Lab 1

Introduction to MATLAB SLRT Toolbox and the Advantec PC 104 computer boards

Introduction

The PC104 standard is for a system of computers that are small in size, and stackable so that any peripheral module with the same standard of size and number of pins can be interfaced to it. The systems we will be using in the lab are made by Advantech. We have two different modules, the older PCM 3350 and the new PCM 3355L boards. They require low power and can be used for general purpose embedded applications. You may read more about their specifications on their respective manuals uploaded on blackboard.

In the lab each group will have access to two PC104 computers. They are installed with Diamond MM A/D D/A boards and screw terminals that provide connections to them. In the first lab you will be familiarizing yourselves with the various functionalities/capabilities of these embedded computer systems. You will also learn how to use the MATLAB Simulink Real-Time Target toolbox for embedded systems programming. For all experiments performed in this lab, we will design models in Simulink and then utilize a C compiler to create executable code from the models. The code will then be ported to the flash memory on the PC104 board and run from there.

By the end of this lab, you should be well informed and completely comfortable in utilizing the various resources these PC104 systems have to offer. All the skills you learn in this lab experiment will be required in future labs for successful completion of the semester.

NOTE: The computer systems in GWC laboratory are such that they lose all information upon restart and also at 12:00 am midnight. As such, it is strongly recommended to carry flash drives and save all your work in them at all times. Also, MATLAB needs to be configured in a particular manner for use in this lab. Since the computers reset, these configurations will have to be done every time you come in.

Setting up Matlab for first use

These procedures must be followed every time you come to the lab.

Step 1. Compiler configuration

NOTE: Throughout these lab manuals, type command or run command means you should enter the command into the MATLAB main command window. MATLAB always has ">>" symbol to indicate where your commands are entered. We use the same symbol in this manual to indicate it is a MATLAB command. To see what licenses and toolboxes are installed in MATLAB you can type "ver" and hit enter. At the eighth option from the bottom you should see the Simulink Real-Time toolbox listed as available. Without them you cannot do any work related to this lab.

A compiler is required to convert our Simulink models into C code and eventually to executables.

Run command:

>> mbuild –setup

The output on the screen will let you know if a compiler has been configured already. If not you may have to install/choose one. For the computers in the GWC 379 lab, the compilers are preconfigured by the TAs.

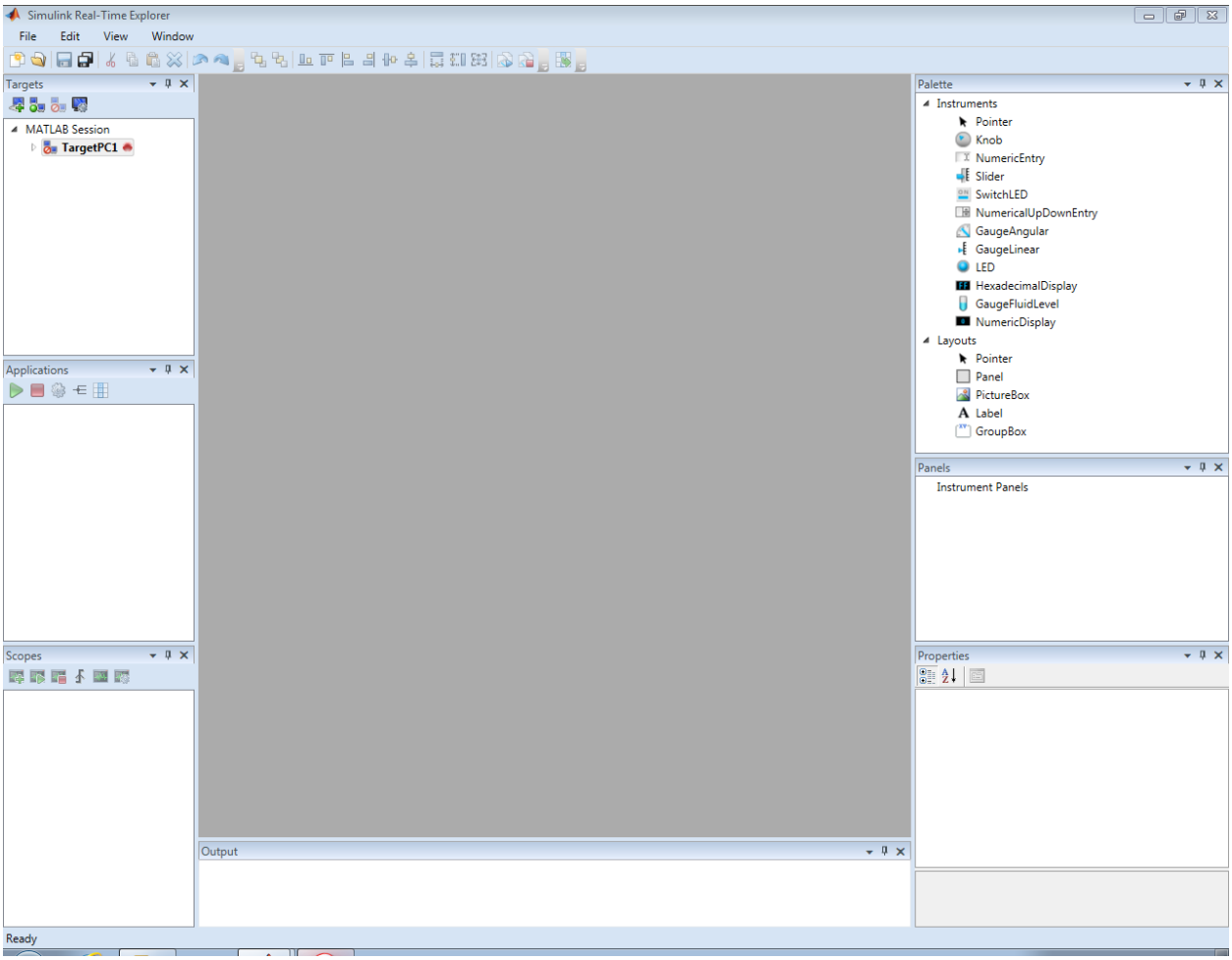
Next run

Make sure you click “Apply” every time, after making any changes. If you move to a different window without clicking “Apply”, the changes are not saved.

Step 2. Target PC configuration

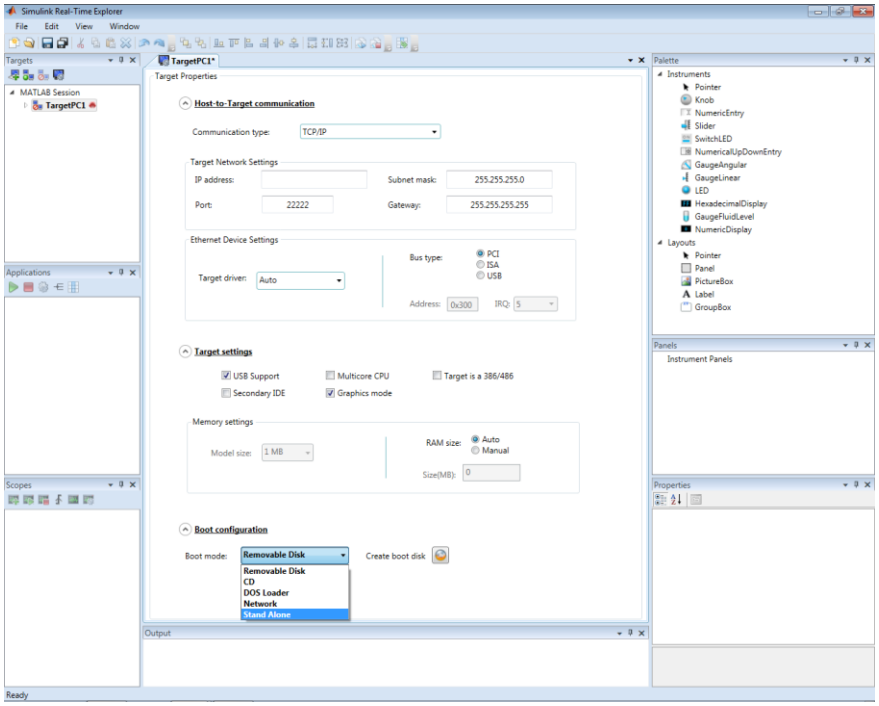
Simulink Real-Time(SLRT) Explorer identifies each PC104 system as a target PC. To run SLRT Explorer enter the command as shown

>>slrtexplr



By default the explorer will start with one target named TargetPC1. Each target PC can be programmed to run in either "Standalone" mode or in "DOS Loader" mode for the purpose of this lab (there are other unused options). We will look at each mode, their uses and how to configure them. Right click on TargetPC1 and select Properties and then expand all properties.

Step 2.1. Standalone Mode



In standalone mode, any program you build into the PC104 board will run as long as the system has power. After a power reset, the same program will start running as soon as the system boots up. No user intervention is required to start the program at all. To enable standalone mode select "Stand Alone" in the Boot configuration label and save the targetPC by going in to File>Save TargetPC1 (or 'save' icon, ctrl+s does not work) as shown in the image. You will learn later to download and run a program on the PC104 board using Standalone mode.

Step 2.2 Dos Loader Mode

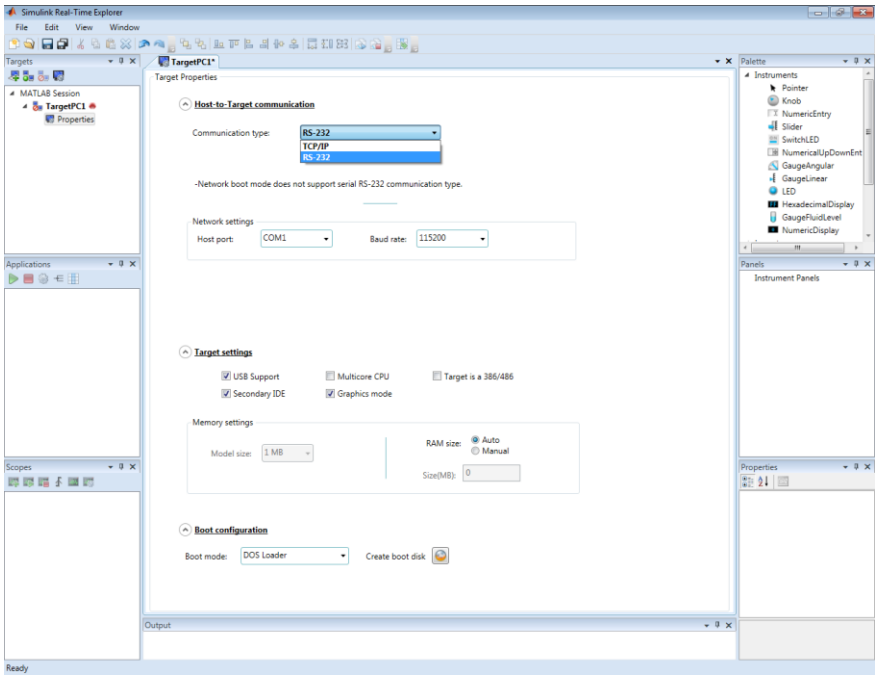
In this mode of target programming, the target computer has to be connected to a host PC at all times. If there is a power reset, then the host PC has to redownload the program to the target or else the target will simply boot and wait.

There are two types of Loader modes. One is Ethernet (TCP/IP) and the other is Serial (RS-232). These two modes determine the connectivity between the host PC and the target computer. To use Dos loader mode, select "DOS Loader" under "Boot Configuration". Once the communication setting is selected as

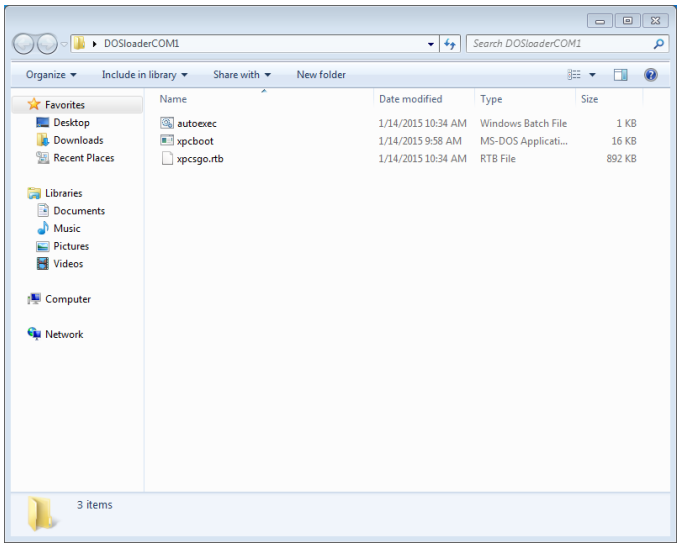
either RS-232 or TCP/IP a dos loader file must be created and copied on to the flash disk. We will now see how to do this, first for serial and then for Ethernet.

Step 2.2.a Serial communication using dos loader

In SLRT Explorer, go to TargetPC1 Properties page and under "Host-to-Target Communication" choose RS-232 (as shown in the figure). Choose the com port and baud rate you would like to set up on the target PC104. The Advantech PC104 has two serial ports (COM1 and COM2). Baudrate determines the speed of data transfer, so choose the highest possible. Save TargetPC1 configurations.

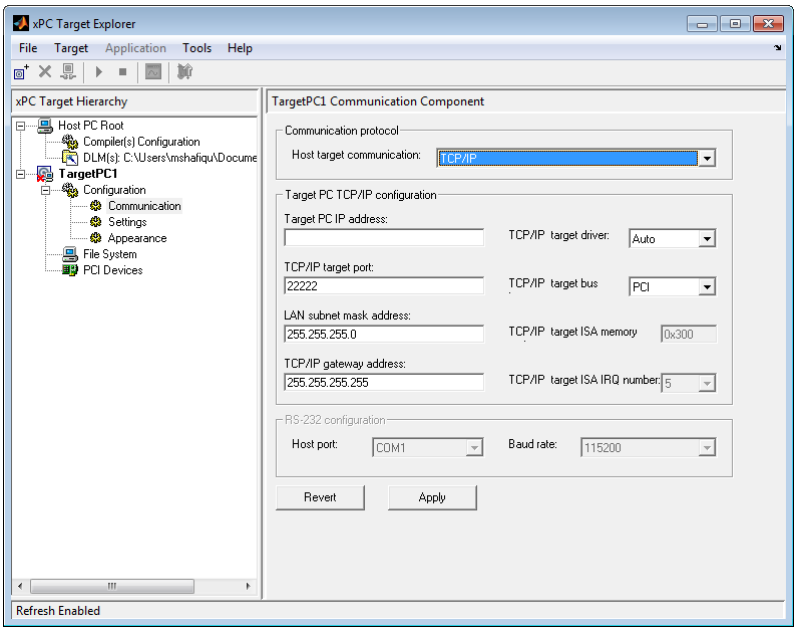


Then go to "Boot Configuration" ensure that "boot mode" is selected to be "DOS Loader". Click on "Create boot disk". In the "Location" box specify a particular folder where you want to create the DOS Loader files (Preferred method is to create a new folder and name it so you know what it is later e.g. RS232Com1DosLoader). Click "ok". This will create three files in the folder that you chose (as shown in the figure). These files must now be copied onto the compact flash card(CF) and then you must insert the card into the PC104 CF slot and boot the PC104 up. On the monitor connected to the PC104 board you should see a message that says "Host-Target interface is RS232". You will later learn how to download a program to the PC104 using RS-232 protocol.



Step 2.2.b TCP/IP communication using dos loader

In SLRT Explorer, go to TargetPC1 Properties page and under “Host-to-Target Communication” choose “TCP/IP” (as shown in the figure below).



TCP/IP uses IPv4 addressing schemes to communicate with different Ethernet devices. We will not go into a discussion about IPv4 addressing but will provide enough information to setup communications between the PC104 boards and the host computer. These are the following options to configure:

Target PC IP address: the IP address labeled next to the PC104 board you are using (e.g 129.219.76.35)

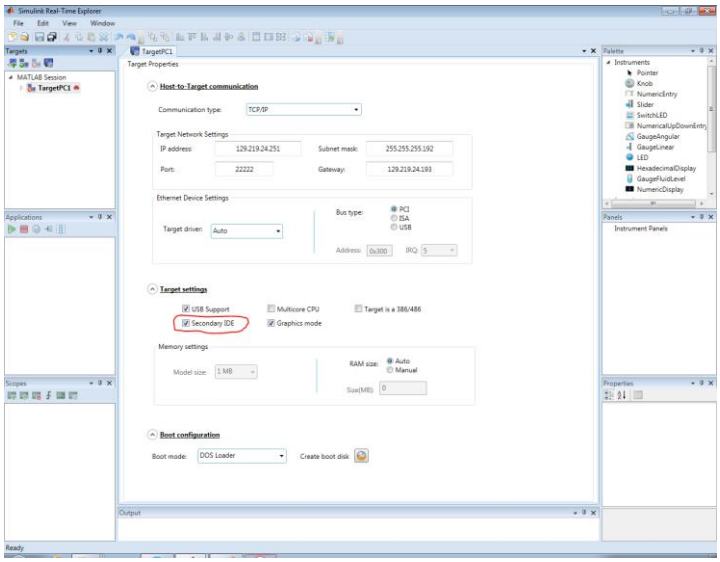
TCP/IP target port : 22222

LAN subnet mask address : 255.255.255.128

TCP/IP gateway address : 129.219.76.1

To create a DOS Loader with Ethernet go to “Boot Configuration”. Choose “DOS Loader” as boot mode and click on “Create boot disk”. This time create folder with a unique name and the last three digits of the IP address e.g. DOSLoaderTCP251. Just as it was in the RS232 DOSLoader, this will create three files in the folder. These files must now be copied onto the compact flash(CF) and the CF card inserted into the PC104 CF slot and the PC104 booted up. On the monitor connected to the PC104 board you should see a message that says “Host-Target interface is TCP/IP Ethernet” and it will specify the IPv4 address, subnet mask and gateway you specified. You will later learn how to download a program to the PC104 using TCP/IP protocol.

Step 2.3 Enabling secondary IDE

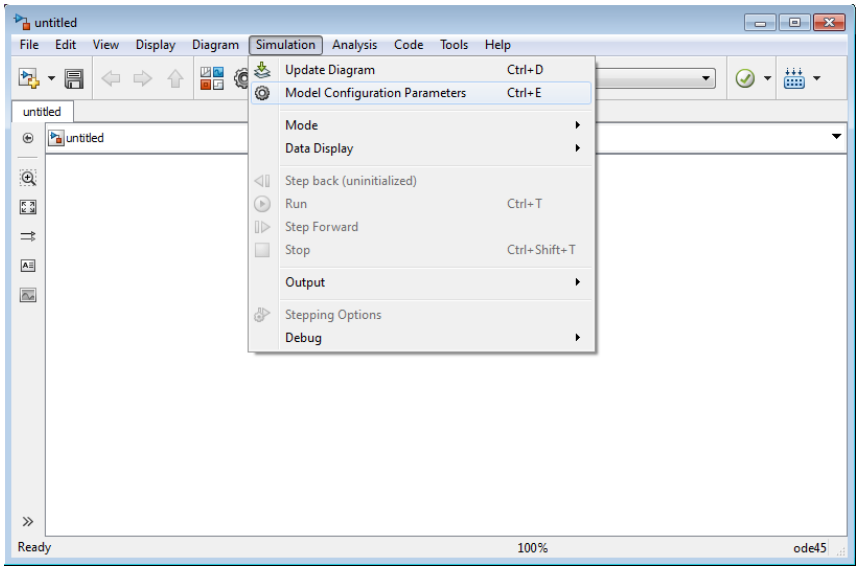


Whether you are using RS232 or TCP/IP DOSLoader mode, you will need to enable "Secondary IDE". This will allow you to store data on the CF card. This data can be anything starting from error messages, A/D readings, serial port inputs etc. In order to enable Secondary IDE simply check the box next to it as shown in the figure.

Step 3 Model file Configuration

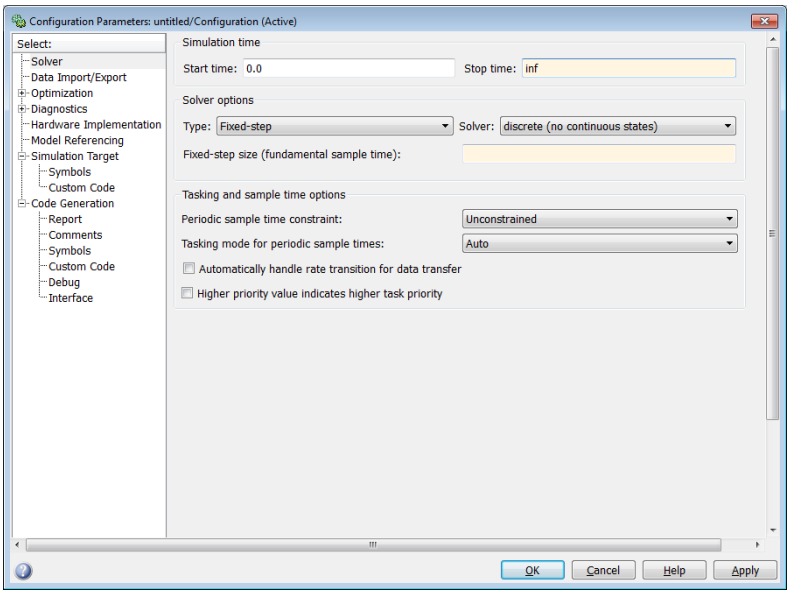
This needs to be done for every real time model you build. However unlike the setup processes in step 1 and step 2, these configurations are local to individual files. Since, the model files will be saved (not on the lab computers); these configurations will not require redoing every time unless you specifically want to change them.

Go to MATLAB main window, then click on File> New> Model



A new Simulink model file will appear. Then go to Simulink> Configuration Parameters as shown above. In the window that appears select “Solver” and under “Solver Options” change “Type” to “Fixed-step” based on the lowest sample time you want to use on your model (see figure below).

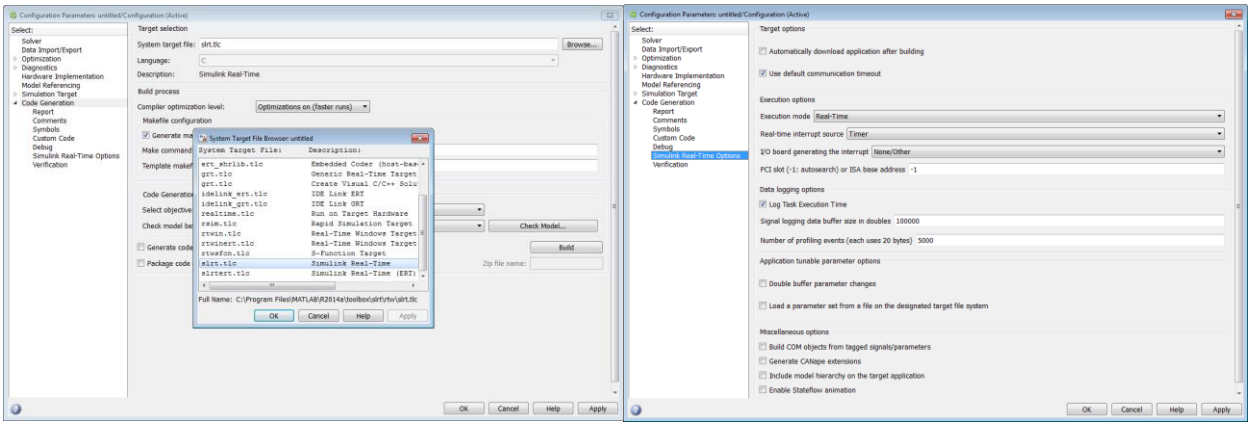
NOTE: It is best practice to ensure sample times are exactly the same all over your model file unless you specifically need different sample rates in different blocks of your Simulink model. If you must use two or more different sample times, they must be integer multiples of what you specify as “Fixed step size”.



Choose “Stop time” based on the amount of time you want to run the embedded program. For this lab, it will be a good practice to leave it at inf (infinite).

Choose Solver options> Type to be Fixed-Step. Variable Step, the other option is to simulate continuous systems and should be chosen only when the simulation is run in MATLAB. For all embedded programs, fixed step should be the step size. The “Solver” needs to be set to “discrete (no continuous states)” unless a continuous time system is modeled (there may be computational issues with this).

Next, go to Code generation. Under Target Selection> System Target file, click browse to select the file slrt.tlc, as shown in the image below. This will allow the MATLAB compiler to generate executable code that is compatible for SLRT compatible embedded systems ; such as Advantech PC104 computer boards. You also need to go to SLRT target options and deselect ‘Automatically download application after building’; doing this will stop MATLAB from automatically trying to download the application file to the

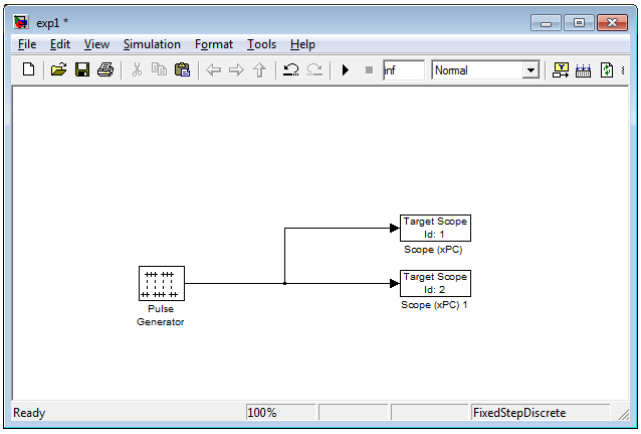


target computer (this prevents errors when multiple targets are connected to the same host). Be sure to click “Apply” at every step and save all changes. You are now ready to build our first application model and use Standalone or DOS Loader mode to run them on the PC104 targets.

Experiment 1.1

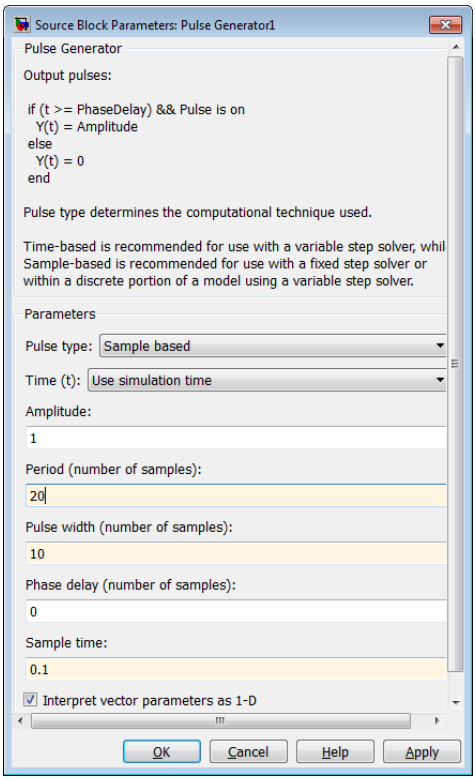
Building familiarity with SLRT toolbox and executing real-time code in PC104.

For your first task you will create a real-time program that produces a square wave with amplitude 1, a period of 2 seconds and sampling time 0.1 seconds. The output should be visible on the monitor connected to the PC 104 board. For the scope, use a 100 point window (load this model file into PC 104 board using Standalone and DOS Loader (RS232 and TCP/IP) modes). Show your TA what you have done after each step.

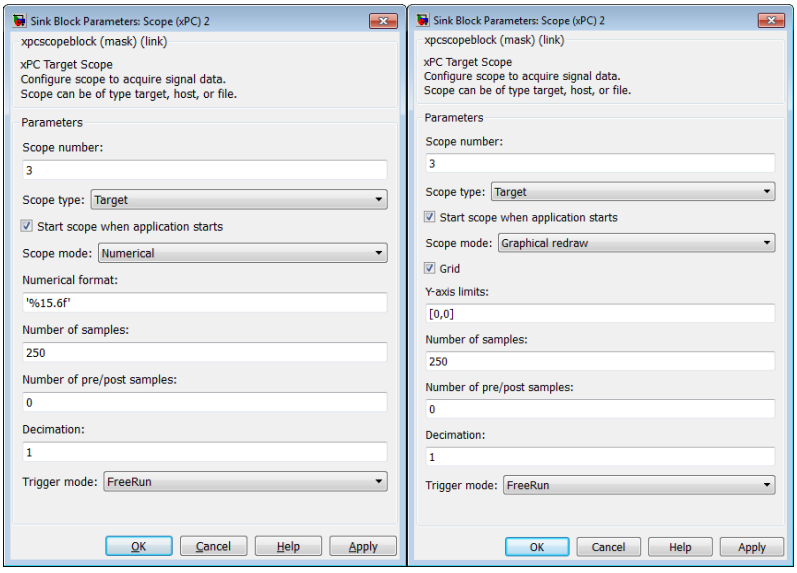


The figure below shows the model you need to create. The steps to build it and where to find each block is given as follows.

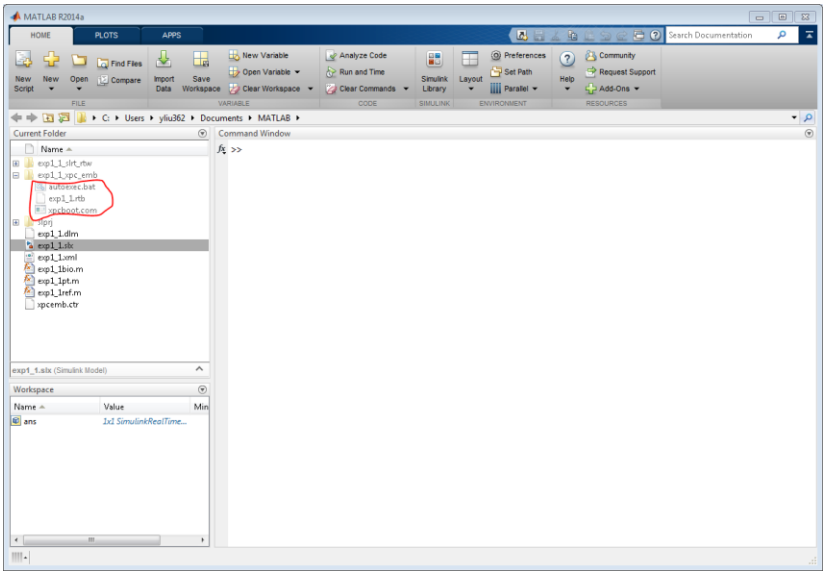
You will find the Pulse generator under : View> LibraryBrowser> Simulink> Sources> Pulse generator. Drag and drop it in the model file. Double click on the block to configure it. See image below the model file. Select pulse type as "Sample based". Set "Sample time" as 0.1, "Amplitude" as 1, period as 20 samples(which corresponds to a period of 2 seconds) and pulse width as 10 samples (as required in the problem).



For SLRT Scopes (these enable you to view graphical/numerical scopes on the monitor connected to the PC104 boards), go to View> Library Browser> Simulink Real-Time>Displays and Logging> Scope. Configure one scope as numerical and the other one as a graphical scope as shown below.



Once the model file is ready, save it and press Ctrl+b to build it. Then copy the three files shown in the below figure into the flash disc.

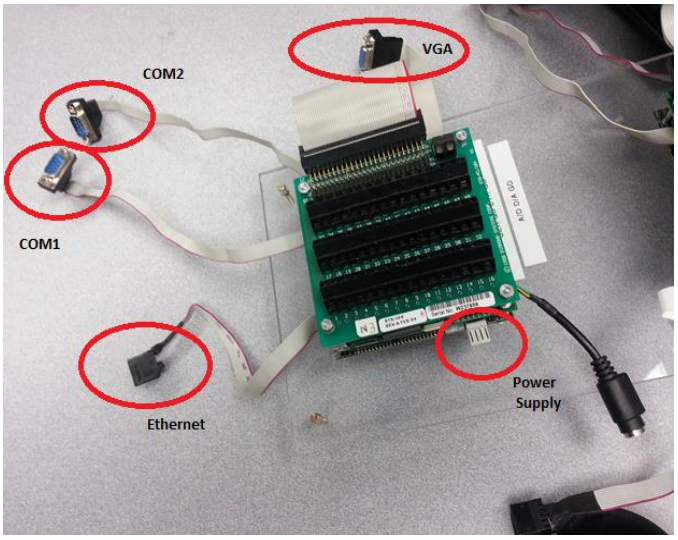


NOTE: The "Current Folder" tab in the MATLAB file explorer shows the active working directory of MATLAB. You must save all your model files in this folder or else MATLAB will not know where the file or its dependencies are. In the example below ExpMdl is the active folder. So the model was saved under it and when a build was performed, all necessary files for creating executables from C code were created there. The executables are always saved under the modelname_slrt_emb folder. You must copy all these files and put them into the CF card for your program to run in Standalone Mode.

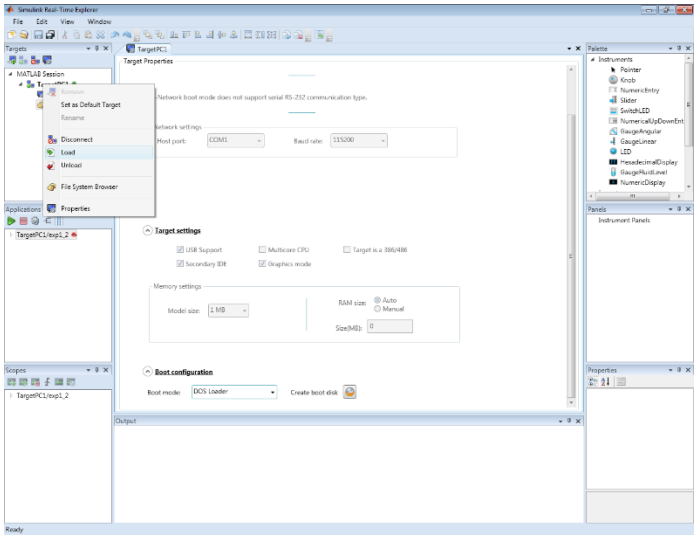
Experiment 1.2

Implement the same model used in exp 1.1 with a sample time of 0.01s, but this time use serial communication to download the file onto the PC104 board.

To do this you need to connect a serial cable (DB-9) from the back of the host computer running MATLAB to one of the two RS-232 ports on the PC104. The image below shows all the connectors you will find on the PC104 board. See step 2.2.a to create DOS Loader files.



Once the DOS Loader files have been created and loaded on to the CF disc, insert it into the PC104 board's CF flash slot and boot it up. When the board is ready, go to SLRT Explorer. Connect to the TargetPC by right clicking and selecting "connect". After the Icon turns green right click on it again and choose "load". In the "Location" box browse and select the .dlm file that is created after building your model. You should see a .dlm file with the name of the model you created (this will only happen if the current working folder in MATLAB is set to the one where you saved your model file).



Wait for a while and you should see the experiment file listed under the Applications section. This means the file has been successfully downloaded to the PC104 board. Once the file is loaded the display on the monitor attached to the target PC should change and you will be able to see your scopes in an uninitialized condition. Right click on the application you downloaded and select "start". You should see a square waveform on the monitor connected to the PC104.

In MATLAB command window type the following commands

>> tg = slrt('TargetPC1')

>> tg.viewTargetScreen

Save the screen shot of the output from PC104 for your report.

Repeat experiment 1.2 with DOS Loader but this time with TCP/IP communication as shown in step 2.2.b. Remember, after setting up the DOS Loader for Ethernet communication you need to build and reload the new DOS Loader files on to the flash disk. Then use the same process to download the file as you did when using RS-232.

NOTE: The first time you try to connect to a PC104 using TCP/IP, you may not be able to communicate with the board. To fix this, on your desktop, go to Start> Run then type "cmd" and press enter. In the DOS command prompt type "ping 129.219.76.xx" (xx being the number of the board you are trying to connect to). If you get a response, go back to SLRT Explorer and try to connect again.

Repeat the experiment by decreasing sampling time by a factor of 10. For every sampling time, record the average TET against sample time in a table for your report.

Record the lowest sampling time before which the PC104 stops working.

Answer the following questions in your report.

Q1. Describe briefly the capabilities of the PC104 board and the Diamond MM A/D, D/A board with mention about its important technical specifications like Processor, A/D range resolutions, serial communication etc. and its application in not more than 150 words.

Q2. Give 5 examples of the practical applications where you can put PC104 boards to use?

Q3. What complier was used to create executables to run on PC104?

Q4. Report the all the sample times you used on the board and average TET of your program before it stopped working. Tabulate your data.

Q5. What does the command tg.viewTargetScreen do?

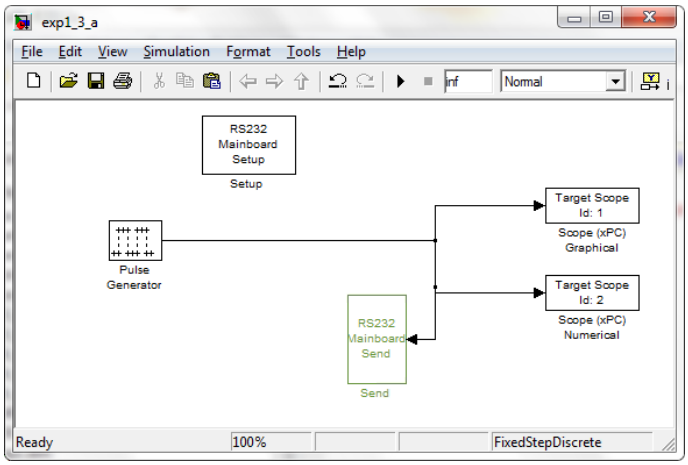
Experiment 1.3

Generate the same pulse wave as you did in exp 1.1 and transmit it from one board then receive the signal on a second PC104 board through serial. This transmission will be done over the serial ports of the PC104. Use TCP/IP DOS Loader for all your work. Sample for this experiment will be 0.1s.

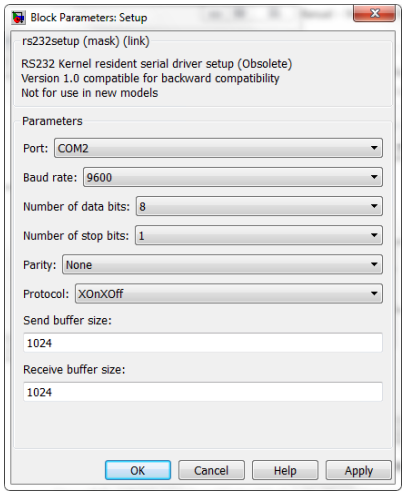
For this experiment you need to create a new target PC to utilize the second PC 104 board on your system. To do this simply click on the "Add Target" Icon under "Targets" in SLRT explorer.

You should now see a new target PC named TargetPC2. For the new target PC configure TCP/IP settings as given on the second board. Create the DOS Loader, copy the files on to the second compact flash and boot the new PC104 up. Right click on the new Target PC and click connect to ensure connectivity to the new board.

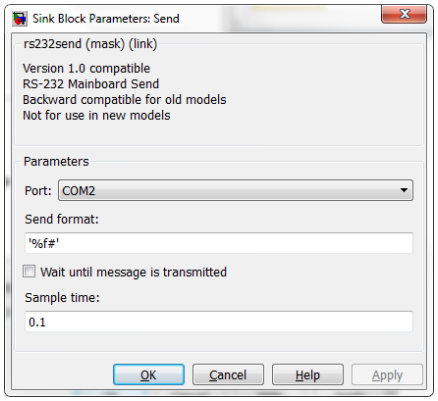
For your experiment, create two new model files as shown below.



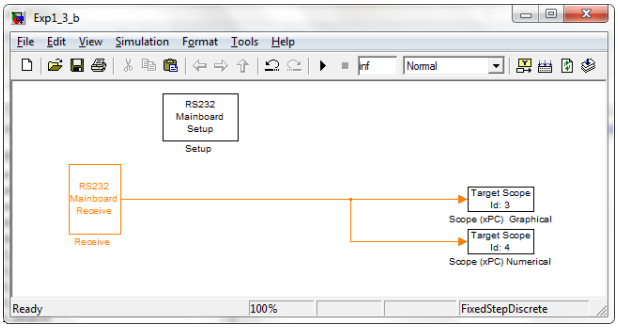
The above model, when implemented on the PC104 board transmits a square pulse using the serial link. You will find RS-232 related blocks in the RS232_blocks model file that is available for download in Blackboard. Configure the RS23s Mainboard Setup block as shown below.



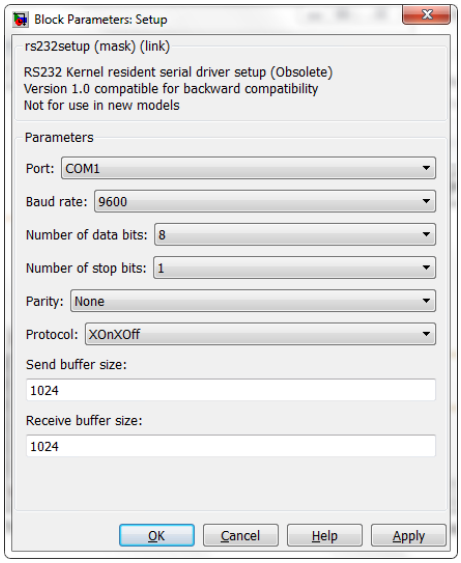
Configure the RS232 Send block as follows.

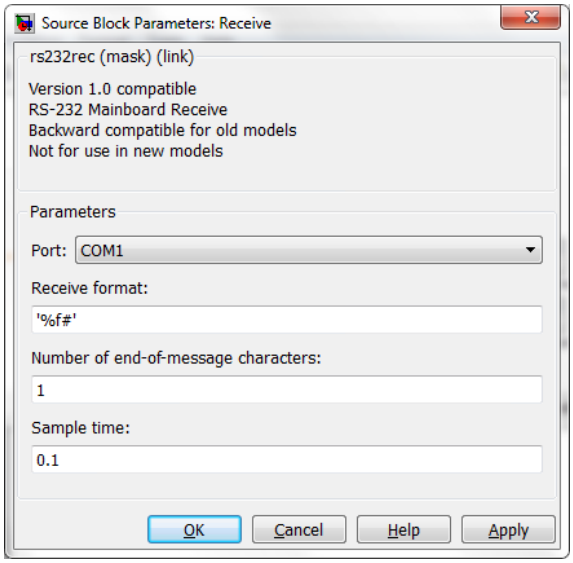


NOTE: use the same COM ports in both RS232 Mainboard Setup and RS232 Mainboard Send block. If they are different, unexpected results may occur.



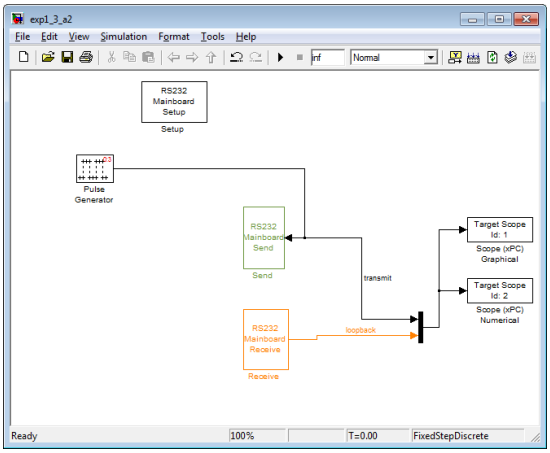
The above model must be implemented on the second board to receive and display the square wave using the serial link. RS232 Mainboard setup and receive block should be configured as shown below.

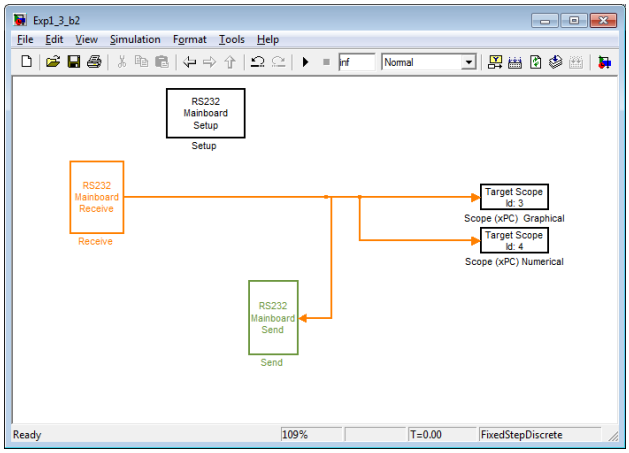




After creating both the model files build them by pressing Ctrl+b while having windows of corresponding models selected. Once the files have successfully been built, download them to their corresponding target PC using TCP/IP. Make sure you have attached a serial cable between the correct COM ports of the boards. Note down the fastest sampling rate with the baud rate of 9600 that the board can support.

Modify your files such that the square wave is looped back from the second board to the first and compare the difference between the two waves you see on the monitor on the first board. This will help you determine the delay in serial communication. The images below show you the two model files you must build to complete this step of the experiment.





You can find the MUX block under library browser Simulink->Signal routing

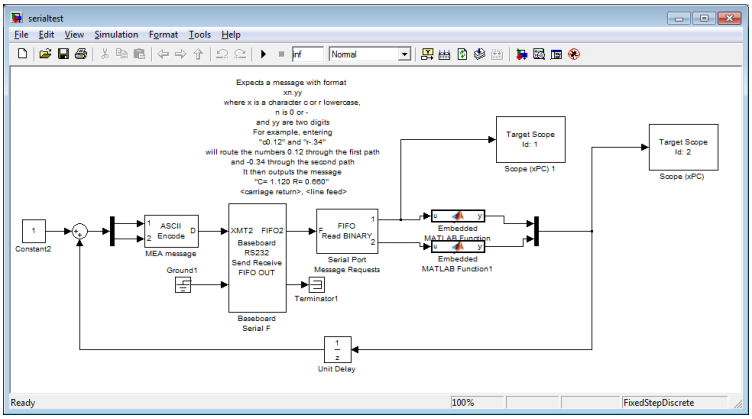
Answer these questions in your report.

Q6. What is the lowest sampling time with 9600 baud in experiment 1.3 that the boards can tolerate (do not use the feedback models)? Why is there a limit?

Q7. What is the delay from data goes from one serial port to another with the sampling time of 0.1

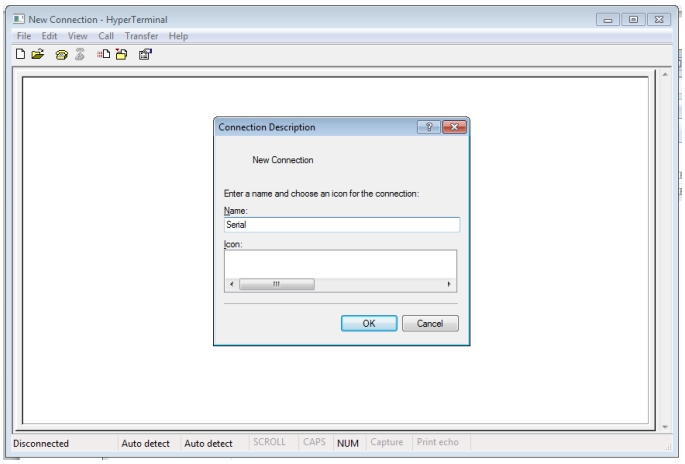
Q8. Change the sampling time to 0.01s and measure the delay again? What is the relationship between delay and sampling time?

The final part of Exp 1.3 is where you send a command to the PC104 board via RS-232 and the board performs compuations based on it. In order to do this download serialtest.mdl from blackboard. Build and download to your target SLRT.



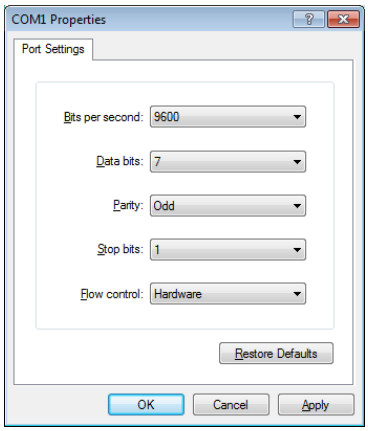
Connect the serial cable to the COM port configured in the baseboard Serial F block to the COM port of the host computer. Start the application on the PC104 board.

Download hyperterminal from blackboard and extract the files. Run the application.



Use anything as Name of the connection and click ok.

Make sure you choose COM1 and click ok. The host PCs in the lab only have one COM port.



Set COM1 properties as above. This configuration has to match with the serial configuration in the simulation. If everything was setup correctly you should see the hyperterminal window printing out C=1.00 R=1.00 in every line.

Type C0.34 and r-.25. You should see the value of R and C change, both on the target monitor and hyperterminal on the host PC.

Q9) In the results in section in your report, describe properly how this experiment worked. Start from typing the command in your hyperterminal to how it gets decoded and encoded in the PC104 and goes back to the hyperterminal. Be as descriptive as possible.

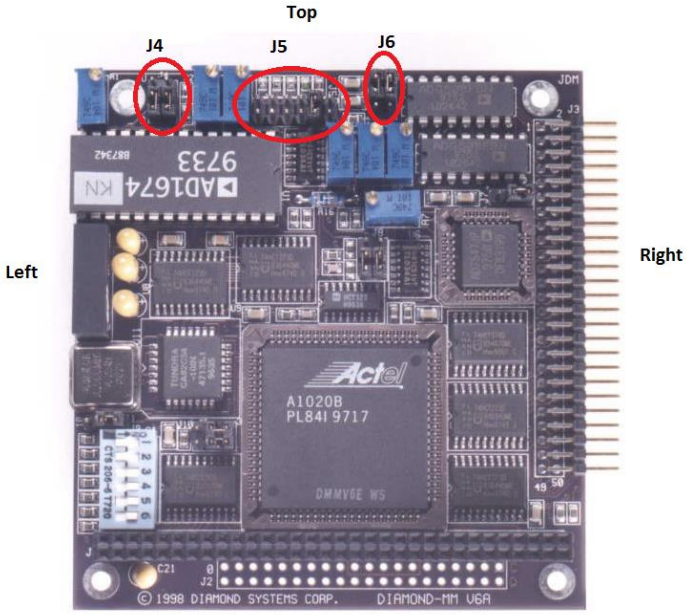
This concludes the first part of Lab 1. Submit a lab report and number it as Lab 1 part A before moving on to Part B. Use the sample lab report template. It is in IEEE double column format. Make your report as descriptive as possible to show how much you were able to grasp the material that has been taught so far.

This is Part B of Lab 1. You will need to submit a separate report for this section.

Experiment 1.4

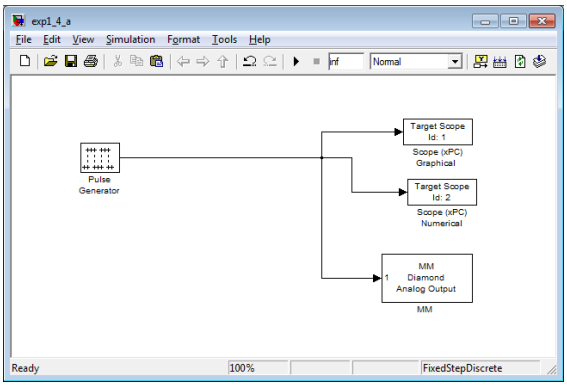
Getting familiar with Diamond MM A/D and D/A boards.

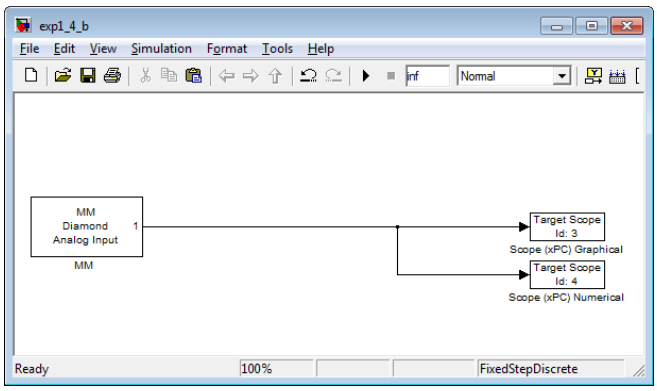
Before building any blocks we need to get familiar with the diamond MM board and its functionalities.



The Diamond MM boards are a multifunction analog and digital I/O module compatible with the PC104 standard. It offers 16 single-ended or 8-differential, bipolar analog inputs (A/D) with 12 bit resolution. It has 2 analog outputs (D/A) with 12-bit resolution, however the D/A is a unipolar peripheral. The board also features 8 digital inputs and 8 digital outputs. Maximum sampling rate that can be achieved using MATLAB programming is 2 KHz, however with DMA operation (cannot be done in MATLAB) 100 KHz sampling rate can be achieved. The image to the left shows how the board will look once you take off the screw terminal on top of it. It is important to read the jumper (J4, J5, J6 etc.) locations based on what is top, left and right for the board as shown in the image. The Diamond MM manual is available in Blackboard and is needed to learn how to configure the board for proper use. Make sure jumpers are at the right position based on the configuration you want, follow instructions from diamond MM manual.

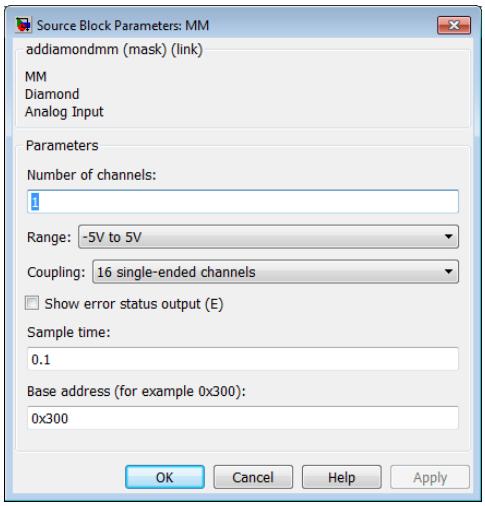
Build and compile the two model files as shown below.

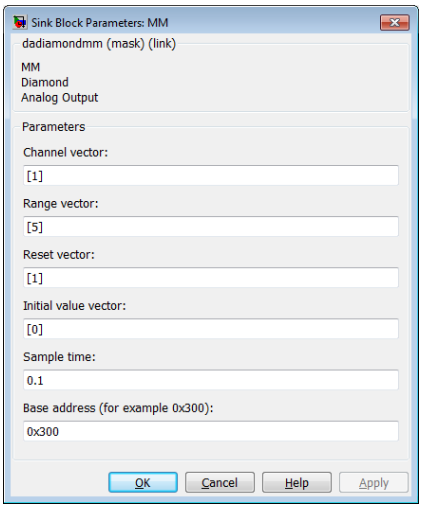




You can find the MM Diamond analog Input block under Simulink library Browser> A/D> Diamond-> MM and the Diamond MM analog Output under Simulink library Browser> D/A> Diamond> MM

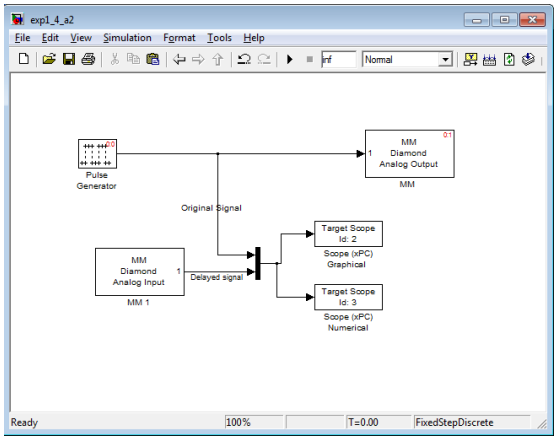
Configure A/D and D/A as follows

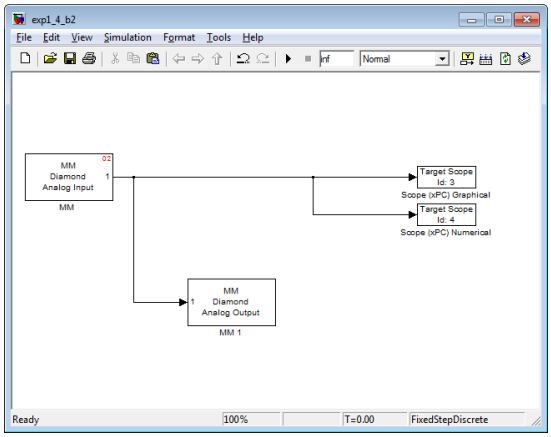




This experiment, as you see is similar to exp 1.3; although, here you are using the D/A and A/D modules to transmit and receive the signal. Remember, that the D/A cannot transmit negative voltages. Connect the ports you configured in the blocks with their corresponding pins on the screw terminal using wires supplied to you in the lab. You should be able to see the waveform on the monitor.

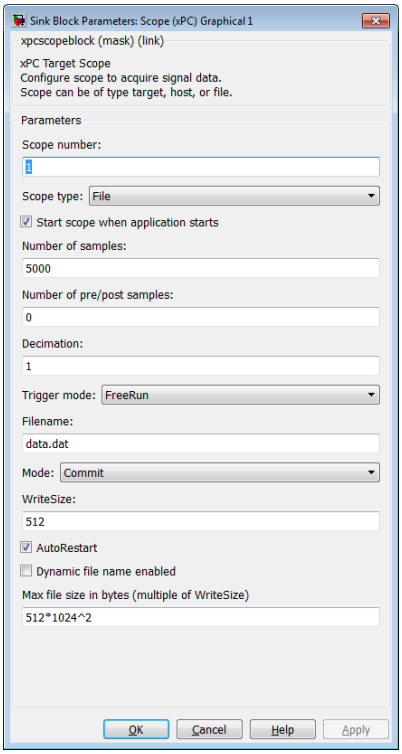
In the second part of this experiment, you will determine the delay in A/D conversion. To do this, modify your model files as shown below.





You may use the xpctargetspy command on the board running exp1_4_a2 then measure the difference in the transmitted and looped back signal. The delay in one side of the conversion is then half of what you measured off the screen.

Now, you will learn how to log and save data, this can be any signal, on to the flash disc. Add a new target scope to the output of your exp1_4_a2 mux (multiplexer). Configure it as shown below. This method configures the scope as a file scope. It saves the data in a file by the name specified in the "Filename" box. Build and run your model and follow the next procedures to have the data transferred to MATLAB workspace.



To transfer the data from the CF card to MATLAB workspace, first right click on target PC in xPC Explorer and select it as default. Type the following commands (it will be advantageous to save these lines of code as a MATLAB script file)

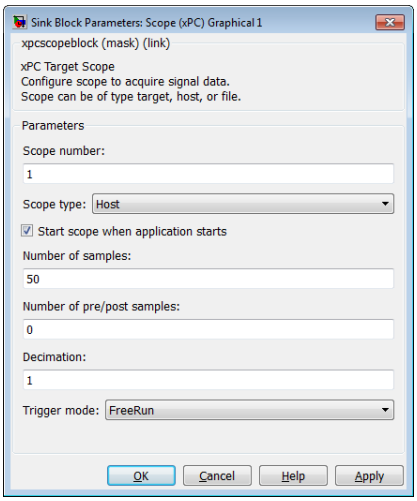
```
% Create file system object
fsys = xpctarget.fs
% Open file on the target file system
h = fsys.fopen('DATA.DAT')

% Read the data from the target file into a MATLAB variable. Note
that
% this data will still be represented in SLRT Target file
% format (i.e. not bytes)
data = fsys.fread(h);
% Close file on file system
fsys.fclose(h);

% Call READXPCFILE to convert the data from SLRT Target
% file format to bytes for use in MATLAB
new_data = readxpcfile(data);
```

At this point the data is saved in "new_data" as a matrix in the MATLAB workspace. The first column is data from the pulse generator, second column is data from the Diamond MM board and third column is the time at which the samples were saved. You can get these signal names by simply typing

```
>> new_data
```



Next, you will learn how to have the SLRT scope running on the host computer. This enables you to get away with no monitor connected to the PC104 board. To do this, change the SLRT Scope type to Host. Build and download the file to your SLRT target; then start the program. Go to SLRT Explorer and expand the tree underneath the "Scopes" section. Look for the target PC that the host scope is associated with and if you expand this tree you should see an option listed as "Host Scope(s)". Right click on the scope you want to see and select "View Scopes". A new window will appear displaying the waveform and they should look the same as the ones you are viewing on the monitor. Play around with different scope settings to get comfortable with adjusting the scope

From this point onwards, you are strongly encouraged to use host scope mode as much as possible since this reduces the requirement of a monitor.

Questions

Q9 For Experiment 1.4 a and 1.4 b report minimum sample time supported by diamond MM boards. Do you think this limitation in sample time can cause bandwidth limitations?

Q10 Mention A/D and D/A capabilities of Diamond boards.

Q11 Do you notice anything wrong (other than the delay) with the signal received through A/D? If so what do you think causes discrepancy in the signal?

Q12 If diamond MM board is used in a feedback control loop will there be any phase lead/lag introduced to the loop due to D/A/A/D?

Q13 Since the D/A in the diamond board does not output negative voltages. Create, compile and test a Simulink model showing how you would transmit a sinusoidal AC wave of amplitude 110 V frequency of 60 hz through D/A on one board and receive it on the second board using A/D. Provide spy screen shots from both boards and Simulink models you created

Q14 When an SLRT scope is configured as a "numerical" target type, what does changing the parameter in "number of samples" do to the values that are being displayed? What does it do when the scope is in target type "graphical mode"?

This concludes Lab 1 Part B. Submit a report on this section. Your TA will let you know of the due dates.

Lab 2

Design of a Cruise Control System for a Simplified Car Model

Introduction

Cruise control is a feature commonly available in many modern vehicles. The system (plant) to be controlled describes the dependence of the output variable of interest (speed) on the manipulated variable (throttle position). Most of this dependence can be expressed with a straightforward application of Newtons law. However, the complexity of the model increases vastly when more detail is desired. Friction, wind resistance, and engine dynamics require considerable effort and experiments to model.

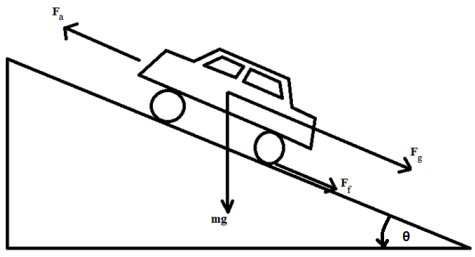


Figure 1: Free body force diagram of a car on an inclined plane.

Newtons law relates the velocity of the car with the sum of the forces applied on it. The free body force diagram in figure 1. shows the forces we are using in our simplified car model; these include throttle acceleration (F_a), friction and wind resistance (F_f) and gravity (F_g). The model takes the form

$$m\frac{dv}{dt} = F_a + F_f + F_g \tag{1}$$

Where, m is the mass of the vehicle, and v is its velocity.

F_a is the force applied on the car due to the torque produced by the engine. This depends on engine and gearbox characteristics, but for our purposes we will simply take that $F_a = c_a u$, where c_a is a proportionality constant and u is the throttle position.[1]

The friction term F_f is typically of the form $F_f = -c_f |v| v$ with c_f being an aggregate friction/wind resistance coefficient. Obviously, this is a gross simplification, but it is sufficient to account for the main contributions during normal operation. Changes in this term due to wind

[1] A more realistic model could be obtained by defining u to be a "commanded force" which is subsequently resolved to throttle position and brake application.

gusts or following other vehicles present perturbations that should be compensated by the control system. We will not even attempt to model the precise aerodynamic forces, although they could be very important in, e.g., racing cars.

The last term serves to describe the effect of gravitational forces when the vehicle moves on an incline. Thus, $F_g = -mg\sin\phi$ where ϕ is the angle between the horizontal and the velocity vector. This angle is treated as an unknown external disturbance for which amplitude and frequency spectrum bounds can be available.

Collecting all the terms, we obtain a model of the form

$$\frac{dv}{dt} = \frac{c_a}{m}u + \frac{-c_f|v|v}{m} - g\sin\phi \tag{2}$$

Typical numerical values for the constants in equation 2 are $m = 1500$, $c_a = 1500$ and c_f =0.5. In the next section we will go over linearization of a nonlinear system.

Linearization

It is imperative to realize that all real systems are nonlinear *i.e.* they are represented by nonlinear functions. However, the design of nonlinear control systems is beyond the scope of this course. So, we restrict ourselves to linear systems and the design of linear controllers for them. To this end, we must linearize the nonlinear dynamics of the simplified car model so we can design and implement a digital controller. To keep things simple yet generic enough, we start with

$$\dot{v}(t) = f(w(t), v(t)) \tag{3}$$

Where, $f(w, v)$ is a time-dependent nonlinear function with input w and output v. For simplicity we will drop the t term and use small letters for variables in the time domain and capital letters for those in the frequency domain.

Let the pair (v_e,w_e) be equilibrium/operating points of $f(w, v)$, computed by setting $\dot{v} = 0$. A nonlinear system must be linearized around its equilibrium. An unforced system (one with no inputs) may have certain fixed equilibria; however a forced system can be made to have equilibrium points at any value by setting w and v appropriately such that $f(w_e, v_e) = 0$. Since linearization is valid only around a small neigborhood of the equilibrium points *i.e* only a small signal input of δw will have an output δv which is linear, we suppose

$$\begin{array}{rlrll} v = v_e + \delta v & and & w = w_e + \delta w, & then & (4) \\ \Rightarrow \dot{v} = 0 + \delta\dot{v} & and & \Rightarrow \dot{w} = 0 + \delta\dot{w} & & \\ \therefore \dot{v} = \delta\dot{v} & and & \therefore \dot{w} = \delta\dot{w} & & \end{array}$$

Then, using Talyor's expansion theorem we get

$$\begin{aligned} \dot{v} &= f(w, v) \\ \delta\dot{v} &= f(w_e, v_e) + \left[\left.\frac{df(w,v)}{dw}\right|_{(w_e,v_e)}\right]\delta w + \left[\left.\frac{df(w,v)}{dv}\right|_{(w_e,v_e)}\right]\delta v + higherorderterms \end{aligned}$$

Since $f(w_e, v_e) = 0$ and we can neglect the higher order terms (because δw is small), we get the following approximation

$$\delta\dot{v} \approx \left[\left.\frac{df(w,v)}{dw}\right|_{(w_e,v_e)}\right]\delta w + \left[\left.\frac{df(w,v)}{dv}\right|_{(w_e,v_e)}\right]\delta v \tag{5}$$

Equation 5 is the linear approximation to our nonlinear equation. For simplicity in writing you may treat $\delta w = u$ and $\delta v = x$. Then, if $\left[\left.\frac{df(w,v)}{dw}\right|_{(w_e,v_e)}\right] = B$ and $\left[\left.\frac{df(w,v)}{dv}\right|_{(w_e,v_e)}\right] = A$, we have the very familiar state-space equation

$$\dot{x} = Ax + Bu$$

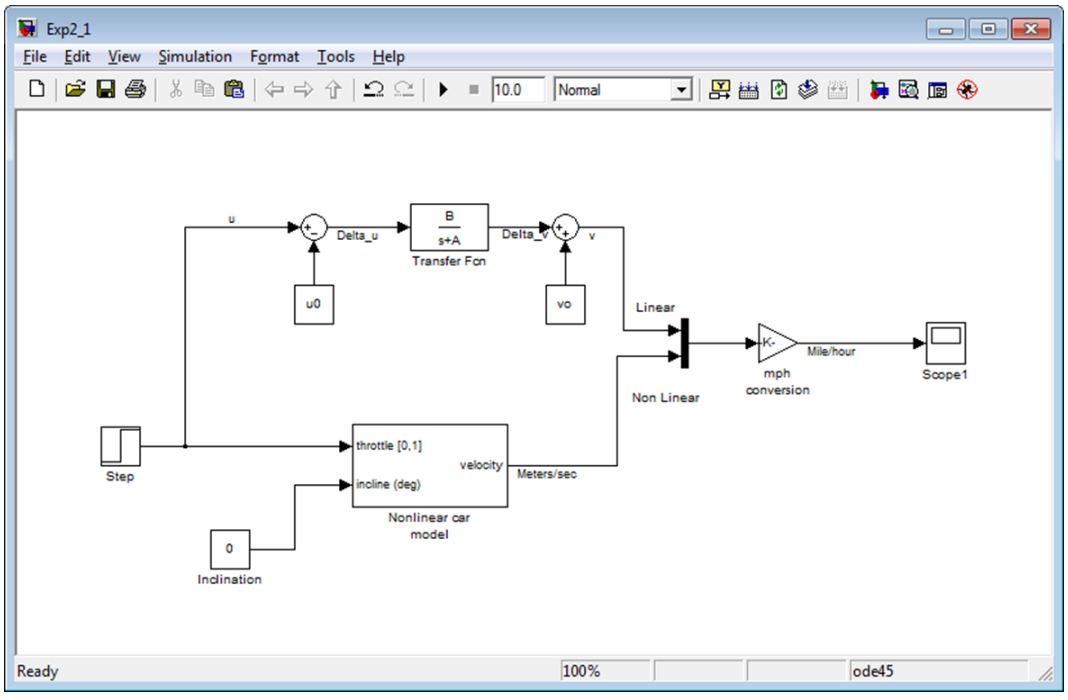
Answer the following questions in your report:

Q1) For the model provided in equation 2 assume $\phi = 0$ and linearize it at an equilibrium of 60 mph. Remember to use SI units when computing u_e and v_e

Q2) Convert the time domain model into a continuous time transfer function. What is the time constant of this system?

Experiment 2.1

You will now setup a simulation experiment where you will observe how the linear model behaves in comparison to the nonlinear one as it moves away from the equilibrium point. Download the file NonLinCar.mdl from BlackBoard and build the following Simulink model.

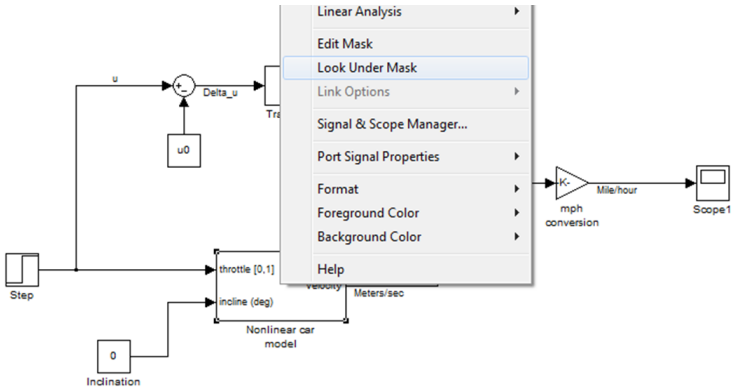


Here is how to find some of the blocks

- Transfer function can be found under Simulink Library Browser → Simulink → Continuous → TransferFcn
- Step can be found under Simulink Library Browser → Simulink → Sources → Step
- Scope can be found under Simulink Library Browser → Simulink → Sinks → Scope

You need to set the state related to the velocity of both models to the initial value of 60 mph (remember to always convert to SI units). In the linearized model this is being done by the

constant blocks u_e and v_e.[2] However, for the nonlinear car you need to do this by right clicking on the block and selecting 'Look Under Mask'.



When the mask opens you will see the model file that implements the nonlinear car as shown in figure 2. The integrator is the state associated with the velocity of the car, and this is the state

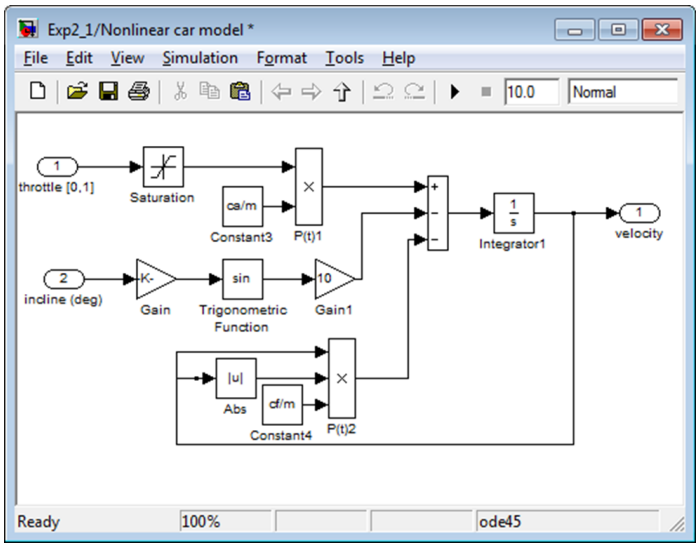
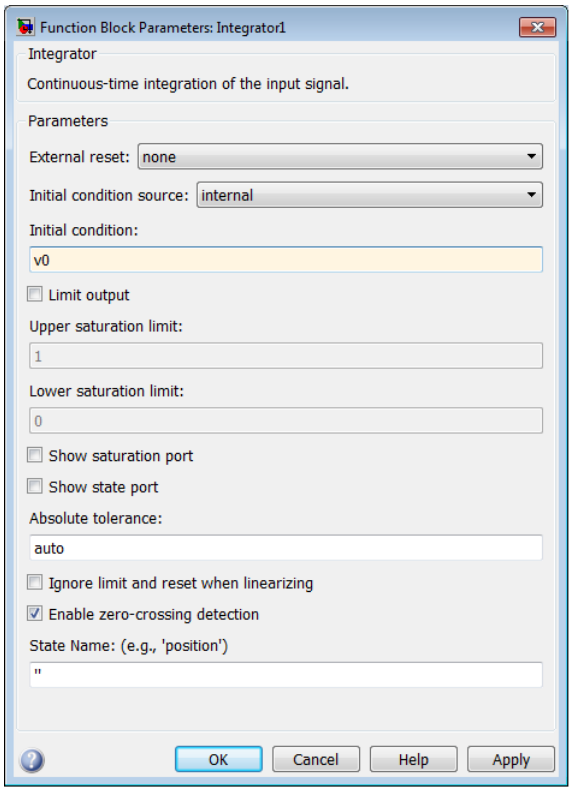


Figure 2: Nonlinear model

you need to initialize to 60mph. Right click on the integrator1 block and specify the value of v_e that you have computed in the initial condition block as shown in the figure below.

Choose the step time to be 10 seconds in the step block. You must understand that these blocks take input u_e that will produce a certain output v_e. So for an initial value of 60 mph in the output you must compute the u_e that will produce this value by setting $f(u_e, v_e) = 0$ and assuming $\phi = 0$. Similarly you must find the value of u that will produce a final value of 61 mph. Run the simulation and see if the linear car behaves similarly to the nonlinear one. Change the final value to different numbers to see how the linear system deviates from the nonlinear as you set the output to different velocities that are further away from 60 mph. Answer the following questions in your report:

[2]Notice that δu and δv are the only inputs this transfer function can take . This is why we are implementing equation 4 before and after the block so that we can directly inject input u and get output v.



Q3) Increase the final value of u and observe how much the linear system deviates from the nonlinear system. At what value of u and v does the output velocity of the linear system differ from the output velocity of the nonlinear one by 10%? Find the values of u and v for which you get the same 10% variation but this time try decreasing u.

Q4) Is the difference in u from u_e the same in both cases in Q3? If not, justify why using your knowledge of nonlinear equations.

Calculations 2.1

Design a continuous-time linear controller with PI structure with a bandwidth of 2 rad/sec and a phase margin of 60°. *Hint:* Bandwidth = 1.5 * Open loop gain crossover frequency.
Note: It will be advantageous to you if you write script for your computations from now on. This will allow you to quickly change some parameters and get the results.

Answer the following questions in your report:

Q5) What are the values of k_p and k_i that you found?

Q6) Provide sensitivity and complementary sensitivity plots for the closed loop system. Plot open loop frequency response and show all stability margins on your plot.

Q7) Provide step response plots and step response characteristics of the closed loop system.

Hint: type 'help bode', 'help step' and 'help stepinfo' to learn how to use these functions. To generate the closed loop transfer function, use the feedback command in matlab (type 'help feedback' to see how it is utilized).

Experiment 2.2

Implement the controller you found in the previous section and compare how well it is able to control both the linear and the nonlinear model. Build a model file that looks like the one in figure 3

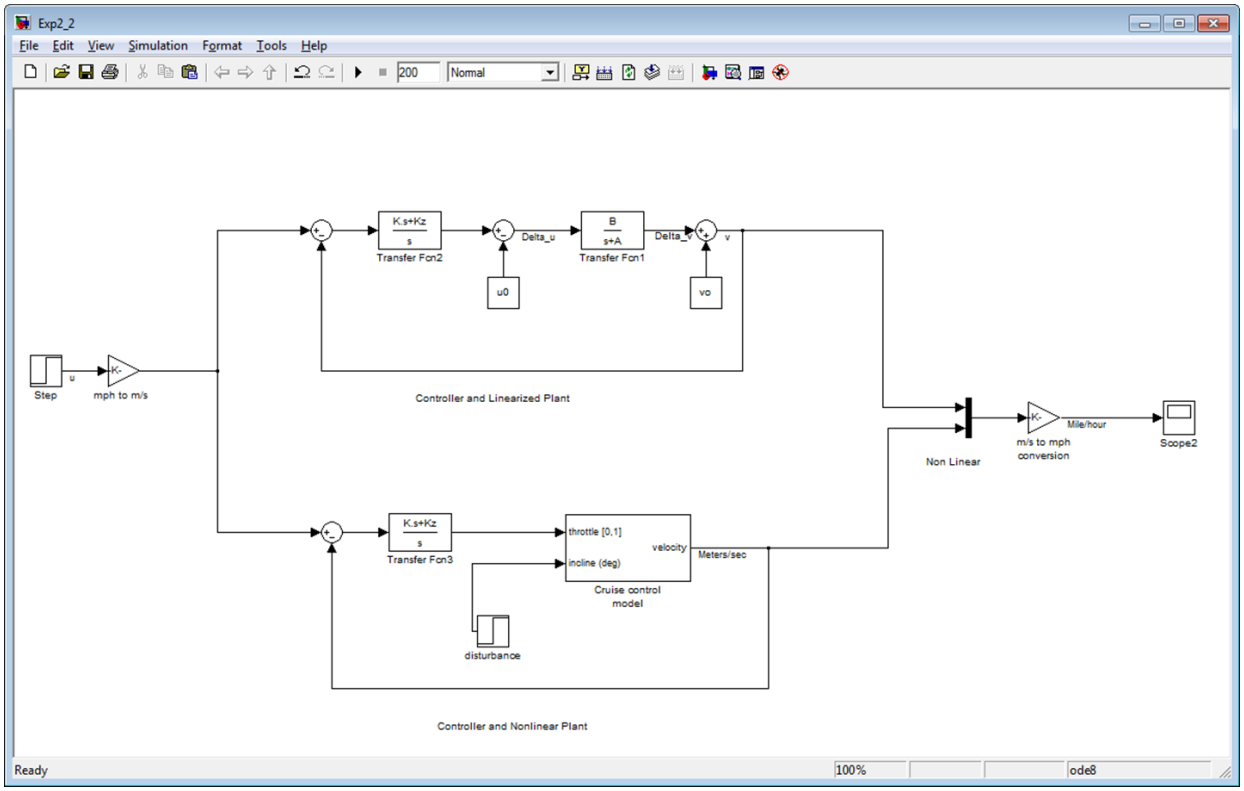


Figure 3: Performance comparison of linear controller on linear and nonlinear model.

Notice that in this model the step block is going to be a reference signal. It no longer will be u. What this means is that you can specify a target velocity (*e.g.*60 mph) and the gain block will convert it to SI units after which the controller will generate the required values of u to get the car to the target velocity. Set step time to 10 seconds initial value to 60 mph, final value to 65 mph and run the simulation to compare the step responses.

Answer the following questions in your report:

Q8) Use the values of v_e, from Q3, that you found to create a 10% deviation in output velocities between the linear and nonlinear model in the step block for final value in Exp 2.2. Do you see any deviations in final velocity in either case? Explain on your findings

Q9) Look closely at the first few seconds of your simulation. There is an undershoot in the response.

Explain why it is there.

Calculations 2.2

We will now attempt to fix the undershoot. To do so, we need the control input u to be u_e and tracking error e=0 at t=0. Any system can be realized in state-space form. As such, we may write the state space equations for the controller as

$$\begin{aligned} \dot{x} &= Ax + Be \\ u &= Cx + De \end{aligned} \tag{6}$$

Where u is the controller output and e is its input. At t=0, we consider a momentary steady state, thus $\dot{x} = 0$. Writing the state-space equation in matrix notation we get the following equation for x_0, the initial condition of the controller

$$\begin{bmatrix} 0 \\ u \end{bmatrix} = \begin{bmatrix} A \\ C \end{bmatrix} x_0 + \begin{bmatrix} B \\ D \end{bmatrix} e \tag{7}$$

Since e = 0 and u = u_e (required conditions stated above)

$$\begin{bmatrix} A \\ C \end{bmatrix} x_0 = \begin{bmatrix} 0 \\ u_e \end{bmatrix} \tag{8}$$

Let $U = \begin{bmatrix} A \\ C \end{bmatrix}$ and $V = \begin{bmatrix} 0 \\ u_e \end{bmatrix}$, then x_0 can be computed as

$$x_0 = (U^T U)^{-1} U^T \cdot V \tag{9}$$

Compute x_0 using equations 8 - 9. This is the initial value that the state related to the integrator in the PI controller must have so that u = u_e at t = 0. However, the model you have at this point implements the controller as a transfer function and in SIMULINK you cannot specify initial conditions for transfer functions. You must replace your controller transfer function block with a state-space block found in

• Simulink library → Simulink → Continuous → State-Space

Also, you will need to get the system matrices [A,B,C,D] for the controller from the transfer function. To do this use the following command:

```
>> Css = ss(C)
```

This creates a state-space object Css for the controller transfer function. Then the system matrices can be accessed by typing Css.a, Css.b, Css.c, Css.d respectively.

Answer the following questions in your report:

Q10) Did using the state-space block with the proper initial conditions for the controller remove the undershoot? Provide step response plots.

Experiment 2.3

In this experiment you will discretize and implement the controller on PC-104 boards for Hardware-In-the-Loop (HIL) testing.

Experiment 2.3.1

You will begin by first discretizing the state-space controller you found in the calculations 2.2 section. To do so use the following command :

```
>> Cd = c2d(Css, Ts, tustin)
```

Cd now has the discrete-time controller state-space which has been discretized from 'Css' using the 'Tustin' (bilinear) transform method at a sampling time of 'Ts' seconds. Use sample time of 0.01 seconds and implement the model shown in figure 4.

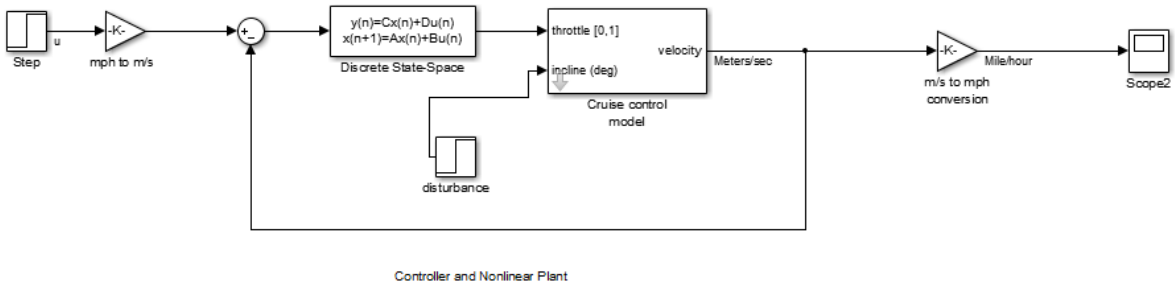


Figure 4: Discrete-time controller in feedback loop.

Run the simulation. If you have not specified the correct initial conditions for this discrete-controller you will see the undershoot in this case as well. To fix this we need to look at the discrete-time state-space equations

$$\begin{aligned} x_{k+1} &= Ax_k + Be_k \\ u_k &= Cx_k + De_k \end{aligned} \tag{10}$$

At steady state $x_{k+1} = x_k$. So, for the state update equation we can write

$$\begin{aligned} x_k &= Ax_k + Be_k \\ \Rightarrow 0 &= (A - I)x_k + Be_k \end{aligned}$$

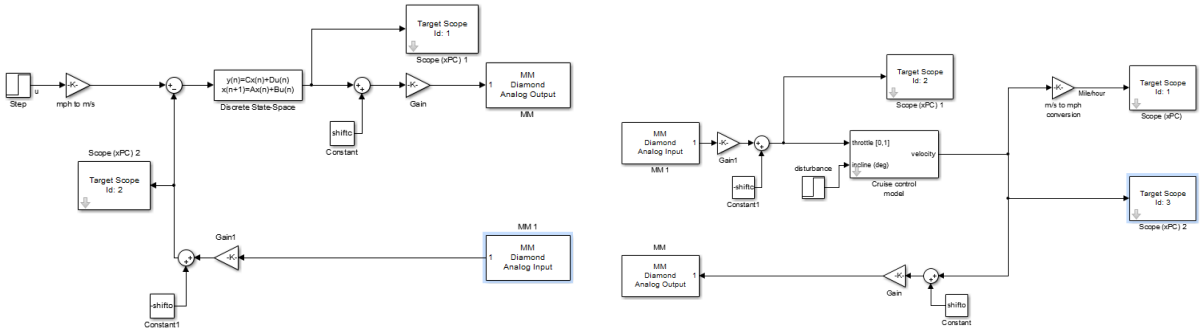
Also, we want $u_k = u_e$ and $e_k = 0$ at t = 0. Thus equation 10 becomes

$$\begin{bmatrix} A - I \\ C \end{bmatrix} x_0 = \begin{bmatrix} 0 \\ u_e \end{bmatrix} \tag{11}$$

Let $U = \begin{bmatrix} A - I \\ C \end{bmatrix}$ and $V = \begin{bmatrix} 0 \\ u_e \end{bmatrix}$, then x_0 can be computed using equation 9. At this point you should be able to get the exact response as you did while using a continuous controller with corrected initial conditions.

Experiment 2.3.2

We are now going to implement this digital controller in the Advantech PC-104 boards. To do this you will have to split up the model file from Exp 2.3.1 so that the controller goes in one board



(a) Controller side

(b) Plant side

Figure 5: Experiment 2.3.2

and the plant goes in the other. Before splitting the model it is recommended that you look at the maximum and minimum values being output from the controller and also from the plant (in m/s). You must condition both control output and plant output so that they conform to the supported ranges of the A/D and D/A boards (Recall Q13 from Lab 1 part B). Your model files should look similar to the ones in figure 5. Implement these files on two PC-104 boards. On the plant side, remember to use 'Fixed step' with ODE5 solver settings. This will ensure the plant model behaves as similar to the continuous time version as possible.

Answer the following questions in your report:

Q11) Use file scopes on both plant and controller side to save the control output and plant output data. Provide plots of these signals against time in your report.

Experiment 2.4

In this experiment you will test the disturbance rejection properties of the controller you have designed. So far in this lab the value of ϕ was set to 0°, implying that the car was travelling on a level road. Use the model file you created in experiment 2.3.1. You will set up the experiment so that the following scenario is simulated - the car initializes at t=0 with a velocity of 60 mph, after 30 seconds its velocity is increased to 65 mph. At 100 seconds it hits a hill with a slope of 1%. Run this simulation on the host computer (HIL testing is advisable but not necessary in this step) Answer the following questions in your report:

Q12) Provide a velocity against time plot showing how your controller behaved in the face of the disturbance. Repeat the experiment by increasing the slope 1% at a time to find when the controller fails. Comment on your observations.

Experiment 2.5

In this experiment you will see how sample time in discretization of the controller can affect system stability.

Experiment 2.5.1

Use the models from Exp 2.3.2. Whenever you want to change the sampling time of your controller you must discretize it again from the continuous time state-space using the c2d command. Iterate Exp 2.3.2 by increasing the sample time on the controller side until you see the controller fail. Answer the following questions in your report:

Q13) Try to find, to the second decimal, the value of Ts that makes the controller fail. Provide screenshots of the unstable response. Q14) Explain clearly, why the controller fails.

Experiment 2.5.2

In this step you will learn how to correct for the controller failure in Exp 2.5.1. Redo your math in calculations 2.1 but this time add a phase correction term for the delay. When disctretizing remember to use the value of Ts from Q13.

Hint: The phase contribution of a time delay in a continous transfer function is expressed as [delay(in sec)$\times$ unity gain crossover frequency (in degrees/sec)]

This is the end of Lab 2

Lab 3

Design of a Liquid Level Control System

Introduction

In this lab you will study the level of liquid in a cylindrical container and design a controller to maintain the liquid level at a specific height. Liquid level control is used in petrochemical industries in re-boilers, waste water treatment plants, bioreactors etc. The output variable of interest is the height of the liquid column in the container. The container can be spherical, cylindrical or any other shape depending on the application. In our model we consider the cylindrical case. Refer to figure 1, the input is a constant flow of liquid into the container through F_{in} by keeping the valve (valve 1) open at all times. The controlled variable is the outflow of liquid from the container, which depends on the position of the valve at F_{out}; we will call this valve 1. Other applications of this system may have F_{in} as the control and F_{out} fixed. The position is controlled by controlling the speed at which valve 2 is opened and closed.

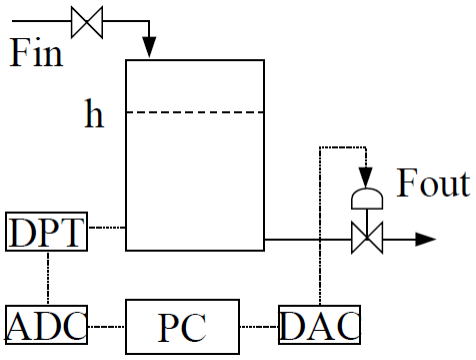


Figure 1: Schematic of liquid level control experiment.

Bernoulli's principle states that the sum of all mechanical energies in a fluid along a streamline is the same at all points on that streamline. Equation 1 puts this in writing.

$$P + \frac{1}{2}\rho v^2 + \rho g h \tag{1}$$

Where, P is pressure, ρ is density of fluid, g is acceleration due to gravity, v is velocity and h is the fluid level. The inlet conditions are as in equaion 2 and equation 3 dictates that the volume of a

fluid is constant under incompressible flow

$$A_i\dot{h}_i = F_{in} - F_{out} \tag{2}$$
$$A_i v_i = A_o v_o \tag{3}$$

Then from Bernoulli's principle, equating input and output at the junction of the bottom of the tank and the outflow pipe, we have

$$P_i + \frac{1}{2}\rho v_i^2 + \rho g h_i = P_o + \frac{1}{2}\rho v_o^2 + \rho g h_o \tag{4}$$

However, $P_i = P_o, h_o = 0$ and if the area of the inlet pipe is much greater than that of the outlet pipe, $A_i >> A_o$ then we have $v_i << v_o$. Equation 4 then becomes

$$2gh_i \cong v_o^2 \tag{5}$$

Applying equation 5 to equation 2 and simplifying it, we get

$$\dot{h}_i = \frac{F_{in}}{A_i} - \frac{A_o}{A_i}\sqrt{2gh_i} \tag{6}$$

It may not be intuitive at this point but the fact is that this model assumes the outlet pipe is fixed so $h_o = 0$, but the inlet pipe is virtually floating on the top of the liquid level in the container. So the level of liquid is $h_i - h_o = h_i$. It is clear from equation 6 that the differential equation describing the height of the liquid column is a nonlinear ordinary differential equation. The square root implies that the tank drains faster if there is less liquid and vice versa[1]. We now look at the controlled variable.

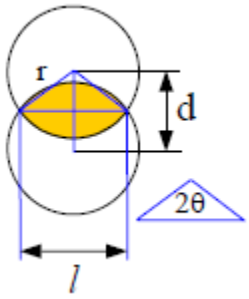


Figure 2: Schematic of liquid level control experiment.

For valve 2 we choose a disc shaped gate which stops the outflow of liquid when the gate is at the lowermost position. Figure 2 shows the configuration of the gate valve under consideration. In a partially open configuration the net area through which liquid flows out is that which we get by subtracting the shaded region from cross-section of the pipe. We express the percent of open area as a function of distance between gate center and pipe center. Then

[1]When the tank is full, it drains slower (as an exponential). The rate of change $\dot{h}/h$ goes to zero (system looks like an integrator). When it is near empty, $\dot{h}/h$ approaches infinity (system looks like a very fast LHP pole), and that is why the tank drains in finite time

$$
\begin{aligned}
SectorArea &= r^2 Cos^{-1}(d/2r) && (7)\\
TriangleArea &= (ld/4) = \frac{d\sqrt{4r^2 - d^2}}{4} && (8)\\
ShadedArea &= 2r^2[Cos^{-1}(D) - D\sqrt{1 - D^2}], where\ D = d/2r && (9)\\
A_o &= \pi r^2 - ShadedArea\\
&= \pi r^2 - 2r^2[Cos^{-1}(D) - D\sqrt{1 - D^2}]\\
&= r^2(\pi - 2[Cos^{-1}(D) - D\sqrt{1 - D^2}]) && (10)
\end{aligned}
$$

The control input u is the speed at which the valve is operated and that in turn controls the position of the valve which determines A_o. Equation 11 is the second ODE in our model

$$\dot{D} = -u \tag{11}$$

Typical values to be used for the unknown parameters in this model are -
$F_{in} = 0.00339m^3/s;\ g = 9.80665m/s^2;\ r_i = 0.15m;\ r_o = 0.02m$

Linearization

Since the ODEs obtained for the liquid level in a tank model are nonlinear, you must linearize them before proceeding to controller design. Recall equations 6 and 11. We will treat these as functions describing our states

$$\dot{x}_1 = f_1(h, D, u) = \dot{h}_i = \frac{F_{in}}{A_i} - \frac{A_o}{A_i}\sqrt{2gh_i} \tag{12}$$

$$\dot{x}_2 = f_2(h, D, u) = \dot{D} = -u \tag{13}$$

Since h is the output of interest, the output equation is

$$y \ = g(h, D, u) \ = h \tag{14}$$

Now, the state space system equations of the linear model will be

$$\dot{x} = Ax + Bu \tag{15}$$

$$y = Cx + Du \tag{16}$$

where

$$A = \left[\begin{matrix} \frac{df_1(h,D,u)}{dh} & \frac{df_1(h,D,u)}{dD} \\ \frac{df_2(h,D,u)}{dh} & \frac{df_2(h,D,u)}{dD} \end{matrix}\right]\Bigg|_{(h_e,D_e,u_e)} \qquad B = \left[\begin{matrix} \frac{df_1(h,D,u)}{du} \\ \frac{df_2(h,D,u)}{du} \end{matrix}\right]\Bigg|_{(h_e,D_e,u_e)}$$

$$C = \left[\begin{matrix} \frac{dg(h,D,u)}{dh} & \frac{dg(h,D,u)}{dD} \end{matrix}\right]\Bigg|_{(h_e,D_e,u_e)} \qquad D = \left[\begin{matrix} \frac{dg(h,D,u)}{du} \end{matrix}\right]\Bigg|_{(h_e,D_e,u_e)}$$

Answer the following questions in your report:

Q1) For the model provided in equation 12 - 14 assume $h_e = 1, D_e = 0.5, u_e = 0$ is an equilibrium

where the water level is steady at 1 m and linearize it to find the linear state-space system equations. Show all calculations

Q2) Convert the continuous time state-space model into a continuous time transfer function and provide the transfer function in your report.

Hint: use MATLAB command tf(ss(A,B,C,D))

Experiment 3.1

MATLAB provides many useful tools for control system design and analysis. One such tool is the 'linmod' command. This linearizes a model at a particular equilibrium. This could effectively help reduce time spent in painstaking linearization calculations for complicated models. Download the simulink model file named 'LiquidLevelModel' from Blackboard. Observe that F_{in} is fixed for our experiment since we assume constant inflow rate for liquid. The 'valve speed' inport and 'level' outport are required for linmod to recognize what are the inputs and what are the outputs of the system. Also, by looking into the mask you will find two integrators associated with h and D. They must set to initial conditions h_e and D_e; this is how linmod knows what equilibrium condition to linearize the system at. Setup your file to the linearization conditions given in Q1 and enter the following command in MATLAB

```
Plin = linmod('LiquidLevelModel')
A = Plin.a; B = Plin.b; C = Plin.c; D = Plin.d
```

Answer the following questions in your report:

Q3) Compare the state-space system representation you calculated in Q1 with what you get using the linmod command. Is there any difference?

Calculations 3.1

Design a PID controller for the linearized plant that has a bandwidth of 10 rad/sec and a phase margin of 60°. Use $\tau = 0.0009$ for the pseudo pole. The structure for a PID controller in continuous time is $(C = \frac{K(s+a)^2}{s(\tau s+1)})$ [2].

Answer the following questions in your report:

Q4) What are the values of K and a? Also write down the values of k_p, k_i and k_d that you found.

Q5) Provide sensitivity and complementary sensitivity plots for the closed loop system. Plot open loop frequency response and show all stability margins on your plot. Provide step response plots and step response characteristics of the closed loop system.

[2] You must show us your calculations but to check for correctness the values of the controller should be K = 71.6833, a = 1.7777

Experiment 3.2

Now that you have designed a PID controller and verified its performance on the linear system, test the performance of your controller on the nonlinear plant by implementing the following model

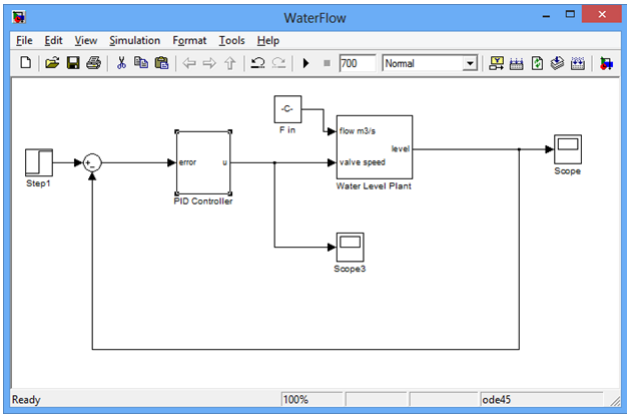


Figure 3: PID controller with nonlinear plant.

• PID controller block can be found in → Simulink → Continuous → PID Controller
Answer the following questions in your report:

Q6) What do you get for the step response when the step value is changed from h_e to h_e+0.1 i.e change of 0.1m in liquid level.?

Q7) Why do you get the response as seen in Q6?

Experiment 3.3

We will now attempt to fix the error in experiment 3.2. The system becomes unstable due to integrator windup. A large change in the step reference causes the control input to increase by a considerable amount. However, the control input gets clipped due to saturation in actuators of the plant, the integrator in the controller keeps adding the error from the plant output compared to the reference and increasing the control input to compensate; nothing happens since the plant cannot respond to the controller's commands. This leads to large overshoots which could lead to instability.

This effect is called integrator windup. One way to cure this is to implement an anti-windup for the integrator in the PID. This is done by switching off integral action of the PID controller whenever the output of the controller exceeds certain lower and upper bounds. To do this in your simulation create a subsystem that looks like the one below and replace it with your PID controller

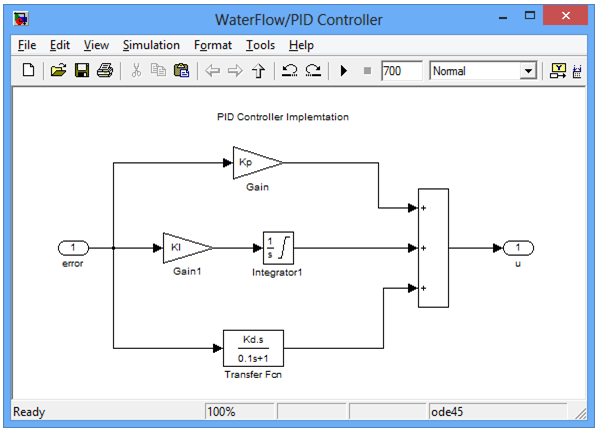
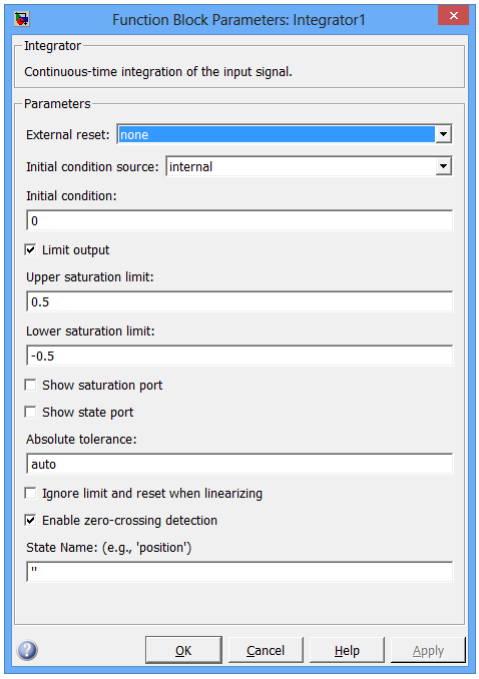


Figure 4: Subsystem implementing antiwindup.

• The limited integrator can be found in → Simulink → Continuous → Limited integrator
Configure the limited integrator as shown



The saturation limits chosen for the limited integrator is exactly the same as those in the plant input.
Answer the following questions in your report:

Q8) Did implementing the limited integrator solve your problem and make the system stable? Provide step response plots.

Q9) Research and list what other methods would help avoid integrator windup related issues.

Experiment 3.4

Discretize your PID controller and test it in a closed loop in SIMULINK with the plant to verify it works. For sampling time use Ts = 0.001 sec. Next, use your knowledge from previous labs to split the controller and plant into two separate models. Build and download them to target PCs. Make necessary connections via A/D, D/A peripherals.
Answer the following questions in your report:

Q10) Does your design work well on the Target PCs? Provide step response plots by gathering input and output data from each Target PC as you deem necessary.

Experiment 3.5

You will now attempt to simulate a more practical controller implementation. Notice that in practice it does not make sense to design and implement a controller that has as reference a fixed change in step input compiled with it i.e. in your case you have specified the step to go from 1 m to 1.1m. However, you may want to have a step from 1m to a different value. It will be cumbersome and inefficient to have to recompile every time a change is needed in the reference. In this part of the lab you will design a simulation where you will be able to type in commands from the host PC connected to the controller target PC. The target PC should be able to recognize your commands and process them as reference inputs to the controller. Use the 'serialtest' experiment from Lab 1 as reference for your solution.
Answer the following questions in your report:

Q11) Provide the simulink block diagram you implemented on the controller board.

Q12) Provide a reference command of 1m. Once the output has stabilized, reference to 1.05 m and after the output stabilizes here command reference to 0.95 m. Plot the input and outputs genereted.

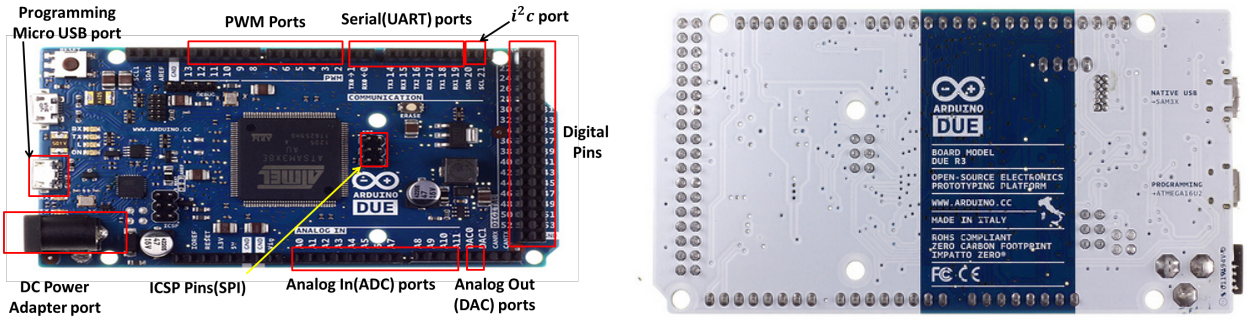
This is the end of Lab 2

Lab 4

Digital Control Systems Design Using Arduino Due Microcontroller Board and Simulink

Introduction

The Arduino Due is a microcontroller board based on a 32-bit ARM core microcontroller (Atmel SAM3X8E ARM Cortex-M3 CPU). It has 54 digital input/output pins (of which 12 can be used as PWM outputs), 12 analog inputs, 4 UARTs (hardware serial ports), an 84 MHz clock, 2 DAC (digital to analog), 2 TWI, a power jack, an SPI header, a JTAG header, a reset button and an erase button. **Unlike other Arduino boards it operates at 3.3V**. It can be powered using a micro USB port (programming port) or AC-DC adapter.



(a) Top view of Arduino Due

(b) Bottom view of Arduino Due

Figure 1: Top and bottom view of the Arduino Due.

The 12 analog inputs of the Arduino Due have a voltage range of 0-3.3V. It can support a resolution of 12 bits on each ADC channel. On the other hand the two DAC channels have a voltage range of $\frac{1}{6}(3.3) - \frac{5}{6}(3.3)$V. You must keep this limitation in mind while doing experiments with the DAC on the Arduino Due. The resolution of the DAC is again 12 bits on each channel.

The Due has 4 serial ports (0-3). Serial port 0 is directly connected to the USB programming port and any data can be passed down or read from the Due using that link and any terminal program on the computer (the Arduino IDE has such a program for convenience). All the serial ports operate at 3.3V as high and 0V as low.

All the necessary pinouts of the Arduino Due are shown in Figure 1 For more details please visit `http://arduino.cc/en/Main/arduinoBoardDue`
Answer the following questions in your report:

Q1) Provide specifications about the hardware peripherals of the Arduino Due microcontroller board.

Q2) What are the limitations of D\A and A\D on this Arduino board?

How to Install the Simulink toolbox for Arduino Due

N.B. You need not perform the tasks in this section for the computers in the GWC379 lab. This is only if you wish to install the Packages on your own PC

Start by downloading the Arduino IDE software from :
`http://arduino.cc/en/Main/OldSoftwareReleases`
Preferrably use the 1.5.8 software, the BETA is actually quite stable. Unzip the Arduino onto a folder. Next, plug in the micro USB end of a cable to the programming port (the one closest to the power jack) and the other end of the cable to the PC. Install drivers for the board using device manager; the drivers can be found under the unzipped Arduino folder inside the Drivers folder. Once the drivers are installed you should see it listed under a COM port inside Device Manager in Windows.

The next step is to install the Simulink Package. In the MATLAB command prompt type the following and hit enter

```
>> supportPackageInstaller
```

The support package installer window will appear as shown in fig 2

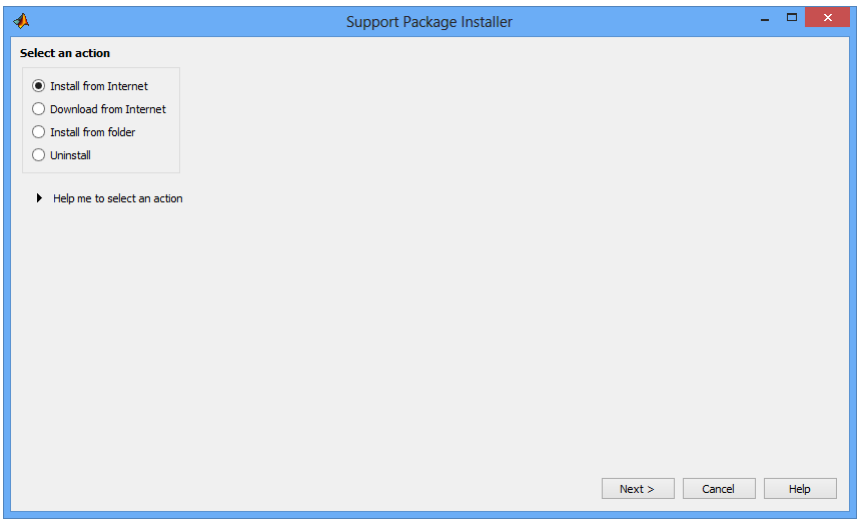


Figure 2: Simulink and Matlab Support Package Installer.

Check the Install from Internet radio button and click next and the following screen should appear (fig 3).

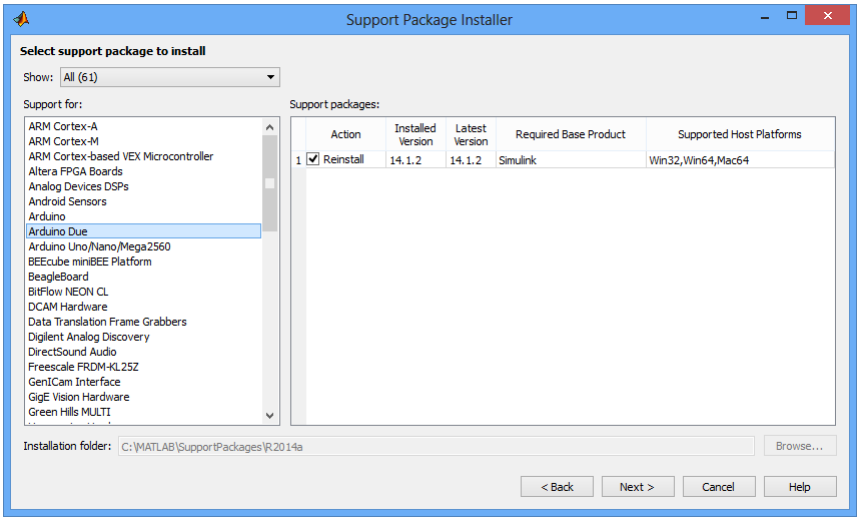


Figure 3: Choose Support Package to install.

Although the Figure shows Reinstall, it will show up as install if the Package has never been installed before on the computer. Select Arduino Due as shown in figure 3 and click next. The next page will ask for a valid Mathworks account to log in(fig 4). If you don't have one, you can create it for free using this link : **Matlab Account Creation**

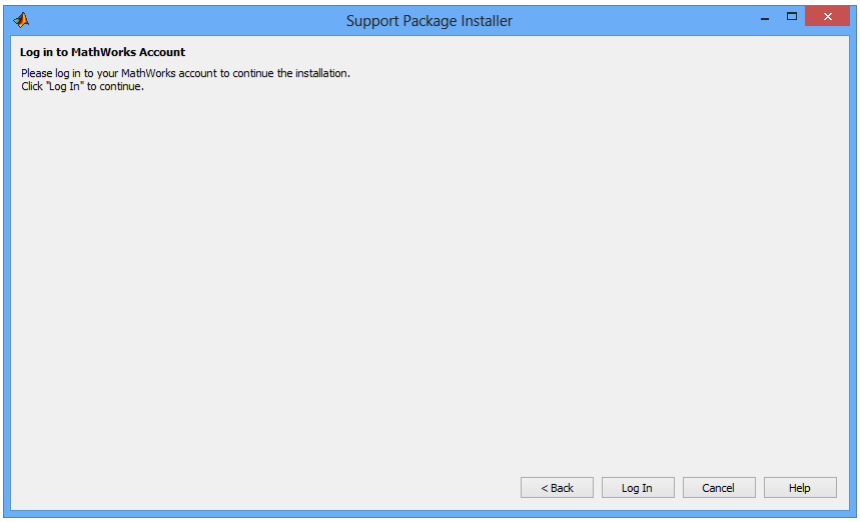


Figure 4: Log in to Mathworks Account.

The next few screens will ask for permission to install and choose installation folders. Complete these steps until the installation process ends.

The Arduino Due package for MATLAB has a bug. With the default downloaded package, two boards cannot be programmed simultaneously. In order to allow MATLAB to program two boards

the following procedure for applying a patch must be performed:
Replace "arduinoSetupDue.p" file at the following path with the file given to you :

```
<Support Package Installed Path>\arduinodue\+realtime\+internal\arduinoSetupDue.p
```

Typically the support packages are installed under

```
"C:\MATLAB\SupportPackages\R2014a\"
```

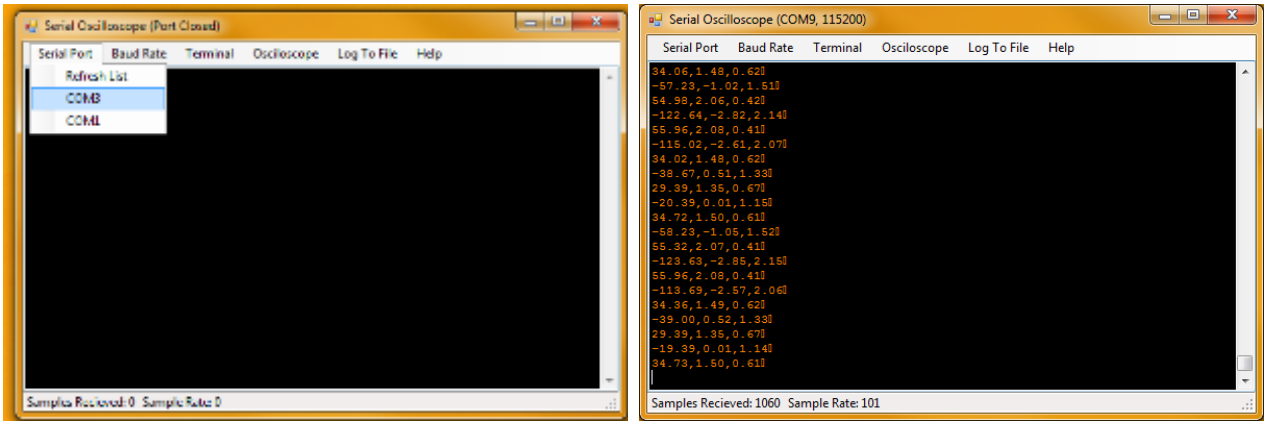
unless you have elected to use a different directory.

N.B. Keep in mind that all the steps provided in this section are only to be performed on your PC.

Viewing Data from the Due boards

Unlike the PC104 computers, the Arduino Due is a simpler microcontroller and thus does not have graphics capability. This limits the way in which we can observe data and plots from them through Simulink. Simulink allows a method of running programs called "External Mode" in which MATLAB installs a simple server and that sends data back to the PC running simulink where you may view the data using a Simulink Scope. The disadvantage to this method is that the server bogs down the Due's processor and it keeps going into overruns unless a very big sample time is used ($\geq$1 sec).
The method we have found that works best is where the data you are interested in viewing is sent out through the serial port of the Due boards and the PC receives and displays it using a software called Serial Oscilloscope. The tool is open source and so is free for download and use through `http://www.x-io.co.uk/serial-oscilloscope/`

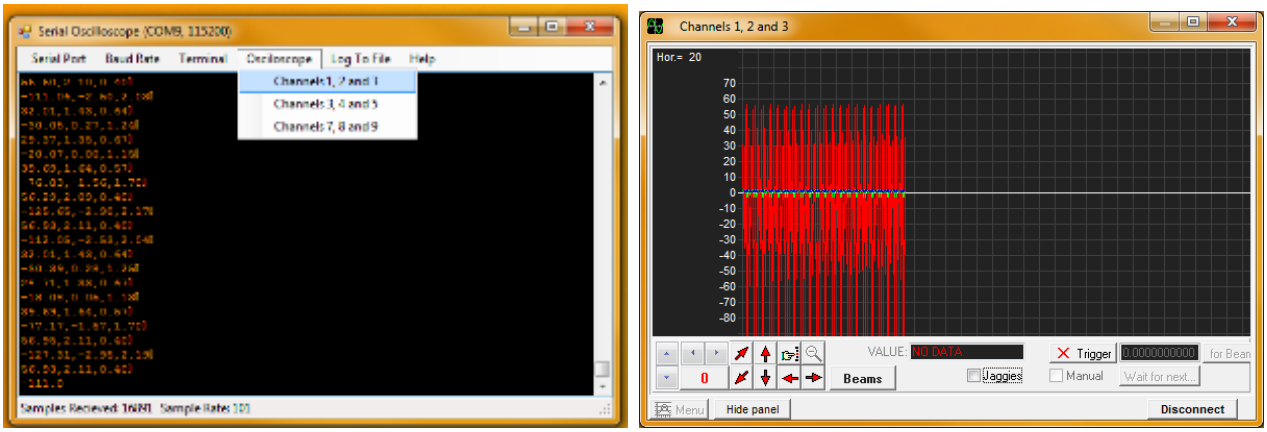
Assuming you have serial data output configured in your model, if you open 'Serial Oscilloscope.exe' and choose the appropriate COM port as shown in figure 5(left) you will see your comma separated data stream being printed on the screen, figure 5(right).



(a) Choose COM port (b) Data received from the Due board

Figure 5: Getting data into Serial Oscilloscope.

Next, in order to view the oscilloscope go to Oscilloscope→ Channels1,2,3 and the Oscilloscope screen should appear.



(a) Choose set of channels

(b) Serial Oscilloscope

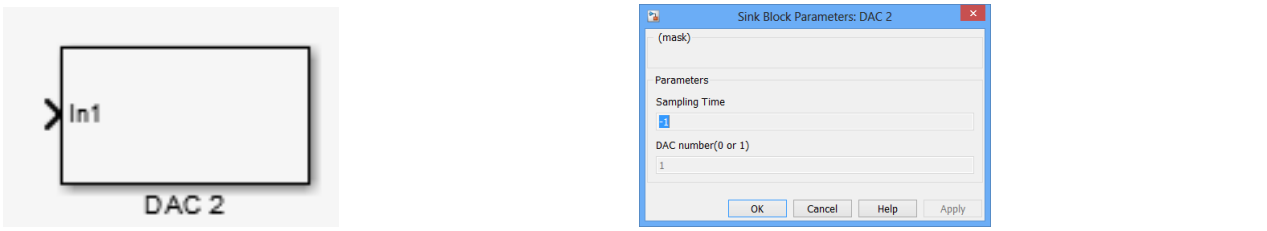
Figure 6: Viewing the Serial Oscilloscope output.

Custom Simulink Library Blocks

In order to make this lab experiment look similar to the ones you have done with PC104 computers we built some custom Simulink blocks. This streamlines the programming process and helps make it intuitive. The library files are available for download from Blackboard. You will need to add them to your MATLAB path every time you use the computers in the GWC 379 lab. A brief description of each block follows.

DAC

The DAC block we designed takes 'double' numbers as input. You may leave the sample time property to -1 (inherited). Keep in mind that the DAC has a voltage range limitation. Either of the DAC 0 or 1 channels may be used with this block.



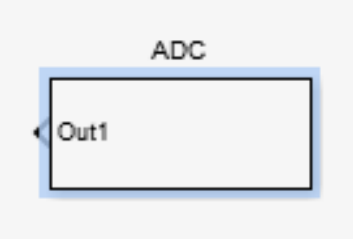
(a) DAC custom Block

(b) DAC properties dialogue box

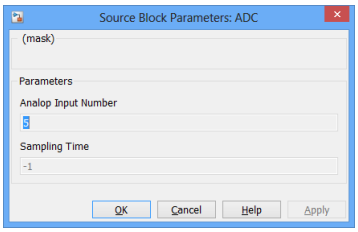
Figure 7: DAC custom block.

ADC Block

The ADC block utilizes the ADC channels on the Arduino Due. The output of the block is in 'double' format. You may specify a sample time or use inherited mode. Choose the appropriate channel for use in your models.



(a) ADC custom Block



(b) ADC properties dialogue box

Figure 8: ADC custom block.

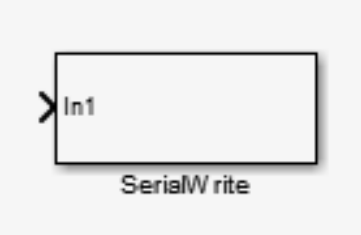
Serial Monitor

This block is what outputs data on to the serial port so it may be viewed using the Serial Oscilloscope program. It takes numbers in double format. Three channels are present in the block. If you do not need a certain input then simply connect a 'ground' or 'constant' block to it.

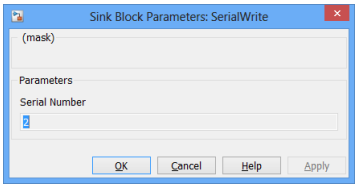
Figure 9: Serial Monitor block.

Serial Read and Write

SerialWrite parses floating point numbers to ASCII and sends the data out over the serial port number selected. The Due has 4 ports of which port 0 is reserved for Serial to USB communication. So use any port other than that.



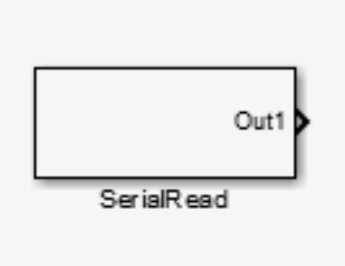
(a) Serial Write block



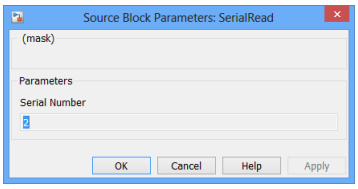
(b) Serial Write properties dialogue box

Figure 10: Serial Write.

This block is used to read incoming serial data from the other Due boards that implement the Serial Write block. It reads from one of three ports which is specified by the Serial Number property.



(a) Serial Read block



(b) Serial Read properties dialogue box

Figure 11: Serial Read.

Instructions for doing experiments

You will be given two Arduino Due boards and two micro USB cables. Please connect both of them to the computer (make sure micro USB is plugged into programming port of Arduino Due). The USB cable sufficiently powers both the boards(no other hardware) so you do not need any external power supply for these experiments. Once you connect the Arduino Due you need to find out the COM port numbers assigned to the boards by the computer (these are COM ports of the computer not Arduino). To do so please look at **My Computer** $\rightarrow$ **properties** $\rightarrow$ **Device Manager** $\rightarrow$ **Ports** . You should see both the Arduino Dues programming port listed there. Please make a note of COM port numbers. This lab section consists of three experiments. For each section you will be provided with the model files and wiring diagram. To download model files on to Arduino Due open Simulation$\rightarrow$ Model Configuration parameters. Select the 'Run on Target Hardware' tab as shown below. Select 'Set host COM port' as 'Manually'. Update COM port number to the COM port number of Arduino due (For example if Arduino due is on COM9 of the computer, update it to 9 as shown below)

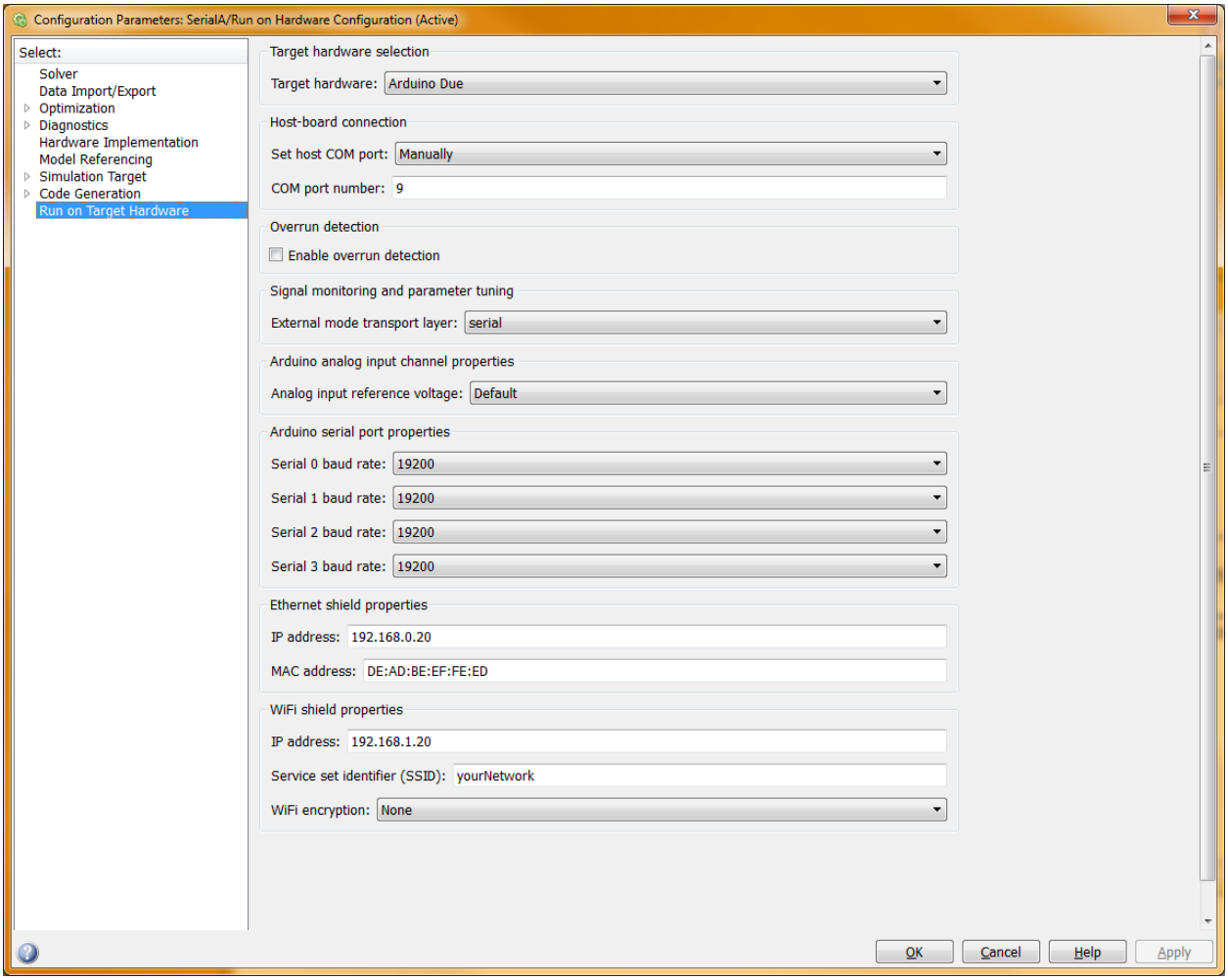


Figure 12: Setting up Simulink for Due programming.

You will have to correctly select the ports for each of the boards you have.

Experiment 4.1

This experiment is similar to what you had done in Lab 1a. You will loop back a square wave over serial trasnsmission using the Due boards.

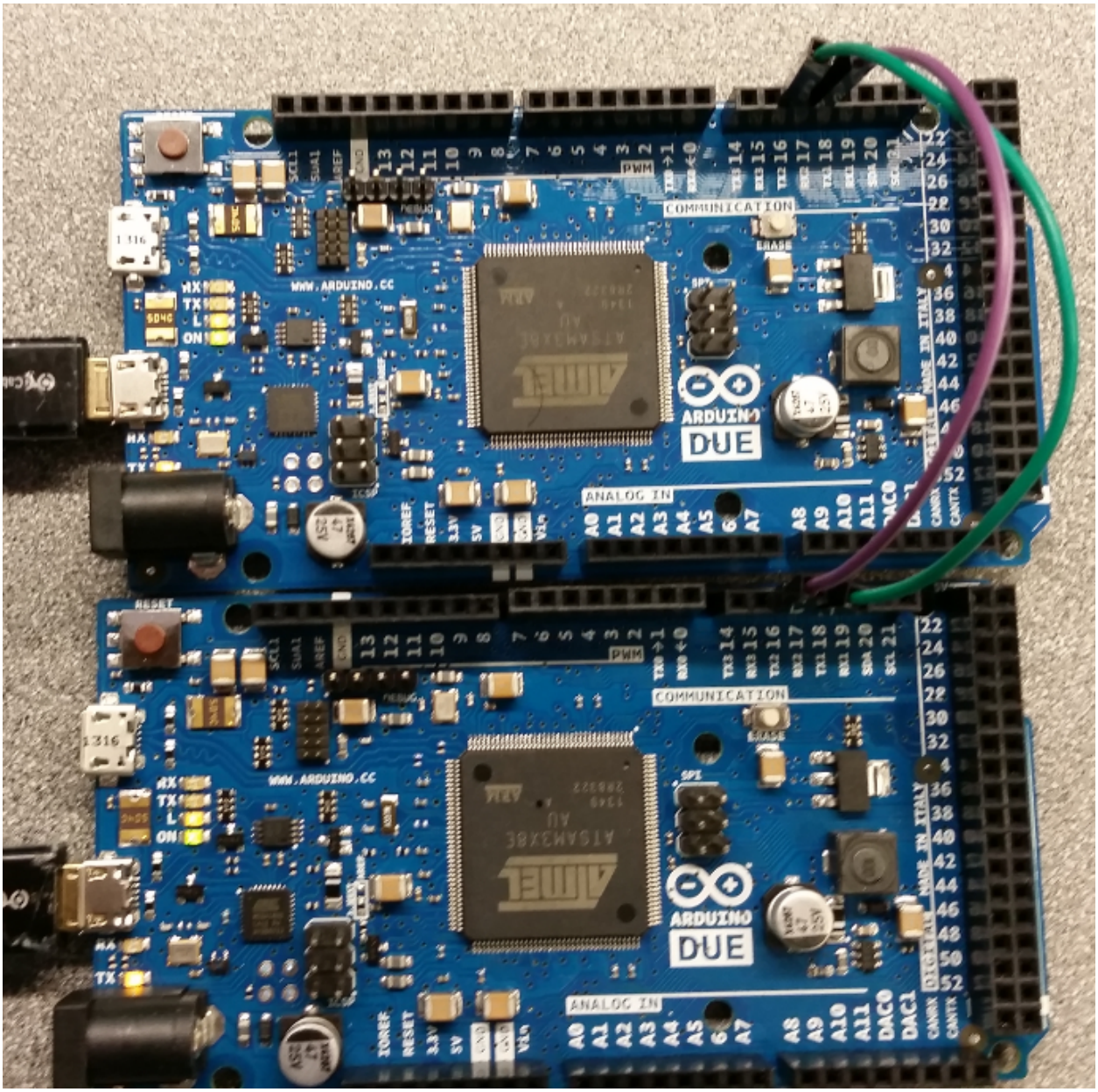


Figure 13: Wiring for Exp 4.1.

The Simulink model diagrams are shown below

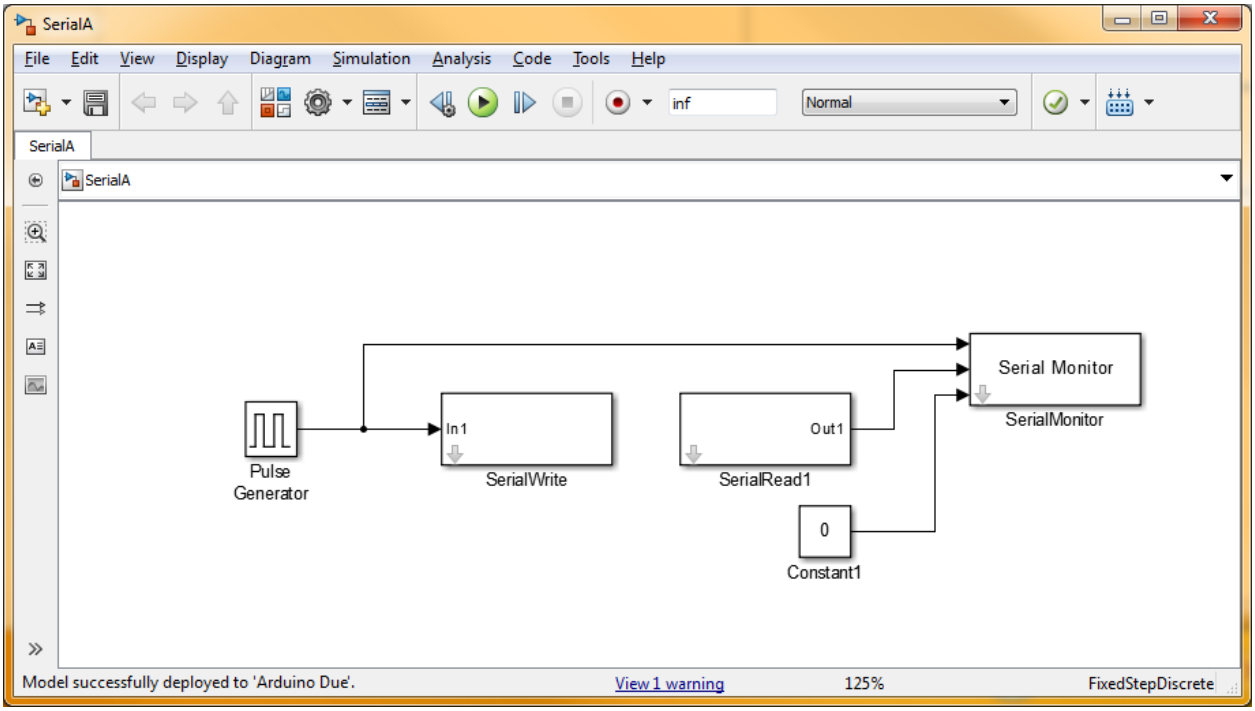


Figure 14: Square wave transmit side.

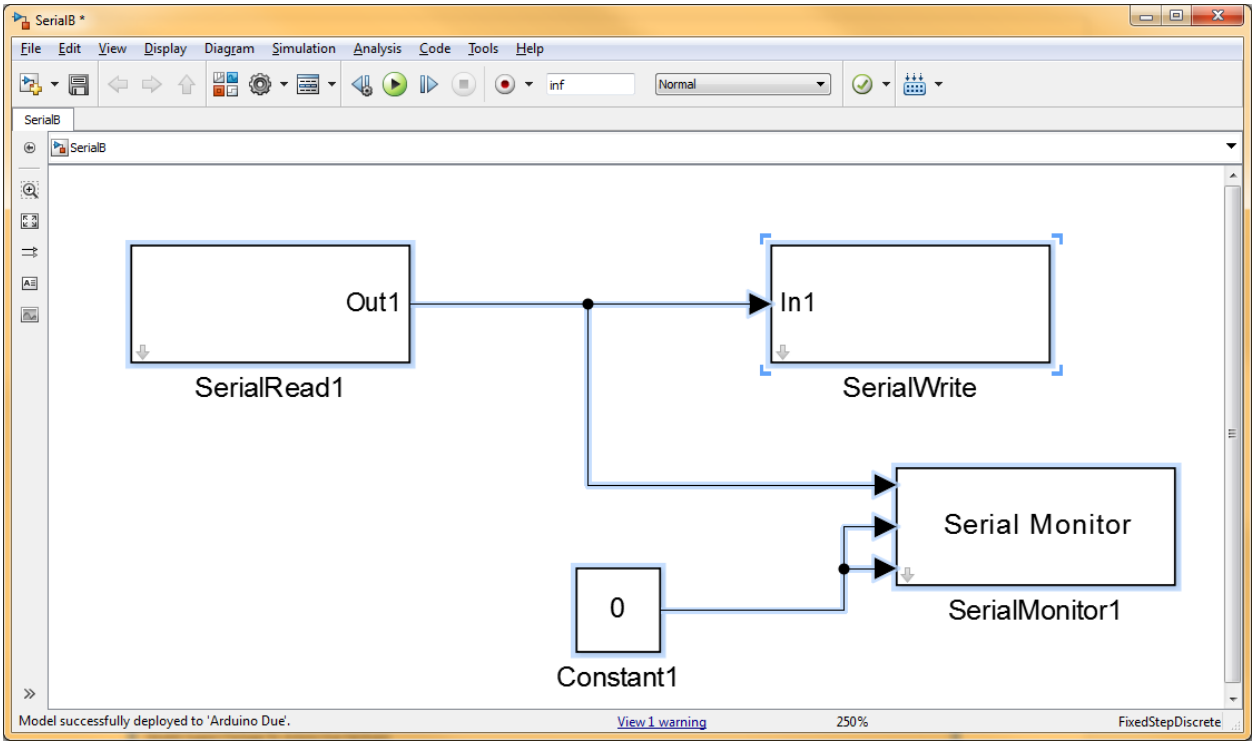


Figure 15: Square wave receive and loopback side.

You may use the Serial Oscilloscope program to monitor, log and plot the signals.

Answer the following questions in your report:

Q3) Provide plot of signals from both the boards.

Experiment 4.2

This experiment is a repetition of 4.1 with a slight change of the transmission hardware. In this case you will use the ADC and DAC peripherals of the Due board. Follow the wiring diagrams as shown below.

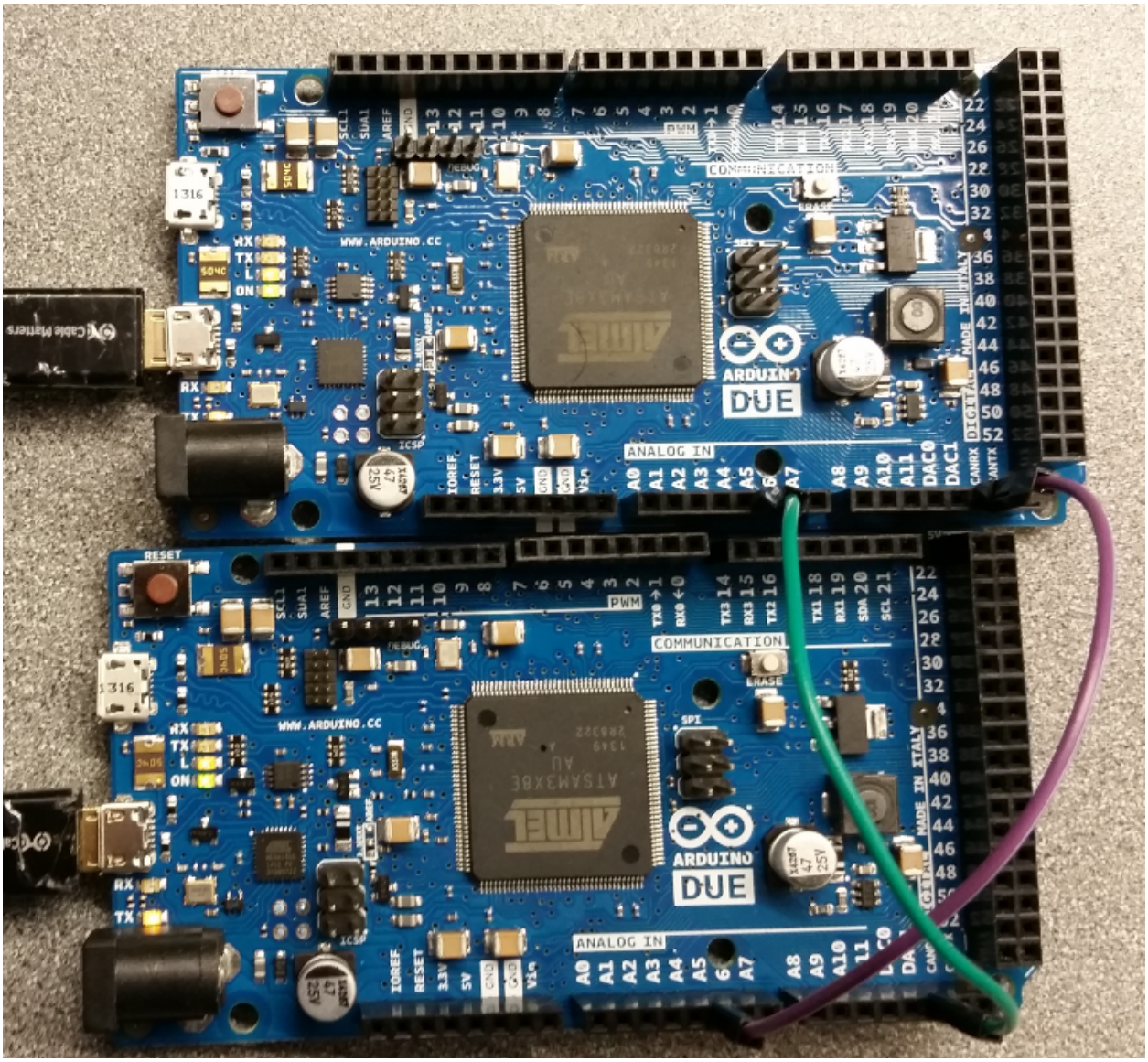


Figure 16: Wiring for Exp 4.1.

Download the two models shown below into each of the boards.

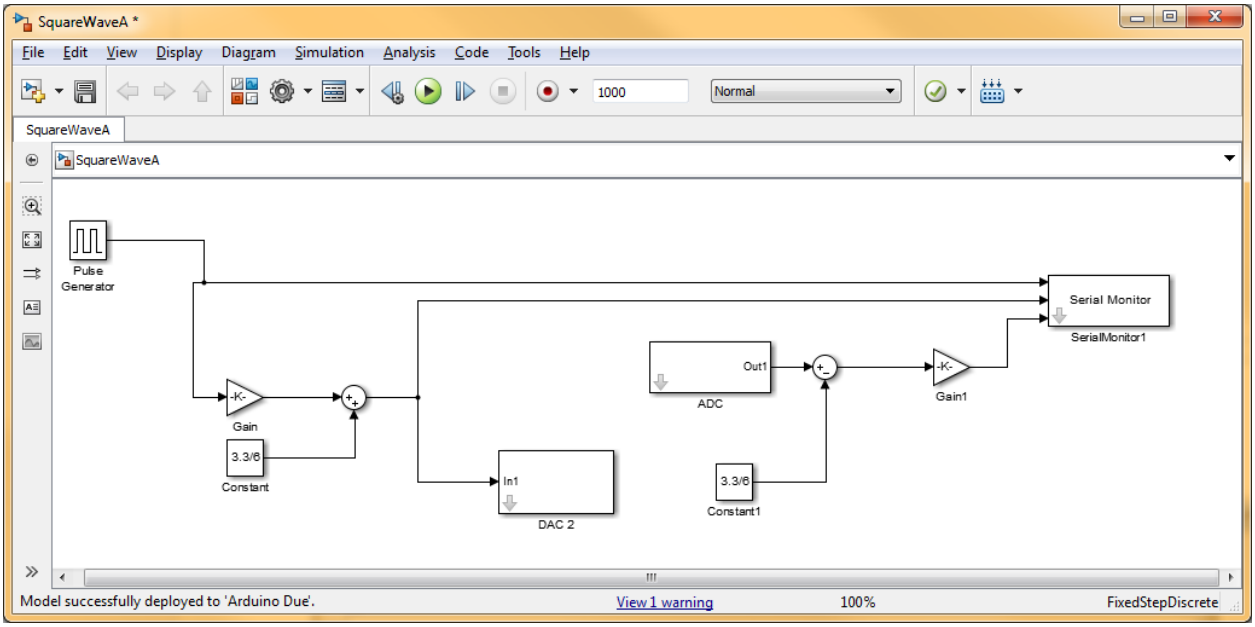


Figure 17: Square wave transmit side.

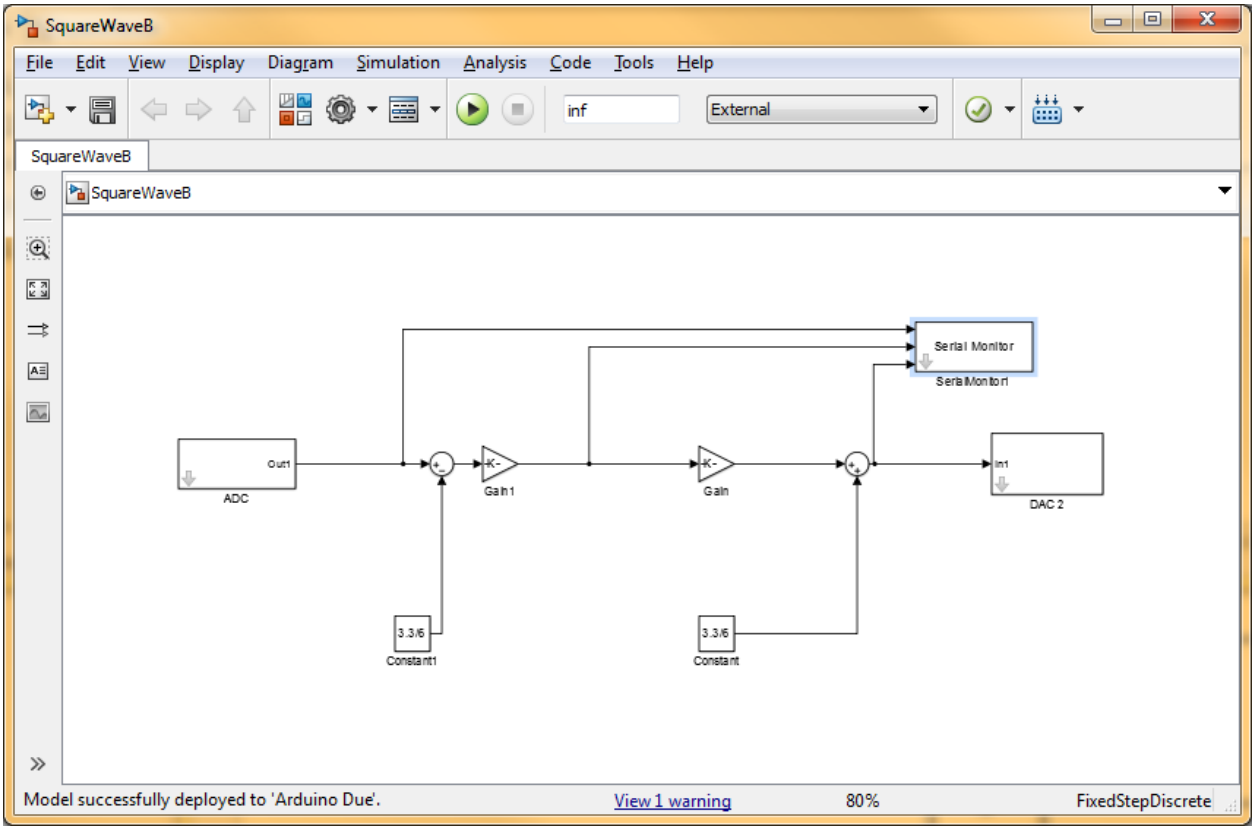


Figure 18: Square wave receive and loopback side.

Answer the following questions in your report:

Q4) Provide plot of signals from both the boards.

Q5) Explain the choice of scaling and shifting values you used.

Experiment 4.3

Now that you are familiarized with the Arduino platform and its Simulink toolbox it's time to perform a control system design and simulation. For this lab, we will use the model of a tubular chamber with a heating equipment inside of it. Suppose that heat is supplied to the tube at an adjustable rate q, e.g., by means of a heating coil. The tube loses energy (heat) to the surroundings at a rate proportional to the surface area times the difference between tube and ambient temperature. Finally the rate of change of the tube energy is proportional to the rate of change of its temperature. Thus, the macroscopic energy balance can be written as

$$mc_p \frac{dT}{dt} = q - h_s A_s (T - T_0) \tag{1}$$

Where m is the mass, c_p is the specific heat (heat capacity) of the material, A_s is the exterior surface area, h_s is the heat transfer coefficient (to the ambient) and T_0 is the ambient temperature. From this we see that the temperature dynamics follow a first-order low-pass model with respect to the supplied heat. Values for coefficients in SI units are m =100 , C_p =0.13, h_s=2 and A_s=0.0063. Ambient temperature (T_0) is 25 degrees Celsius and T initial is 450° Celsius. Please look at 0.4 section of the following document for further details `http://tsakalis.faculty.asu.edu/notes/models.pdf` .

Assume T_0 is fixed and derive transfer function between q and temperature T. Design a PI controller for bandwidth of 0.15 rad/s and phase margin 60°. Discretize the transfer function for sampling time of 0.01 seconds. Update parameters of the controller in the model files given to you for this experiment. Download the file onto the Arduino due and plot the step response of closed loop system for step size of 50 degree Celsius.

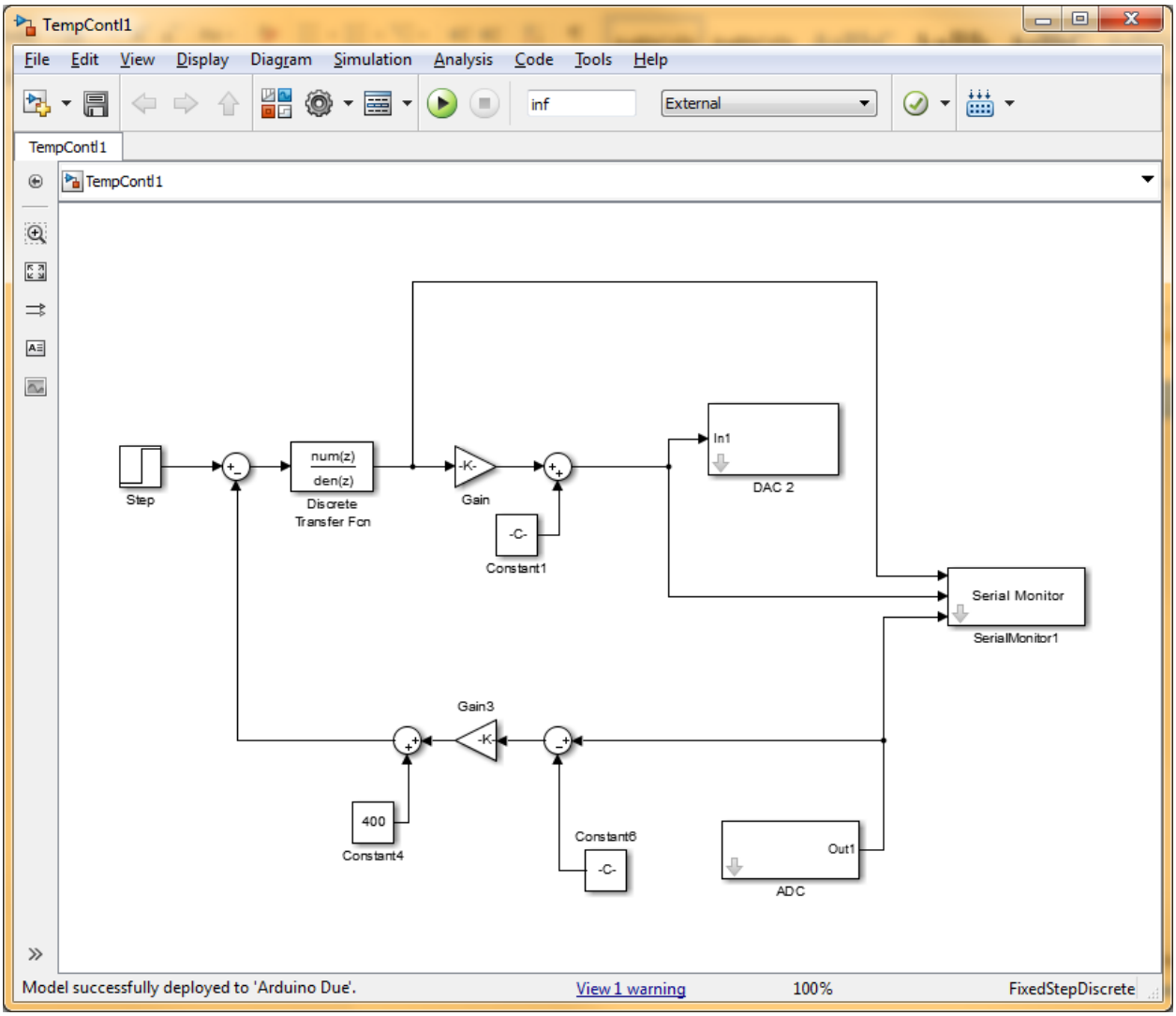


Figure 19: Temperature controller.

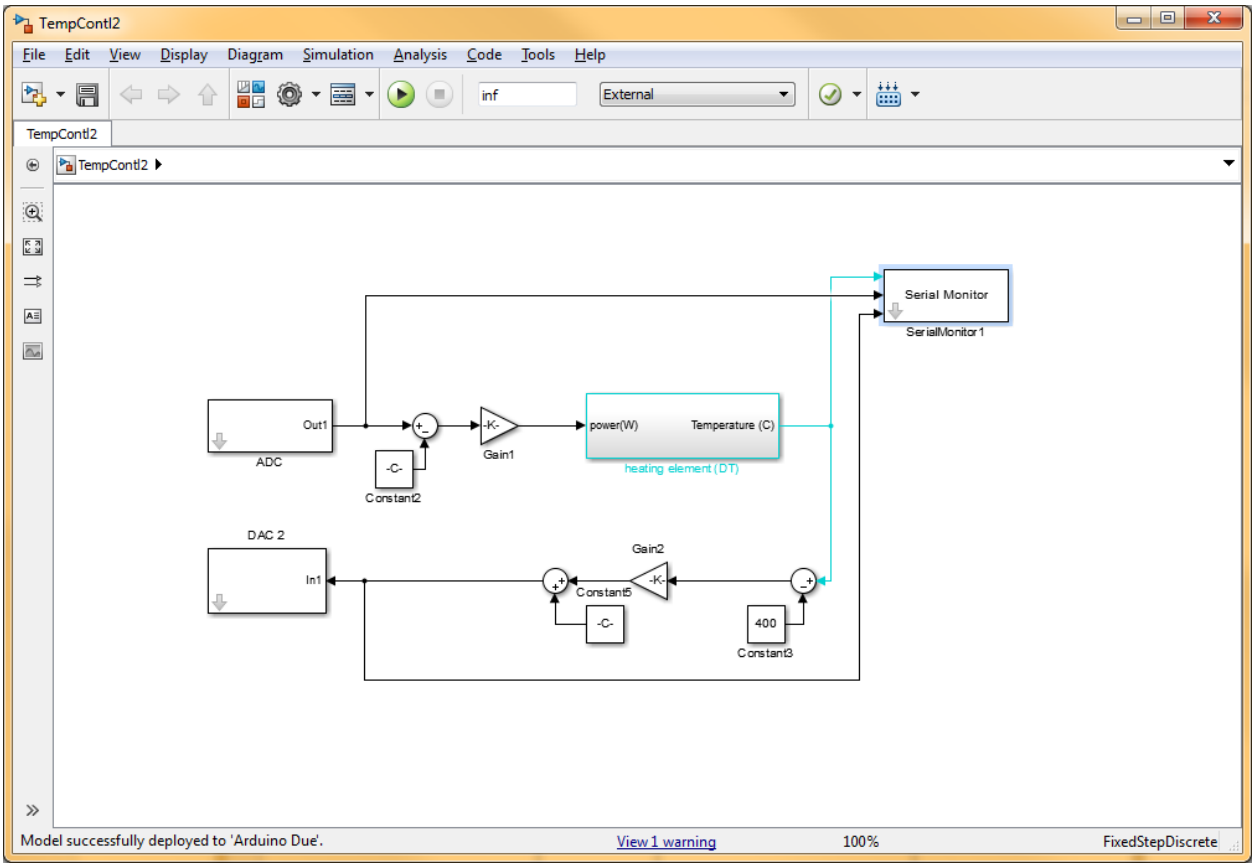


Figure 20: Heating system plant model.

Answer the following questions in your report:

Q6) Show all calculations for control design and report the values for K_p and K_i.

Q7) Show Closed loop Sensitivity and Complementary Sensitivity plots.

Q8) Show plots for the step response when a step of 50 is applied at the refernce input. Also, show the response of the system when a disturbance is applied as a step of 30 degrees.

Q9) Change the disturbance input into a sinusoid of frequency 10hz and show the output response. Do the same again for a frequency of 0.01 Hz. Comment on the difference of the two observations .

Q10) Show plots from the recorded data of the implemented models of the control input and plant output .