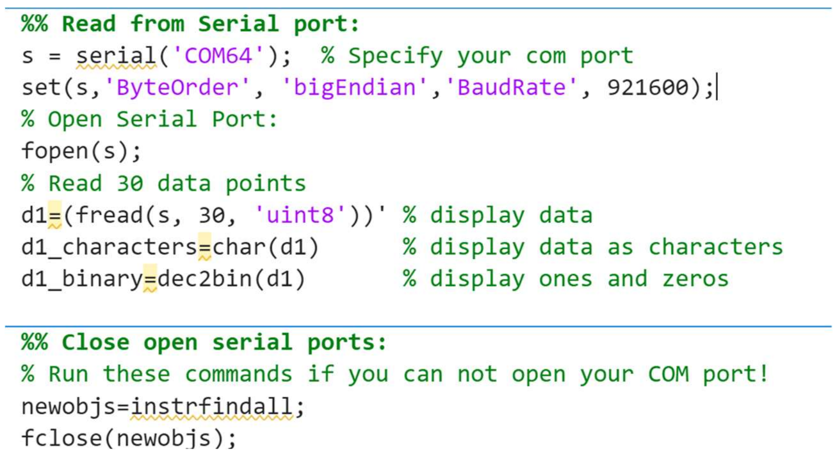
```
%% Read from Serial port:
s = serial('COM64');  % Specify your com port
set(s,'ByteOrder', 'bigEndian','BaudRate', 921600);
% Open Serial Port:
fopen(s);
% Read 30 data points
d1=(fread(s, 30, 'uint8'))' % display data

%% Close open serial ports:
% Run these commands if you can not open your COM port!
newobjs=instrfindall;
fclose(newobjs);
```

The output on the MATLAB command line should be:



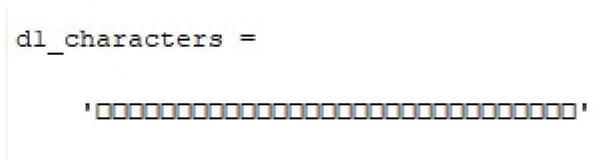
```
d1 =

  Columns 1 through 21

     4     4     4     4     4     4     4     4     4     4

  Columns 22 through 30

     4     4     4     4     4     4     4     4     4
```

To display the ASCII equivalent characters, or the binary form of ones and zeros:



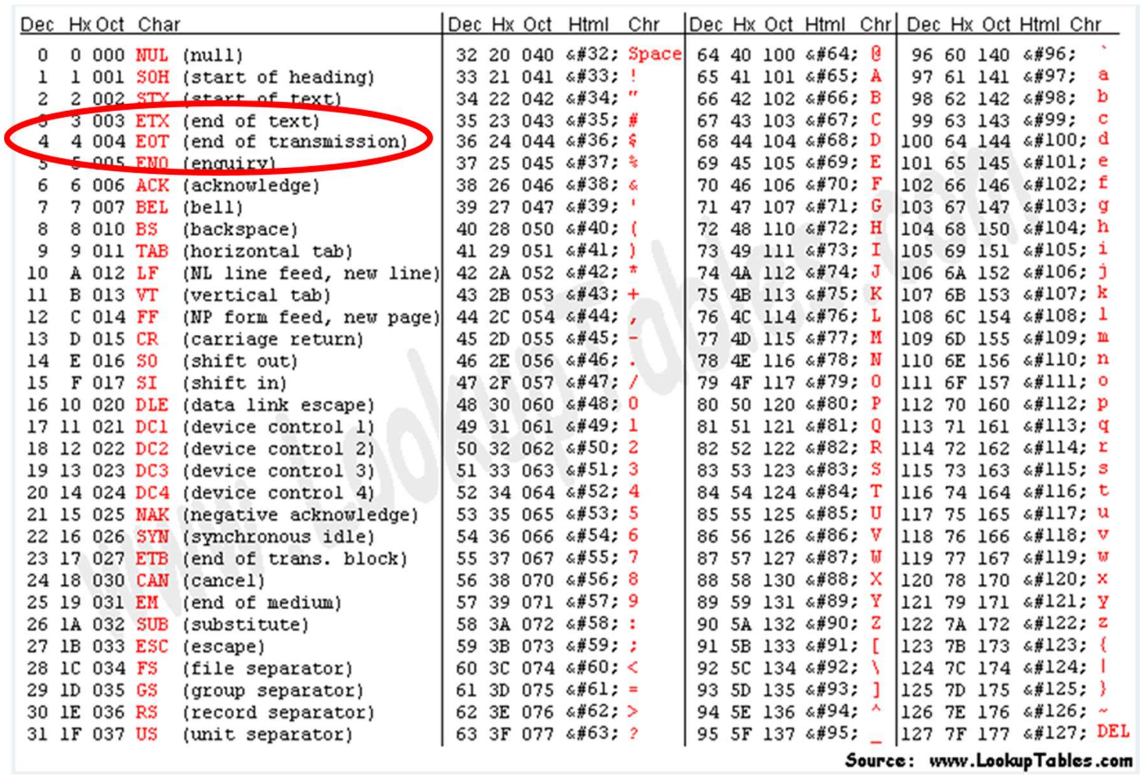
```
%% Read from Serial port:
s = serial('COM64');  % Specify your com port
set(s,'ByteOrder', 'bigEndian','BaudRate', 921600);
% Open Serial Port:
fopen(s);
% Read 30 data points
d1=(fread(s, 30, 'uint8'))' % display data
d1_characters=char(d1)      % display data as characters
d1_binary=dec2bin(d1)       % display ones and zeros

%% Close open serial ports:
% Run these commands if you can not open your COM port!
newobjs=instrfindall;
fclose(newobjs);
```

The result



```
d1_characters =

    '□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□'
```

You notice it will display the characters that the bytes represent instead of the numerical binary values. The characters represented by a byte can be found from a Ascii character table:

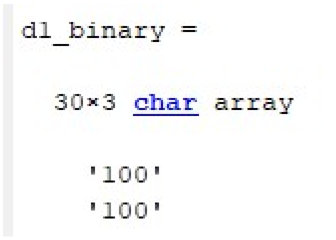


Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	&#32;	Space	64	40	100	&#64;	@	96	60	140	&#96;	`
1	1	001	SOH (start of heading)	33	21	041	&#33;	!	65	41	101	&#65;	A	97	61	141	&#97;	a
2	2	002	STX (start of text)	34	22	042	&#34;	"	66	42	102	&#66;	B	98	62	142	&#98;	b
3	3	003	ETX (end of text)	35	23	043	&#35;	#	67	43	103	&#67;	C	99	63	143	&#99;	c
4	4	004	EOT (end of transmission)	36	24	044	&#36;	$	68	44	104	&#68;	D	100	64	144	&#100;	d
5	5	005	ENQ (enquiry)	37	25	045	&#37;	%	69	45	105	&#69;	E	101	65	145	&#101;	e
6	6	006	ACK (acknowledge)	38	26	046	&#38;	&	70	46	106	&#70;	F	102	66	146	&#102;	f
7	7	007	BEL (bell)	39	27	047	&#39;	'	71	47	107	&#71;	G	103	67	147	&#103;	g
8	8	010	BS (backspace)	40	28	050	&#40;	(	72	48	110	&#72;	H	104	68	150	&#104;	h
9	9	011	TAB (horizontal tab)	41	29	051	&#41;	)	73	49	111	&#73;	I	105	69	151	&#105;	i
10	A	012	LF (NL line feed, new line)	42	2A	052	&#42;	*	74	4A	112	&#74;	J	106	6A	152	&#106;	j
11	B	013	VT (vertical tab)	43	2B	053	&#43;	+	75	4B	113	&#75;	K	107	6B	153	&#107;	k
12	C	014	FF (NP form feed, new page)	44	2C	054	&#44;	,	76	4C	114	&#76;	L	108	6C	154	&#108;	l
13	D	015	CR (carriage return)	45	2D	055	&#45;	-	77	4D	115	&#77;	M	109	6D	155	&#109;	m
14	E	016	SO (shift out)	46	2E	056	&#46;	.	78	4E	116	&#78;	N	110	6E	156	&#110;	n
15	F	017	SI (shift in)	47	2F	057	&#47;	/	79	4F	117	&#79;	O	111	6F	157	&#111;	o
16	10	020	DLE (data link escape)	48	30	060	&#48;	0	80	50	120	&#80;	P	112	70	160	&#112;	p
17	11	021	DC1 (device control 1)	49	31	061	&#49;	1	81	51	121	&#81;	Q	113	71	161	&#113;	q
18	12	022	DC2 (device control 2)	50	32	062	&#50;	2	82	52	122	&#82;	R	114	72	162	&#114;	r
19	13	023	DC3 (device control 3)	51	33	063	&#51;	3	83	53	123	&#83;	S	115	73	163	&#115;	s
20	14	024	DC4 (device control 4)	52	34	064	&#52;	4	84	54	124	&#84;	T	116	74	164	&#116;	t
21	15	025	NAK (negative acknowledge)	53	35	065	&#53;	5	85	55	125	&#85;	U	117	75	165	&#117;	u
22	16	026	SYN (synchronous idle)	54	36	066	&#54;	6	86	56	126	&#86;	V	118	76	166	&#118;	v
23	17	027	ETB (end of trans. block)	55	37	067	&#55;	7	87	57	127	&#87;	W	119	77	167	&#119;	w
24	18	030	CAN (cancel)	56	38	070	&#56;	8	88	58	130	&#88;	X	120	78	170	&#120;	x
25	19	031	EM (end of medium)	57	39	071	&#57;	9	89	59	131	&#89;	Y	121	79	171	&#121;	y
26	1A	032	SUB (substitute)	58	3A	072	&#58;	:	90	5A	132	&#90;	Z	122	7A	172	&#122;	z
27	1B	033	ESC (escape)	59	3B	073	&#59;	;	91	5B	133	&#91;	[	123	7B	173	&#123;	{
28	1C	034	FS (file separator)	60	3C	074	&#60;	<	92	5C	134	&#92;	\	124	7C	174	&#124;	|
29	1D	035	GS (group separator)	61	3D	075	&#61;	=	93	5D	135	&#93;	]	125	7D	175	&#125;	}
30	1E	036	RS (record separator)	62	3E	076	&#62;	>	94	5E	136	&#94;	^	126	7E	176	&#126;	~
31	1F	037	US (unit separator)	63	3F	077	&#63;	?	95	5F	137	&#95;	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

Here we can see that the number 4 represents the represents "EOT" character which we see displayed as □□□□ Note the same data is always being sent, we are simply changing how it is displayed on the screen – as a character "EOT", or as the numerical binary data 4.

You can also have MATLAB show the actual ones and zeros corresponding to the number 4:



```
dl_binary =

  30×3 char array

    '100'
    '100'
```

This is explained in a more detail in the section "Reading the Data with RealTerm"

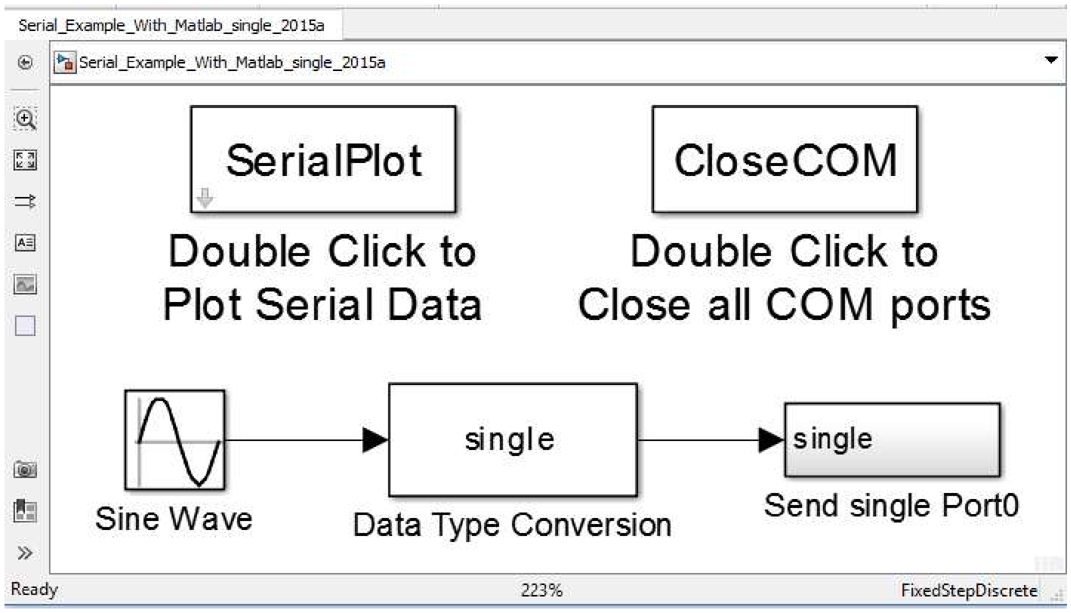
Part 2: Sending and Receiving Data with RASPlib

Objectives:

- Use RASPlib to send, plot and store data. Data can be sent and received faster than "Monitor and Tune" (external mode). The exact number of bytes sent each time step is specified.

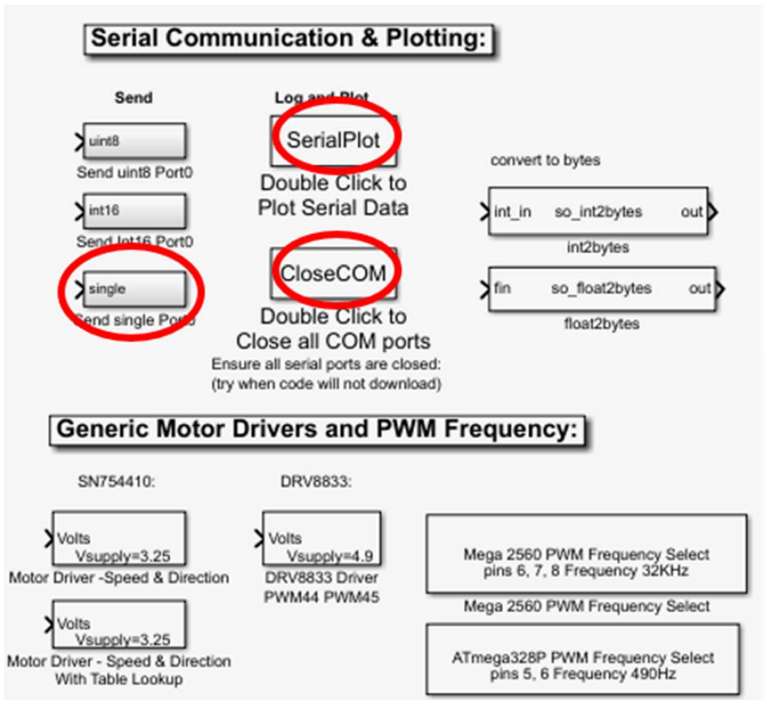
Simulink Model:

- Build the following Simulink diagram: (note if your Simulink diagram does not show the block names by default you can right click the block and select 'format', 'show block name', 'on'.)

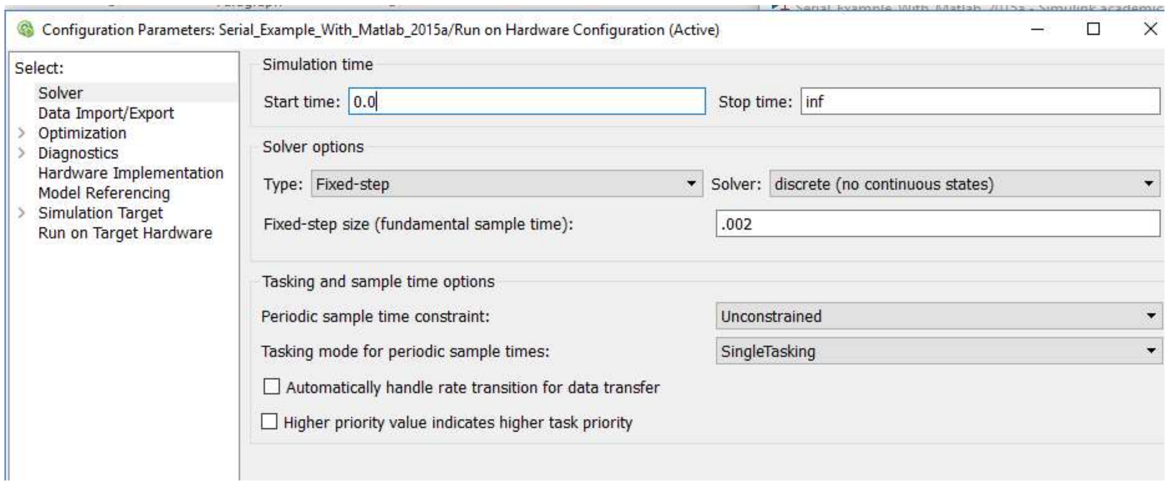

Serial_Example_With_Matlab_single_2015a
Serial_Example_With_Matlab_single_2015a
SerialPlot
Double Click to Plot Serial Data
CloseCOM
Double Click to Close all COM ports
Sine Wave
single
Data Type Conversion
single
Send single Port0
Ready
223%
FixedStepDiscrete

SerialPlot, CloseCOM, and "Send single Port0" are all obtained from RASPlib. Open RASPLib by navigating inside your RASPLib installation folder and opening RASPlib.slx. Be sure you also add the RASPLib folder to your path (see RASPLib installation instructions).

Then drag the blocks to you diagram:


Serial Communication & Plotting:
Send
uint8
Send uint8 Port0
int16
Send int16 Port0
single
Send single Port0
Log and Plot
SerialPlot
Double Click to Plot Serial Data
CloseCOM
Double Click to Close all COM ports
Ensure all serial ports are closed:
(try when code will not download)
convert to bytes
int_in so_int2bytes out
int2bytes
fin so_float2bytes out
float2bytes
Generic Motor Drivers and PWM Frequency:
SN754410:
DRV8833:
Volts Vsupply=3.25
Motor Driver -Speed & Direction
Volts Vsupply=4.9
DRV8833 Driver PWM44 PWM45
Mega 2560 PWM Frequency Select pins 6, 7, 8 Frequency 32KHz
Mega 2560 PWM Frequency Select
Volts Vsupply=3.25
Motor Driver - Speed & Direction With Table Lookup
ATmega328P PWM Frequency Select pins 5, 6 Frequency 490Hz

Use solver settings:


Configuration Parameters: Serial_Example_With_Matlab_2015a/Run on Hardware Configuration (Active)

Simulation time

Start time: 0.0 Stop time: inf

Solver options

Type: Fixed-step Solver: discrete (no continuous states)

Fixed-step size (fundamental sample time): .002

Tasking and sample time options

Periodic sample time constraint: Unconstrained

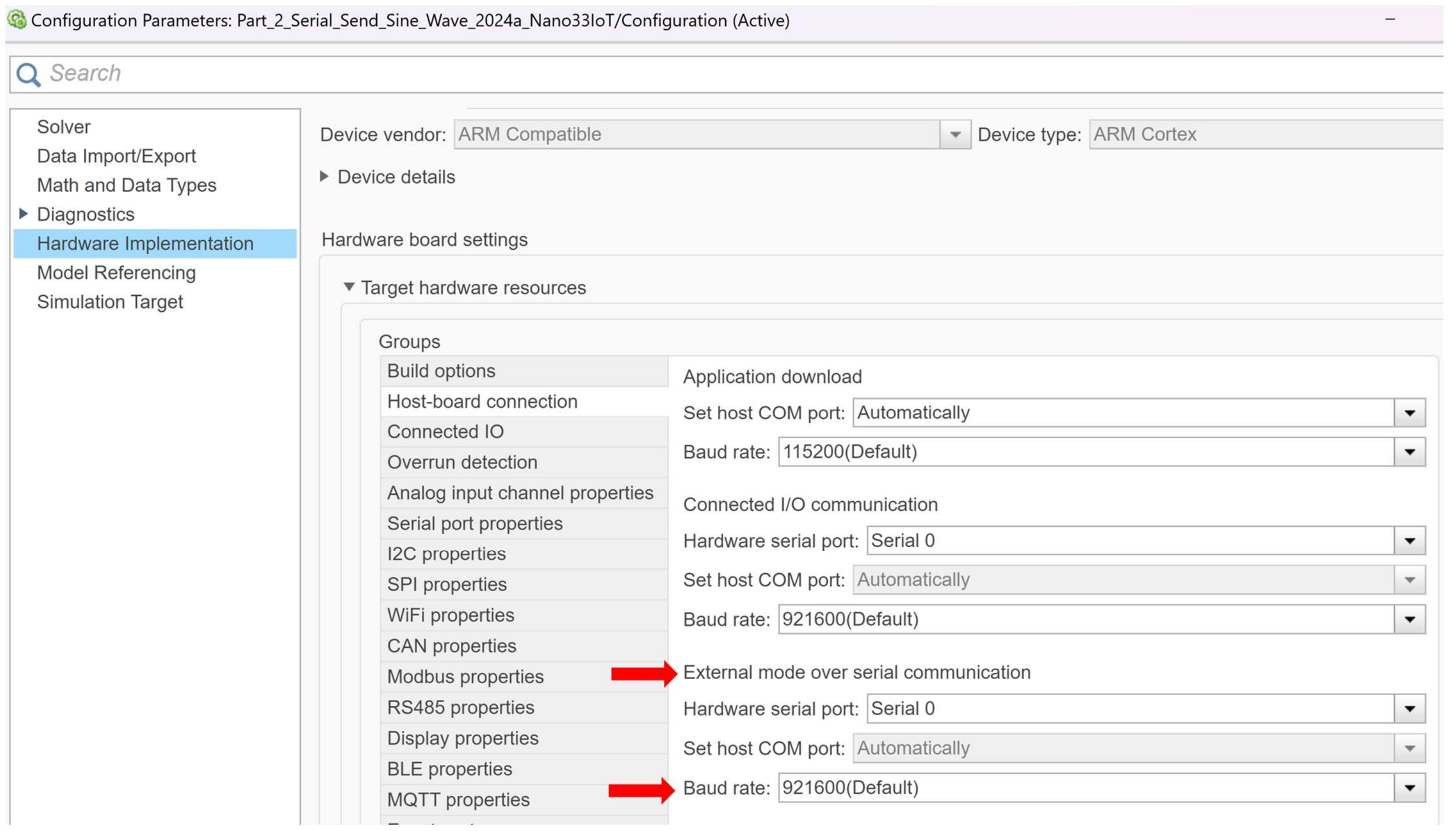
Tasking mode for periodic sample times: SingleTasking

☐ Automatically handle rate transition for data transfer

☐ Higher priority value indicates higher task priority

BAUD Rate Calculation

The default setting for the BAUD rate is 921600. In the above case the sample time is .002 second, and we are sending a single which is 4 bytes. The amount of bits (8 bites in a byte) per second is then 4*8/.002 = 16000 bits per second. This means the minimum BAUD rate for a single data channel at .002 is 1600. 921600 is more than fast enough. Sometimes these rates have to be lowered if you start start having hardware issues and the data sseem to be less reliable. Access this menu form the “Hardware” Tab then “Hardware Settings”


Configuration Parameters: Part_2_Serial_Send_Sine_Wave_2024a_Nano33IoT/Configuration (Active)

Device vendor: ARM Compatible Device type: ARM Cortex

Device details

Hardware board settings

Target hardware resources

Application download

Set host COM port: Automatically

Baud rate: 115200(Default)

Connected I/O communication

Hardware serial port: Serial 0

Set host COM port: Automatically

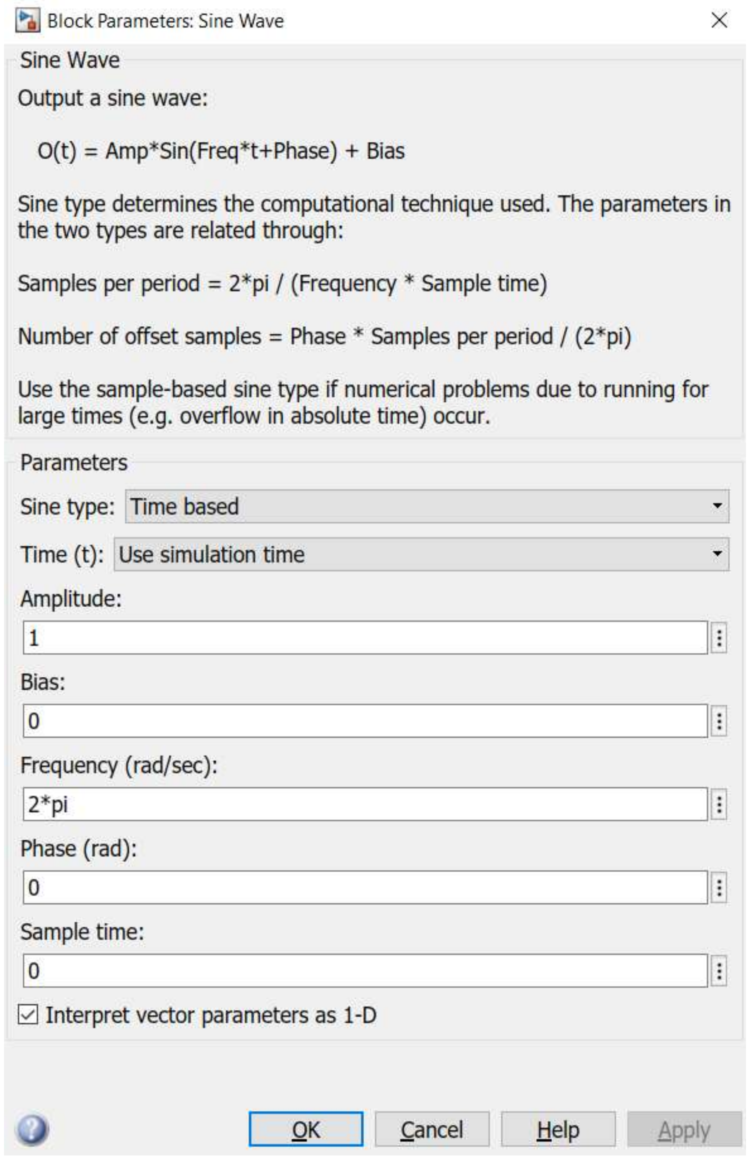
Baud rate: 921600(Default)

External mode over serial communication

Hardware serial port: Serial 0

Set host COM port: Automatically

Baud rate: 921600(Default)



Block Parameters: Sine Wave

Sine Wave

Output a sine wave:

O(t) = Amp*Sin(Freq*t+Phase) + Bias

Sine type determines the computational technique used. The parameters in the two types are related through:

Samples per period = 2*pi / (Frequency * Sample time)

Number of offset samples = Phase * Samples per period / (2*pi)

Use the sample-based sine type if numerical problems due to running for large times (e.g. overflow in absolute time) occur.

Parameters

Sine type: Time based

Time (t): Use simulation time

Amplitude: 1

Bias: 0

Frequency (rad/sec): 2*pi

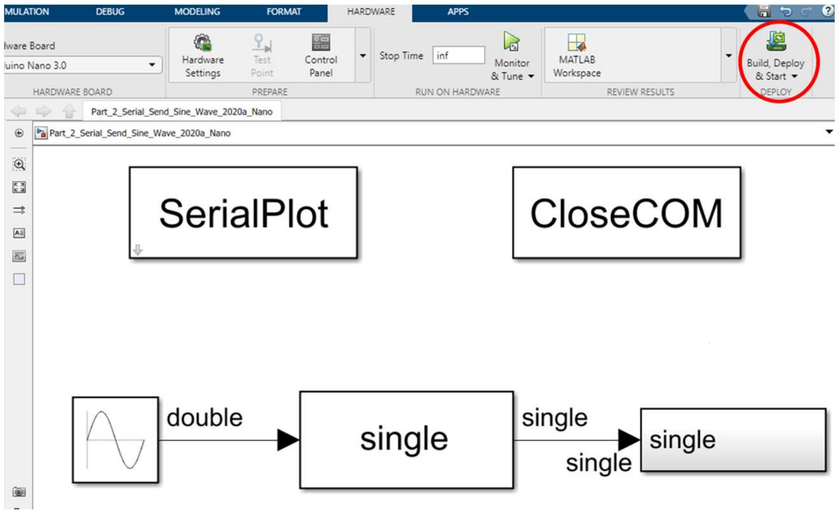
Phase (rad): 0

Sample time: 0

☑ Interpret vector parameters as 1-D

And a sine wave with magnitude 1 and frequency 2*pi:

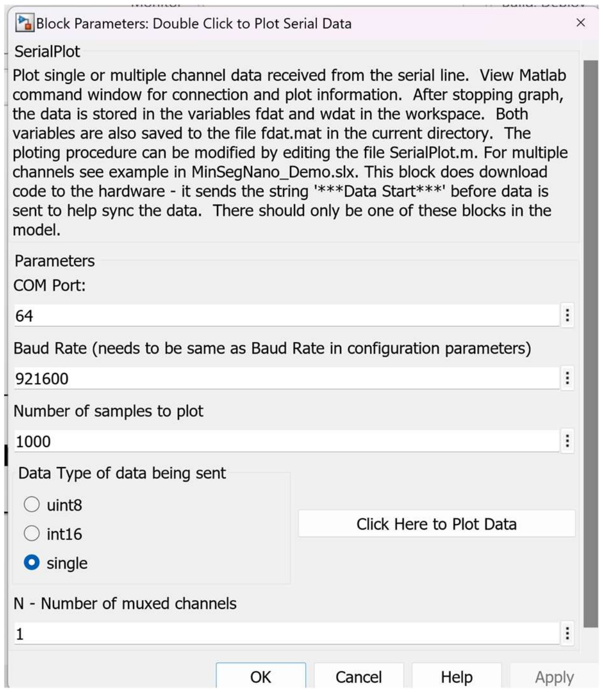
Next download the code to the hardware:



Note that the “Build, Deploy & Start” button downloads the code to the hardware only. After it downloads it will only execute the code you specify, in this case sending data from a sine wave to port 0 as a “single” floating point number (4 bytes).

After the code has been downloaded the “Tx” light on the board should light up indicating data is being sent to the computer.

- To capture and plot the data double click the “SerialPlot” block and set the following parameters. Be sure to use your COM port and the BAUD rate as specified in the configuration parameters.



Block Parameters: Double Click to Plot Serial Data

SerialPlot

Plot single or multiple channel data received from the serial line. View Matlab command window for connection and plot information. After stopping graph, the data is stored in the variables fdat and wdat in the workspace. Both variables are also saved to the file fdat.mat in the current directory. The ploting procedure can be modified by editing the file SerialPlot.m. For multiple channels see example in MinSegNano_Demo.slx. This block does download code to the hardware - it sends the string '***Data Start***' before data is sent to help sync the data. There should only be one of these blocks in the model.

Parameters

COM Port: 64

Baud Rate (needs to be same as Baud Rate in configuration parameters): 921600

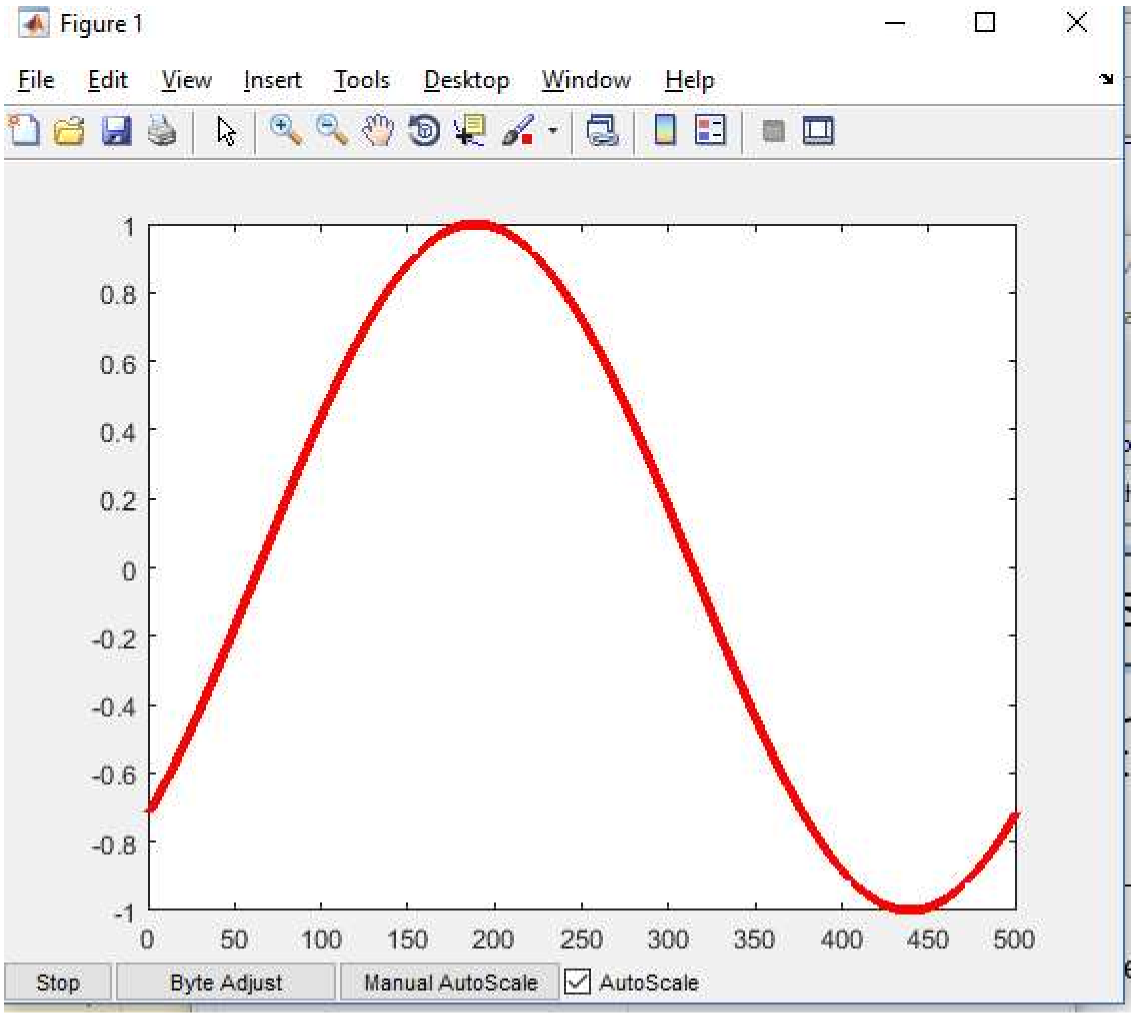
Number of samples to plot: 1000

Data Type of data being sent: uint8 int16 [✓ single]

N - Number of muxed channels: 1

- Then click “Click Here to Plot Data”

You should observe a sine wave of frequency 2*pi rad/sec – which should repeat every second:



- Click "Stop" to stop recording data.
- The data is stored in file "dat.mat" in the current directory
- The complete data is stored in the fdat variable and the last window data in the wdata variable. You can then easily plot the stored data:

```
>> plot(fdat)
>> plot(wdat)
```

There are 3 possible data types you might want to send

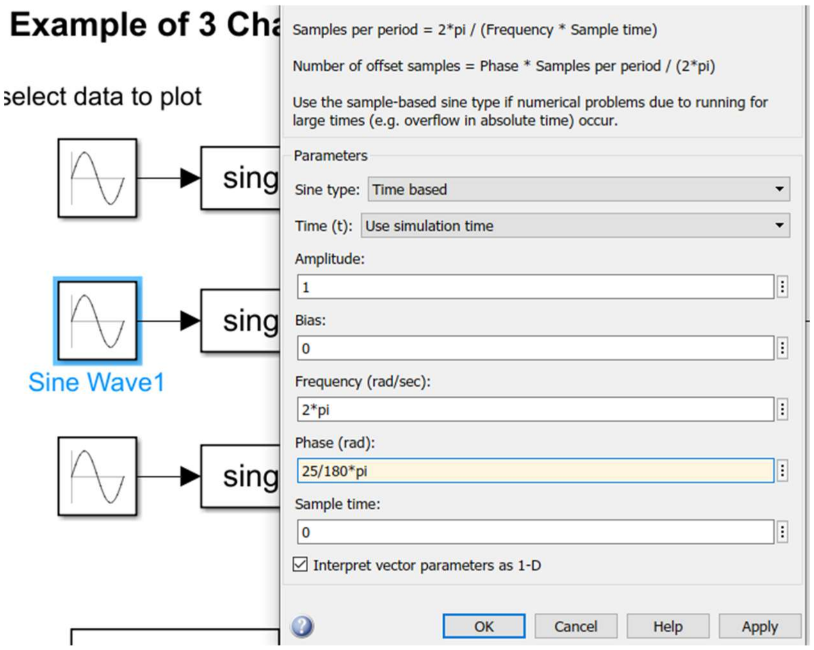
Type	Bytes	Range
uint8	1	0 to 255
int16	2	-32768 to 32767
single	4	-1.401298E-45 to 1.401298E-45

In general single should be used, but if data is needed faster other types can be used as long as the data is scaled to within the range of the data type.

Sending Multiple Data channels with RASPLib

Open the example "Three_Channel_Example.slx" located in your Blackboard. Deploy to hardware and verify it plots all 3 channels.

Next modify the 3 inputs to be sine waves, each shifted by 25 degrees from the previous:



Samples per period = 2*pi / (Frequency * Sample time)

Number of offset samples = Phase * Samples per period / (2*pi)

Use the sample-based sine type if numerical problems due to running for large times (e.g. overflow in absolute time) occur.

Parameters

Sine type: Time based

Time (t): Use simulation time

Amplitude: 1

Bias: 0

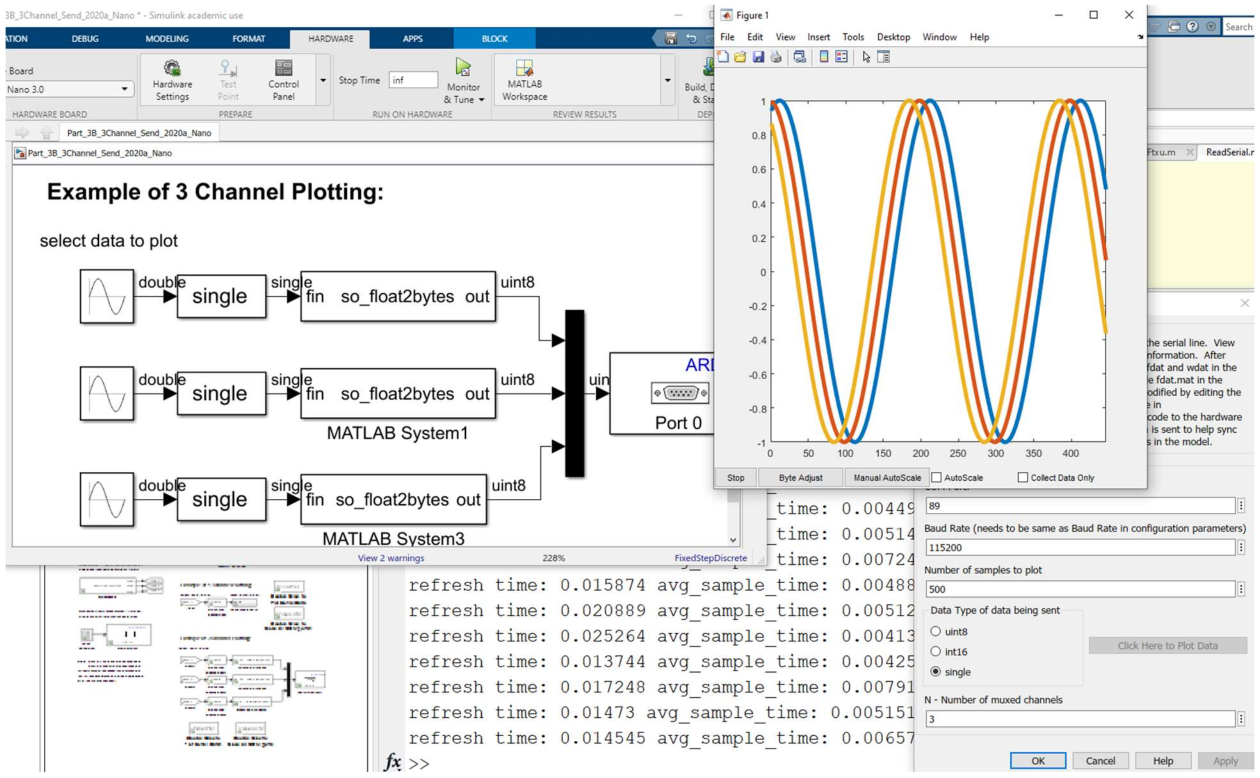
Frequency (rad/sec): 2*pi

Phase (rad): 25/180*pi

Sample time: 0

☑ Interpret vector parameters as 1-D

Download to the board and use Serial Plot to observe the results:



```
_time: 0.00449
_time: 0.00514
_time: 0.00724
refresh time: 0.015874 avg_sample_time: 0.00488
refresh time: 0.020889 avg_sample_time: 0.00512
refresh time: 0.025264 avg_sample_time: 0.00413
refresh time: 0.013744 avg_sample_time: 0.00425
refresh time: 0.017248 avg_sample_time: 0.00791
refresh time: 0.01473 avg_sample_time: 0.005151
refresh time: 0.014545 avg_sample_time: 0.00657
fx >>
```

the serial line. View
nformation. After
fdat and wdat in the
e fdat.mat in the
odified by editing the
e in
code to the hardware
is sent to help sync
s in the model.

89

Baud Rate (needs to be same as Baud Rate in configuration parameters): 115200

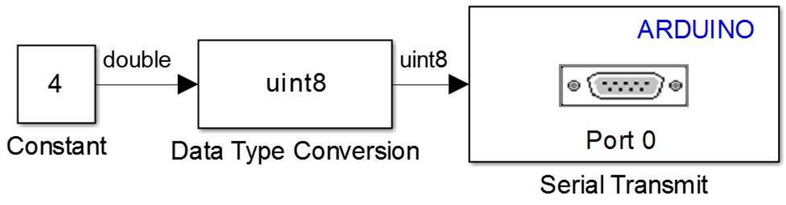
Number of samples to plot: 500

Data Type of data being sent: ○ uint8 ○ int16 [✓ single]

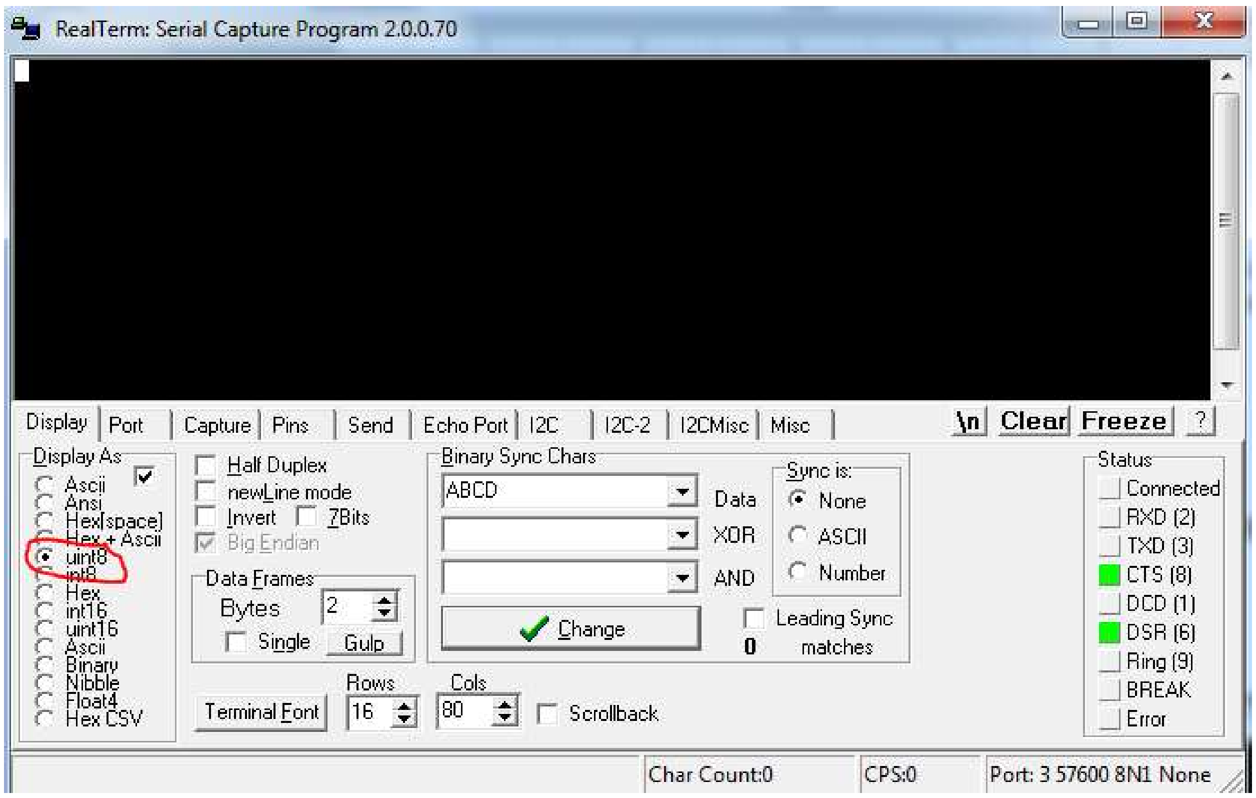
N - Number of muxed channels: 3

Part 3: Reading the Data with RealTerm (optional)

Another way to obtain data is RealTerm – although any serial terminal program will work. First download the following code to the Arduino:



After installing RealTerm open it in administrator mode.



Display As: Ascii, Ansi, Hex[space], Hex + Ascii, [✓ uint8], int8, Hex, int16, uint16, Ascii, Binary, Nibble, Float4, Hex CSV

☐ Half Duplex
☐ newLine mode
☐ Invert ☐ 7Bits
☑ Big Endian

Data Frames: Bytes 2 ☐ Single Gulp

Binary Sync Chars: Data ABCD; XOR; AND; Change

Sync is: [✓ None] ASCII Number

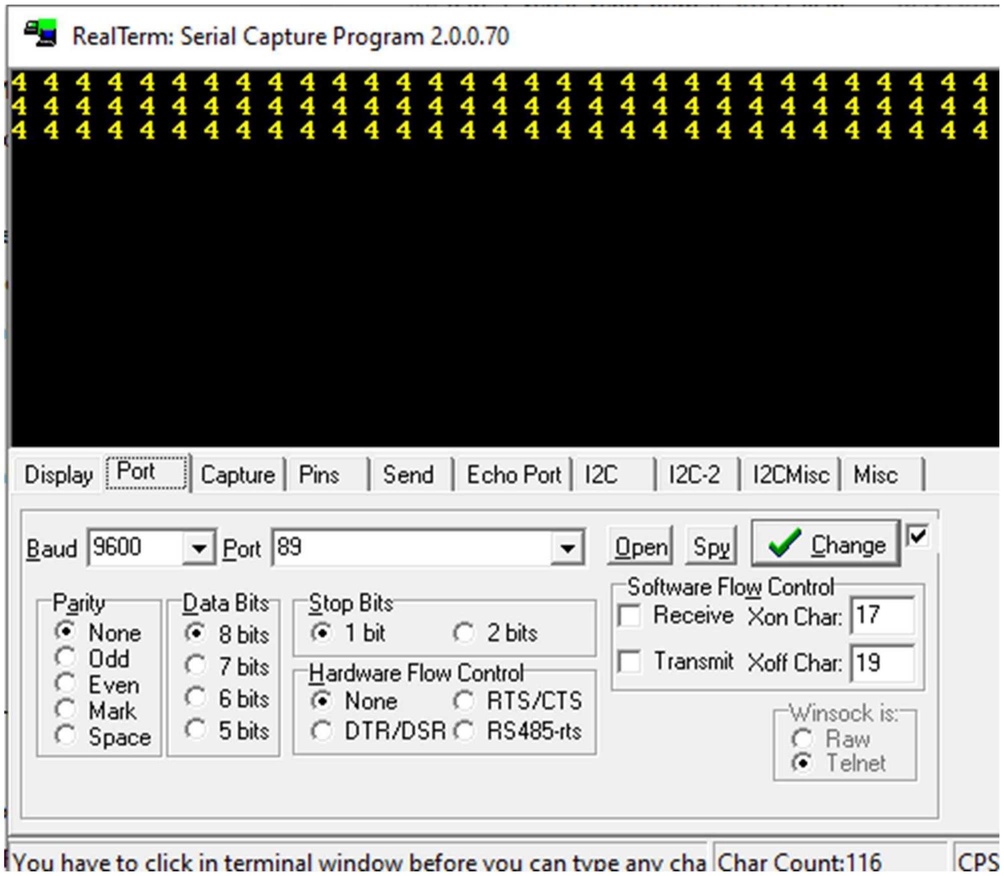
☐ Leading Sync 0 matches

Terminal Font; Rows 16; Cols 80; ☐ Scrollback

Status: Connected, RXD (2), TXD (3), 🟩 CTS (8), DCD (1), 🟩 DSR (6), Ring (9), BREAK, Error

Char Count:0 | CPS:0 | Port: 3 57600 8N1 None

- 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:


RealTerm: Serial Capture Program 2.0.0.70

```
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
```

Baud: 9600 Port: 89

Parity: [✓ None] Odd Even Mark Space

Data Bits: [✓ 8 bits] 7 bits 6 bits 5 bits

Stop Bits: [✓ 1 bit] 2 bits

Hardware Flow Control: [✓ None] RTS/CTS DTR/DSR RS485-rts

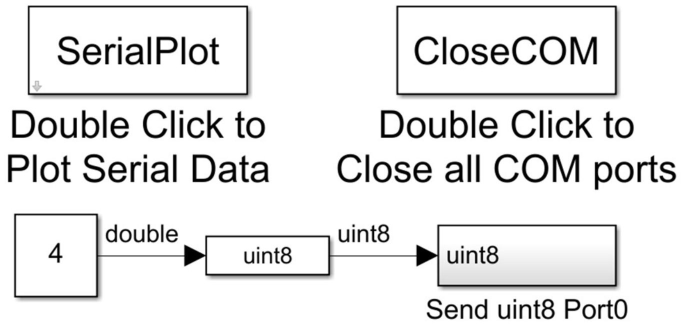
Software Flow Control: Receive ☐ Xon Char: 17; Transmit ☐ Xoff Char: 19

Winsock is: Raw [✓ Telnet]

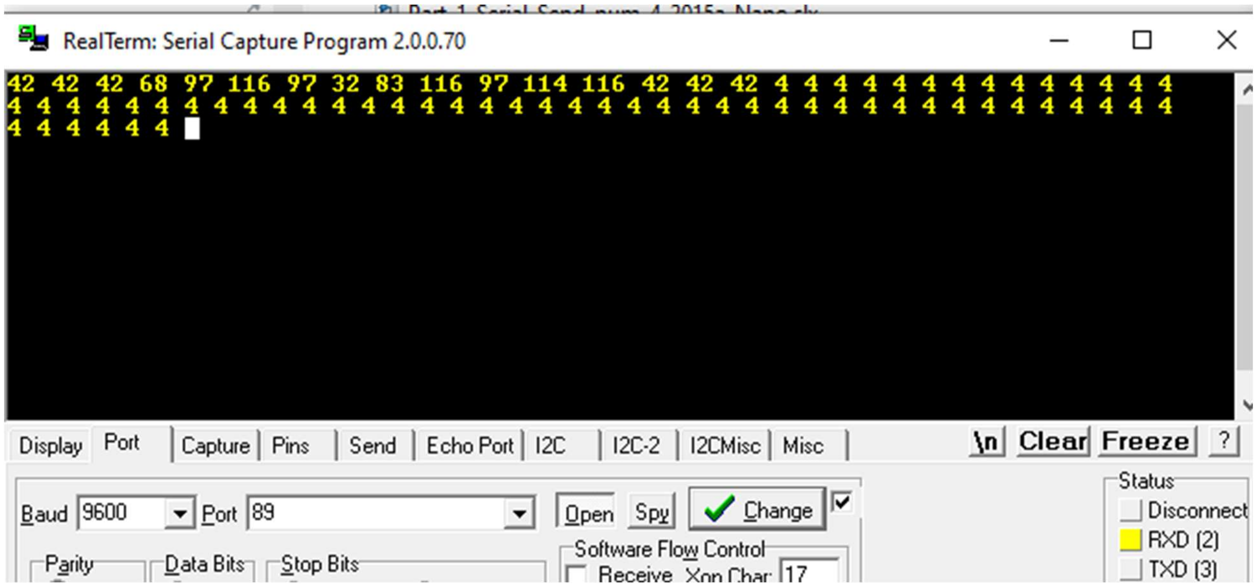
You have to click in terminal window before you can type any cha Char Count:116 CPS

By specifying “uint8” RealTerm knows the data type so it can display it correctly.

Next modify the first Simulink diagram to use the RASPLib block to send the number 4.


SerialPlot
Double Click to Plot Serial Data
CloseCOM
Double Click to Close all COM ports
4
double
uint8
uint8
uint8
Send uint8 Port0

Use the same settings and open RealTerm: (sovler of .05 seconds, Baud rate of 9600):

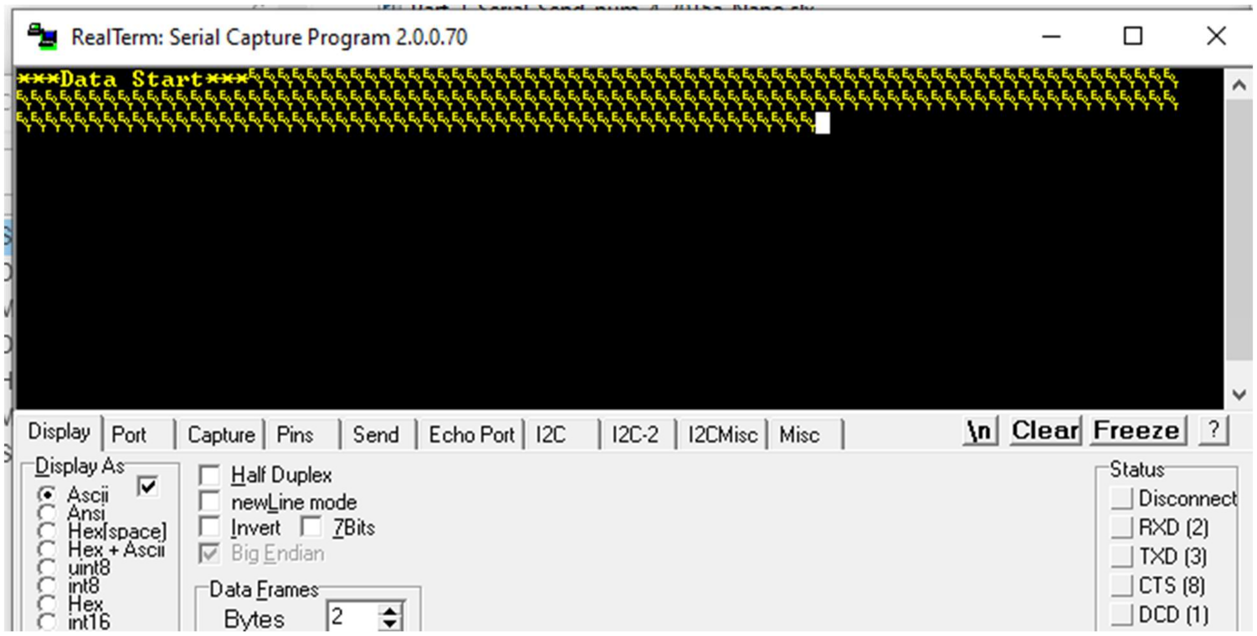

RealTerm: Serial Capture Program 2.0.0.70

```
42 42 42 68 97 116 97 32 83 116 97 114 116 42 42 42 4 4 4 4 4 4 4 4 4 4 4 4 4 4
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
4 4 4 4 4 4
```

Baud 9600 Port 89

Notice there are a bunch of numbers befor our data of fours “4 4 4 4”.

Change the “Display As” to “Ascii” and open the port again:

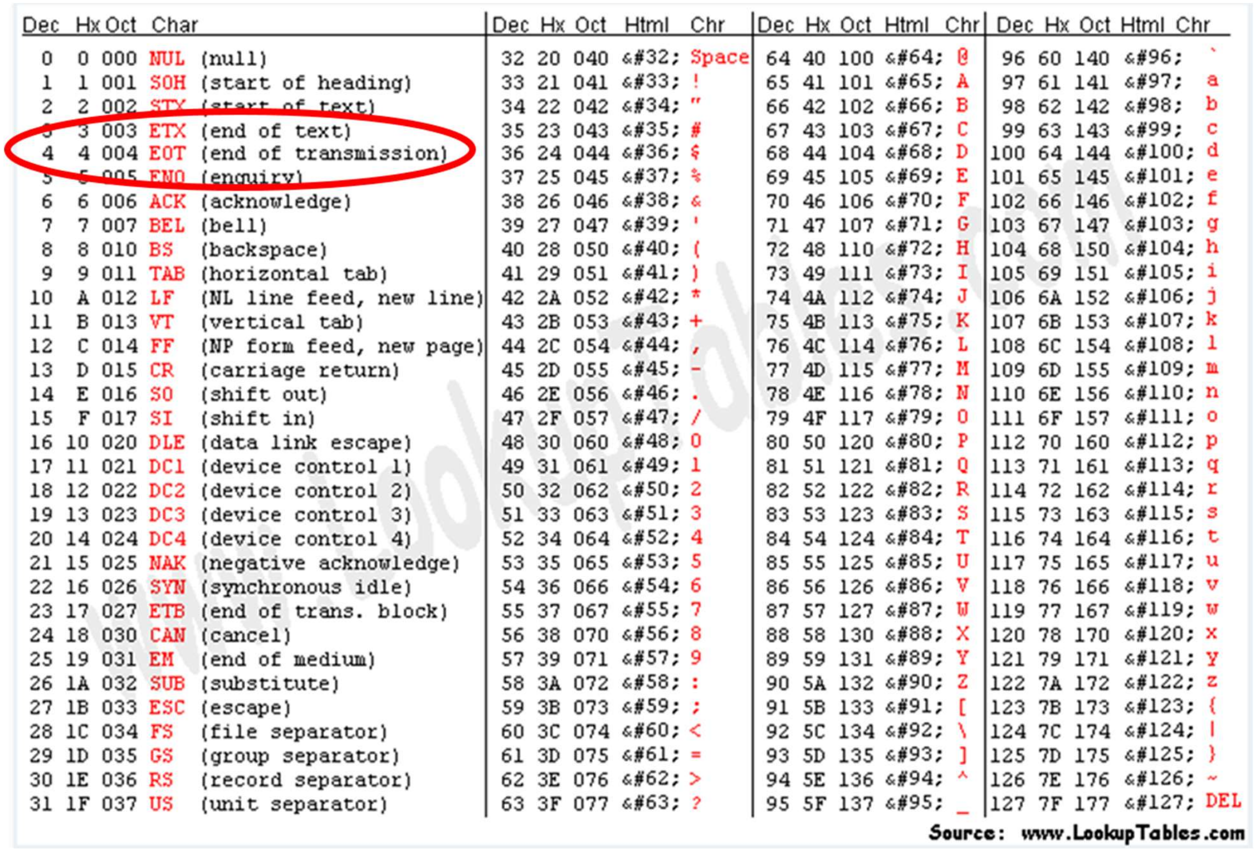

RealTerm: Serial Capture Program 2.0.0.70

```
***Data Start***
```

Display As: [✓ Ascii]

The program sends the message “***Data Start***” before sending data. This is so that the program that receives and plots that data can synchronize it by reading the “***Data Start***” sequence before plotting the data!

You notice it will display the characters that the bytes represent instead of the numerical binary values. The characters represented by a bye can be found from a Ascii character table:



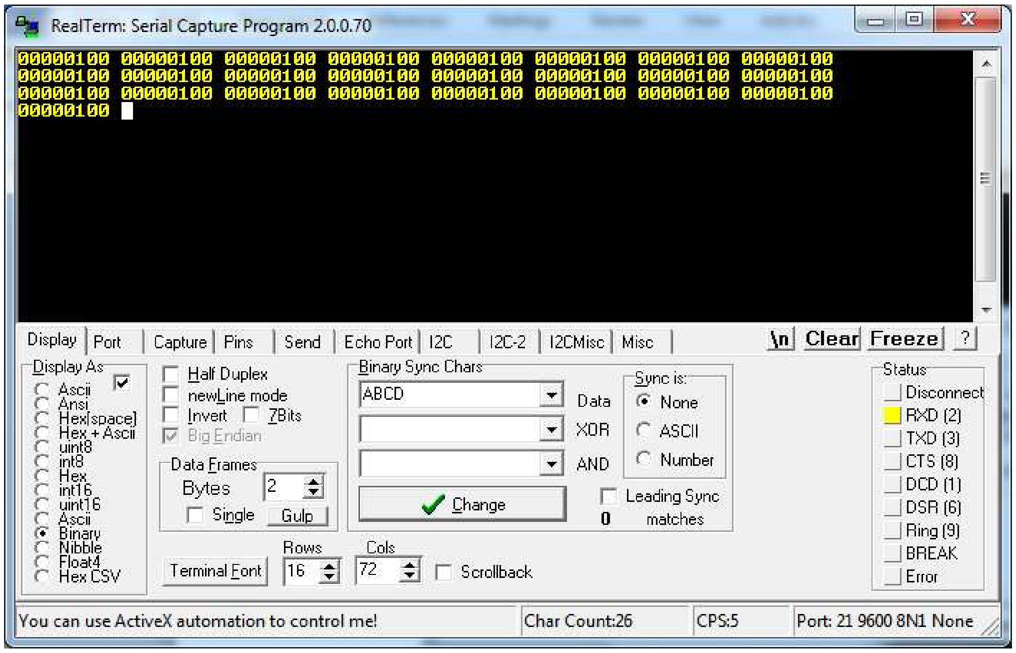
Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	&#32;	Space	64	40	100	&#64;	@	96	60	140	&#96;	`
1	1	001	SOH (start of heading)	33	21	041	&#33;	!	65	41	101	&#65;	A	97	61	141	&#97;	a
2	2	002	STX (start of text)	34	22	042	&#34;	"	66	42	102	&#66;	B	98	62	142	&#98;	b
3	3	003	ETX (end of text)	35	23	043	&#35;	#	67	43	103	&#67;	C	99	63	143	&#99;	c
4	4	004	EOT (end of transmission)	36	24	044	&#36;	$	68	44	104	&#68;	D	100	64	144	&#100;	d
5	5	005	ENQ (enquiry)	37	25	045	&#37;	%	69	45	105	&#69;	E	101	65	145	&#101;	e
6	6	006	ACK (acknowledge)	38	26	046	&#38;	&	70	46	106	&#70;	F	102	66	146	&#102;	f
7	7	007	BEL (bell)	39	27	047	&#39;	'	71	47	107	&#71;	G	103	67	147	&#103;	g
8	8	010	BS (backspace)	40	28	050	&#40;	(	72	48	110	&#72;	H	104	68	150	&#104;	h
9	9	011	TAB (horizontal tab)	41	29	051	&#41;	)	73	49	111	&#73;	I	105	69	151	&#105;	i
10	A	012	LF (NL line feed, new line)	42	2A	052	&#42;	*	74	4A	112	&#74;	J	106	6A	152	&#106;	j
11	B	013	VT (vertical tab)	43	2B	053	&#43;	+	75	4B	113	&#75;	K	107	6B	153	&#107;	k
12	C	014	FF (NP form feed, new page)	44	2C	054	&#44;	,	76	4C	114	&#76;	L	108	6C	154	&#108;	l
13	D	015	CR (carriage return)	45	2D	055	&#45;	-	77	4D	115	&#77;	M	109	6D	155	&#109;	m
14	E	016	SO (shift out)	46	2E	056	&#46;	.	78	4E	116	&#78;	N	110	6E	156	&#110;	n
15	F	017	SI (shift in)	47	2F	057	&#47;	/	79	4F	117	&#79;	O	111	6F	157	&#111;	o
16	10	020	DLE (data link escape)	48	30	060	&#48;	0	80	50	120	&#80;	P	112	70	160	&#112;	p
17	11	021	DC1 (device control 1)	49	31	061	&#49;	1	81	51	121	&#81;	Q	113	71	161	&#113;	q
18	12	022	DC2 (device control 2)	50	32	062	&#50;	2	82	52	122	&#82;	R	114	72	162	&#114;	r
19	13	023	DC3 (device control 3)	51	33	063	&#51;	3	83	53	123	&#83;	S	115	73	163	&#115;	s
20	14	024	DC4 (device control 4)	52	34	064	&#52;	4	84	54	124	&#84;	T	116	74	164	&#116;	t
21	15	025	NAK (negative acknowledge)	53	35	065	&#53;	5	85	55	125	&#85;	U	117	75	165	&#117;	u
22	16	026	SYN (synchronous idle)	54	36	066	&#54;	6	86	56	126	&#86;	V	118	76	166	&#118;	v
23	17	027	ETB (end of trans. block)	55	37	067	&#55;	7	87	57	127	&#87;	W	119	77	167	&#119;	w
24	18	030	CAN (cancel)	56	38	070	&#56;	8	88	58	130	&#88;	X	120	78	170	&#120;	x
25	19	031	EM (end of medium)	57	39	071	&#57;	9	89	59	131	&#89;	Y	121	79	171	&#121;	y
26	1A	032	SUB (substitute)	58	3A	072	&#58;	:	90	5A	132	&#90;	Z	122	7A	172	&#122;	z
27	1B	033	ESC (escape)	59	3B	073	&#59;	;	91	5B	133	&#91;	[	123	7B	173	&#123;	{
28	1C	034	FS (file separator)	60	3C	074	&#60;	<	92	5C	134	&#92;	\	124	7C	174	&#124;	|
29	1D	035	GS (group separator)	61	3D	075	&#61;	=	93	5D	135	&#93;	]	125	7D	175	&#125;	}
30	1E	036	RS (record separator)	62	3E	076	&#62;	>	94	5E	136	&#94;	^	126	7E	176	&#126;	~
31	1F	037	US (unit separator)	63	3F	077	&#63;	?	95	5F	137	&#95;	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

Here we can see that the number 4 represents the represents "EOT" character which we see displayed. Note the same data is always being sent, we are simply changing how it is displayed on the screen – as a character "EOT", or as the numerical binary data 4 – but they ones and zeros are always the same.

To see the actual ones and zeros

- go back to the Display tab and select Binary



RealTerm: Serial Capture Program 2.0.0.70

```
00000100 00000100 00000100 00000100 00000100 00000100 00000100 00000100
00000100 00000100 00000100 00000100 00000100 00000100 00000100 00000100
00000100 00000100 00000100 00000100 00000100 00000100 00000100 00000100
00000100
```

Display As: Ascii, Ansi, Hex[space], Hex + Ascii, uint8, int8, Hex, int16, uint16, Ascii, [✓ Binary], Nibble, Float4, Hex CSV

Half Duplex, newLine mode, Invert, 7Bits, [✓ Big Endian]

Data Frames: Bytes 2, Single, Gulp

Binary Sync Chars: ABCD Data; XOR; AND

Sync is: [✓ None], ASCII, Number

Leading Sync; 0 matches

Change

Terminal Font; Rows 16; Cols 72; Scrollback

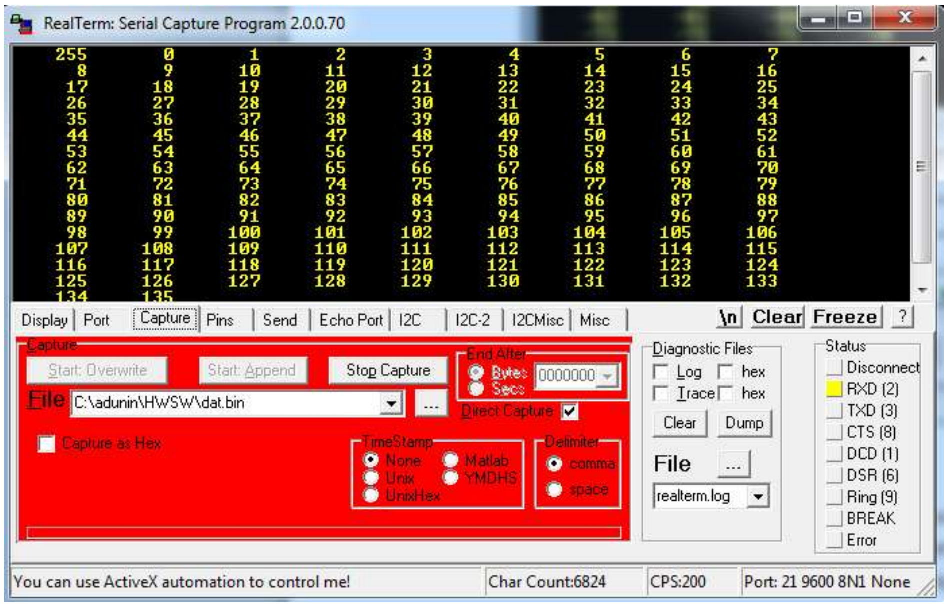
Status: Disconnect, RXD (2), TXD (3), CTS (8), DCD (1), DSR (6), Ring (9), BREAK, Error

You can use ActiveX automation to control me! | Char Count:26 | CPS:5 | Port: 21 9600 8N1 None

Now the actual binary representation of the data is seen: 00000100 is the number 4, or the character "EOT", but the ones and zeros are always the same.

Writing the data to a file with RealTerm:

- Open RealTerm, set the baud rate and port, and connect
- 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



```
255    0    1    2    3    4    5    6    7
  8    9   10   11   12   13   14   15   16
 17   18   19   20   21   22   23   24   25
 26   27   28   29   30   31   32   33   34
 35   36   37   38   39   40   41   42   43
 44   45   46   47   48   49   50   51   52
 53   54   55   56   57   58   59   60   61
 62   63   64   65   66   67   68   69   70
 71   72   73   74   75   76   77   78   79
 80   81   82   83   84   85   86   87   88
 89   90   91   92   93   94   95   96   97
 98   99  100  101  102  103  104  105  106
107  108  109  110  111  112  113  114  115
116  117  118  119  120  121  122  123  124
125  126  127  128  129  130  131  132  133
134  135
```

0000000

File C:\adunin\HWSW\dat.bin

Direct Capture ✓

Capture as Hex

TimeStamp: None ● / Unix / UnixHex / Matlab / YMDHS

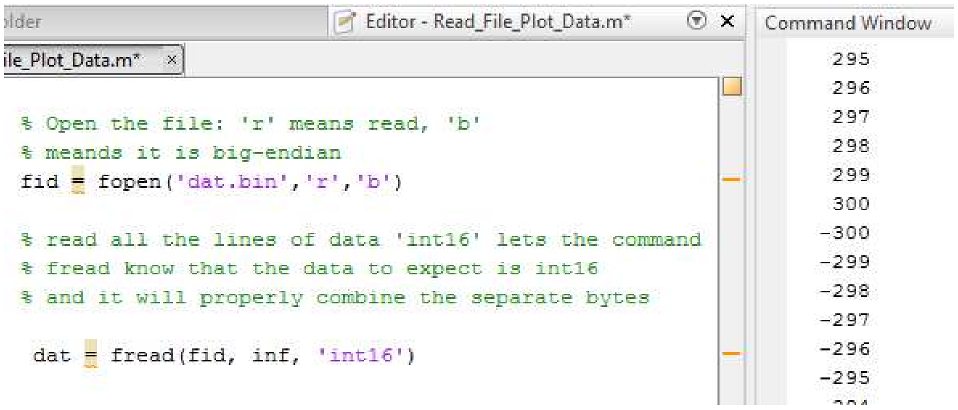
Delimiter: comma ● / space

realterm.log

You can use ActiveX automation to control me! | Char Count:6824 | CPS:200 | Port: 21 9600 8N1 None

Reading the data file with Matlab:

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



```
% Open the file: 'r' means read, 'b'
% meands 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

 dat = fread(fid, inf, 'int16')
```

```
 295
 296
 297
 298
 299
 300
-300
-299
-298
-297
-296
-295
```

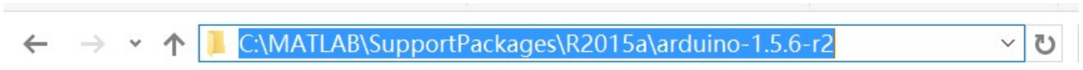
Part 4: Sending data with the Arduino IDE (optional)

Objectives:

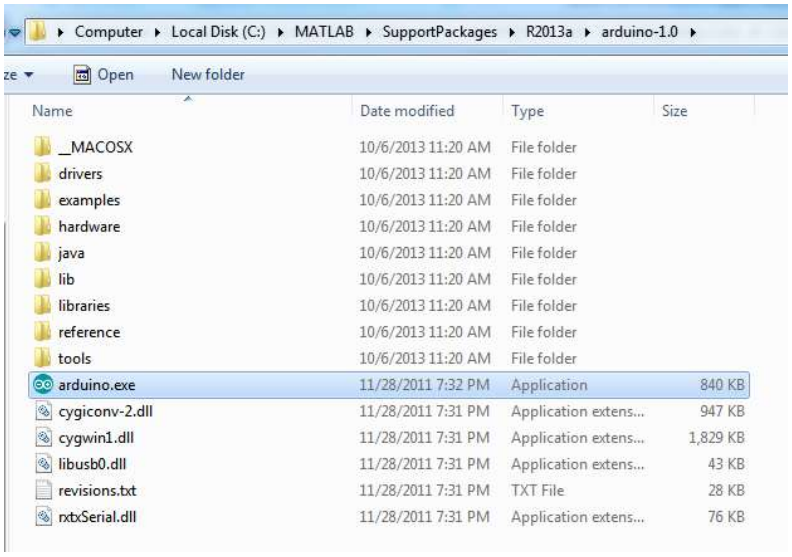
- Use the Arduino IDE to send ascii encoded data and binary data

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(R2014a)\arduino-1.0(arduino-1.0.5)
    - C:\MATLAB\SupportPackages\R2015a\arduino-1.5.6-r2 (You can copy this path and directly paste it in "This computer" to find "arduino.exe")
        - 2015a has both a arduino-1.0 folder an arduino-1.5 folder – either will work but 1.5 is the latest version so use this.

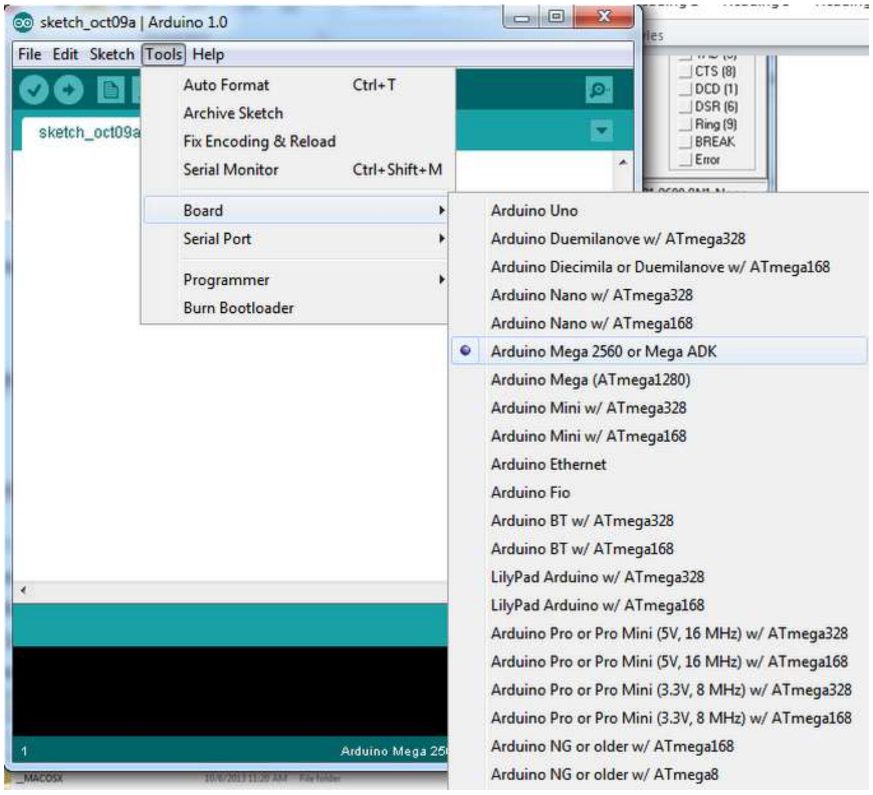


- The Arduino IDE can be opened from here:

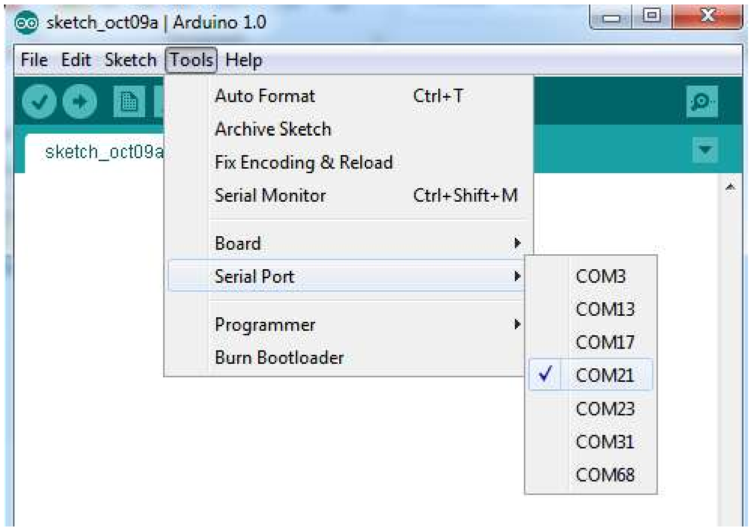

Computer ▸ Local Disk (C:) ▸ MATLAB ▸ SupportPackages ▸ R2013a ▸ arduino-1.0 ▸
Open
New folder

Name	Date modified	Type	Size
__MACOSX	10/6/2013 11:20 AM	File folder	
drivers	10/6/2013 11:20 AM	File folder	
examples	10/6/2013 11:20 AM	File folder	
hardware	10/6/2013 11:20 AM	File folder	
java	10/6/2013 11:20 AM	File folder	
lib	10/6/2013 11:20 AM	File folder	
libraries	10/6/2013 11:20 AM	File folder	
reference	10/6/2013 11:20 AM	File folder	
tools	10/6/2013 11:20 AM	File folder	
arduino.exe	11/28/2011 7:32 PM	Application	840 KB
cygiconv-2.dll	11/28/2011 7:31 PM	Application extens...	947 KB
cygwin1.dll	11/28/2011 7:31 PM	Application extens...	1,829 KB
libusb0.dll	11/28/2011 7:31 PM	Application extens...	43 KB
revisions.txt	11/28/2011 7:31 PM	TXT File	28 KB
rxtxSerial.dll	11/28/2011 7:31 PM	Application extens...	76 KB

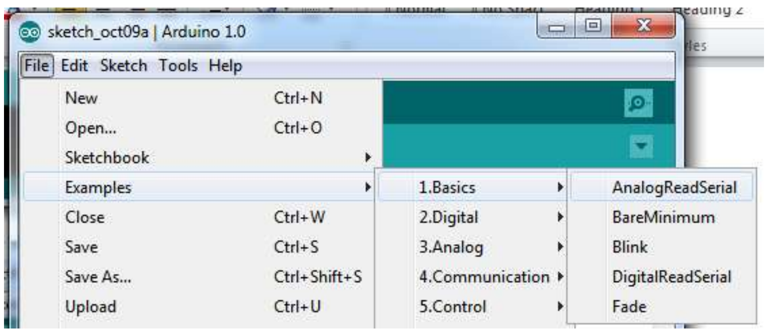
- Select your target board:


sketch_oct09a | Arduino 1.0
File Edit Sketch Tools Help
sketch_oct09a
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Serial Monitor Ctrl+Shift+M
Board
Serial Port
Programmer
Burn Bootloader
CTS (8)
DCD (1)
DSR (6)
Ring (9)
BREAK
Error
Arduino Uno
Arduino Duemilanove w/ ATmega328
Arduino Diecimila or Duemilanove w/ ATmega168
Arduino Nano w/ ATmega328
Arduino Nano w/ ATmega168
● Arduino Mega 2560 or Mega ADK
Arduino Mega (ATmega1280)
Arduino Mini w/ ATmega328
Arduino Mini w/ ATmega168
Arduino Ethernet
Arduino Fio
Arduino BT w/ ATmega328
Arduino BT w/ ATmega168
LilyPad Arduino w/ ATmega328
LilyPad Arduino w/ ATmega168
Arduino Pro or Pro Mini (5V, 16 MHz) w/ ATmega328
Arduino Pro or Pro Mini (5V, 16 MHz) w/ ATmega168
Arduino Pro or Pro Mini (3.3V, 8 MHz) w/ ATmega328
Arduino Pro or Pro Mini (3.3V, 8 MHz) w/ ATmega168
Arduino NG or older w/ ATmega168
Arduino NG or older w/ ATmega8
1
Arduino Mega 25

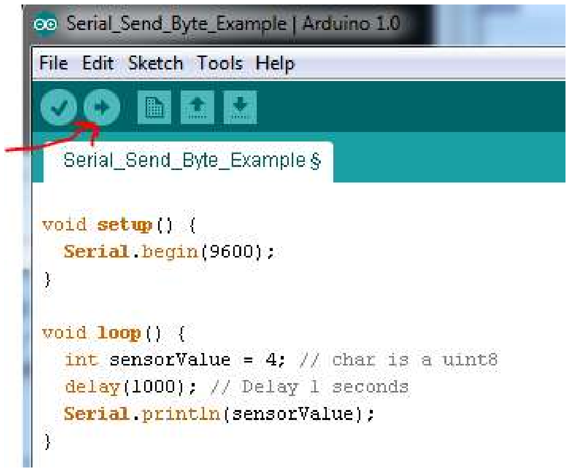
- Select the serial port:



- Open a basic serial example:

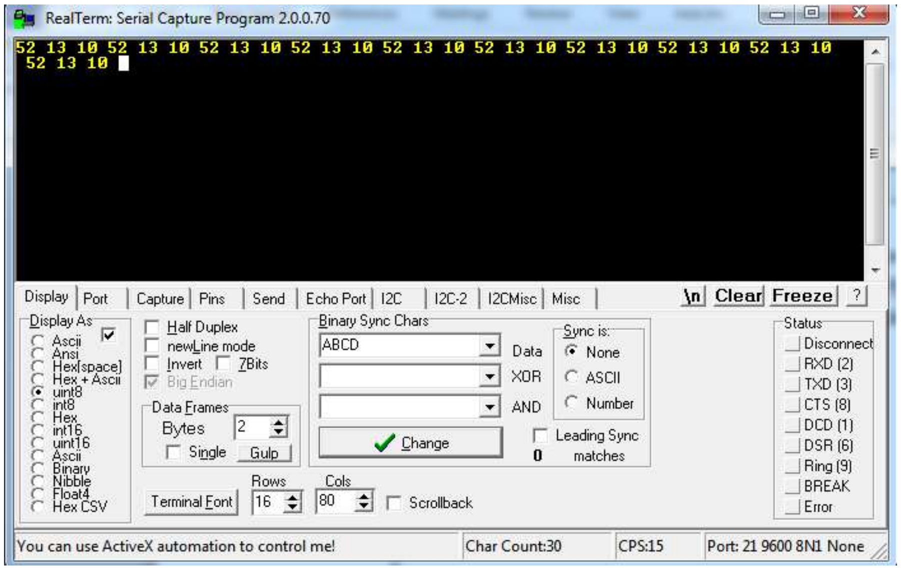


- **Modify** the code to send only the number 4:
    - Download the code to the board with the Right arrow
    - Note – you must make sure the serial port from RealTerm is closed – it cannot download code when the serial port is open in another application

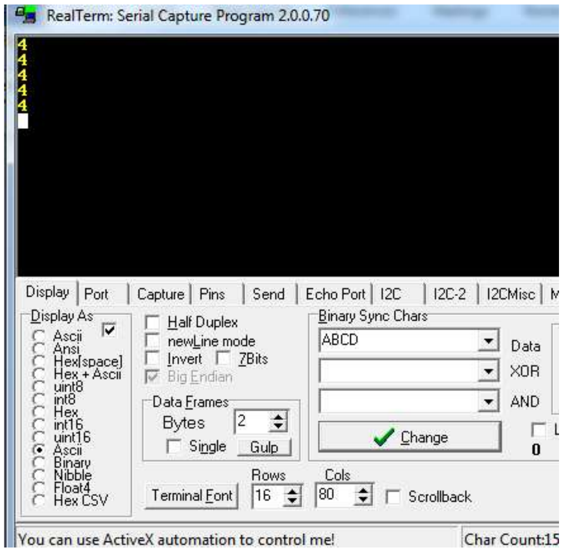

Serial_Send_Byte_Example | Arduino 1.0
File Edit Sketch Tools Help
Serial_Send_Byte_Example §

```
void setup() {
  Serial.begin(9600);
}

void loop() {
  int sensorValue = 4; // char is a uint8
  delay(1000); // Delay 1 seconds
  Serial.println(sensorValue);
}
```

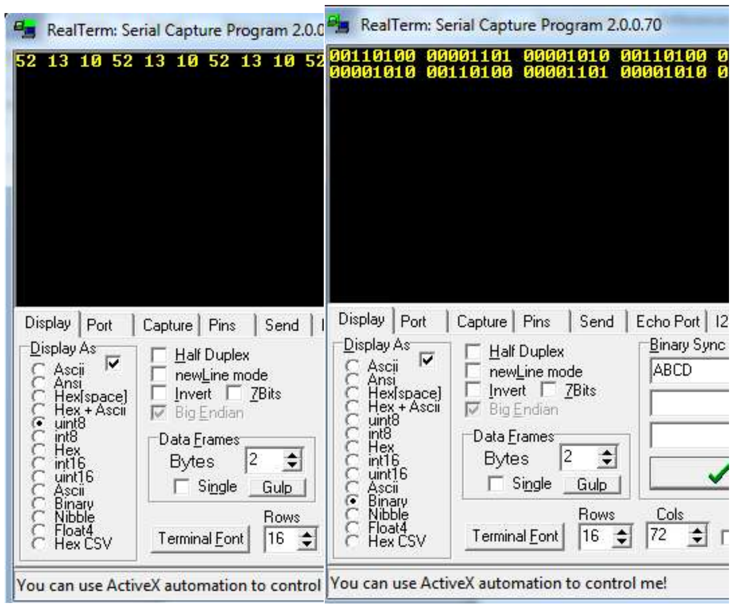
    - Open the terminal program to view


```
52 13 10 52 13 10 52 13 10 52 13 10 52 13 10 52 13 10 52 13 10 52 13 10 52 13 10
 52 13 10
```

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


```
4
4
4
4
4
```
RealTerm: Serial Capture Program 2.0.0.70
Display As: [✓ Ascii]
Bytes 2
Rows 16
Cols 80
You can use ActiveX automation to control me!
Char Count:15

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):



```
52 13 10 52 13 10 52 13 10 52
```

```
00110100 00001101 00001010 00110100 0
00001010 00110100 00001101 00001010 0
```

RealTerm: Serial Capture Program 2.0.0
RealTerm: Serial Capture Program 2.0.0.70
Display As: uint8
Display As: Binary
Binary Sync: ABCD
Data Frames Bytes 2
Rows 16
Cols 72
You can use ActiveX automation to control me!

If you want to write the data in binary (that is 4 as 0000100) use the Serial.Write command.

Questions

A analog sensor records a vlaue 16 (the result of the Analog to Digital conversion). You want to send this data (the number 16) from the microprocessor to your computer.

- How many bytes does it take to send this data in binary format?
  - What is the binary (ones and zeros) representaiton of this data?
- How many bytes does it take to send this data ASCII encoded?
  - What is the binary (ones and zeros) representation of the ASCII encoded data?

Now the sensors reads 32768.

- How many bytes does it take to send this data in binary format?
  - What is the binary (ones and zeros) representaiton of this data?
- How many bytes does it take to send this data ASCII encoded?
  - What is the binary (ones and zeros) representation of the ASCII encoded data?

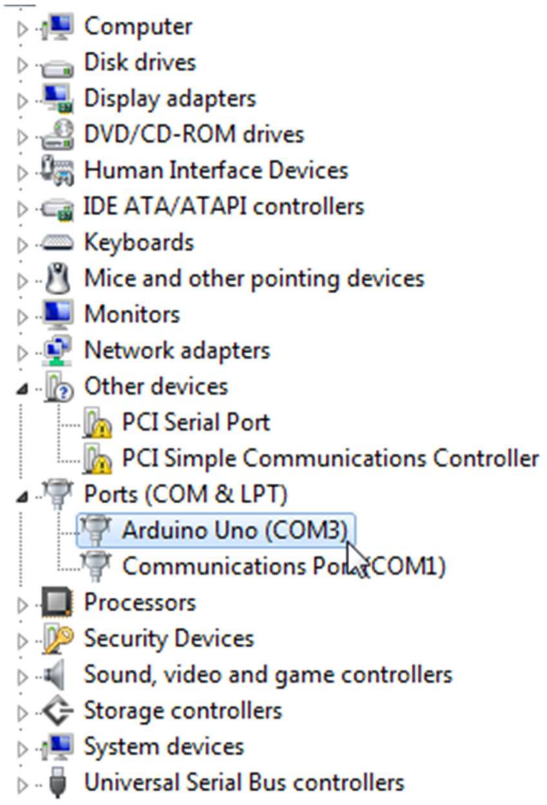
- Which of these formats would be “fastest” to send?

Find Port Number on Windows (Mathworks)

1. Connect the Arduino board to one of the USB ports on your computer running MATLAB® Support Package for Arduino Hardware.

   If you are connecting the board for the first time, ensure that the driver installation is also complete.

2. Open **Device Manager** (In the search box on the Windows Taskbar, type `Device Manager`).
3. In the Device Manager window, expand the `Ports (COM & LPT)` list.
4. Note the port name (`COM#`) that displays the Arduino board's name.



You will be using this name (for example, "`COM3`") as the value in the lab m-files and the value "3" is used in the hardware settings.

Find Port Number on Macintosh (Mathworks)

1. Connect the Arduino board to one of the USB ports on your Mac running MATLAB Support Package for Arduino Hardware.

   If you are connecting the board for the first time, ensure that the driver installation is also complete.
2. Open Terminal in macOS. To do this, you can either use Spotlight Search (search for `Terminal`) or use the Finder window (launch Finder, click Go > Utilities, and double-click **Terminal**).
3. At the command prompt in the Terminal, go to the root directory (run this command: `cd /`).
4. In the root directory, run this command: `ls /dev/*`.
5. Note the port name `/dev/tty.usbmodem*` or `/dev/tty.usbserial*`.

   You will be using this name (for example, `/dev/tty.usbmodem1421`) as the value of `Port` in the m-files.

Works Cited

Mathworks. (n.d.). Retrieved from https://www.mathworks.com/help/matlab/supportpkg/find-arduino-port-on-windows-mac-and-linux.html