

RENSSELAER MECHATRONICS
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. 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 external mode.

This lab explores sending data with Simulink using both built in blocks, system object blocks and with the Arduino IDE.

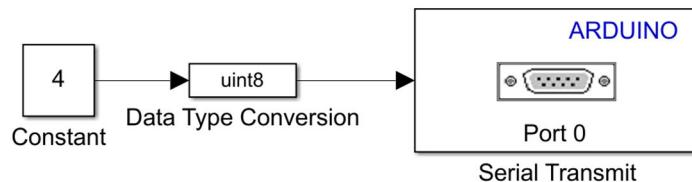
Part 1: Sending 8 bit data with Simulink

Objectives:

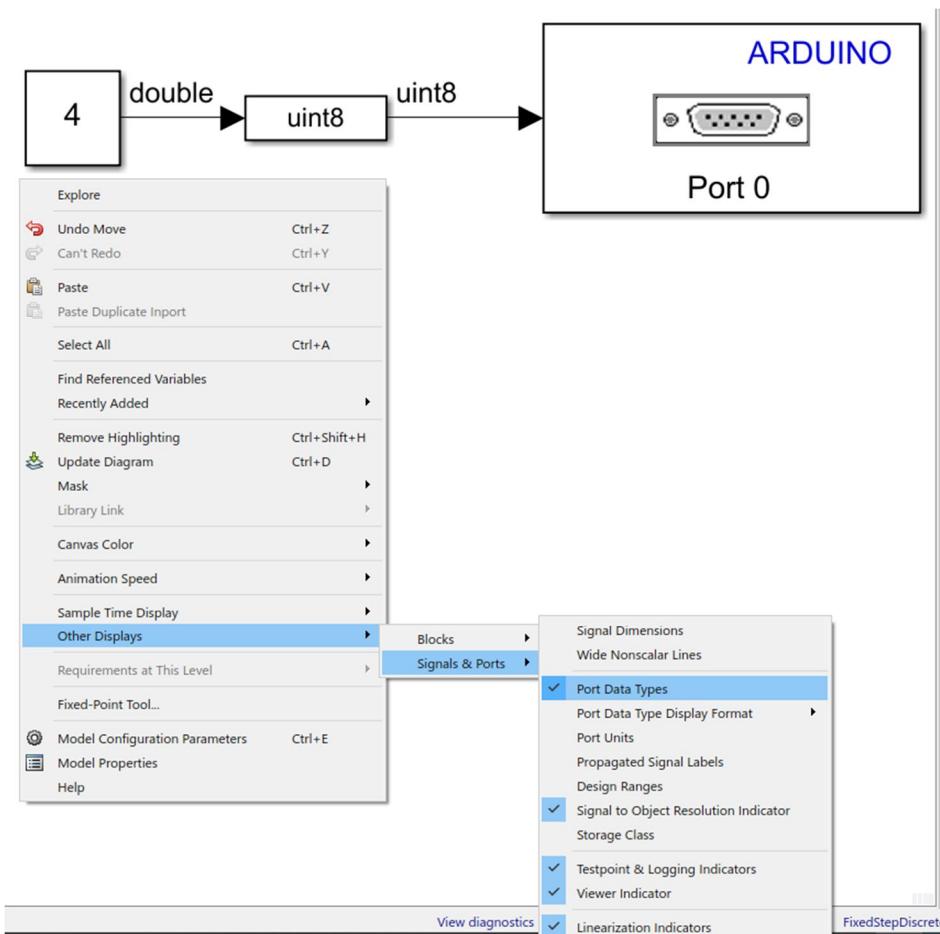
- Send 8 bit data over the serial link using the supported Simulink blocks

Simulink Model:

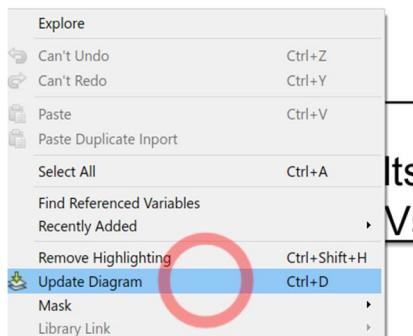
- Build the Simulink diagram form a “Constant”, “Data Type Conversion” and a “Serial Transmit” block. Be sure to use the blinking LED Simulink diagram and create this new one from that one.



- 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”:

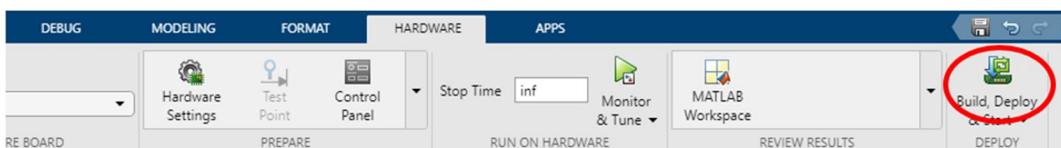


- Press “Control+D” to update the diagram see the data types, right-click in the diagram to Update the Diagram:

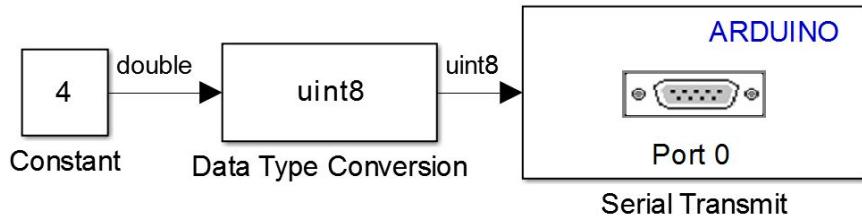


The input data type for a Serial Transmit is uint8 (a single byte, 8 bits, of data). This means that data must be converted to unit8 before sending across the serial line with this block.

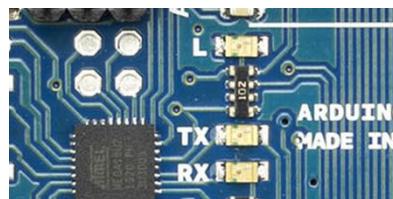
- Build and run the following Simulink diagram.
 - Click the the ‘Build, Deploy & Start’ button:



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

```

1 % Read from Serial port 14:
2 s = serial('COM14');
3 set(s,'ByteOrder', 'bigEndian','BaudRate', 9600);
4 % Open Serial Port:
5 fopen(s);
6 % Read 30 data points
7 d1=fread(s, 30, 'uint8'))'
8
9 % Close the serial port:
10 newobjs=instrfindall;
11 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      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('COM10');
set(s,'ByteOrder', 'bigEndian','BaudRate', 9600);
% 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 the serial port:
newobjs=instrfindall;
fclose(newobjs);

```

The result

```

d1_characters =

'oooooooooooooooooooooooooooooooooooo'

```

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	000	NUL (null)	32	20 040	000	 	Space	64	40 100	000	@	Ø	96	60 140	000	`	`
1	1 001	001	SOH (start of heading)	33	21 041	001	!	!	65	41 101	001	A	A	97	61 141	001	a	a
2	2 002	002	STX (start of text)	34	22 042	002	"	"	66	42 102	002	B	B	98	62 142	002	b	b
3	3 003	003	ETX (end of text)	35	23 043	003	#	#	67	43 103	003	C	C	99	63 143	003	c	c
4	4 004	004	EOT (end of transmission)	36	24 044	004	$	\$	68	44 104	004	D	D	100	64 144	004	d	d
5	5 005	005	ENQ (enquiry)	37	25 045	005	%	%	69	45 105	005	E	E	101	65 145	005	e	e
6	6 006	006	ACK (acknowledge)	38	26 046	006	&	&	70	46 106	006	F	F	102	66 146	006	f	f
7	7 007	007	BEL (bell)	39	27 047	007	'	'	71	47 107	007	G	G	103	67 147	007	g	g
8	8 010	010	BS (backspace)	40	28 050	010	((72	48 110	010	H	H	104	68 150	010	h	h
9	9 011	011	TAB (horizontal tab)	41	29 051	011))	73	49 111	011	I	I	105	69 151	011	i	i
10	A 012	012	LF (NL line feed, new line)	42	2A 052	012	*	*	74	4A 112	012	J	J	106	6A 152	012	j	j
11	B 013	013	VT (vertical tab)	43	2B 053	013	+	+	75	4B 113	013	K	K	107	6B 153	013	k	k
12	C 014	014	FF (NP form feed, new page)	44	2C 054	014	,	,	76	4C 114	014	L	L	108	6C 154	014	l	l
13	D 015	015	CR (carriage return)	45	2D 055	015	-	-	77	4D 115	015	M	M	109	6D 155	015	m	m
14	E 016	016	SO (shift out)	46	2E 056	016	.	.	78	4E 116	016	N	N	110	6E 156	016	n	n
15	F 017	017	SI (shift in)	47	2F 057	017	/	/	79	4F 117	017	O	O	111	6F 157	017	o	o
16	10 020	020	DLE (data link escape)	48	30 060	020	0	0	80	50 120	020	P	P	112	70 160	020	p	p
17	11 021	021	DC1 (device control 1)	49	31 061	021	1	1	81	51 121	021	Q	Q	113	71 161	021	q	q
18	12 022	022	DC2 (device control 2)	50	32 062	022	2	2	82	52 122	022	R	R	114	72 162	022	r	r
19	13 023	023	DC3 (device control 3)	51	33 063	023	3	3	83	53 123	023	S	S	115	73 163	023	s	s
20	14 024	024	DC4 (device control 4)	52	34 064	024	4	4	84	54 124	024	T	T	116	74 164	024	t	t
21	15 025	025	NAK (negative acknowledgement)	53	35 065	025	5	5	85	55 125	025	U	U	117	75 165	025	u	u
22	16 026	026	SYN (synchronous idle)	54	36 066	026	6	6	86	56 126	026	V	V	118	76 166	026	v	v
23	17 027	027	ETB (end of trans. block)	55	37 067	027	7	7	87	57 127	027	W	W	119	77 167	027	w	w
24	18 030	030	CAN (cancel)	56	38 070	030	8	8	88	58 130	030	X	X	120	78 170	030	x	x
25	19 031	031	EM (end of medium)	57	39 071	031	9	9	89	59 131	031	Y	Y	121	79 171	031	y	y
26	1A 032	032	SUB (substitute)	58	3A 072	032	:	:	90	5A 132	032	Z	Z	122	7A 172	032	z	z
27	1B 033	033	ESC (escape)	59	3B 073	033	;	:	91	5B 133	033	[[123	7B 173	033	{	{
28	1C 034	034	FS (file separator)	60	3C 074	034	<	<	92	5C 134	034	\	\	124	7C 174	034	|	
29	1D 035	035	GS (group separator)	61	3D 075	035	=	=	93	5D 135	035]]	125	7D 175	035	}	}
30	1E 036	036	RS (record separator)	62	3E 076	036	>	>	94	5E 136	036	^	^	126	7E 176	036	~	~
31	1F 037	037	US (unit separator)	63	3F 077	037	?	?	95	5F 137	037	_	_	127	7F 177	037		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 =
30x3 char array
'100'
'100'
```

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

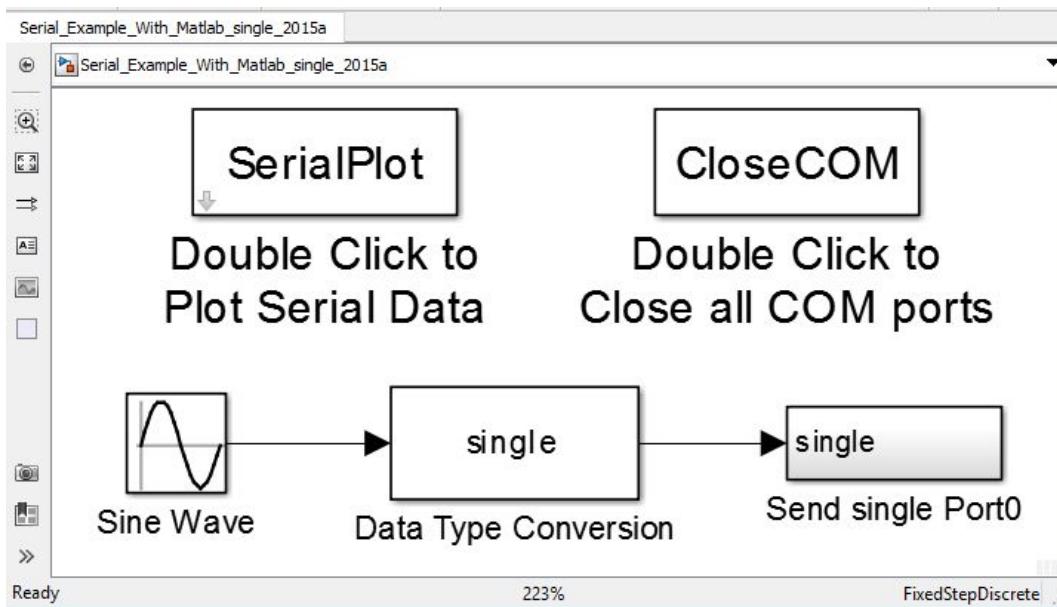
Part 2: Sending and Receiving Data with RASPlib

Objectives:

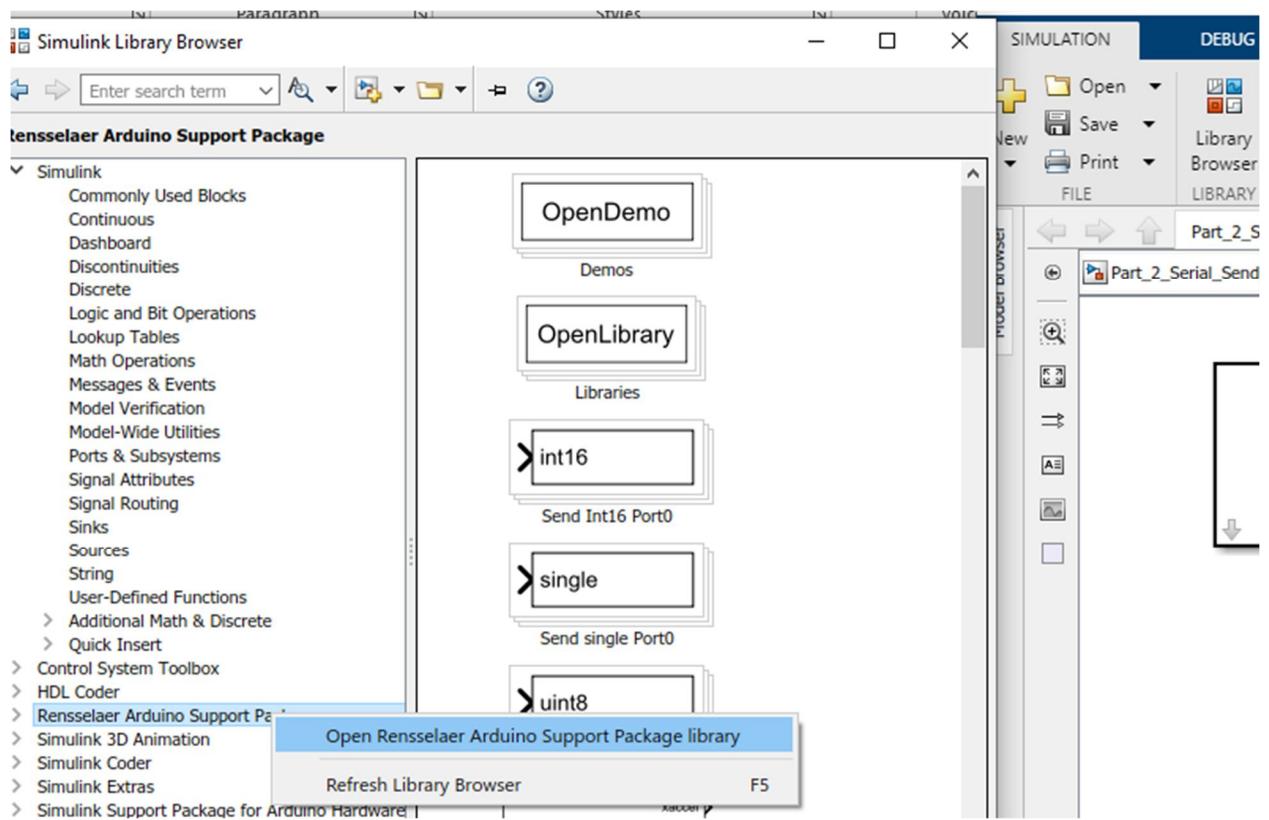
- Use RASPlib to send, plot and store data. Data can be sent and received faster than external mode. The exact number of bytes sent each time step is specified.

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’.)



SerialPlot, CloseCOM, and “Send single Port0” are all obtained from RASPlib. Open Rasplib from the “Simulation” tab and select “Library Browser” then right click “Rensselaer Arduino Support Package Library”:



Then drag the blocks to your diagram:

 RASLib ▶

Device Specific Libraries:

[OpenLibrary](#) [OpenDemo](#)

Libraries **Demos**

I2C Sensors:

- soHMC5883L: xmag, ymag, zmag
- soMPU6050Gyro: xvel, yvel, zvel
- soMPU6050Accel: xaccel, yaccel, zaccel
- soMPU6050Temp: temp
- Temperature: Temperature
- soMPU9250RAW: xaccel, yaccel, zaccel, xgyro, ygyro, zgyro, xmag, ymag, zmag, temp
- MPU9250 Raw: xmag, ymag, zmag
- soMPU9250Mag: xmag, ymag, zmag
- VL53L0X Distance Sensor: soVL53L0X, d_mm

Serial Communication & Plotting:

- soBMP280 Barometer: pres_Pa, absAlt_m, temp_C
- soBMP180 Barometer: pres_Pa, absAlt_m, temp_C
- MS5611 Barometer: pres_Pa, temp_C, absAlt_m, relAlt_m
- soMS5611BaroRAW: pres_Pa, temp_C
- soMS5611BaroPT: pres_Pa, temp_C
- BMI160 Gyroscope: xvel, yvel, zvel

Send: uint8, int16, single

Plot: SerialPlot, CloseCOM

Double Click to Plot Serial Data

Double Click to Close all COM ports

Ensure all serial ports are closed:
(try when code will not download)

Generic Motor Drivers and PWM Frequency:

- Motor Driver - Speed & Direction: Volts, Vsupply=3.25
- Motor Driver - Speed & Direction With Table Lookup: Volts, Vsupply=3.25

Mega 2560 PWM Frequency Select pins 6, 7, 8 Frequency 32KHz

Mega 2560 PWM Frequency Select pins 5, 6 Frequency 490Hz

Uno or Nano PWM Frequency Select

Generic Sensors:

soHCsr04Sonar_d_cm: Sonar Distance Sensor

Generic IO:

Analog Digital Read y: Mega Read Analog as Digital

Analog Digital Read y: Uno Nano Read Analog as Digital

Digital Pulup Read y: Digital Read with Pulup Enabled

Note: MPU9250 sensors can use all MPU6050 sensor blocks (recommended)
MPU9250 Magnetometer readings are slower than gyro and accel readings,
so fast applications should use MPU6050 blocks when reading MPU9250

Use solver settings:

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

Select:

- Solver
- Data Import/Export
- > Optimization
- > Diagnostics
- Hardware Implementation
- Model Referencing
- > Simulation Target
- Run on Target Hardware

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

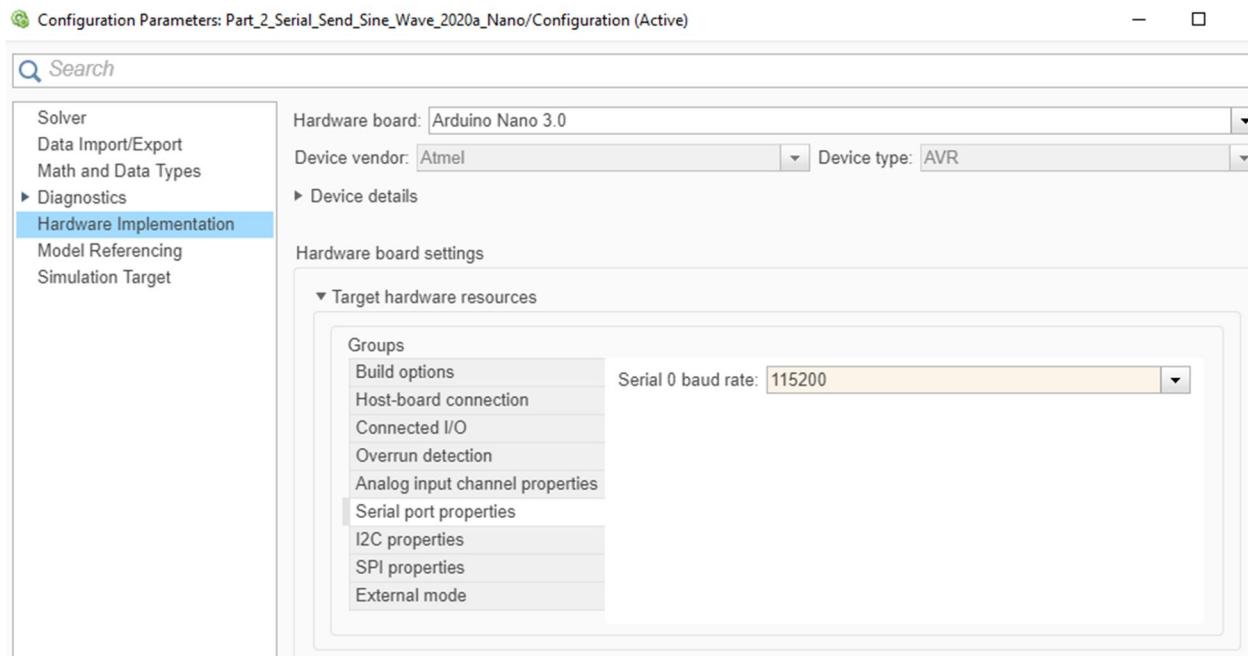
Tasking mode for periodic sample times: SingleTasking

Automatically handle rate transition for data transfer

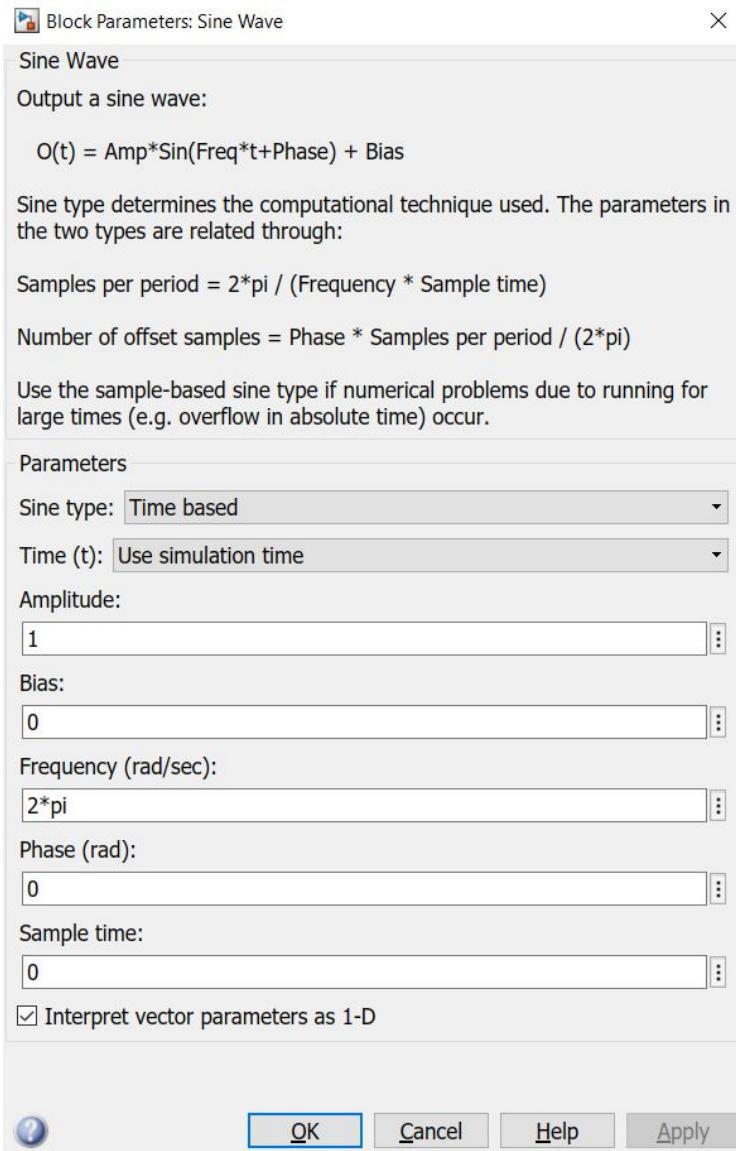
Higher priority value indicates higher task priority

BAUD Rate Calculation

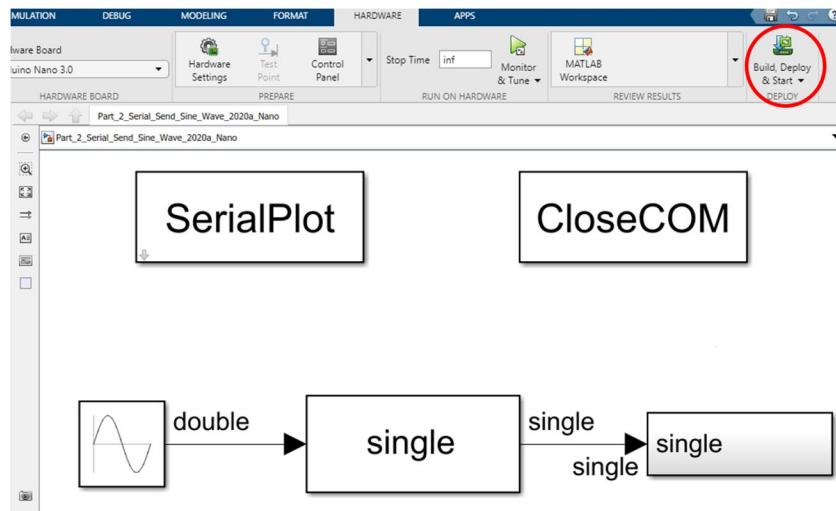
The default setting for the BAUD rate is 9600. In the above case the sample time is .002 second, and we are sending a single which is 4 bytes. The amount of bits (8 bites in a byte) per second is then $4*8/.002 = 16000$ bits per second. This means the minimum BAUD rate for a single data channel at .002 is 1600. Choose 115200, which is more than fast enough. Rates faster than 115200 seem to start having hardware issues and seem to be less reliable. Access this menu from the “Hardware” Tab then “Hardware Settings”



And a sine wave with magnitude 1 and frequency 2π :



Next download the code to the hardware:

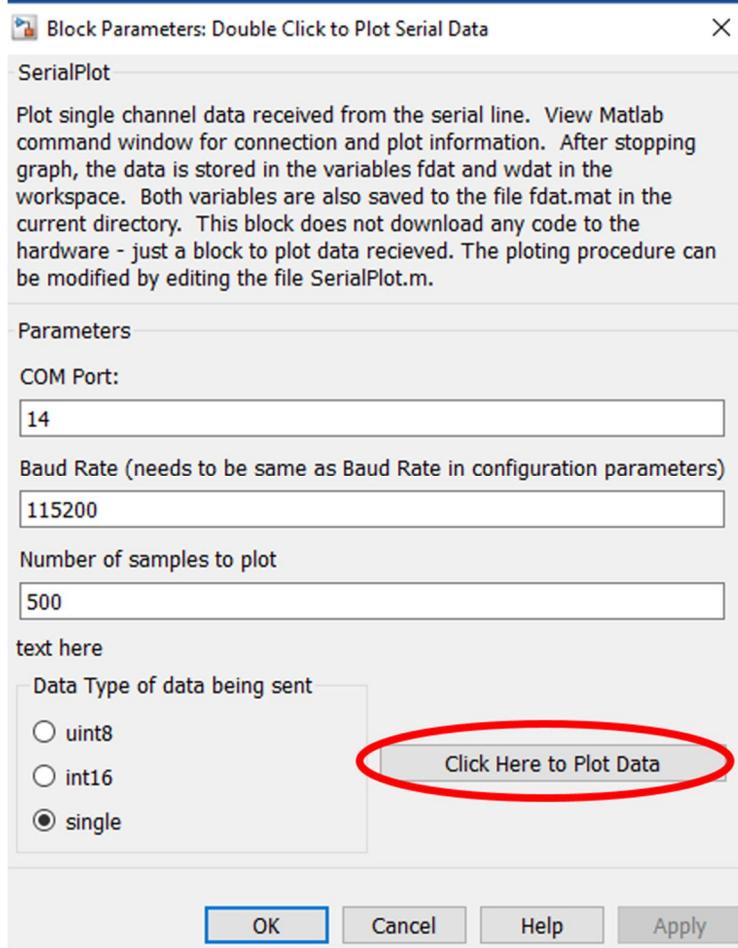


Note that the “Build, Deploy & Start” button downloads the code to the hardware only. After it downloads it will only execute the code you specify, in this case sending data from a sine wave to port 0 as a “single” floating point number (4 bytes).

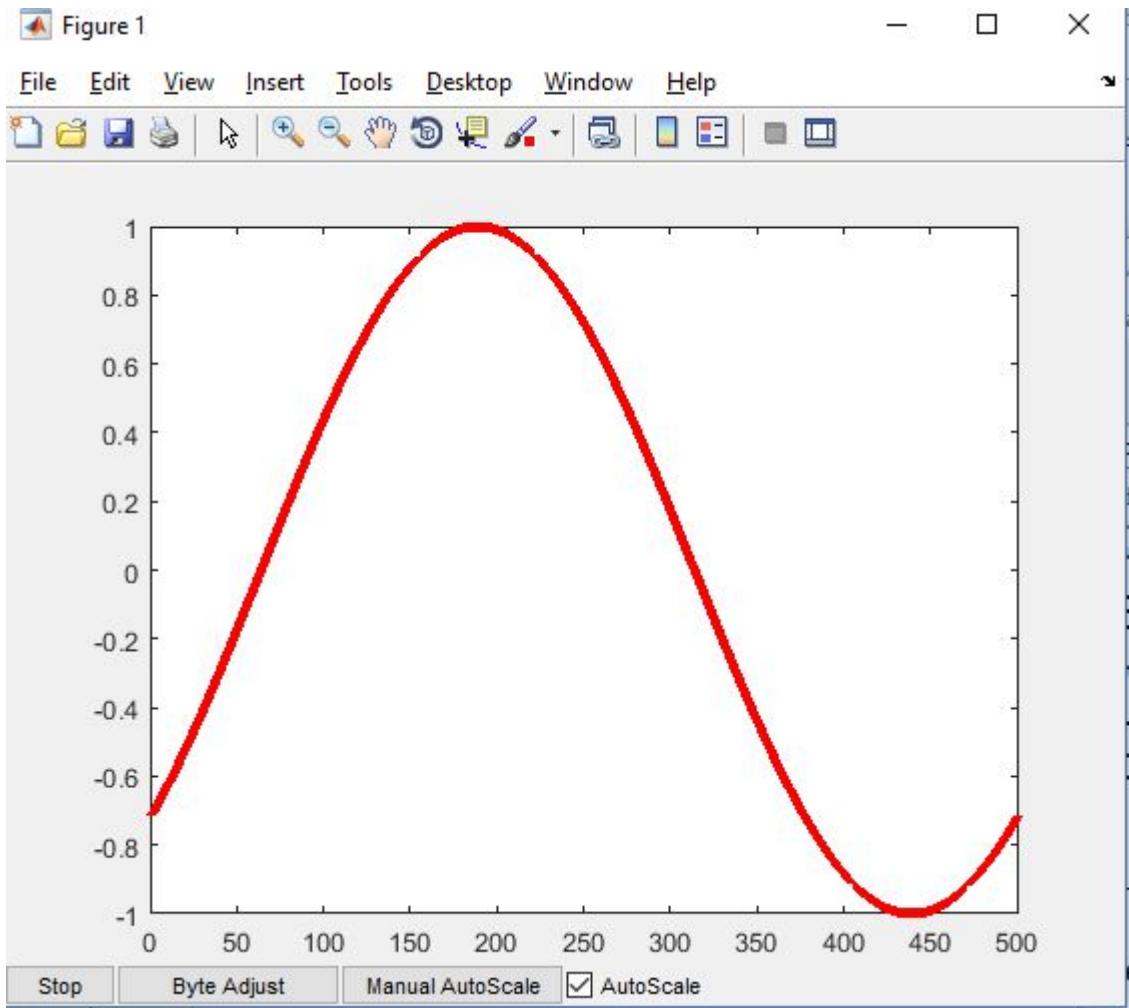
After the code has been downloaded the “Tx” light on the board should light up indicating data is being sent to the computer.

- 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.

- Then click “Click Here to Plot Data”



You should observe a sine wave of frequency 2π 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

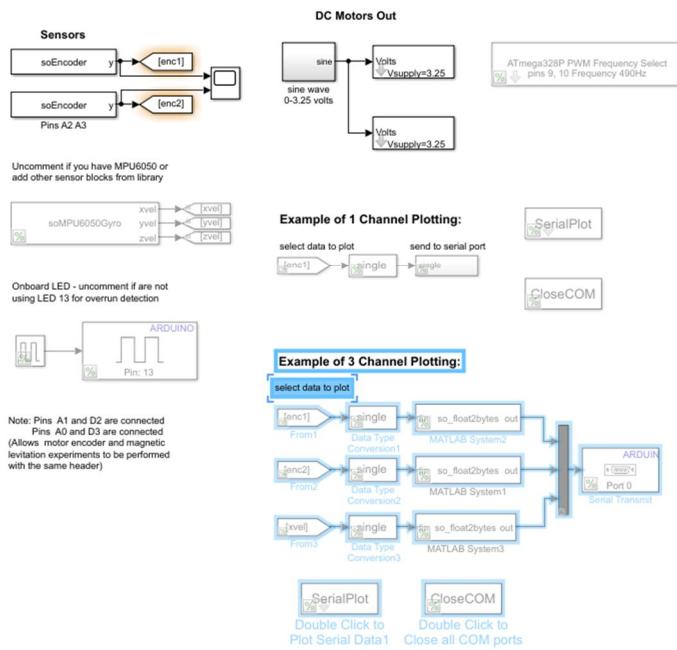
Open the example “Demo_MinSegNano.slx” located in the RASPLib\examples folder:

Current Folder			
	Name	Date Modified	Type
	README.txt	8/31/2016 5:28 PM	TXT File
	license	3/26/2014 6:12 AM	File
	COPYING	3/25/2014 8:53 PM	File
	slblocks.m	3/5/2014 11:33 AM	Function
+	blocks	9/16/2020 7:36 AM	Folder
+	include	9/8/2020 12:16 PM	Folder
+	slprj	9/8/2020 12:16 PM	Folder
+	src	9/8/2020 12:16 PM	Folder
-	examples	9/8/2020 12:16 PM	Folder
	Demo_MinSegNano_V3.slx	8/17/2020 5:03 PM	Simulink Model
	Demo_M1V4_9250.slx	8/17/2020 4:40 PM	Simulink Model
	Demo_M2V5.slx	8/17/2020 4:20 PM	Simulink Model
	Demo_MinSegMega_V3....	8/17/2020 1:09 PM	Simulink Model
	Demo_MinSegNano.slx	10/6/2019 7:41 PM	Simulink Model
	Closed_Loop_DC_Motor_...	5/30/2019 1:29 PM	Simulink Model
	Closed_Loop_DC_Motor_...	5/30/2019 1:29 PM	Simulink Model
	Motor_Step_Response_L...	5/30/2019 1:28 PM	Simulink Model
	Motor_Step_Response_L...	5/30/2019 1:27 PM	Simulink Model

Copy the contents of “Example of 3 Channel plotting”

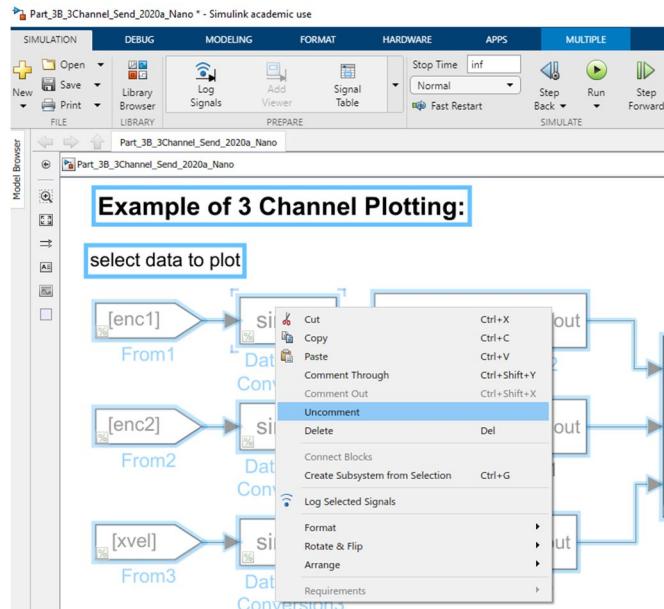
IMPORTANT: If the USB to serial converter is hardware based such as the CH340 or the PL2303 you must:
 1. Manually Specify COM Port in the Model Configuration Parameters
 2. For later versions of MATLAB Run the command: `coder.targetarduino.base.registry.setBaudRate(gcs,115200)`
 at the command line to set the baud rate for any new Simulink diagram or always use this file to start new projects.
 It always runs this command after the simulation diagram loads.
 You can change this behavior by selecting "Tools" -> "Model Explorer", "Callbacks", "PostLoadOn"

Note: Later versions of MATLAB you can use External mode with Nano
 for previous versions send and plot data using SerialPlot below.



Paste these into a current Simulink Project for the Arduino, and save the new file with a new name in a an appropriate directory.

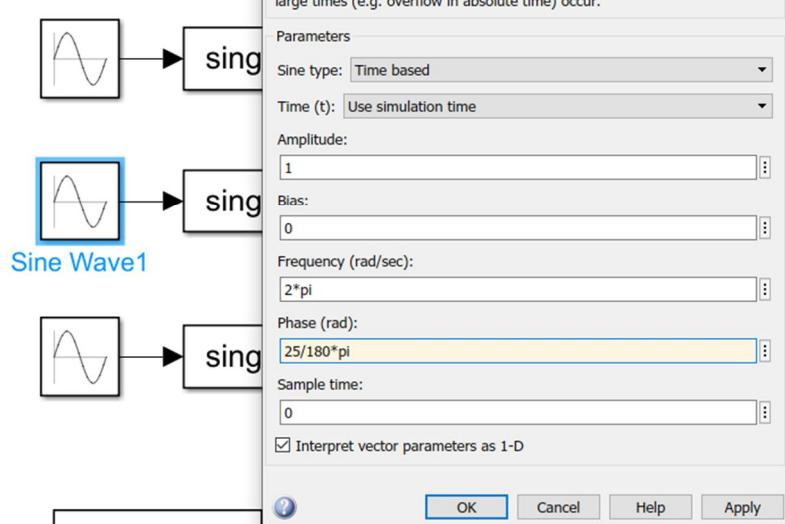
Select all the code and right-click it to uncomment it:



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

Example of 3 Channels

select data to plot



Make sure overrun detection is on:

The screenshot shows the "Configuration Parameters" dialog for a project named "Part_3B_3Channel_Send_2020a_Nano/Configuration (Active)".

Solver
Data Import/Export
Math and Data Types

Diagnostics

Hardware Implementation (selected)

Model Referencing
Simulation Target

Hardware board: Arduino Nano 3.0
Device vendor: Atmel
Device type: AVR

Device details

Hardware board settings

Target hardware resources

Groups

- Build options Enable overrun detection
- Host-board connection
- Connected I/O

Overrun detection

- Analog input channel properties
- Serial port properties
- I2C properties
- SPI properties
- External mode

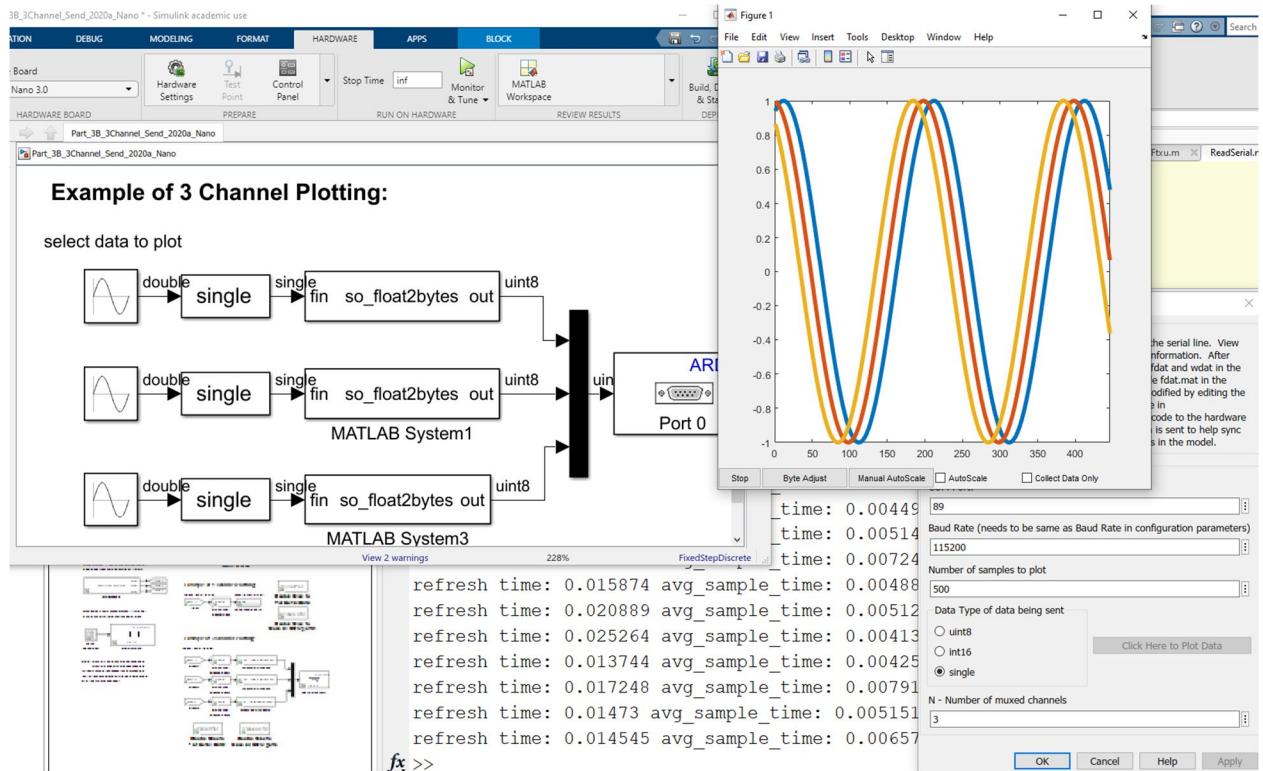
Baud rate is set to 115200

The screenshot shows the 'Hardware Implementation' tab selected in the left sidebar. The main panel displays hardware board settings for an 'Arduino Nano 3.0'. Under 'Target hardware resources', the 'Serial port properties' section is expanded, showing the 'Serial 0 baud rate' set to '115200'.

And the sample time is set to 5ms:

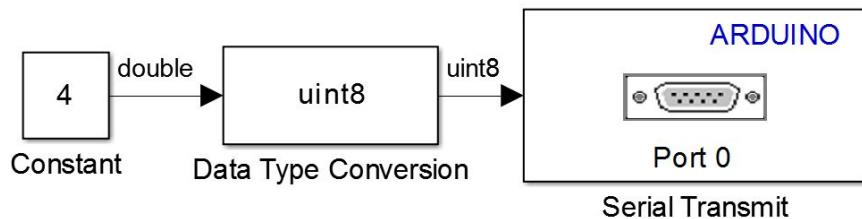
The screenshot shows the 'Configuration Parameters' dialog for a model named 'Part_3B_3Channel_Send_2020a_Nano'. The 'Solver' tab is selected in the left sidebar. In the 'Simulation time' section, the 'Start time' is set to '0.0' and the 'Stop time' is set to 'inf'. In the 'Solver selection' section, the 'Type' is set to 'Fixed-step' and the 'Solver' is set to 'discrete (no continuous states)'. Under 'Solver details', the 'Fixed-step size (fundamental sample time)' is set to '.005'. In the 'Tasking and sample time options' section, the 'Periodic sample time constraint' is set to 'Unconstrained'. The checkbox 'Treat each discrete rate as a separate task' is checked, while 'Allow tasks to execute concurrently on target' and 'Automatically handle rate transition for data transfer' are unchecked.

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

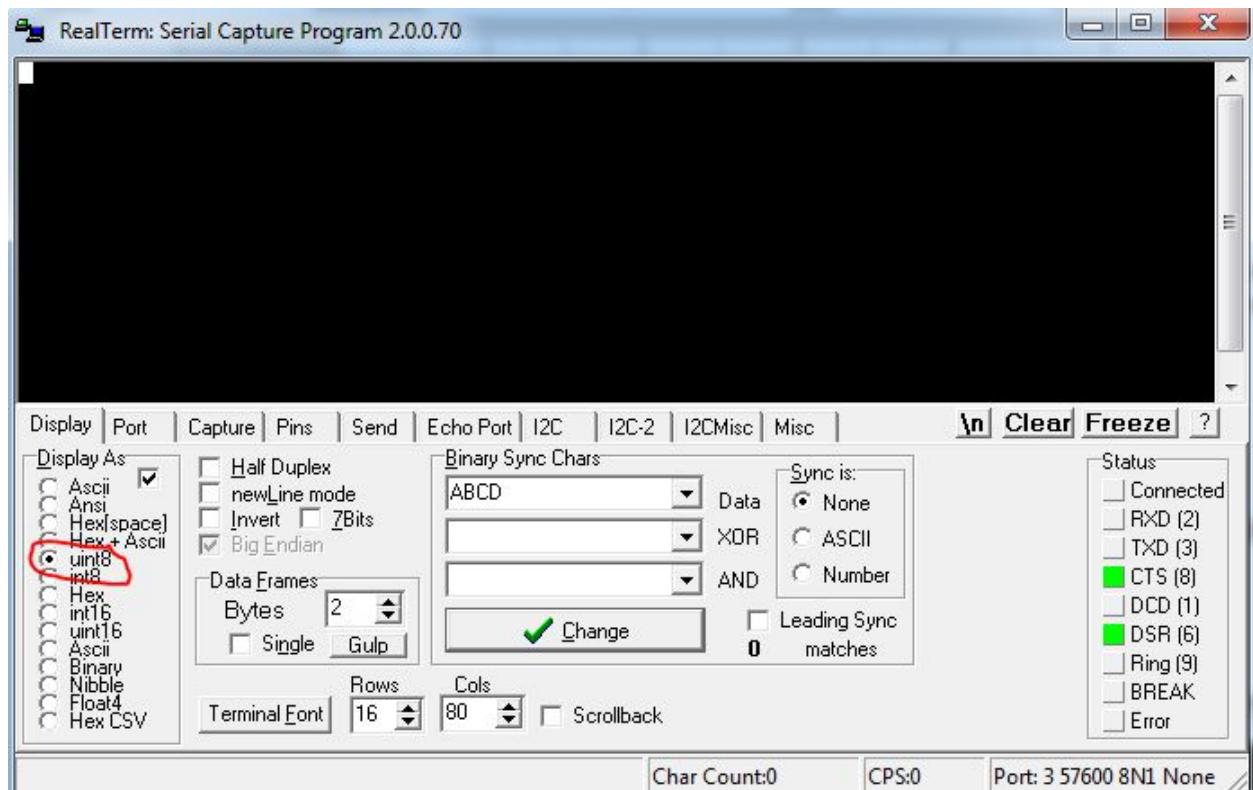


Part 3: Reading the Data with RealTerm (optional)

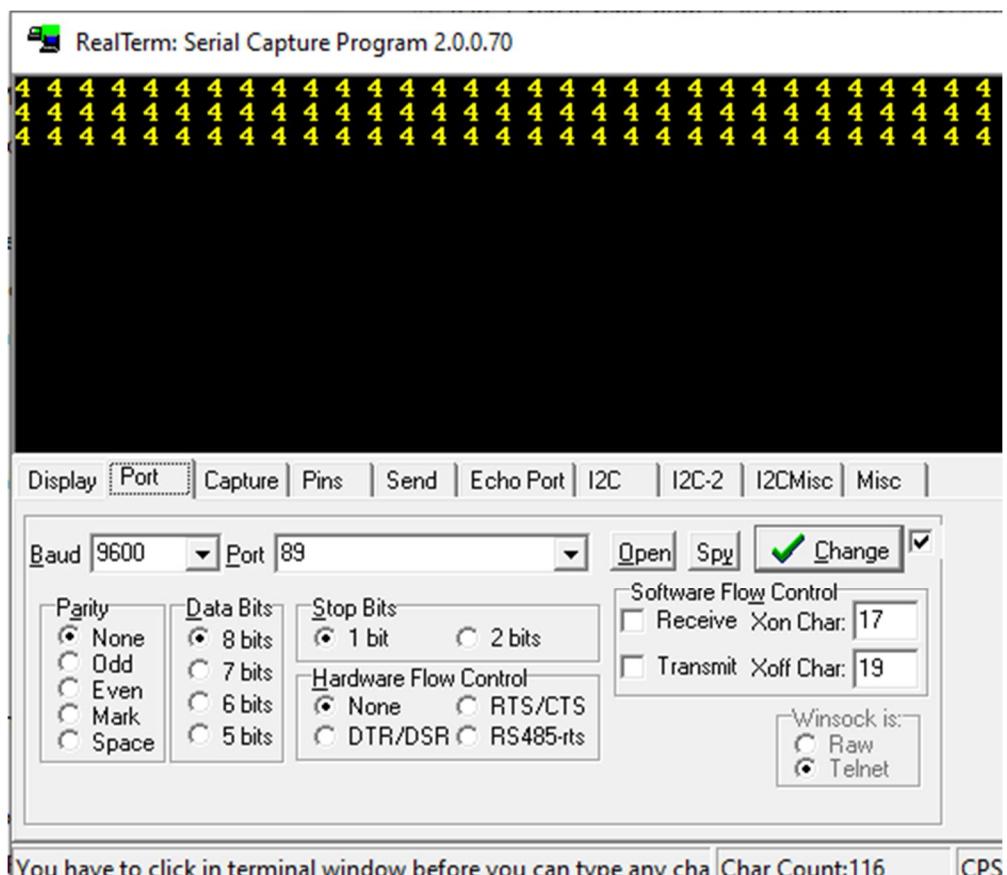
Another way to obtain data is RealTerm – although any serial terminal program will work. First download the following code to the Arduino:



After installing RealTerm open it in administrator mode.

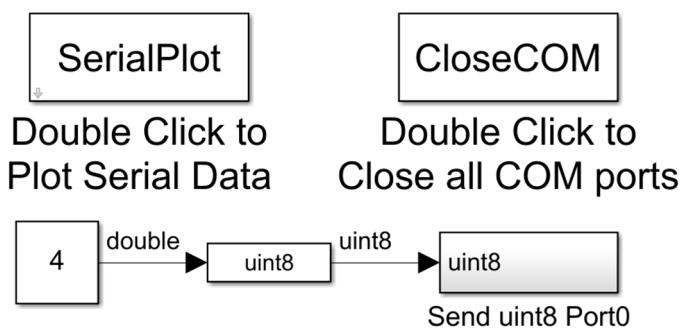


- On the left select “uint8”. This indicates to realterm that the data type to expect is uint8 and it will display the results appropriately.
- Click the “port” tab, change the baud rate to 9600
- select the serial port of your hardware
- If the Open tab is depressed (as in the figure below) click it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

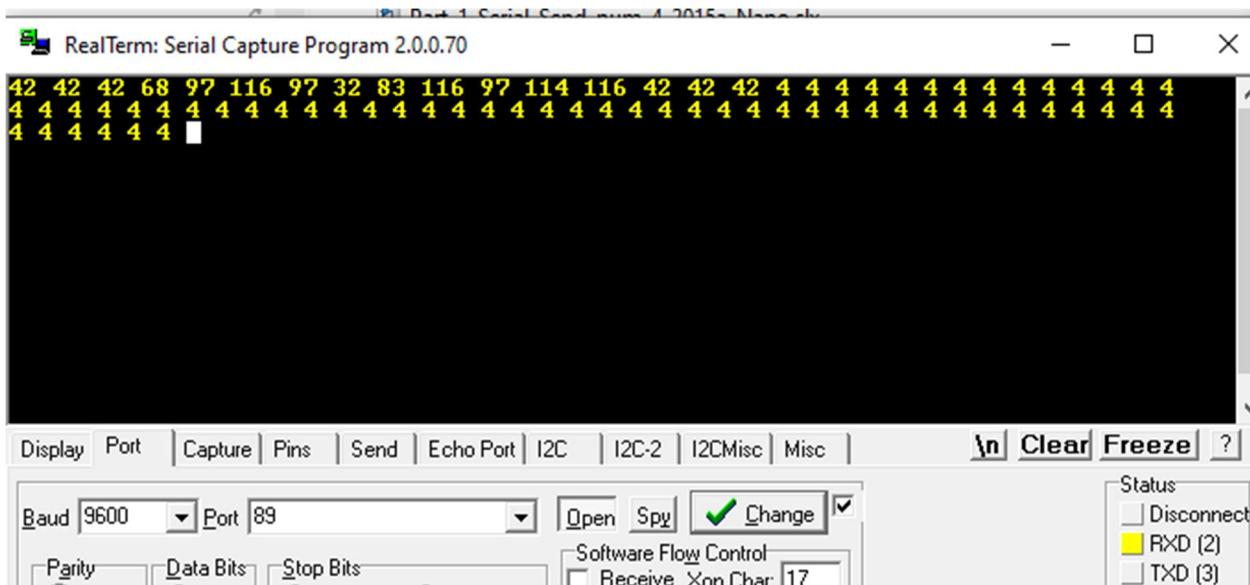


By specifying "uint8" RealTerm knows the data type so it can display it correctly.

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

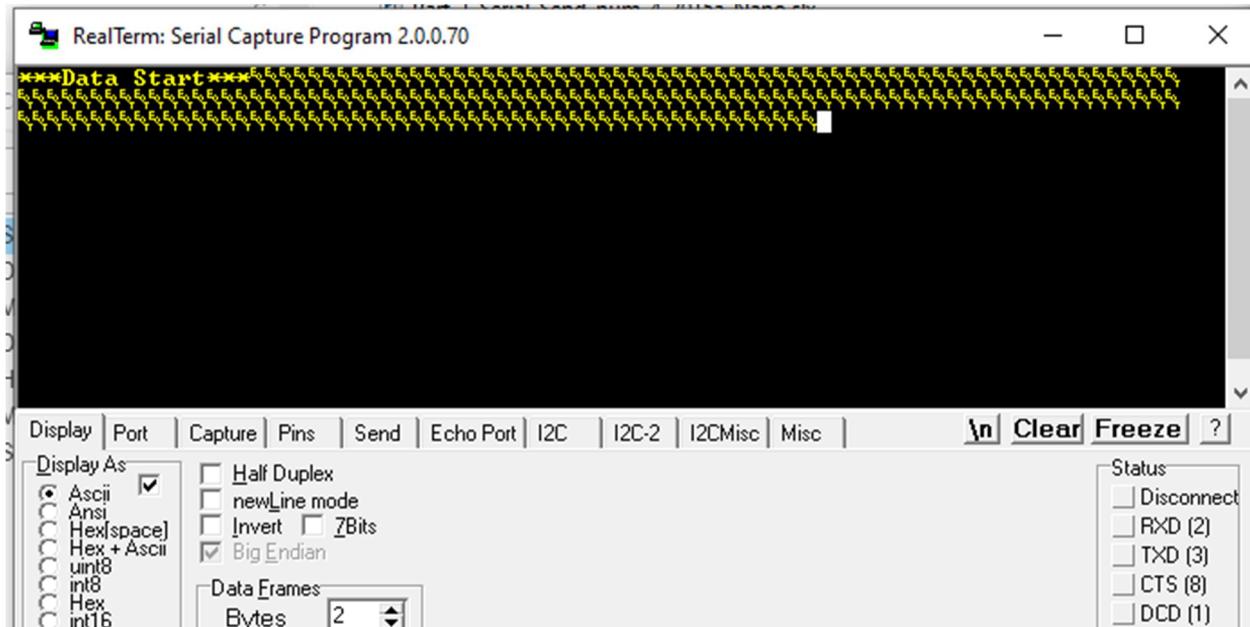


Use the same settings and open RealTerm: (soyler of .05 seconds, Baud rate of 9600):



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

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



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

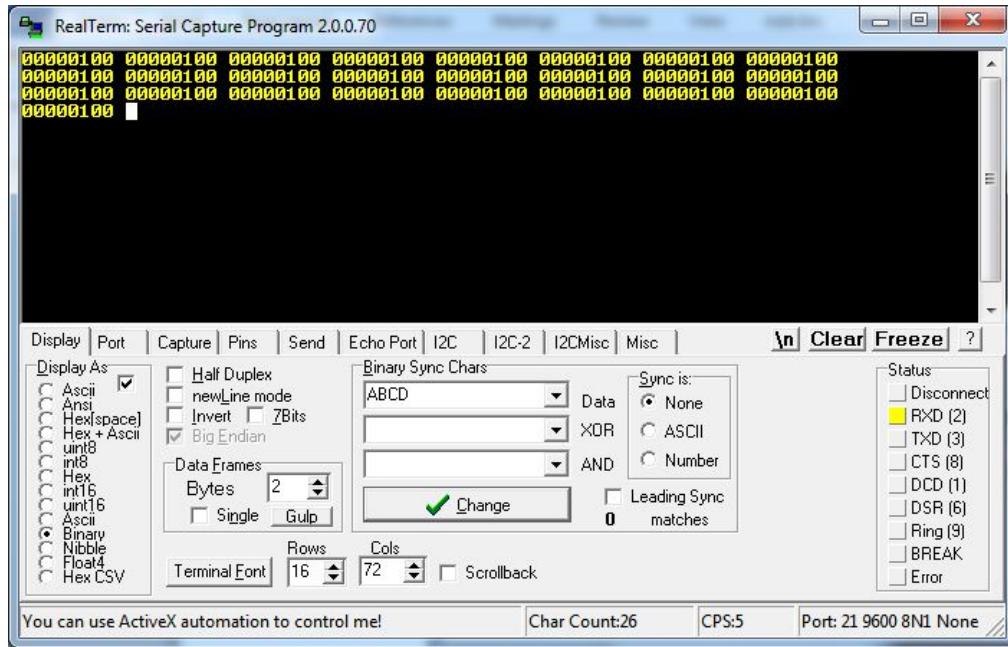
Source: www.LookupTables.com

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

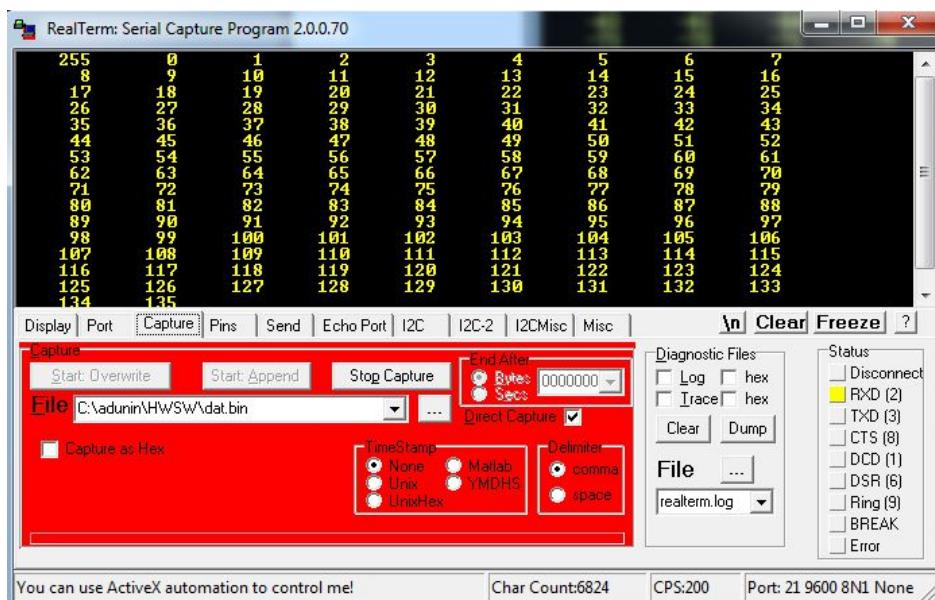
- go back to the Display tab and select Binary



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.

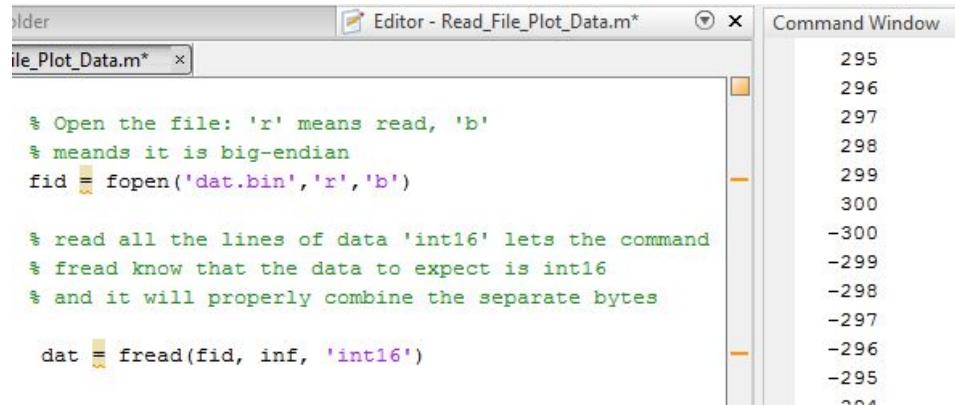
Writing the data to a file with RealTerm:

- 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



Reading the data file with Matlab:

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



The screenshot shows the MATLAB Editor window with a script named 'Read_File_Plot_Data.m'. The code reads a binary file 'dat.bin' and stores the data in a variable 'dat' as int16 values. The command window on the right shows the output of the script, which consists of a series of integers ranging from -300 to 300.

```
% 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')
```

Value
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:

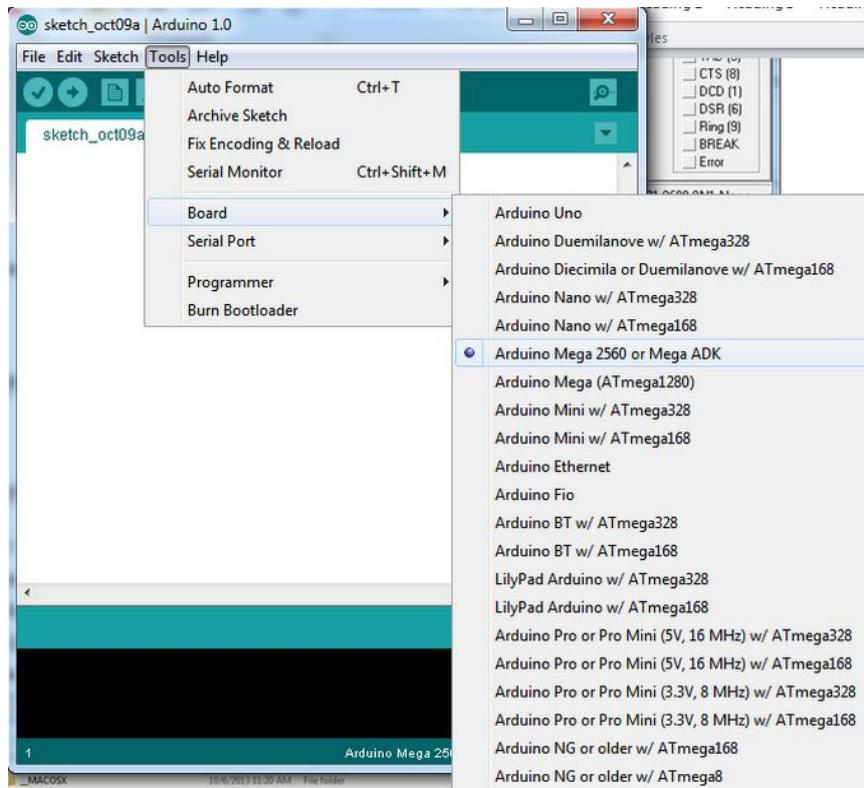
- Matlab uses the Arduino IDE and Arduino libraries. When it installs the Arduino Simulink blocks it installs the Arduino IDE typically in this location
 - C:\MATLAB\SupportPackages\R2013a(R2014a)\arduino-1.0(arduino-1.0.5)
 - C:\MATLAB\SupportPackages\R2015a\arduino-1.5.6-r2 (You can copy this path and directly paste it in "This computer" to find "arduino.exe")
 - 2015a has both a arduino-1.0 folder and arduino-1.5 folder – either will work but 1.5 is the latest version so use this.



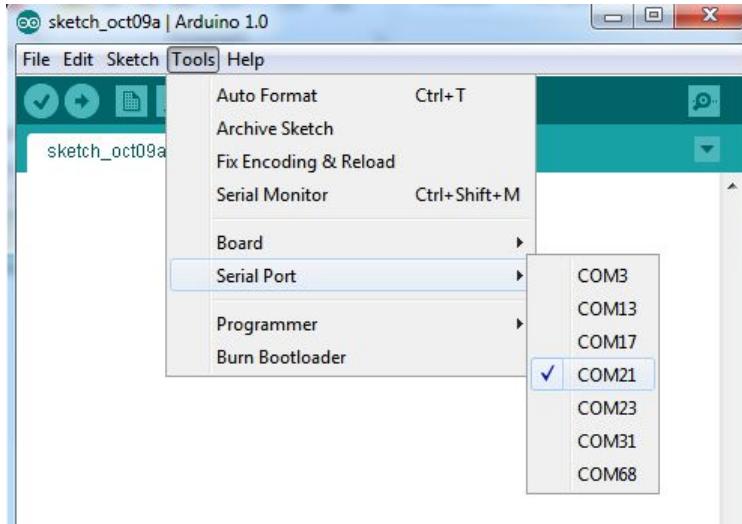
- The Arduino IDE can be opened from here:

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
nxtxSerial.dll	11/28/2011 7:31 PM	Application extens...	76 KB

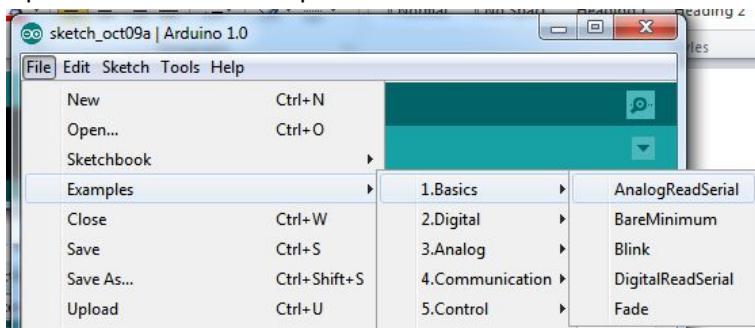
- Select your target board:



- 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

```

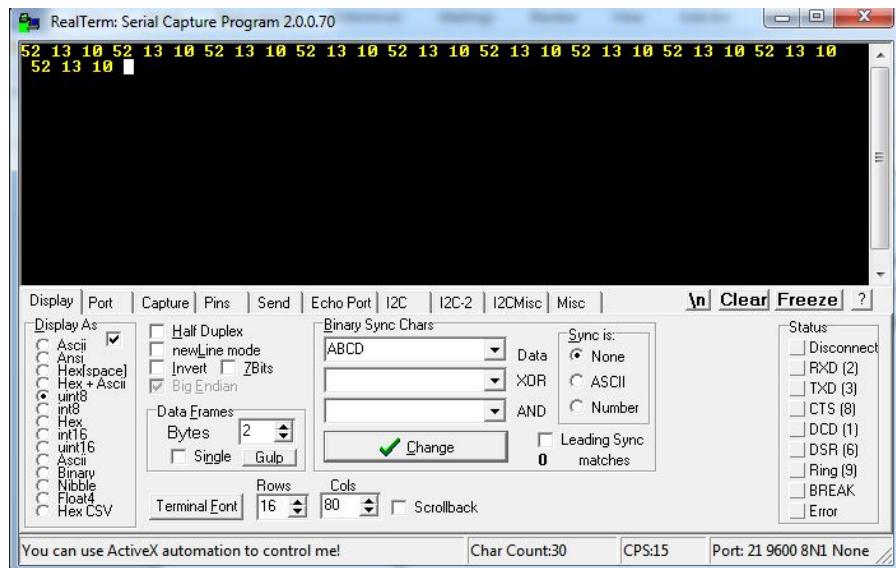
Serial_Send_Byte_Example | Arduino 1.0
File Edit Sketch Tools Help
[Upload Icon] [Run Icon] [Save Icon] [Upload] [Save]
Serial_Send_Byte_Example §

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

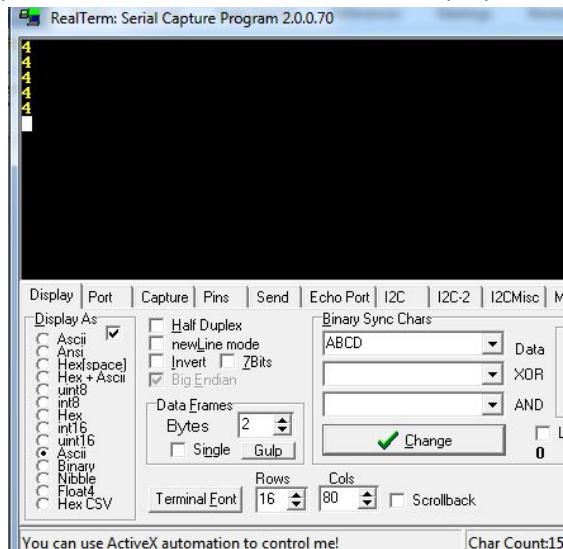
void loop() {
  int sensorValue = 4; // char is a uint8
  delay(1000); // Delay 1 seconds
  Serial.println(sensorValue);
}

```

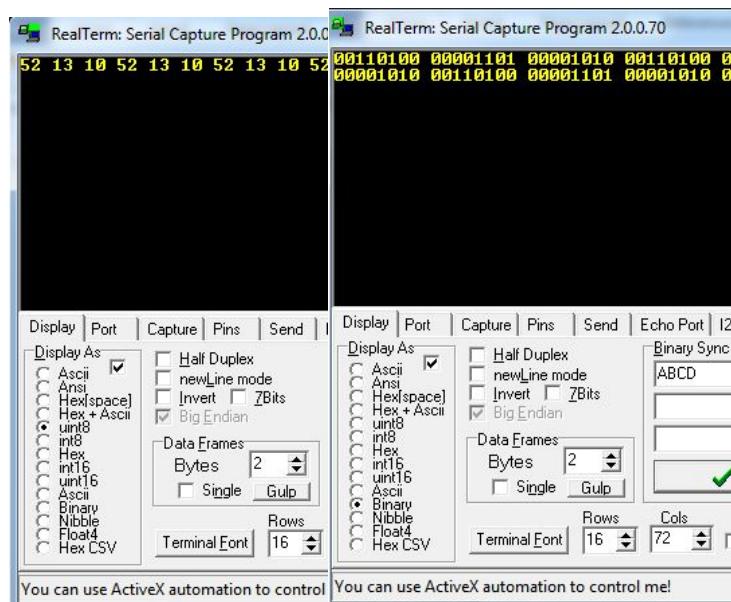
- Open the terminal program to view



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):



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 value 16 (the result of the Analog to Digital conversion). You want to send this data 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) representation 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 sensor reads 32768.

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