HANcoder advanced workshop

Version 0.5
7/12/2016

HANcoder advanced workshop

A guide to use model referencing, variants and HANcoder in order to use the advantages of model based design.

Index

HANCODER ADVANCED WORKSHOP

Contents

1 INTRODUCTION

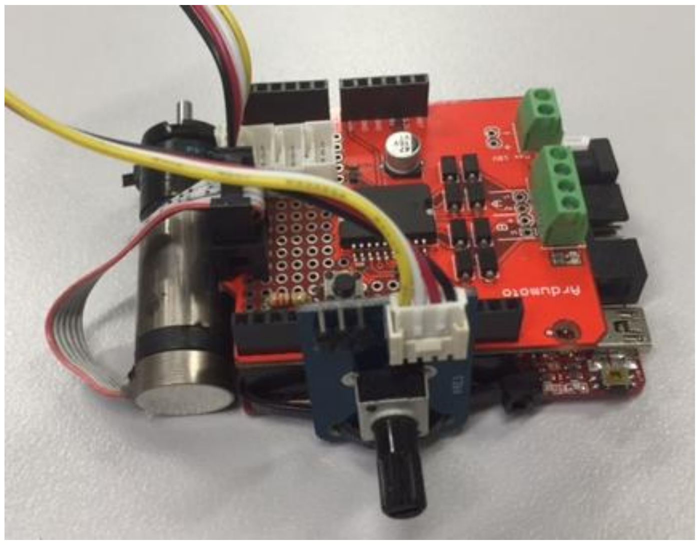
This document describes how to do simulations in Simulink of the hardware in combination with the a controller made with the HANcoder toolbox. By using this approach it will be possible to quickly test the controller in simulations. This is especially valuable when the hardware is not available of unsafe situations can occur.

The system to be controlled is a simple DC-motor. The objective is to use a model of the DC motor in Simulink in combination with a controller using the HANcoder blocks. The controller can be tuned and tested in Simulink before checking the results on the real system.

If you are not familiar with the HANcoder tools and the workflow it is advised to first read the Getting Started document and then do the 'basic' workshop.

The following subjects will be treated in the workshop: Model referencing, variant subsystems, Simulink busses, logging in HANtune, data import in Matlab and System Identification with Matlab.

2 HARDWARE

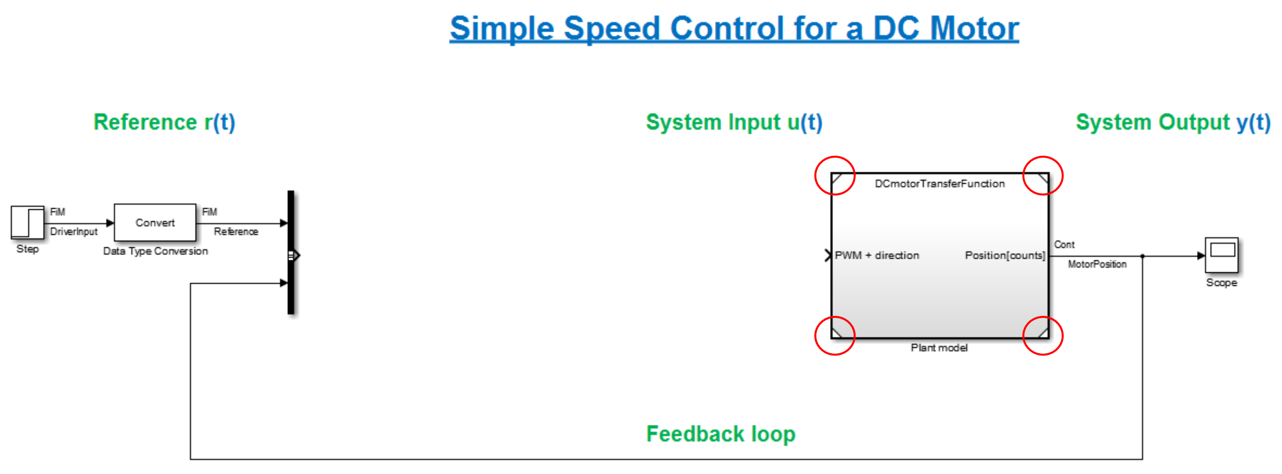


For this workshop an Olimexino will be used. On top of the Olimexino board a shield is added with a motor driver or H-bridge, the Ardumoto. A signal on pin D3 controls the duty cycle to the motor and a digital signal on pin D12 controls the direction. A small DC motor with an encoder connected to the shaft on one side and a gearbox on the other side is connected to the H-bridge. The encoder is connected to pins D6 and D7 and gives around 14000 pulses per rotation of the output shaft. Next to that there is a potentiometer attached to the Olimexino on pin A3. The potentiometer will serve as a set point for the motor speed. The further the potentiometer is turned the faster the motor should turn. This very simple system was also used in the 'basic' workshop.

3 SIMULINK MODEL

The model which will be used for the workshop is missing certain parts. The idea is that you will have to make the model work and by doing so all important steps to enable simulation and code generation will be treated. The resulting model will allow the user to check the controller performance both in Simulink as in the real world.

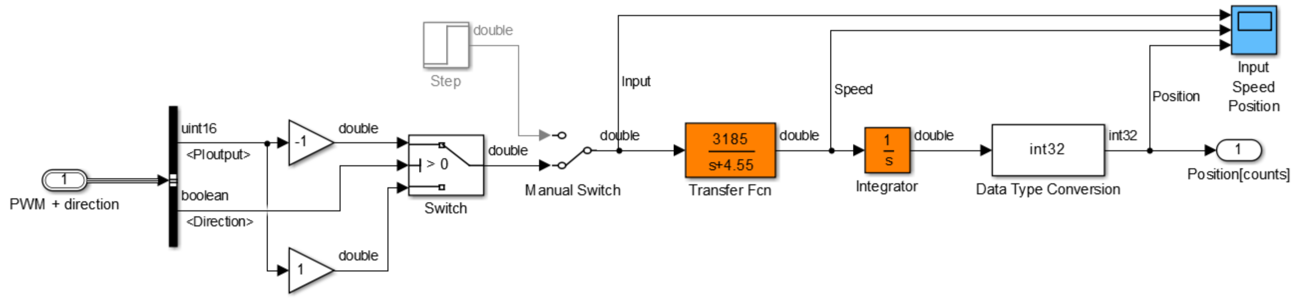
The first step is to open the prepared model by double clicking the 'Top_Model.slx' model file, located in the folder HANcoder_STM32Target\Target:



In the base model you can see a subsystem called Plant model. This is not an ordinary subsystem but a model reference. This means a model can act as a subsystem in another model. A model reference can be recognized by the triangles in the corners of the block.

In this case the Plant model subsystem is a reference to a model called DCmotorTransferFunction.slx. Please double click on this model reference to view its contents.

3.1 Plant model



The model is opened in a new screen. The heart of the model is a transfer function which represents the input to output relation. The transfer function is the relation between the PWM duty cycle and the speed of the motor (without load). An integrator is added because in the real system not the speed but the position is measured. This (black box) model of the electric motor is heavily simplified and is only valid for duty cycles under the 30%. This is to keep this workshop as simple as possible and keep the focus on the workflow.

With a manual switch you can choose the input to the transfer function: either a step or an input coming from another switch. Double clicking the **manual switch** changes the input to the transfer function to the step input. After running the model the speed and position response on the step input can be inspected using the scope. (running this model will only be possible after finishing this workshop)
The other input of the manual switch depends on the signals Ploutput and Direction, if Direction is true the input to the transfer function is Ploutput and if Direction is false it is minus Ploutput. This represents the control of the H-bridge as described in the introduction.

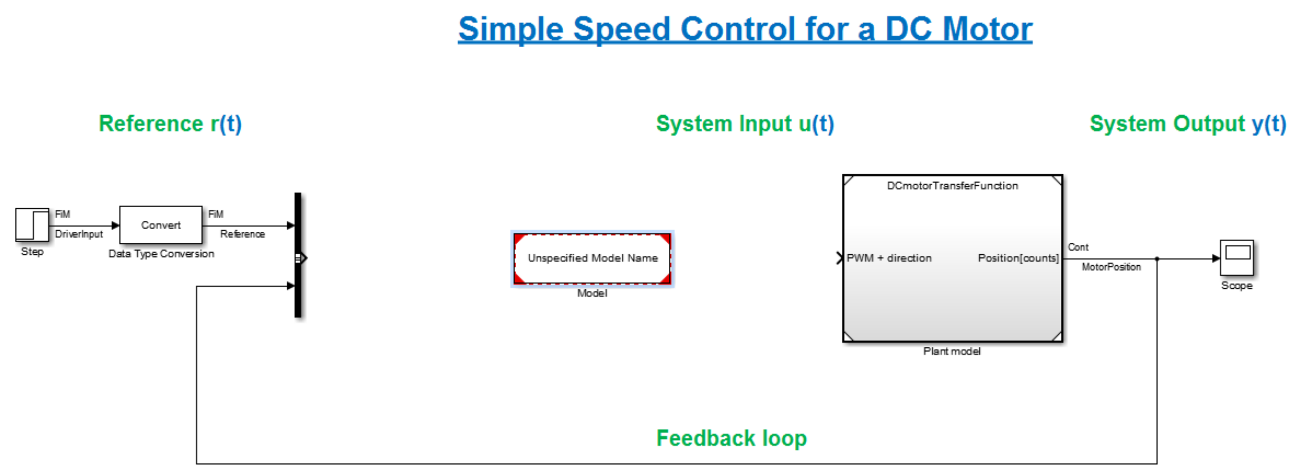
The signals Ploutput and Direction are coming from a bus selector. A bus in Simulink is equivalent to a structure in C, in different words it is a bundle of different signals. With the bus selector it is possible to extract the signals from the bus. More about buses will follow later in this workshop.

Please set the manual switch to the bottom position and close the DCmotorTransferFunction.slx.

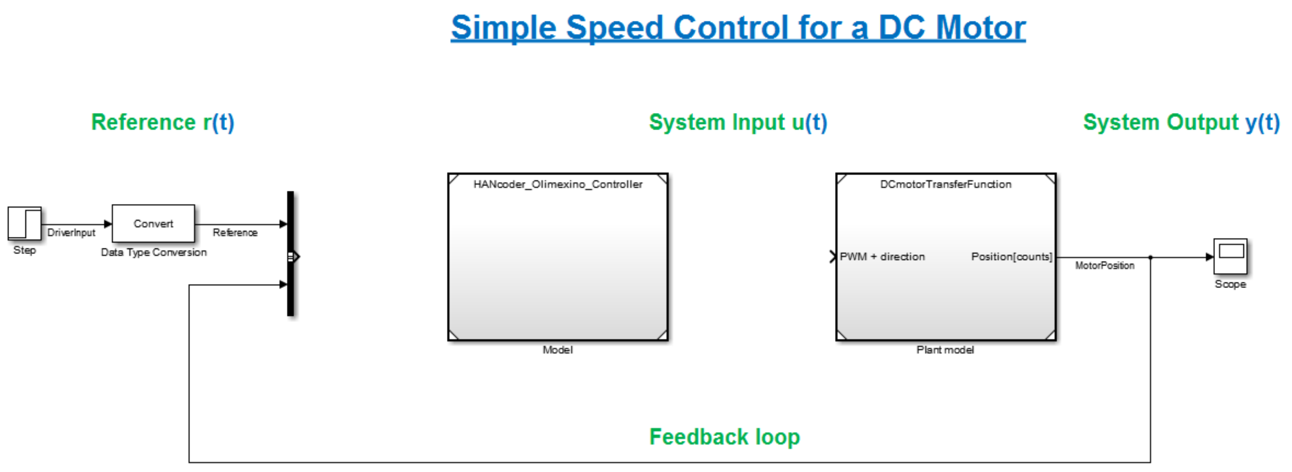
3.2 Adding a model reference

Now we will add a model reference subsystem in the base model. This referenced model will contain the control algorithm and the HANcoder blocks to automatically generate code.

Go to the library browser and go to Simulink/Ports & Subsystems. Next, choose the block 'Model' and place the block in the model left of plant model.



Double click on the newly added block to choose the model it should reference to. Using the browse button navigate to the model 'HANcoder_Olimexino_Controller.slx'. Click OK. The name of the model now appears in the model reference subsystem. Resize the block so the name becomes readable.



We now have two subsystems which are a link to other models, the HANcoder_Olimexino_Controller.slx and the DCmotorTransferFunction.slx.

Double click on the HANcoder_Olimexino_Controller subsystem to go into the controller model, this will bring you to the start screen of the HANcoder model.

3.3 Controller model

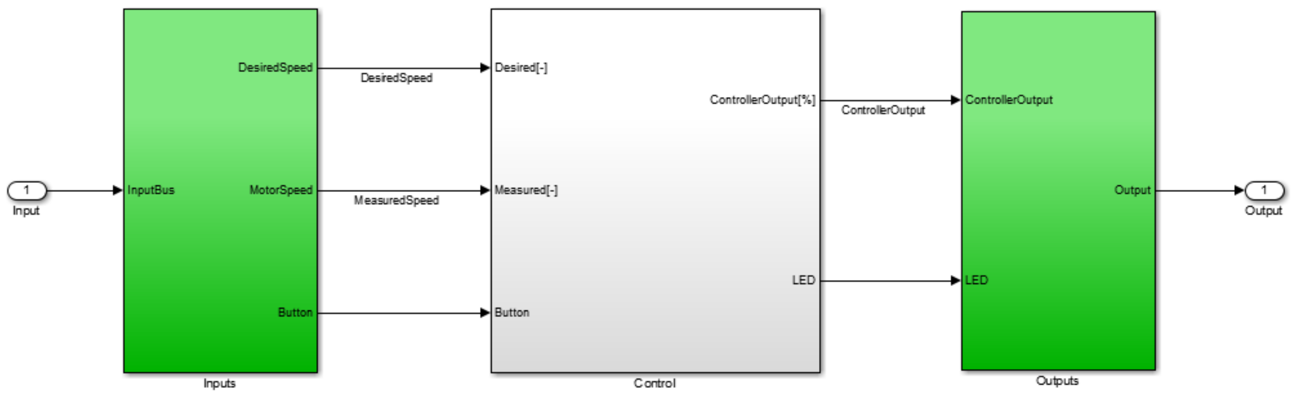
Hogeschool van Arnhem en Nijmegen
HAN University of Applied Sciences

HANcoder STM32 Target - Olimexino-STM32 algorithm

End-User License Agreement
please read before use

Read more on HANcoder

Double click on the picture of the Olimexino to enter the algorithm. The model exists out of some system settings blocks which are grey, a subsystem for the inputs, a subsystem for the actual control algorithm and lastly a subsystem for the outputs. The separation of inputs and outputs has a purpose. If there is a new version of the HANcoder blockset or if the user wants to switch from the Olimexino to the Rexroth, the control subsystem can be copied into the new model and only the input and output subsystems have to be remade.

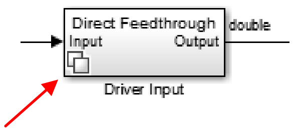


Go into the Control subsystem. There are two separate control algorithms here. The top one controls the motor speed with a PI controller. The input for the controller is the difference called error between the desired speed also called the reference, and the measured speed.

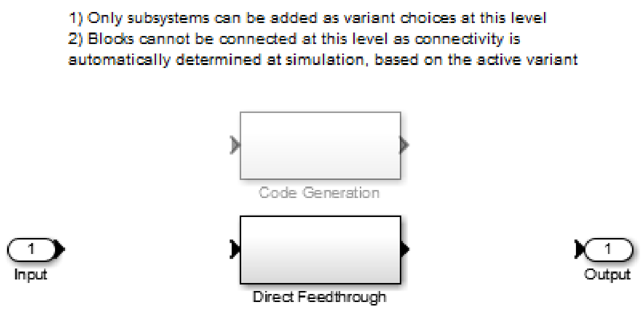
The other algorithm controls the speed of the LED by switching when the button is pressed.

Now go back and take a look in the Inputs subsystem. In the Inputs subsystem there are three inputs: the first two are the Reference, from the potentiometer, and the MotorPosition from the encoder. These inputs are the ones needed for the motor control. The third input is the button for switching between the LED frequencies.

3.4 Variant subsystems



The DesiredSpeed comes from another special kind of subsystem, a **variant subsystem** named Driver Input. A variant subsystem can be recognized by the two squares in the lower left corner. Inside this subsystem there are again two other subsystems. One is called **Code Generation** and the other one **Direct Feedthrough**. Only one of these two subsystems is active while the other one is inactive or 'commented out'. Blocks that are commented out are faded out / lighter.

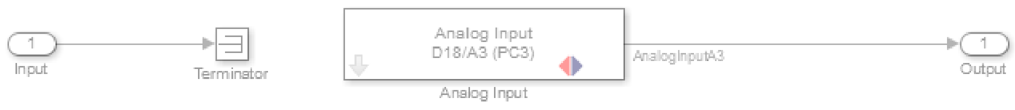

1) Only subsystems can be added as variant choices at this level
2) Blocks cannot be connected at this level as connectivity is automatically determined at simulation, based on the active variant
Code Generation
1
Input
Direct Feedthrough
1
Output

3-1 Inside the Driver Input variant subsystem

The currently *active variant* is **Direct Feedthrough**. At this moment it can be seen that the **Direct Feedthrough** is active, this is also displayed on the parent block **Driver Input**.

What is remarkable is that the input and output blocks don't seem to be connected to the subsystems. The connections between in- and outputs will be automatically resolved by Simulink.

3.1.4 The code generation variant

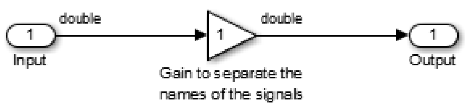


3-2 The code generation variant

Inside the **Code Generation** subsystem the above blocks can be found. The Code Generation subsystem will be used when generating code for the Olimexino. The HANcoder block for reading the voltage of the potentiometer is located in this block. The **input** to this subsystem comes from the Top Model and is only useful during simulation, it can be neglected when generating code. Terminating the signal is used to prevent warnings from Simulink about unconnected blocks.

3.2.4 The Direct Feedthrough variant

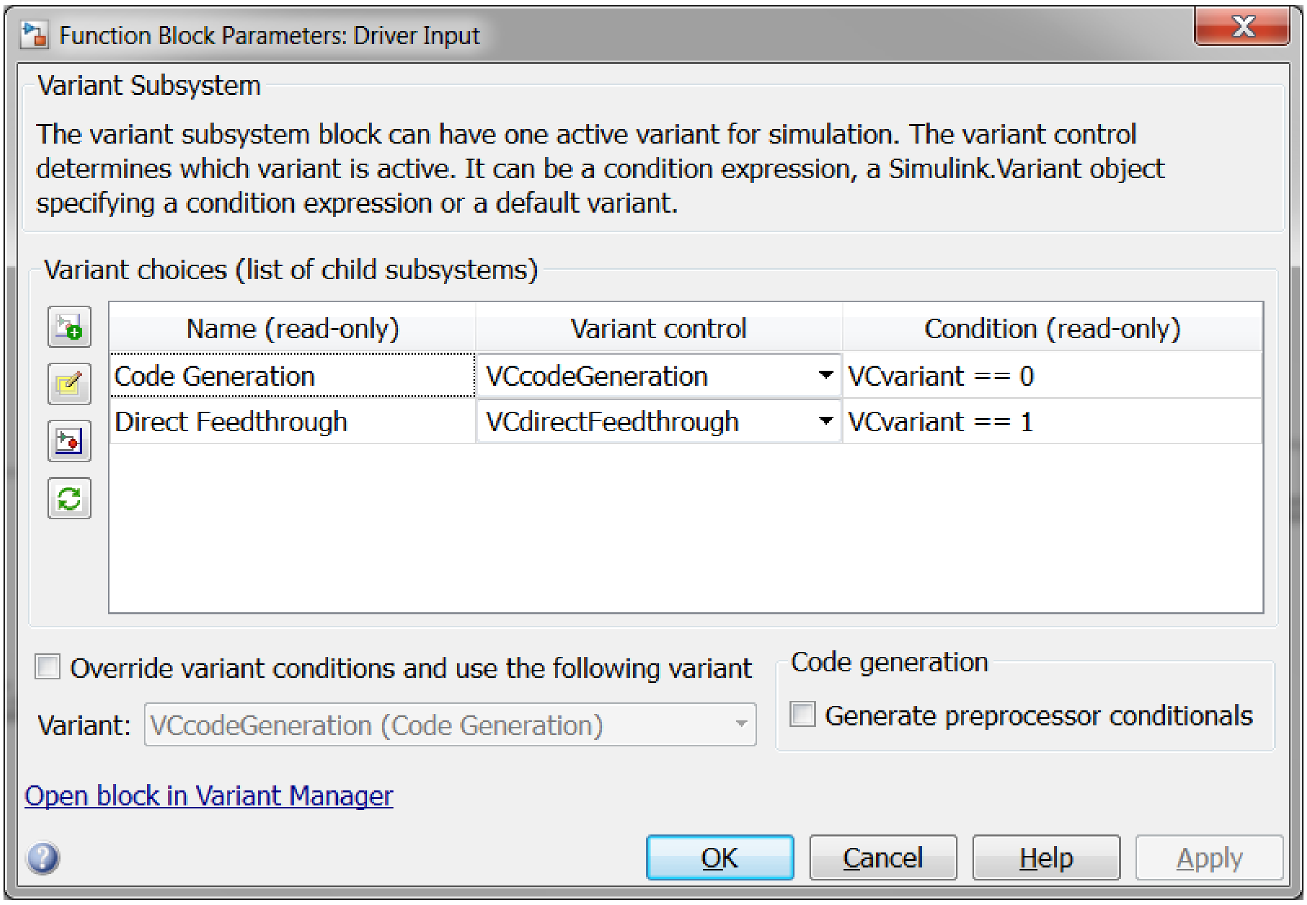
The other subsystem is the **Direct Feedthrough** variant. In this subsystem the output is directly connected to the input. When running a simulation this means that the signal from the Top Model is directly routed into the controller. The signal from the Top Model should have the same characteristics as the value from the measurement in the Code



Generation block.

3.3.4 Variant control

Simulink uses Simulink.Variant objects to determine which subsystem is active, these are defined in the workspace. Please go back to the screen where you can see the Variant Subsystem **Driver Input**. Now right click on the Driver Input variant subsystem and choose Block Parameters (Subsystem). The following screen will appear:



Function Block Parameters: Driver Input

Variant Subsystem

The variant subsystem block can have one active variant for simulation. The variant control determines which variant is active. It can be a condition expression, a Simulink.Variant object specifying a condition expression or a default variant.

Variant choices (list of child subsystems)

Name (read-only)	Variant control	Condition (read-only)
Code Generation	VCcodeGeneration	VCvariant == 0
Direct Feedthrough	VCdirectFeedthrough	VCvariant == 1

☐ Override variant conditions and use the following variant

Variant: VCcodeGeneration (Code Generation)

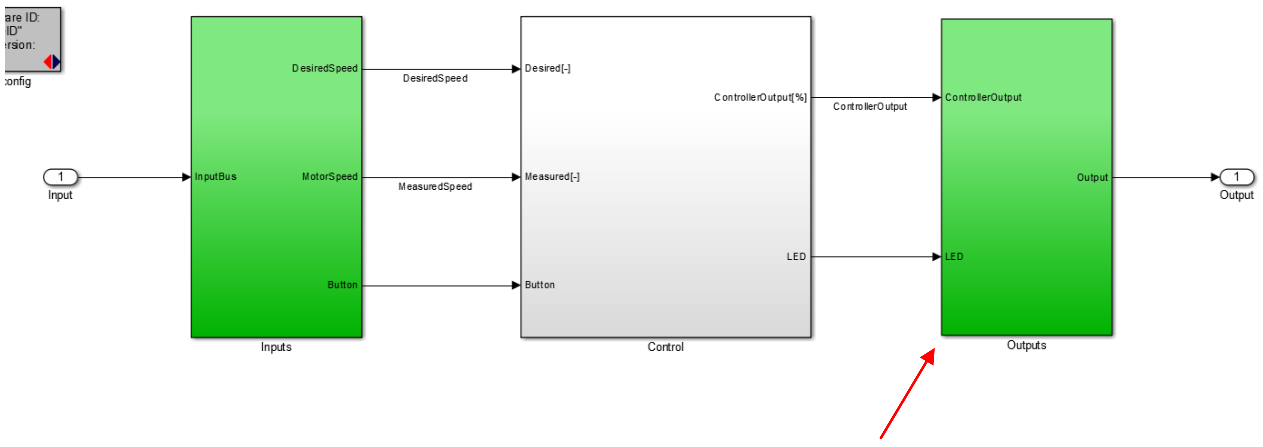
Code generation

☐ Generate preprocessor conditionals

Open block in Variant Manager

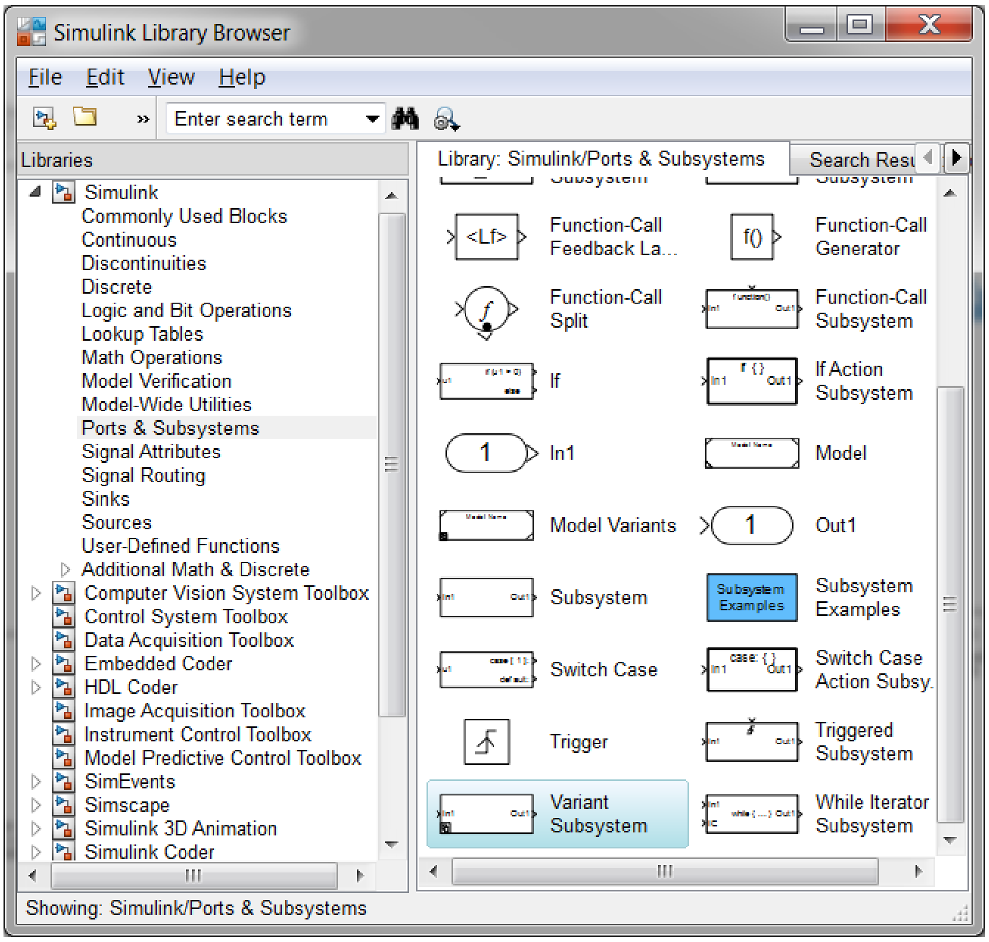
Here the variant control can be defined. Each subsystem is controlled with a Simulink.Variant object, specified in the second column. When the condition of these Variant controls are met the Variant Subsystem is activated. Press cancel to close this screen.

Next we will set up a variant control for one of the outputs. Please navigate to the **Outputs** subsystem.



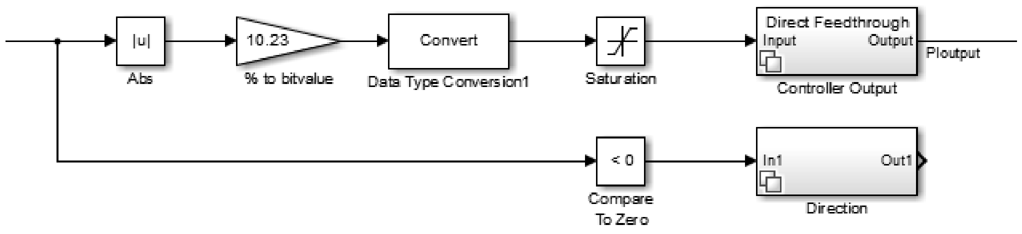
3.4.4 Creating a variant subsystem

As mentioned before the direction can be controlled by toggling pin D12. The current model can only control the speed of the motor in one direction. If the motor should also be able to turn backwards it makes sense to implement this in the controller model. For this a new variant subsystem will be made. Go to the library browser and from Simulink/Ports & Subsystems choose the **Variant Subsystem** block.



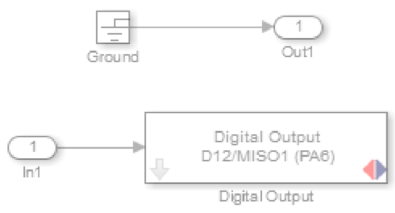
Put the block under the Controller Output subsystem and name it **Direction.**

The direction pin should toggle when the output of the controller is less than 0. So from the library Simulink →Logic and Bit Operations take the **Compare To Zero** block. Connect the input to the block with the controller output, before the absolute block. Double click the block and select the smaller than sign, <. The output of the Compare To Zero block serves as the input to the new Variant Subsystem 'Direction'.



3-3 New Variant subsystem added

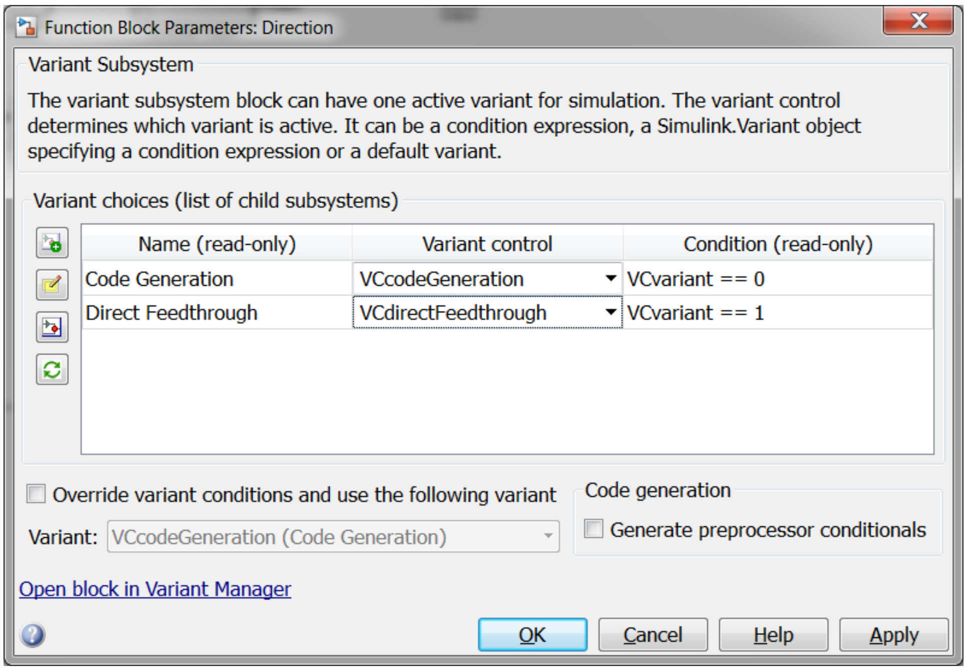
Now go into the **Variant Subsystem,** Direction, and add two 'normal' subsystems from the library Simulink → Ports & Subsystems. Name the first one Code Generation and the second one Direct Feedthrough.



Open the Code Generation subsystem and add a Digital Output block from the library HANcoder STM32 Target→Olimexino STM32 →Digital Outputs. Connect the input to the Digital Output block. Set the Digital Output to the correct pin by double clicking it and choosing pin D12 from the pull-down menu. Take a Ground block from the Library Simulink→Sources and connect it to the output block. This is again to prevent Simulink warnings during code generation. The Code Generation subsystem is now ready.

The other subsystem, Direct Feedthrough, is already done since the standard setup in Simulink is to connect the input directly to the output, just as we want in our model.

Now the variant subsystem still needs to know under which conditions to use which variant. Go up to the level where you can see the base block for the variant subsystem, **Direction**. Right-click on the block and select Block Parameters (Subsystem). The same screen as before appears. Now type the correct Simulink.Variant object at the subsystem; VCcodeGeneration for the Code Generation and VCdirectFeedthrough for the direct feedthrough. If the Variant control was recognized the condition will be automatically updated. Click OK when done.


Function Block Parameters: Direction

Variant Subsystem

The variant subsystem block can have one active variant for simulation. The variant control determines which variant is active. It can be a condition expression, a Simulink.Variant object specifying a condition expression or a default variant.

Variant choices (list of child subsystems)

Name (read-only)	Variant control	Condition (read-only)
Code Generation	VCcodeGeneration	VCvariant == 0
Direct Feedthrough	VCdirectFeedthrough	VCvariant == 1

☐ Override variant conditions and use the following variant

Variant: VCcodeGeneration (Code Generation)

Code generation

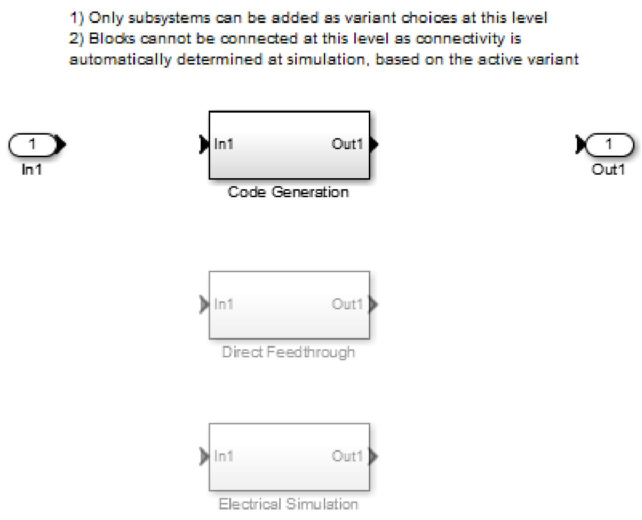
☐ Generate preprocessor conditionals

Open block in Variant Manager

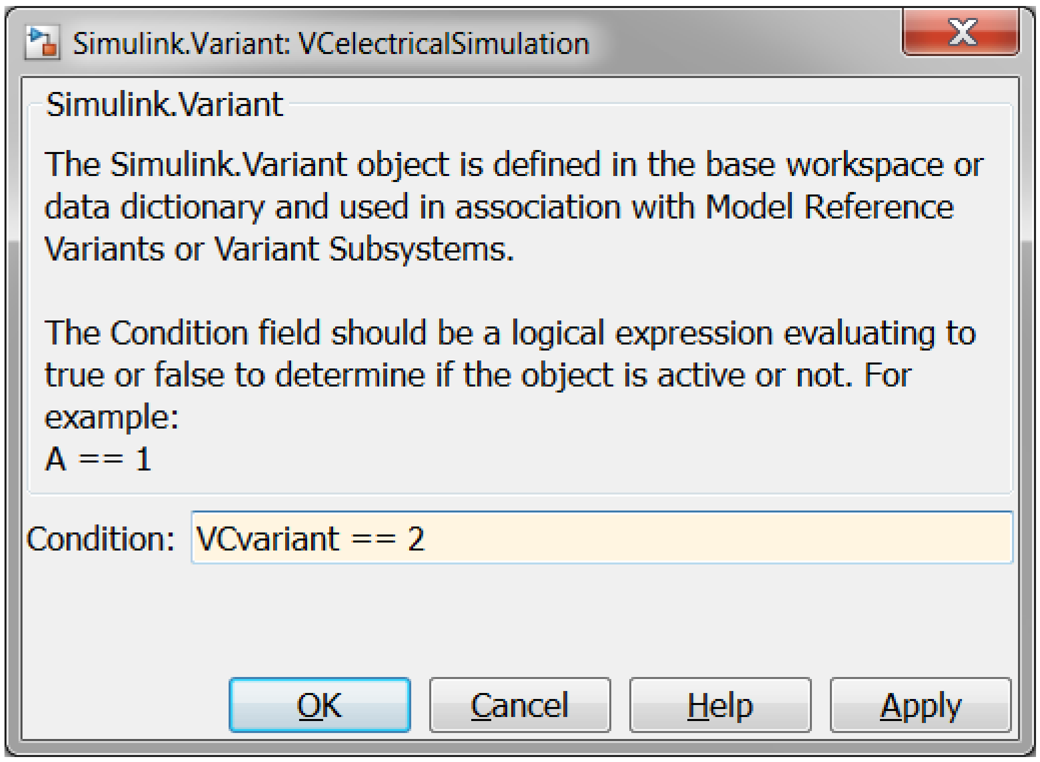
To test if it all works correctly change the condition for the variants by typing 'VCvariant = 0' in the command window. The block should update its text and when you enter the block the Code Generation subsystem should be activated. (the text will be updated when the model window is active)

3.5.4 Adding an extra variant

When you have a (white box) model of a physical system and the outputs of this plant model are in physical units like Nm or Watts you can choose to add a model of the sensor/actuator in another variant. Let's add an extra variant to the model as an exercise. Go into the variant subsystem 'Direction' and add another subsystem, give it the name Electrical Simulation. Leave the contents unchanged.


1) Only subsystems can be added as variant choices at this level
2) Blocks cannot be connected at this level as connectivity is automatically determined at simulation, based on the active variant
1
In1
In1
Out1
Code Generation
1
Out1
In1
Out1
Direct Feedthrough
In1
Out1
Electrical Simulation

The condition when this variant should be used is not defined yet, this is done by creating a Simulink.Variant object by typing: 'VCelectricalSimulation = Simulink.Variant;' in the command window. Now the condition can be set by double clicking on the newly created Simulink Variant object in the workspace:


Simulink.Variant: VCelectricalSimulation

Simulink.Variant

The Simulink.Variant object is defined in the base workspace or data dictionary and used in association with Model Reference Variants or Variant Subsystems.

The Condition field should be a logical expression evaluating to true or false to determine if the object is active or not. For example:
A == 1

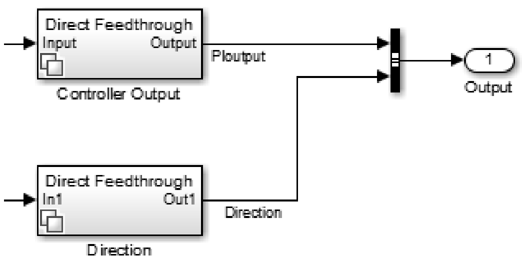
Condition: VCvariant == 2

Type 'VCvariant == 2' in the window and click OK. Now go back to the model and open the Block Parameters (Subsystem) window again. (right click on the Variant Subsystem 'Direction'). In the second column behind Electrical Simulation type the new Simulink Variant object: VCelectricalSimulation. The new condition should automatically be shown. Press OK to close the window.
Test the new variant by setting VCvariant to 2 (type VCvariant =2; in the command window).

Set the VCvariant back to 1 to enable the direct feedthrough. (Note: the actual Actuator model was not made)

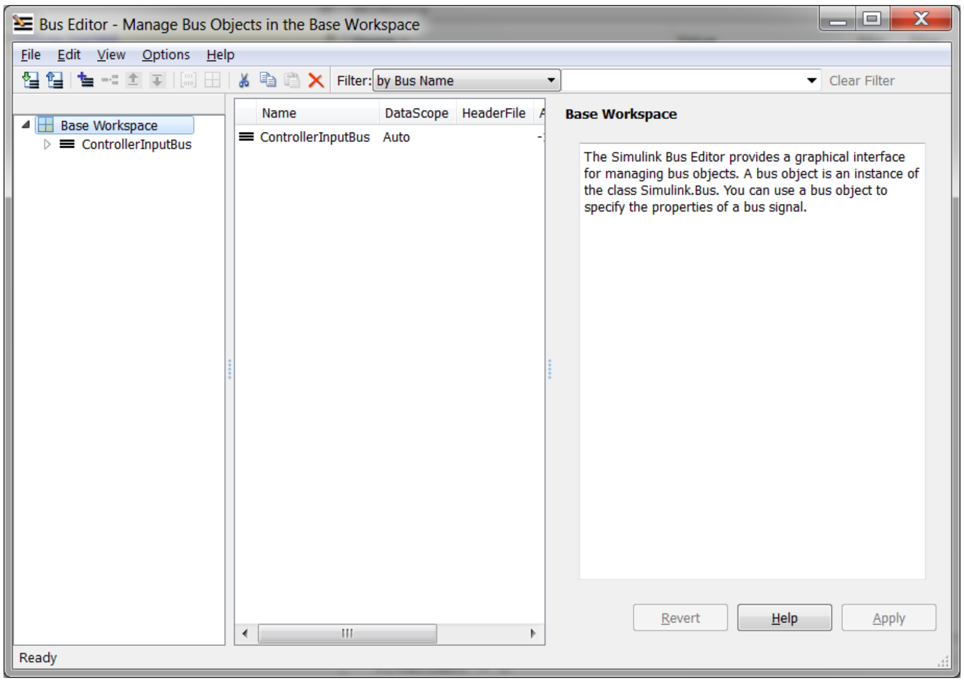
3.5 Creating a bus

The Direction variant was successfully created, next the output from the Direction block needs to be routed. For this we will use a Simulink bus.



In this model it isn't really necessary because not many signals are used but in larger, more complex models, buses will really help with keeping the model readable and maintainable. To create a bus use the Bus Creator from the Simulink→Signal Routing library. Disconnect the Ploutput signal from the Output block and connect it to the first input of the bus creator. Connect the output of the Direction subsystem to the second input of the bus creator. Give this signal a name by double clicking it and typing 'Direction'. The line will turn bleu when the signal is selected. The output of the bus creator can now be connected to the Output block.

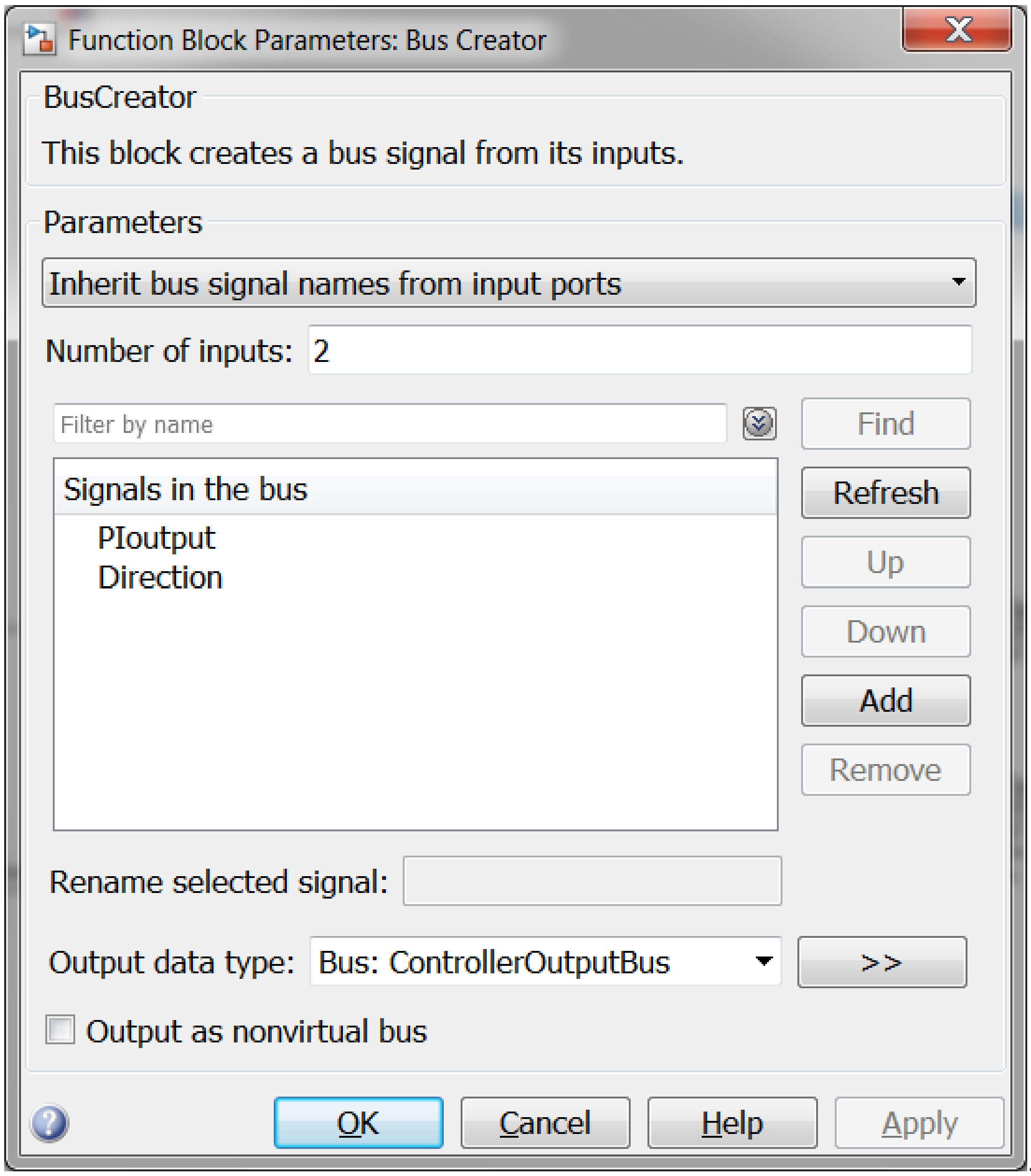
The bus that has been created is only known in the controller model. This bus will be used as an output to the Top Model. In the Top Model it is not known which signals are in the bus and Simulink will not know how to deal with this bus. To solve this a Simulink.Bus can be created. This is a definition or description of the bus so that all blocks and models which use this bus will know exactly what the bus contains. To create this bus a tool can be used from Matlab, start the tool by typing 'buseditor' in the command window. The Bus Editor is now started.



The Simulink Bus Editor provides a graphical interface for managing bus objects. A bus object is an instance of the class Simulink.Bus. You can use a bus object to specify the properties of a bus signal.

To create a new bus in the Base Workspace click the add bus button, , in the toolbar. Now in the right side of the window give the bus the name: ControllerOutputBus. Click Apply to confirm. Now the signals can be added to the bus, in a bus these are called bus elements. Click the add element button, . Give the element the name Ploutput, change the data type to uint16 and click apply. Now add a Direction element to the same bus and set the data type to Boolean. Click apply before exiting the bus editor.

To use the defined bus go to the bus creator in Simulink and double click it. Select the ControllerOutputBus from the drop down menu as Output data type. See below:


Function Block Parameters: Bus Creator

BusCreator

This block creates a bus signal from its inputs.

Parameters

Inherit bus signal names from input ports

Number of inputs: 2

Signals in the bus
- PIoutput
- Direction

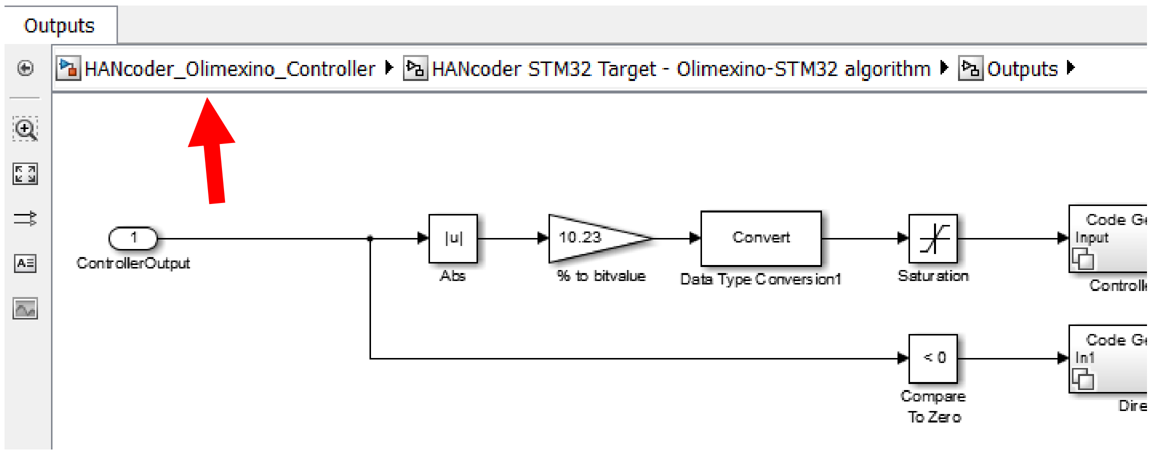
Rename selected signal:

Output data type: Bus: ControllerOutputBus

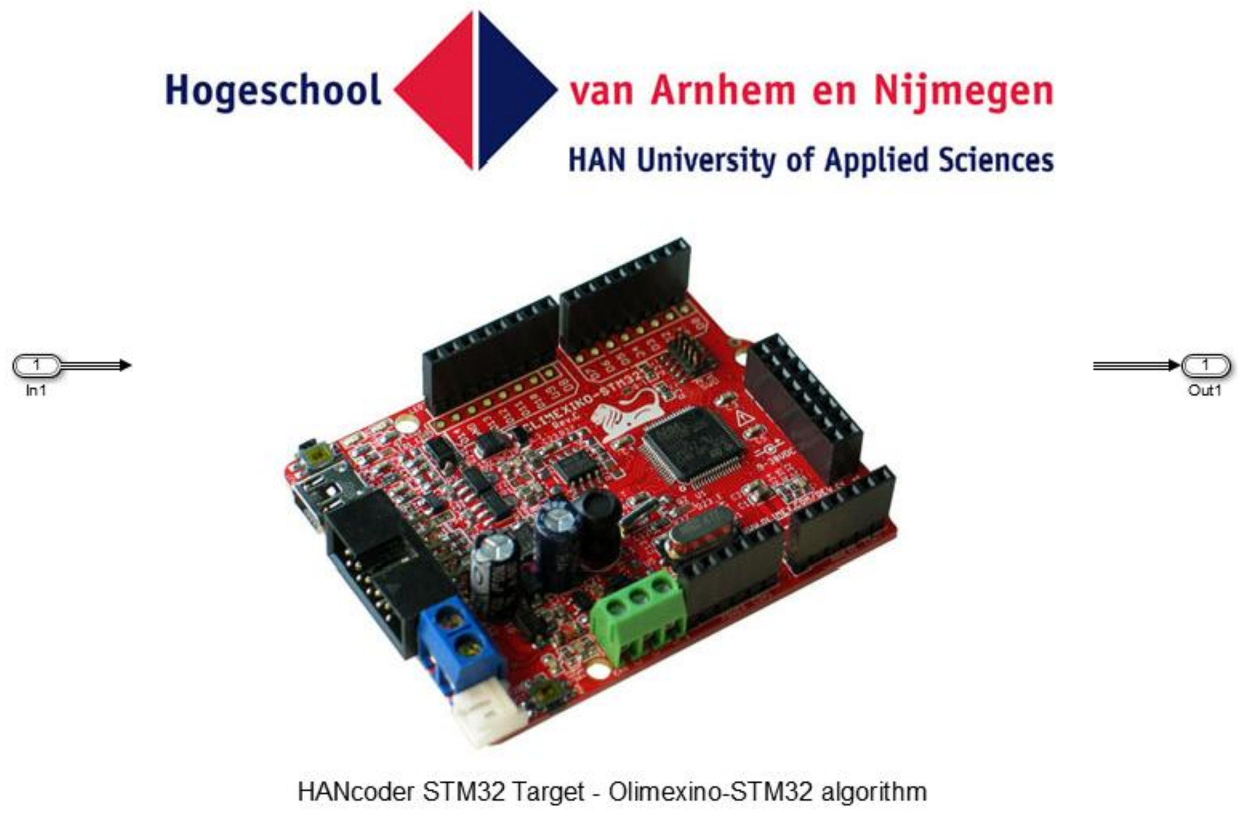
Output as nonvirtual bus: ☐

3.6 Defining inputs and outputs

To enable interaction between the controller model and the top model some in- and outputs are needed. These need to be defined at the highest level of the controller model, this is the level where the picture of the Olimexino is visible. Go there by clicking the ⇧ or by clicking HANcoder_Olimexino_Controller in the bar above the model.



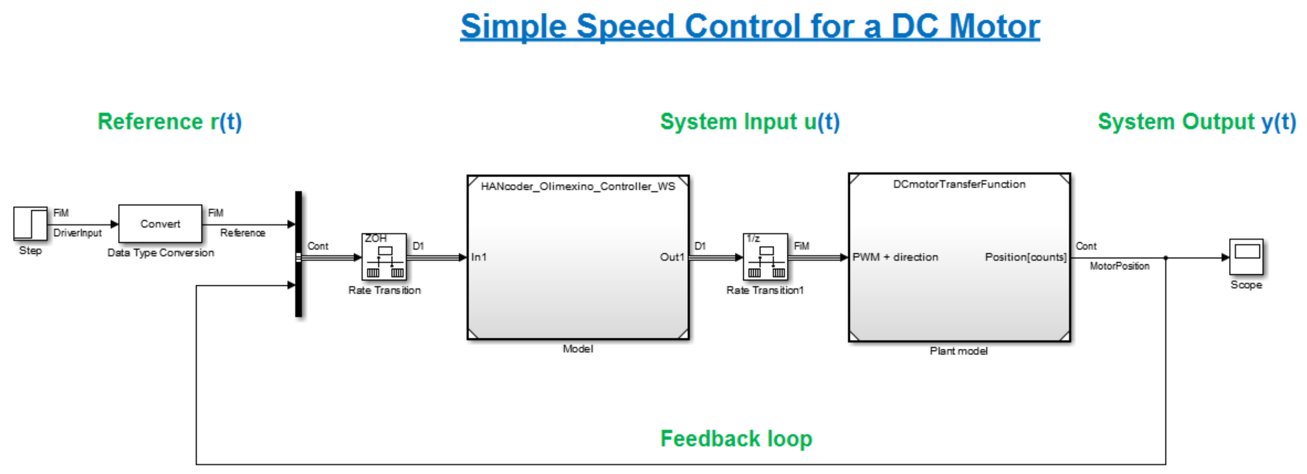
Add and connect an input on the left side of the model and an output at the right side. The input block can be found at the Simulink->Sources library and the output at the Simulink->Sinks library. Because these in- and outputs are connected to the top model the data types must be defined. Double click both blocks and change the data type in the Signal Attributes tab to ControllerInputBus for the input and ControllerOutputBus for the output.


Hogeschool van Arnhem en Nijmegen
HAN University of Applied Sciences
1
In1
1
Out1
HANcoder STM32 Target - Olimexino-STM32 algorithm

Run the model to check if there are no errors. The lines to the input and output now get thicker, this indicates that this line represents a bus.
Save the model and go back to the top model.

3.7 Connecting the controller model

In the Top Model the input and output have now been added to the model reference of the controller model. These can't be directly connected to the reference input and the plant model because the sample times are different. The controller works with time steps, it samples its inputs and updates its outputs at a certain frequency. The controller is a digital or discrete system. The plant model and reference represent the real world and they are therefore simulated using continuous time, it is an analog or continuous system. To solve the difference use the Rate Transition block from the Simulink→Signal Attributes library. Place one of these blocks at the input and one at the output of the controller model. Now connect all lines and run the model. Simulink now automatically determines how to deal with each rate transfer.



Note: Getting a default HANcoder model to operate as referenced model requires changing 3 settings. These are already adapted in the prepared model for this workshop, to see which settings are needed to be changed please refer to Appendix 1: Model settings for model reference

3.8 Simulating and tuning

The model is now ready for simulation. Please press the Run button in the Top Model. Check to see if the model ran correctly by checking the Scope showing the MotorPosition. If the model didn't seem to run check if VCvariant is set to 1.

Now try to find values for the proportional and integral parts of the PI controller that result in a stable and fast step response without too much overshoot. Pay attention that the Kp and Ki parameters are Simulink.Parameters so you can't simply type Kp = 0.4; Instead to change the value either use the dialog that appears when you double-click the name in the workspace or type: Kp.Value = 1;

The results of the simulation can be seen on the Scope located inside the **Plant model**. Note that the speed is indicated in pulses per second and because the encoder gives around 14000 pulses per revolution the values for the speed get very high.

Once the values for Kp and Ki are found it is time to move on to test the controller in the real world.

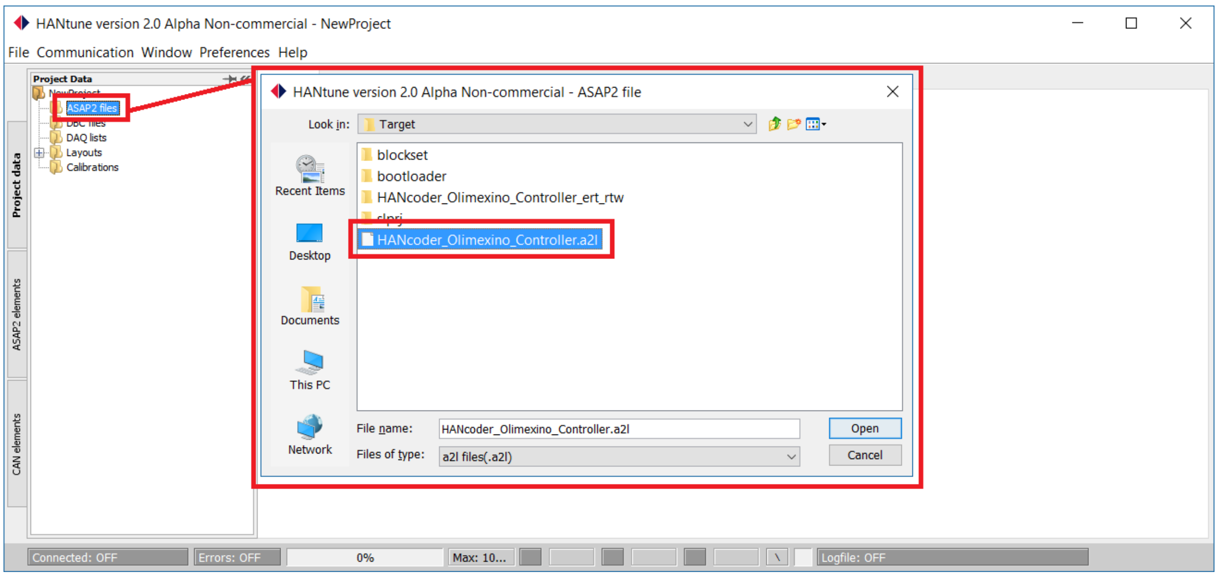
4 TESTING THE COMPLETED CONTROLLER

To generate code for the Olimexino first the variant subsystems should be set to Code Generation. Type: VCvariant = 0; in the Matlab command window. This will switch the variant subsystems to Code Generation. Now the controller model can be build by pressing the incremental build button, , or by pressing Ctrl+B in the HANcoder_Olimexino_Controller window. When the build process is successfully completed the program can be flashed onto the Olimexino in the usual way using MicroBoot. (MicroBoot will be started automatically)

Test if the program works by turning the potentiometer. If the motor starts oscillation turn the potentiometer back to the zero position and reset the controller. These oscillations occur when an instable controller is used.

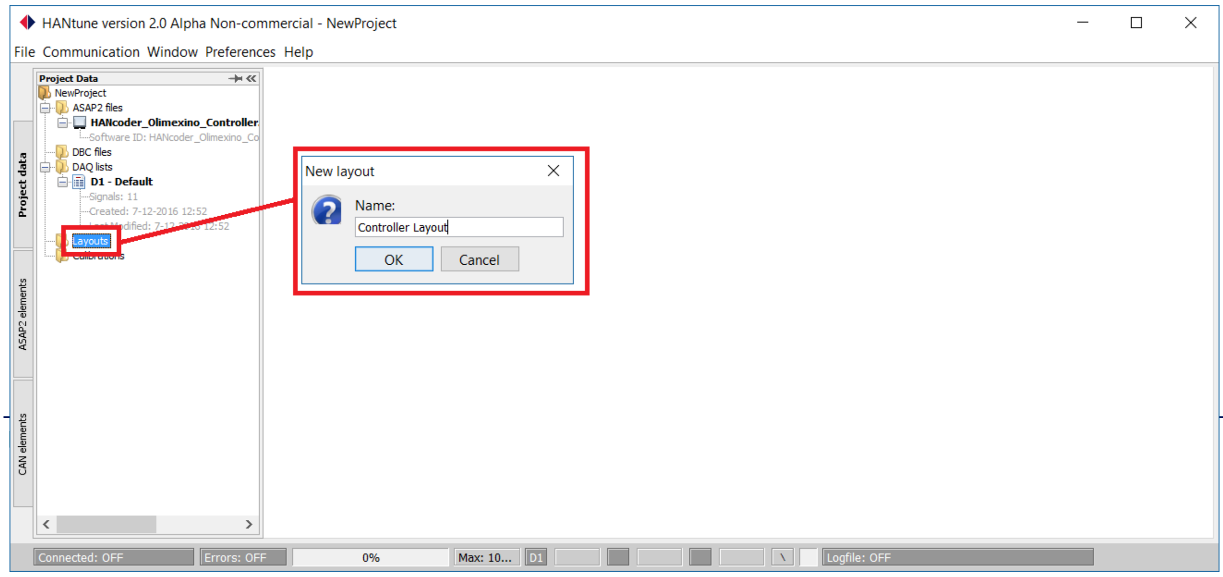
4.1 Creating a HANtune layout

To see what is happening inside the controller HANtune will be used. To connect to the Olimexino with HANtune, the ASAP2 file (.a2l) has to be loaded in HANtune. To do this, start-up HANtune and right click on **ASAP2 files**. Click **add ASAP2 file** and search for the .a2l file in the target directory of HANcoder.



File name: HANcoder_Olimexino_Controller.a2l
Files of type: a2l files(.a2l)

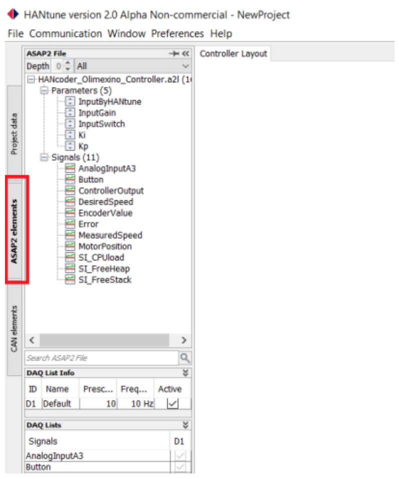
After the ASAP2 file is loaded, it appears in the ASAP2 folder. Double click **HANcoder_Olimexino_Controller.a2l** to load all signals and parameters from the a2l file. When it is loaded, the name will turn bold. After the program is loaded, a layout must be added. To add a layout, right click **layouts** and click on **New Layout**. HANtune will ask to name the Layout, name the Layout **Controller Layout** and click on OK.



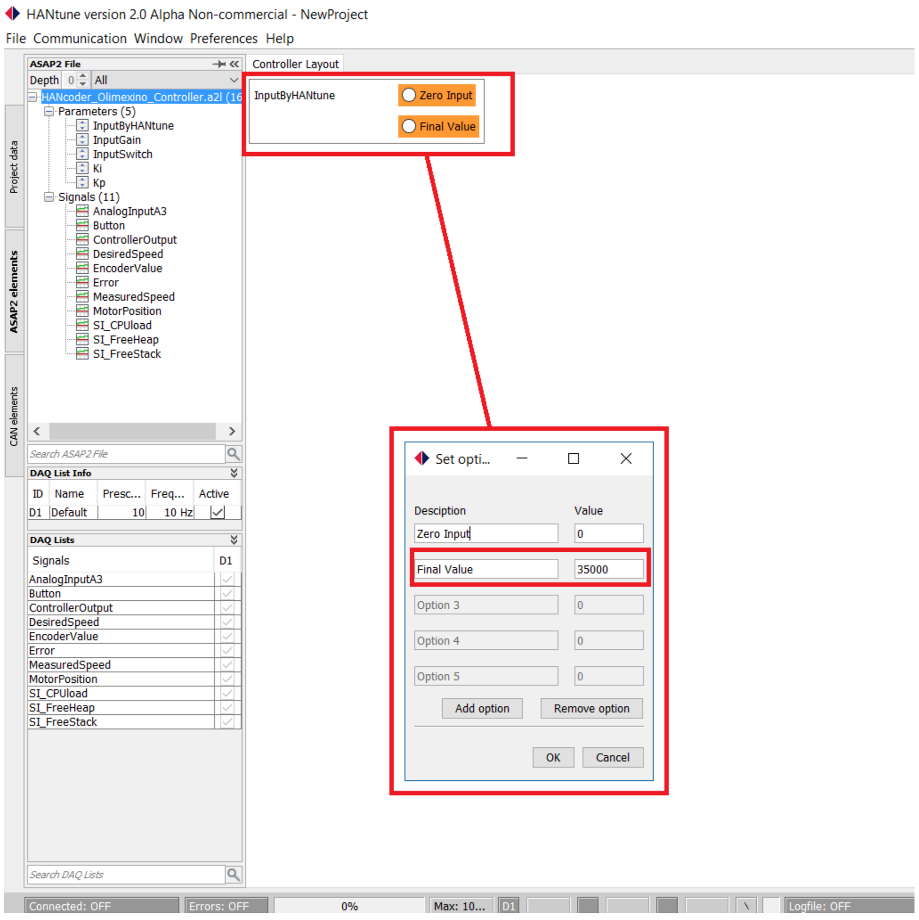
Name: Controller Layout

The layout is now added to the layouts folder. To load the layout, double click on **Controller Layout**. When the layout is loaded, the name will turn bold.

To add editors and viewers to the layout, click on the ASAP2 elements tab on the left of the window. In here the signals and parameters are shown.



Add the **InputByHANtune** parameter by dragging it into the layout. HANtune will ask what kind of editor is desired. Click on **RadioButtonEditor**. When this editor is added to the dashboard, it first has to be given the right settings. To do this right-click on **option 1** in the editor. Go to **Modify options** and name option 1: Zero Input with value is 0. Name option 2: Final Value. This value must be the same as the **step block** in the top model (35000) so the response from the simulation can be compared with that from the real world.



Desciption	Value
Zero Input	0
Final Value	35000
Option 3	0
Option 4	0
Option 5	0

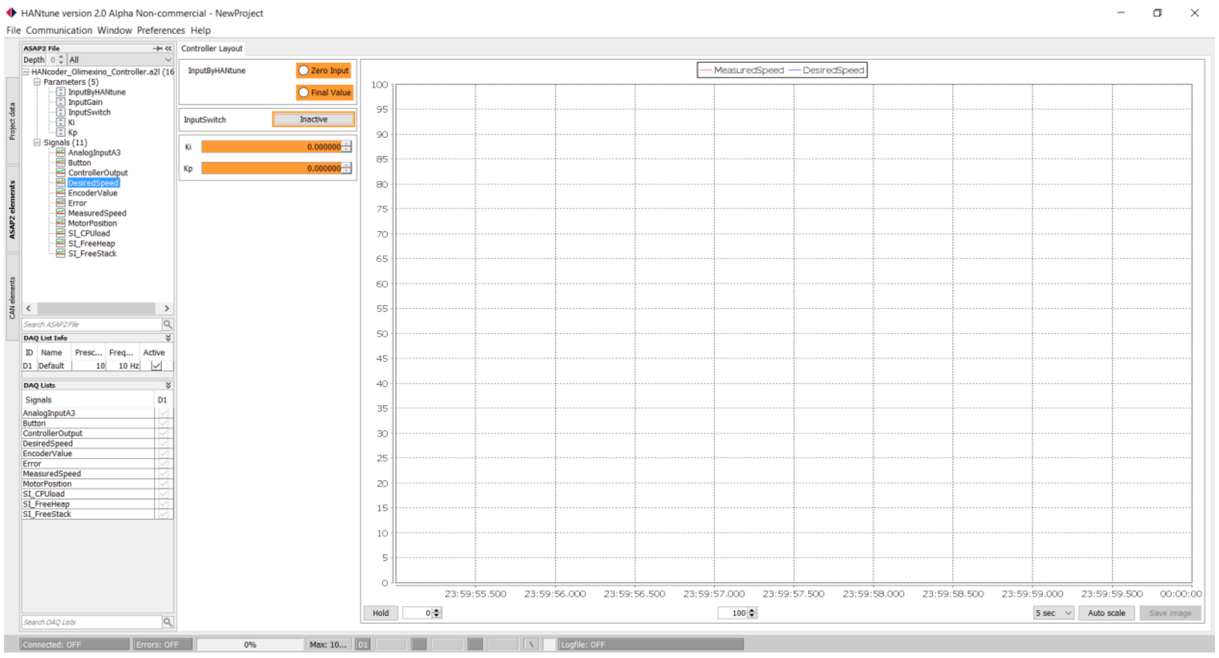
Add the **InputSwitch** parameter to the layout and make it a **ButtonEditor.** Modify the active and inactive value by right-clicking the button → modify values. Set the active value to 1 and the inactive value to 0.

Add the Ki parameter and Kp parameter to the layout by dragging them together into the layout and choosing **MultiEditor**.

Now it is time to add the signals. Drag the signal **MeasuredSpeed** into the layout and make it a ScopeViewer. Drag the signal **DesiredSpeed** into the ScopeViewer to add the signal. The ScopeViewer has two important features for this workshop: the Auto scale feature and the Hold feature.

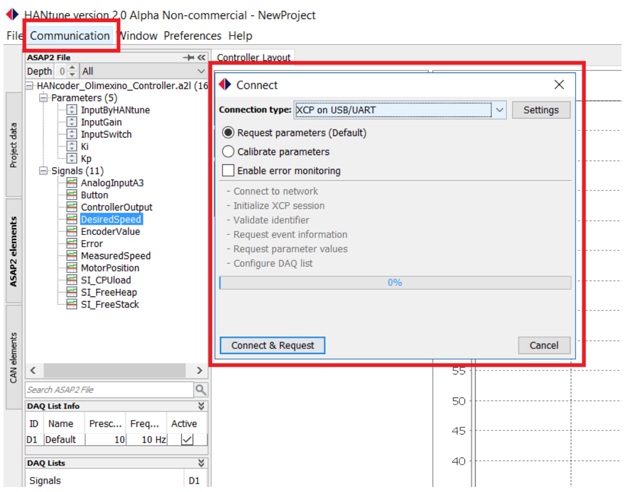
To drag, delete or scale an editor or viewer, use **Ctrl+r**.

Now the layout is finished:



4.2 Connect to the Olimexino

After the layout is done, it is time to connect HANtune to the setup. This is done by clicking on The Communication tab → connect, or by pressing F5.

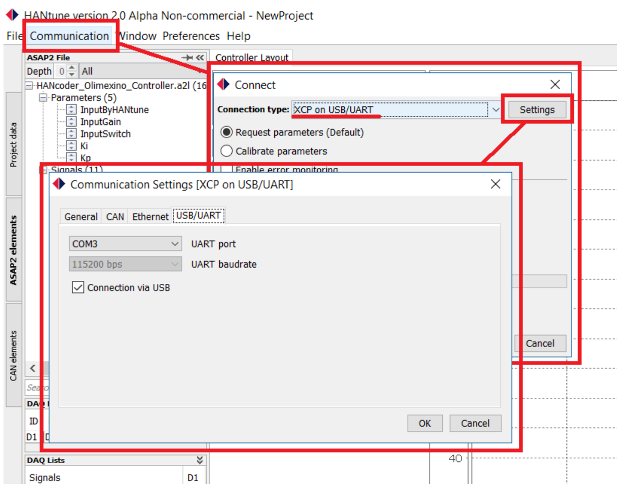


Connection type: XCP on USB/UART

(●) Request parameters (Default)
( ) Calibrate parameters
[ ] Enable error monitoring

- Connect to network
- Initialize XCP session
- Validate identifier
- Request event information
- Request parameter values
- Configure DAQ list

0%

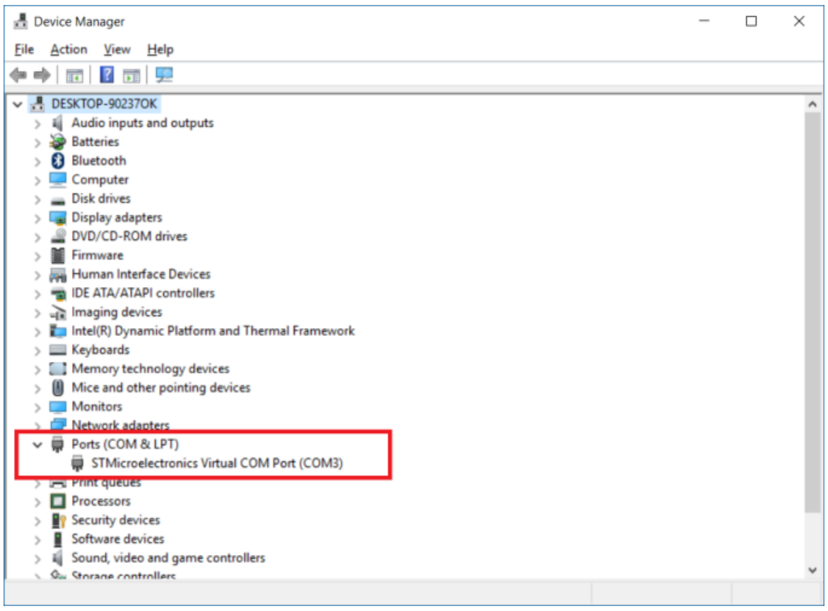

HANtune version 2.0 Alpha Non-commercial - NewProject
File
Communication
Window
Preferences
Help
Connect
Settings
Communication Settings [XCP on USB/UART]
General
CAN
Ethernet
USB/UART
OK
Cancel

Connection type: XCP on USB/UART

COM3 — UART port
115200 bps — UART baudrate
[✓] Connection via USB

To set up the connection, a few settings have to be adjusted. Set the Connection type on **XCP on USB/UART** and click on settings.

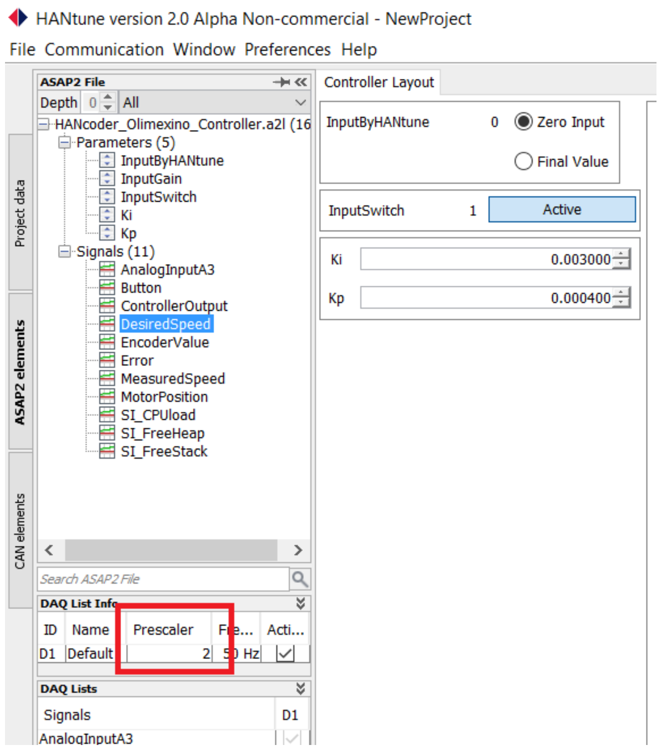
To make sure that the right COMport is selected, go to the Windows device manager (in Dutch: Apparaatbeheer) → Expand the Ports (COM & LPT) dropdown → the COMport is shown.



DESKTOP-90237OK
- Audio inputs and outputs
- Batteries
- Bluetooth
- Computer
- Disk drives
- Display adapters
- DVD/CD-ROM drives
- Firmware
- Human Interface Devices
- IDE ATA/ATAPI controllers
- Imaging devices
- Intel(R) Dynamic Platform and Thermal Framework
- Keyboards
- Memory technology devices
- Mice and other pointing devices
- Monitors
- Network adapters
- Ports (COM & LPT)
  - STMicroelectronics Virtual COM Port (COM3)
- Print queues
- Processors
- Security devices
- Software devices
- Sound, video and game controllers
- Storage controllers

After the COMport is set, click on OK and then Connect & Request.

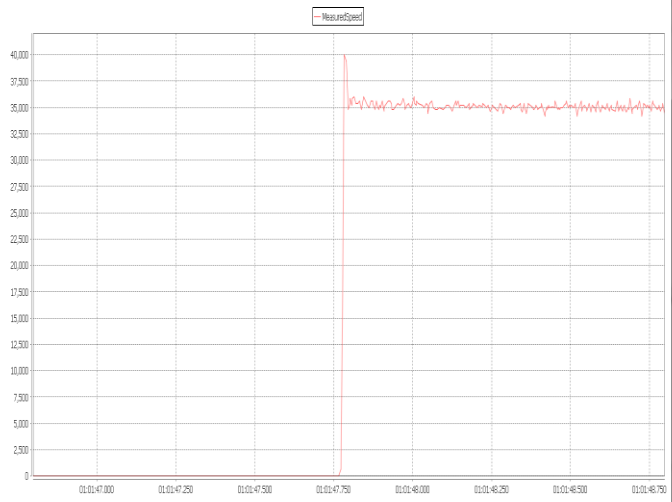
To make the scope more precise, the DAQlist prescaler can be adjusted. Set the Prescaler to 2 by clicking on the section and typing in 2. The model is run every 10ms, so at 100Hz. HANtune by default only requests the signal values from the controller once every ten times the model is run. Changing the prescaler to 2 will make HANtune request the signals every 2 cycles or at 50Hz.

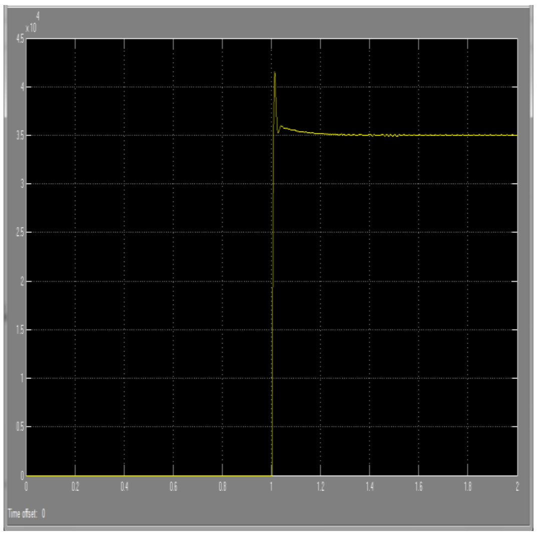


ID	Name	Prescaler	Fre...	Acti...
D1	Default	2	50 Hz	✓

4.3 Test the controller

When the **InputSwitch** parameter is set to 1 the setup can be tested and compared with the simulation. To do this, switch the RadioButton from **Zero Input** to **Final Value** while connected. Press the **Hold** button to stop the scope if desired. Example results for both real world test (left) and simulation (right) are shown below:





Note: The results can differ from the simulation because of the first-order model that was used does not adequately represent the reality.

Further tune the controller gains Kp and Ki until a satisfactory step response has been realised. If desired you can switch back to controlling the motor speed with the potentiometer by pressing the ButtonEditor which controls the **InputSwitch.**

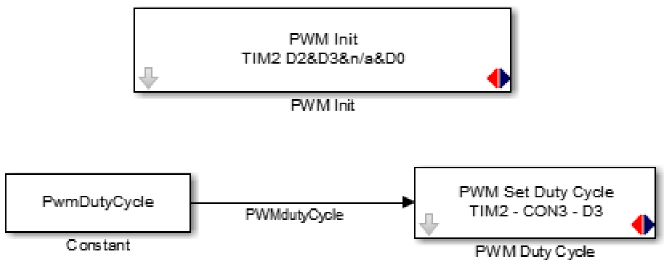
5 CREATING THE PLANT MODEL

In this part a simple linear model from the system will be made. Only the in- and output data is used to construct this model. Instead of the model used in the workshop here the model input will be PWM duty cycle and the output will be motor speed.

To acquire the data first a test model will be build. HANtune's logging function will be used to acquire the data. The resulting log file (csv format) is then imported in Matlab with the *uiimport* command. From this imported data the model will be estimated using the System Identification Toolbox. This chapter serves only as an illustration on how to use the tools to come to a plant model, the plant model will not be used.

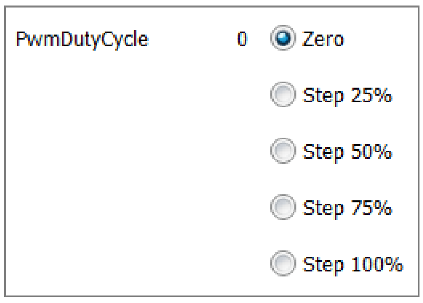
5.1 Making the test setup

First a HANcoder test model is needed to generate the desired data. The output is controlled by a HANtune parameter, the output of this parameter is given a name so it can be logged in HANtune. The input is the encoder value, this is read by using the Quadrature Encoder block. This gives the position of the motor but the speed is needed, a discrete derivative block is used to obtain this. Both the position and speed signals are given a name so they can be logged in HANtune. Also the base sample time is lowered to 1ms for faster logging.



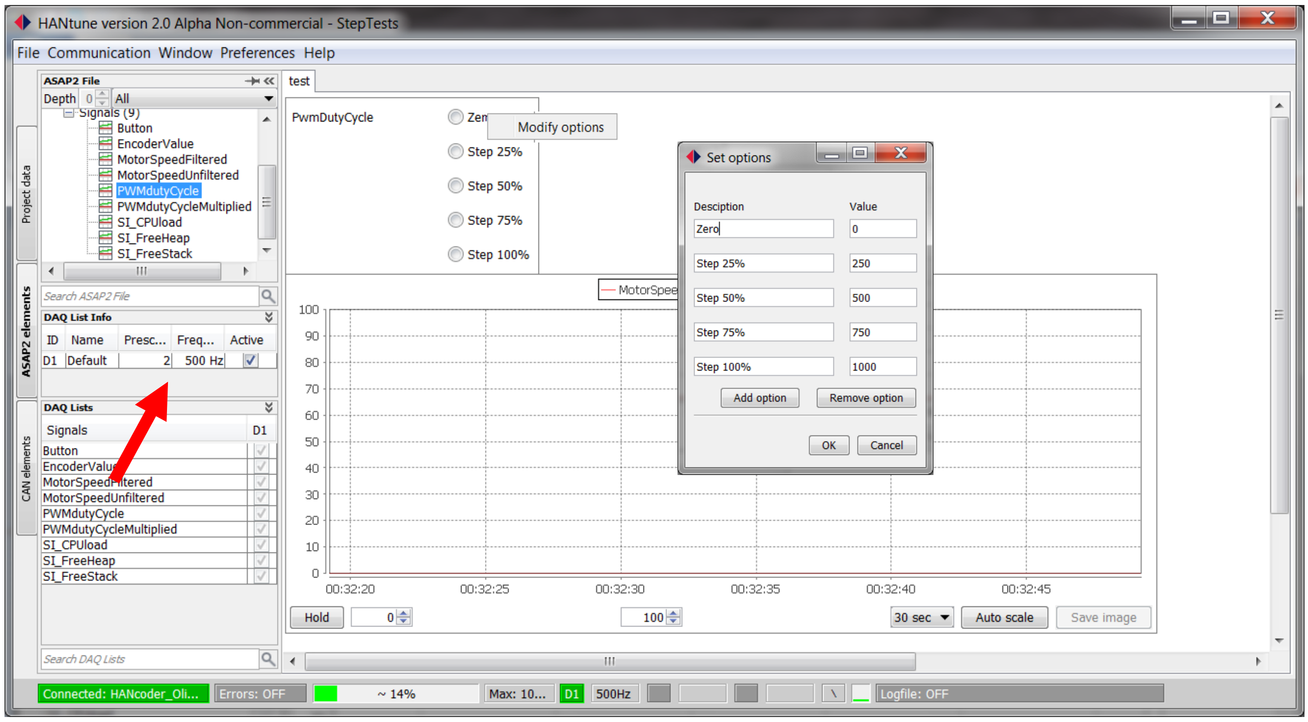
The motor direction will not be changed during the tests, it is assumed that the behaviour is equal in both directions. (This assumptions should be checked)

A HANtune layout needs to be made to facilitate testing. A Radio button editor can be used to quickly be able to repeat the same step tests. Add a Radio button editor by dragging the PwmDutyCycle parameter from the ASAP2 elements sidebar into the layout and select Radio button editor. Right click on one of the options to **Modify** the settings. Add 3 more options by clicking the **Add option** button. Now we can give each option a value and a description.



Next we need a way to see what the system is doing, a scope is probably the best way for this. Add a **Scope viewer** by dragging the **PWMdutyCycle** into the layout and select scope viewer. Add the MotorSpeed by dragging this signal into the Scope window. Press Ctrl+R to go into Resize and Move mode and place the viewers in a convenient location.

Since the reaction of the electric motor is quite fast it wouldn't make sense to log data at low frequencies, the Base Sample Time of the model was already set to 1ms but HANtune by default only requests data once in every 10 cycles. In our case this would mean it would sample at 100Hz or every 10ms. To increase the frequency the **Prescaler** in the DAQ List Info window can be adapted. Set this value to 2 to log at 500Hz, faster isn't possible because of the limited bandwidth over USB. See the next page for the location of this setting.



Signals (9)
- Button
- EncoderValue
- MotorSpeedFiltered
- MotorSpeedUnfiltered
- PWMdutyCycle
- PWMdutyCycleMultiplied
- SI_CPUload
- SI_FreeHeap
- SI_FreeStack

DAQ List Info

ID	Name	Presc...	Freq...	Active
D1	Default	2	500 Hz	✓

DAQ Lists

Signals	D1
Button	✓
EncoderValue	✓
MotorSpeedFiltered	✓
MotorSpeedUnfiltered	✓
PWMdutyCycle	✓
PWMdutyCycleMultiplied	✓
SI_CPUload	✓
SI_FreeHeap	✓
SI_FreeStack	✓

Zero
Step 25%
Step 50%
Step 75%
Step 100%

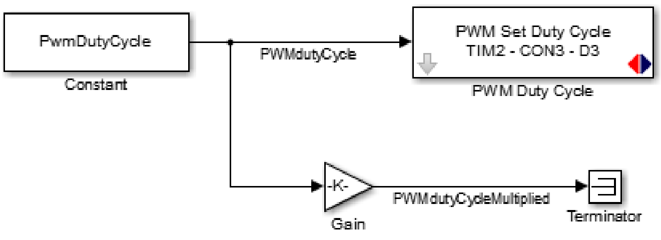
Set options

Description	Value
Zero	0
Step 25%	250
Step 50%	500
Step 75%	750
Step 100%	1000

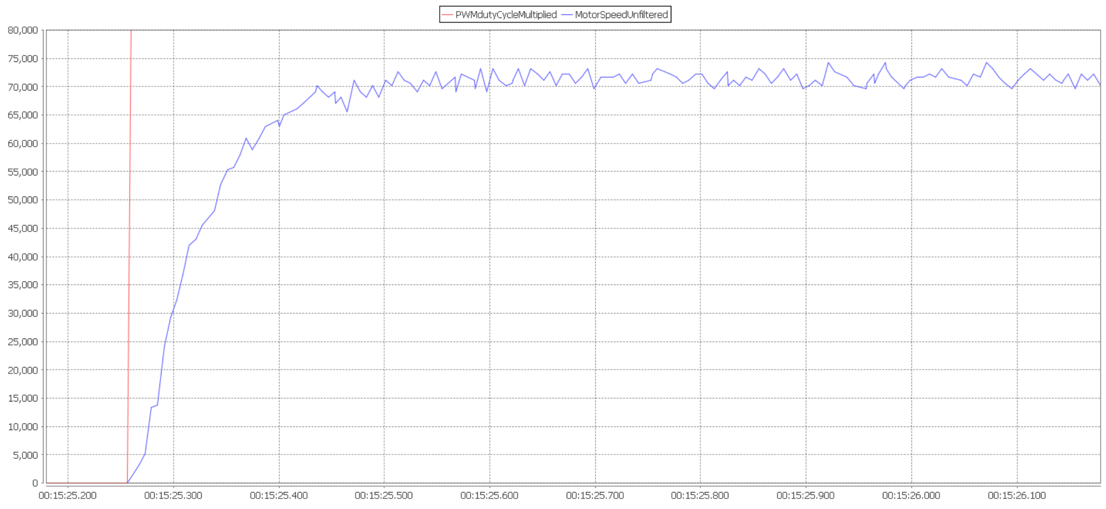
Add option Remove option
OK Cancel

Connected: HANcoder_Oli... | Errors: OFF | ~ 14% | Max: 10... | D1 | 500Hz | Logfile: OFF

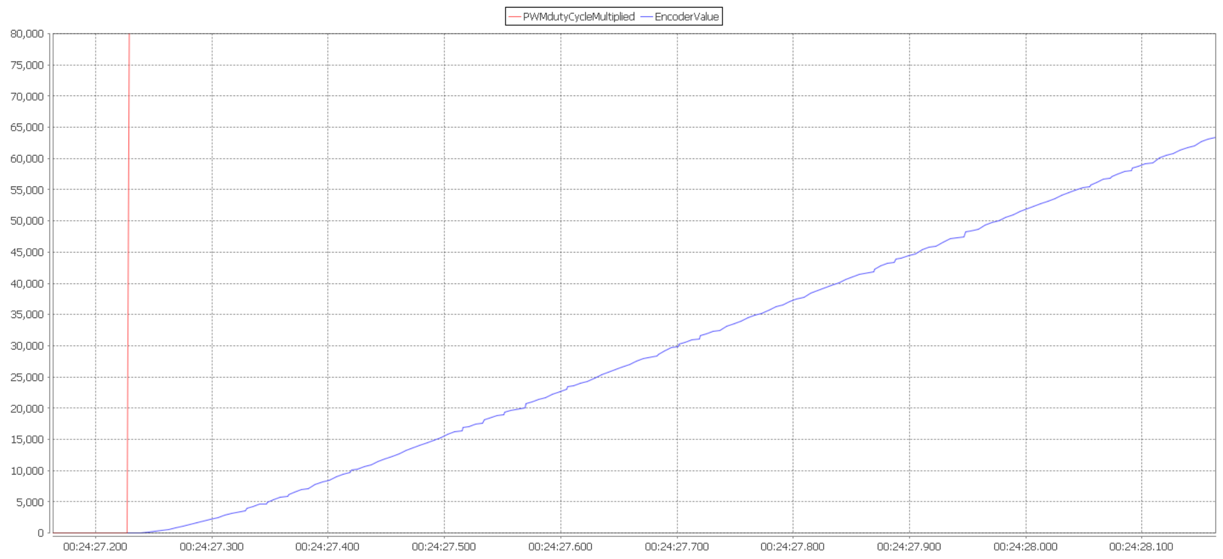
The first test is just giving the system a simple step input to look at the response of the system. Unfortunately because of the difference in magnitude between the PWMdutyCycle and the MotorSpeed it is difficult to see when the input was given. To fix this you can is multiply the **PWMdutyCycle** in the Simulink model by 1000 so it becomes clearly visible on the scope in HANtune.



This isn't strictly necessary. The following Scope image can be obtained when giving a step input to the electric motor.

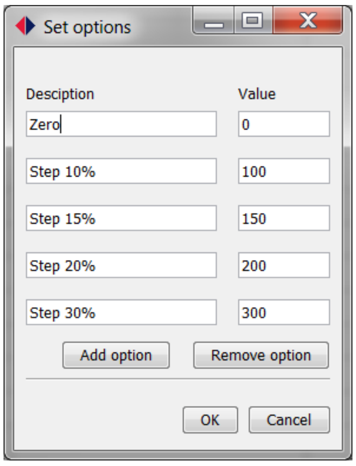


This first test shows a step response of a second order system. It is evident from this graph that the second order is relatively small compared to the first order. Also there seems to be a lot of noise on the signal. This is probably because a derivative is used to determine the speed and there is a little noise on the encoder value. To check this assumption the same test is performed and a graph of the encoder value is created.



It turns out that on the encoder value there is also some noise, this is the source for the noise on the motor speed. At the moment it is unclear where this disturbance is coming from and it will be ignored for now.

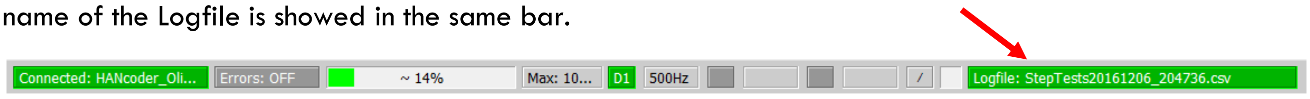
When the Radio button editor is used with the settings recommended above, the step from 0 to 25% causes the motor speed to increase to 90.000 (encoder pulses per second) The step from 25% to 50% only causes an increase in speed of 5000. It's clear that the motor is highly non-linear at the higher speeds. To keep the modeling simple only inputs up to 30% will be used. In this part the motor is still fairly linear and thus easy to model. The Radio button editor should reconfigured to make the following steps: 10%, 15%,20% and 30%. Right click on one of the options to reconfigure the settings.



Desciption	Value
Zero	0
Step 10%	100
Step 15%	150
Step 20%	200
Step 30%	300

5.2 Logging data with HANtune

The controller is reset to and HANtune is reconnected to reset all values to zero. Logging is now started by clicking the Logfile bar on the bottom of HANtune. The Logfile bar is green when logging is enabled. The name of the Logfile is showed in the same bar.


Connected: HANcoder_Oli...
Errors: OFF
~ 14%
Max: 10...
D1
500Hz
/
Logfile: StepTests20161206_204736.csv

When logging has started it is time to give the system a number of input steps. This can now easily be done with the RadioButton editor. In this example the following step sequence is used:

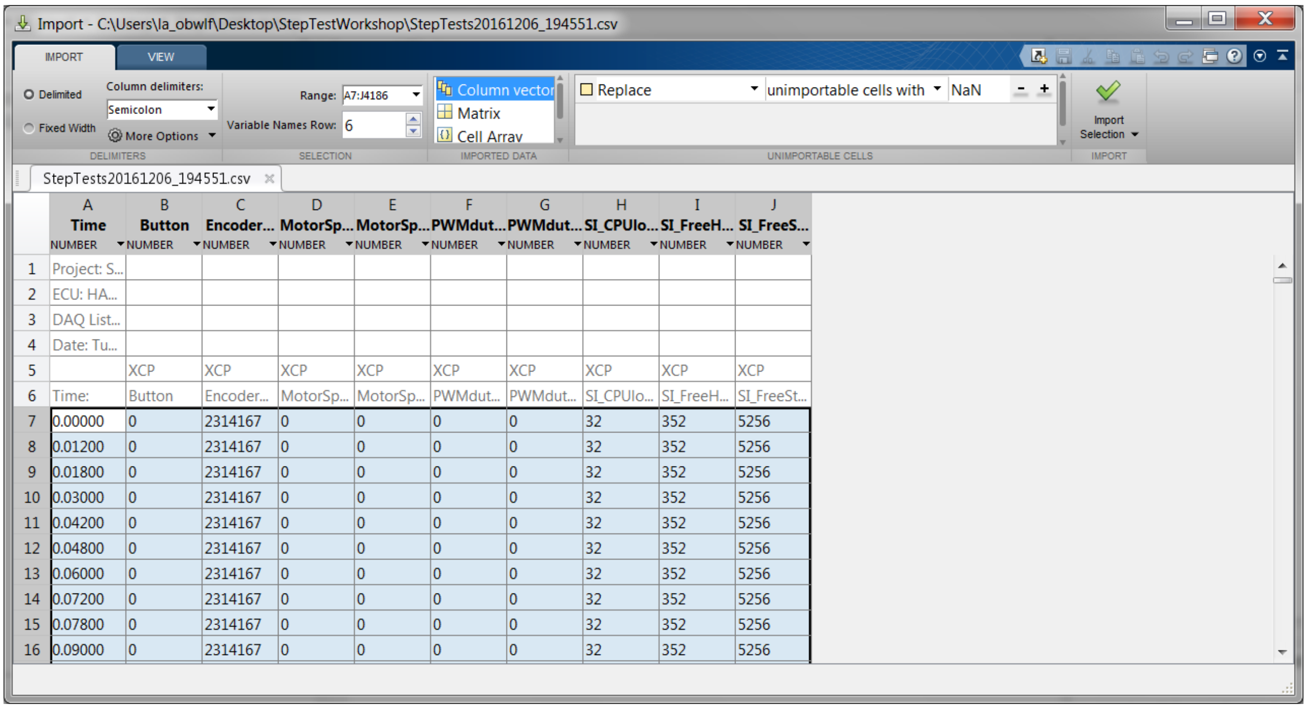
Input: [0, 10%, 0, 10%, 0, 10%, 0, 15%, 0, 15%, 0, 15%, 0, 20%, 0, 20%, 0, 20%, 0, 30%, 0, 30%, 0, 30%]

Between each change is approximately 2 seconds, this is because the fall time of the system is approximately 1.2 seconds. After the sequence is done, press the Logfile bar again to stop logging.

5.3 Reading the data in Matlab

Now it is time to read the data from the logfile. For this Matlab has the convenient data import wizard: in the Matlab command window type: *uiimport* Choose **File** and select the csv file that HANtune has created. The name of this file begins with the HANtune project name (newProject if the project hasn't been saved yet) and ends in the date and time. For example: ProjectName20161206_194551.csv

When the file is opened select **Semicolon** at the **Column delimiters** field and set the **Variable Names Row** to **6.** Next select the data, easiest is to select the first row and then press Ctrl+Shift+End.



Column delimiters: Semicolon

Range: A7:J4186

Variable Names Row: 6

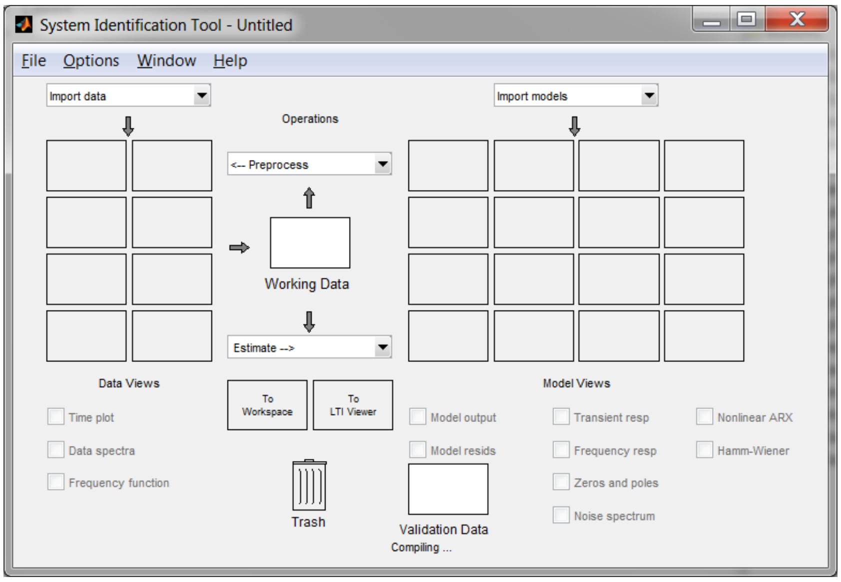
Replace unimportable cells with NaN

	A	B	C	D	E	F	G	H	I	J
	Time	Button	Encoder...	MotorSp...	MotorSp...	PWMdut...	PWMdut...	SI_CPUlo...	SI_FreeH...	SI_FreeS...
	NUMBER	NUMBER	NUMBER	NUMBER	NUMBER	NUMBER	NUMBER	NUMBER	NUMBER	NUMBER
1	Project: S...									
2	ECU: HA...									
3	DAQ List...									
4	Date: Tu...									
5		XCP	XCP	XCP	XCP	XCP	XCP	XCP	XCP	XCP
6	Time:	Button	Encoder...	MotorSp...	MotorSp...	PWMdut...	PWMdut...	SI_CPUlo...	SI_FreeH...	SI_FreeSt...
7	0.00000	0	2314167	0	0	0	0	32	352	5256
8	0.01200	0	2314167	0	0	0	0	32	352	5256
9	0.01800	0	2314167	0	0	0	0	32	352	5256
10	0.03000	0	2314167	0	0	0	0	32	352	5256
11	0.04200	0	2314167	0	0	0	0	32	352	5256
12	0.04800	0	2314167	0	0	0	0	32	352	5256
13	0.06000	0	2314167	0	0	0	0	32	352	5256
14	0.07200	0	2314167	0	0	0	0	32	352	5256
15	0.07800	0	2314167	0	0	0	0	32	352	5256
16	0.09000	0	2314167	0	0	0	0	32	352	5256

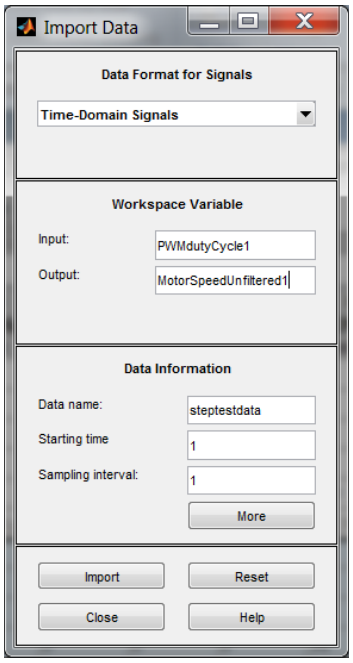
Next click on the Import Selection button and the Data will automatically be added to your workspace. If the variable names already exist in the workspace the wizard will add a number to the end.

5.4 Using the System Identification Toolbox

With the system identification toolbox from Matlab it is possible to estimate a model from just the input and output data. Type: *ident* in the command window to start the System Identification tool. (If the warning: *Undefined function or variable 'ident'.* shows, the toolbox might not be installed)



Next select the data by clicking on the pull-down menu in the upper left corner and selecting **Time Domain Data…** A dialog appears where the input and output variables can be chosen. Select the PWMdutyCycle as input and the MotorSpeed as output. Click import to send the data to the System Identification Tool. When the data is imported successfully the Import Data window can be closed again.


Import Data

Data Format for Signals

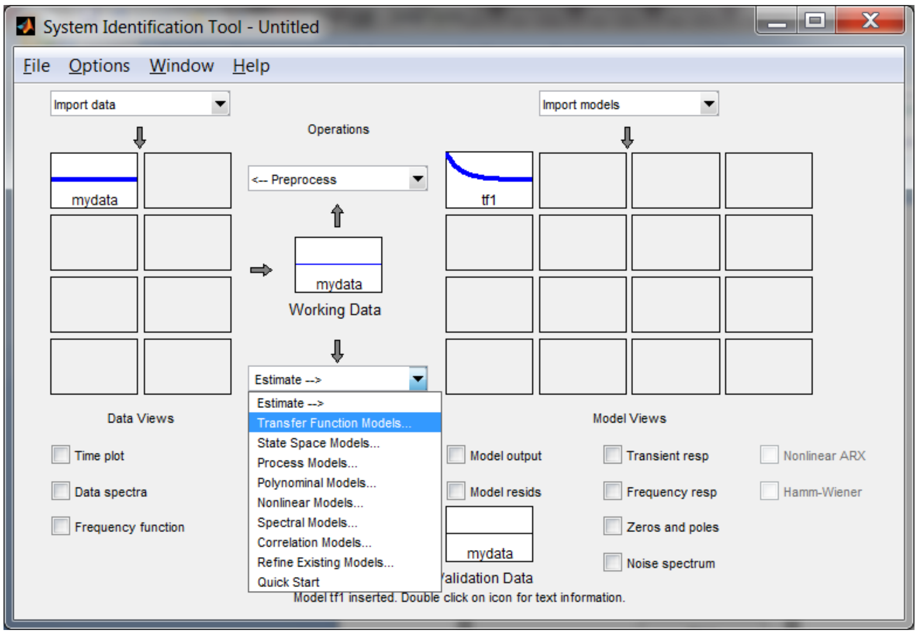
Time-Domain Signals

Workspace Variable

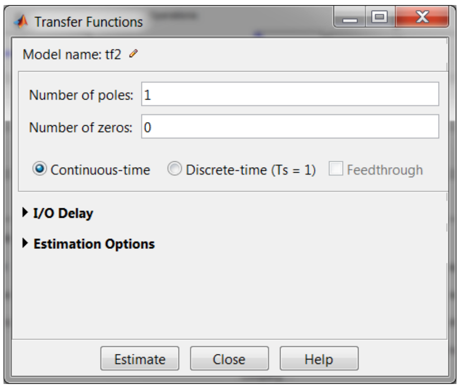
Input:	PWMdutyCycle1
Output:	MotorSpeedUnfiltered1

Data Information

Data name:	steptestdata
Starting time	1
Sampling interval:	1



From the imported data the model can be estimated. Click on the drop down menu and select **Transfer Function Models…**



Model name: tf2

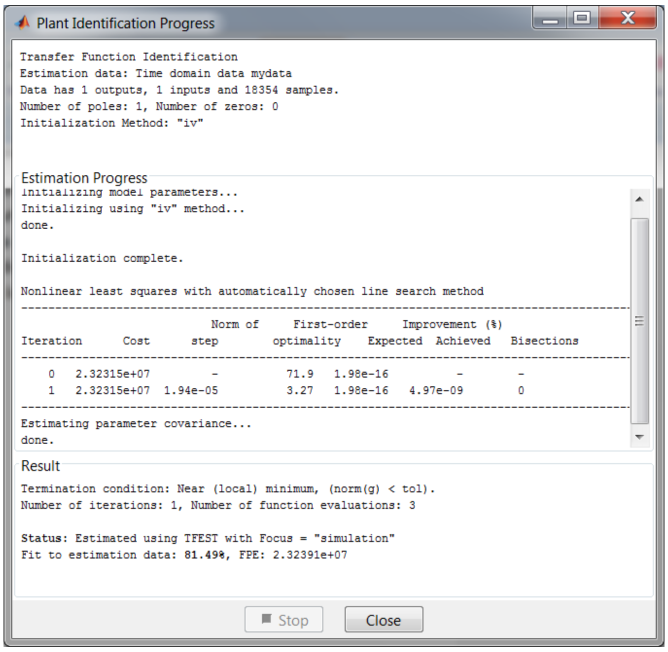
Number of poles: 1

Number of zeros: 0

[✓ Continuous-time] Discrete-time (Ts = 1) Feedthrough

A dialog appears where the numbers of poles and zeros can be chosen. Choose 1 pole and no zeros, although this is wrong the motor will be modeled as a first order system.

Click on estimate to start the estimation process. The time it takes will depend on the data points and the order of the system.



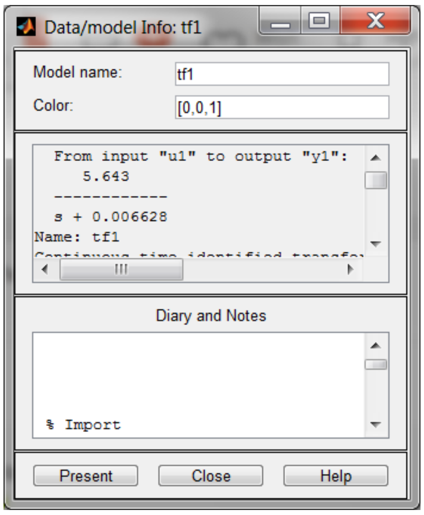
```
Transfer Function Identification
Estimation data: Time domain data mydata
Data has 1 outputs, 1 inputs and 18354 samples.
Number of poles: 1, Number of zeros: 0
Initialization Method: "iv"
```

```
Initializing model parameters...
Initializing using "iv" method...
done.

Initialization complete.

Nonlinear least squares with automatically chosen line search method
------------------------------------------------------------------------------------------
                            Norm of      First-order     Improvement (%)
Iteration      Cost          step          optimality    Expected  Achieved    Bisections
------------------------------------------------------------------------------------------
    0   2.32315e+07          -            71.9    1.98e-16          -           -
    1   2.32315e+07  1.94e-05             3.27    1.98e-16   4.97e-09           0
------------------------------------------------------------------------------------------
Estimating parameter covariance...
done.
```

```
Termination condition: Near (local) minimum, (norm(g) < tol).
Number of iterations: 1, Number of function evaluations: 3

Status: Estimated using TFEST with Focus = "simulation"
Fit to estimation data: 81.49%, FPE: 2.32391e+07
```

Once the estimation process is finished it will show the fit to estimation percentage. This percentage tells how close the models comes to the input output data. In this example 81% is achieved. The noise that was on the output signal could be a possible cause for the poor result. The tool has options to filter the data prior to model estimation. This should be enough to make a decent controller. If the controller requirements, like overshoot and rise time, are though it is necessary to get a better model. The progress report can now be closed.

To check the transfer function simply double click on the result in the System Identification Tool. See next page



Data/model Info: tf1

Model name: tf1

Color: [0,0,1]

```
  From input "u1" to output "y1":
     5.643
  ------------
  s + 0.006628
Name: tf1
Continuous-time identified transfer
```

Diary and Notes

```
 % Import
```

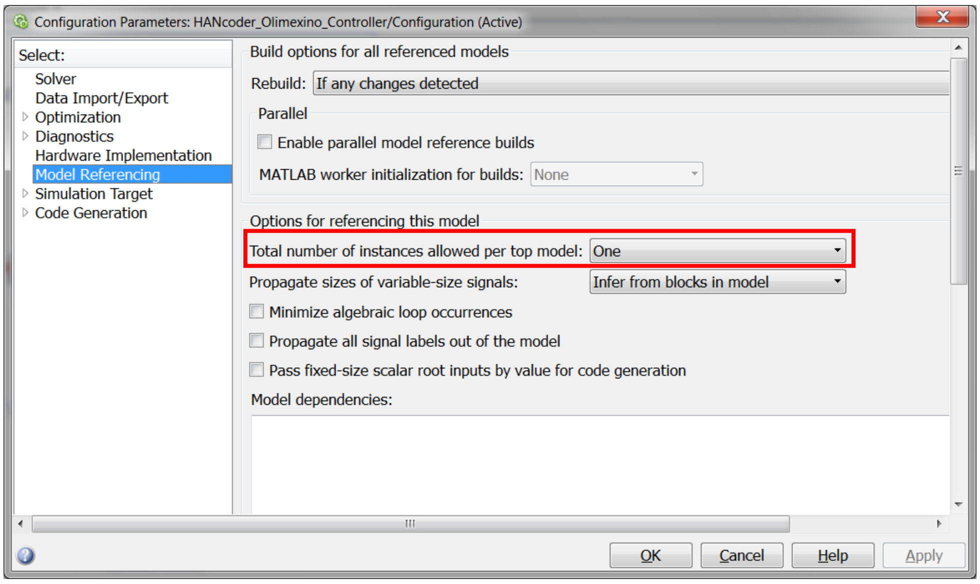
This transfer function can be used in a plant model. Be aware that this is the input output relation from duty cycle to motor speed instead of from duty cycle to motor position [encoderpulses].

APPENDIX 1: MODEL SETTINGS FOR MODEL REFERENCE

Although in this workshop the HANcoder_Olimexino_Controller.slx is already prepared for use as a referenced model, the settings that are changed are shown in this section so users can make the changes to their own models if model referencing is needed.

All the variables in the workspace can be 'accessed' by all referenced models used by Simulink. This means that if a model would be referenced twice the variable used by this model would be read/written twice. This could give serious data integrity problems. To prevent this Matlab forces the user to set the maximum number of instances allowed to 1. An error will show when the referenced model uses global variables (in the base workspace) and the maximum number of allowed instances is set to 'Multiple'. The setting can be changed in Configuration Parameters --> Model Referencing

If multiple instances of the model are used the model workspace instead of the base workspace should be used. This workshop will not treat this subject any further.


Configuration Parameters: HANcoder_Olimexino_Controller/Configuration (Active)

Select: Solver, Data Import/Export, Optimization, Diagnostics, Hardware Implementation, Model Referencing, Simulation Target, Code Generation

Build options for all referenced models

Rebuild: If any changes detected

Parallel

☐ Enable parallel model reference builds

MATLAB worker initialization for builds: None

Options for referencing this model

Total number of instances allowed per top model: One

Propagate sizes of variable-size signals: Infer from blocks in model

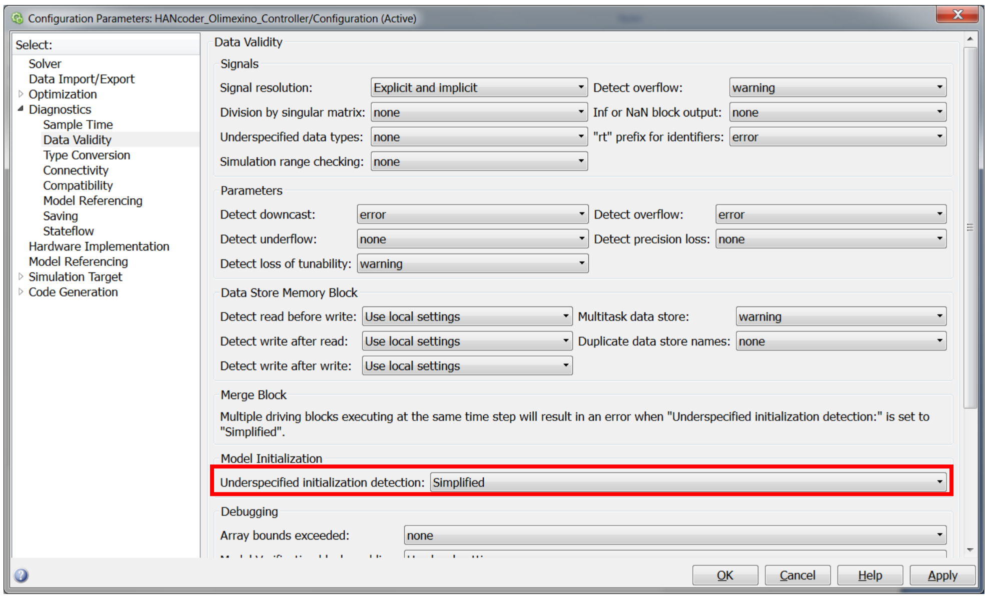
☐ Minimize algebraic loop occurrences

☐ Propagate all signal labels out of the model

☐ Pass fixed-size scalar root inputs by value for code generation

Model dependencies:

The next setting that needed to be changed was the way Simulink initializes the initial conditions for conditionally executed subsystems, Merge blocks, subsystem elapsed time, and Discrete-Time Integrator blocks. The default setting is 'Classic', this mainly ensures compatibility with older models, the setting needs to be changed to 'Simplified' in order to make model referencing work. Future versions of HANcoder will have the correct setting by default.


Configuration Parameters: HANcoder_Olimexino_Controller/Configuration (Active)

Select:
- Solver
- Data Import/Export
- Optimization
- Diagnostics
  - Sample Time
  - Data Validity
  - Type Conversion
  - Connectivity
  - Compatibility
  - Model Referencing
  - Saving
  - Stateflow
- Hardware Implementation
- Model Referencing
- Simulation Target
- Code Generation

Data Validity

Signals

Signal resolution:	Explicit and implicit	Detect overflow:	warning
Division by singular matrix:	none	Inf or NaN block output:	none
Underspecified data types:	none	"rt" prefix for identifiers:	error
Simulation range checking:	none		

Parameters

Detect downcast:	error	Detect overflow:	error
Detect underflow:	none	Detect precision loss:	none
Detect loss of tunability:	warning		

Data Store Memory Block

Detect read before write:	Use local settings	Multitask data store:	warning
Detect write after read:	Use local settings	Duplicate data store names:	none
Detect write after write:	Use local settings		

Merge Block

Multiple driving blocks executing at the same time step will result in an error when "Underspecified initialization detection:" is set to "Simplified".

Model Initialization

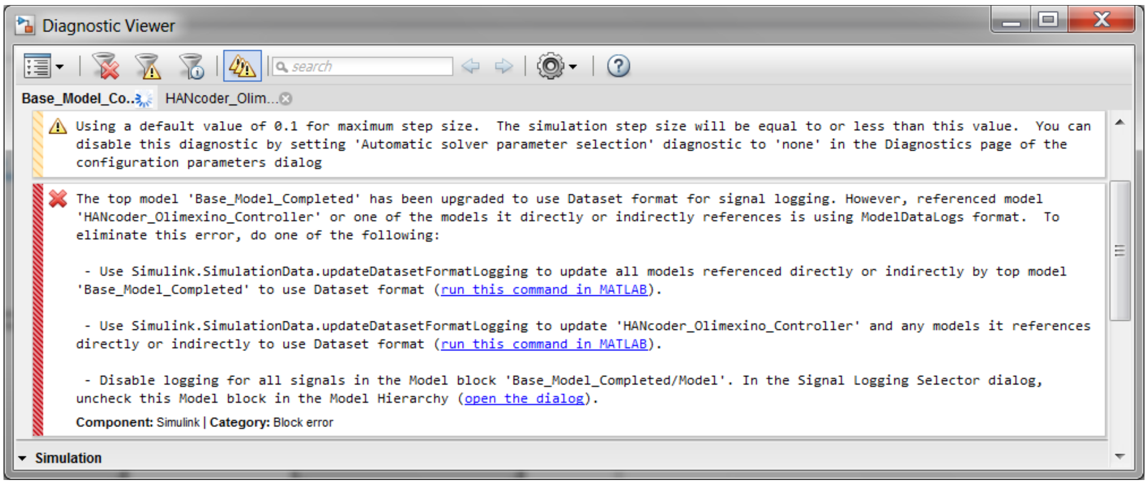
Underspecified initialization detection: Simplified

Debugging

Array bounds exceeded: none

OK Cancel Help Apply

The last error that can occur is an error about signal logging. Matlab upgraded the way it logs signals and the HANcoder models have not yet been upgraded to ensure backwards compatibility. When running the system an error shows in the diagnostics window. To solve the error simply choose the option to update all models referenced directly or indirectly by the top model.


Diagnostic Viewer

Base_Model_Co... HANcoder_Olim...

```
Using a default value of 0.1 for maximum step size.  The simulation step size will be equal to or less than this value.  You can
disable this diagnostic by setting 'Automatic solver parameter selection' diagnostic to 'none' in the Diagnostics page of the
configuration parameters dialog
```

```
The top model 'Base_Model_Completed' has been upgraded to use Dataset format for signal logging. However, referenced model
'HANcoder_Olimexino_Controller' or one of the models it directly or indirectly references is using ModelDataLogs format.  To
eliminate this error, do one of the following:

 - Use Simulink.SimulationData.updateDatasetFormatLogging to update all models referenced directly or indirectly by top model
'Base_Model_Completed' to use Dataset format (run this command in MATLAB).

 - Use Simulink.SimulationData.updateDatasetFormatLogging to update 'HANcoder_Olimexino_Controller' and any models it references
directly or indirectly to use Dataset format (run this command in MATLAB).

 - Disable logging for all signals in the Model block 'Base_Model_Completed/Model'. In the Signal Logging Selector dialog,
uncheck this Model block in the Model Hierarchy (open the dialog).
```

Component: Simulink | **Category:** Block error

Simulation