Hogeschool van Arnhem en Nijmegen

Workshop HANcoder STM32 Target

Version 0.5
10/17/2016

Workshop HANcoder STM32 Target

A guide to build a simple motor control model and use HANtune with it.

Workshop HANcoder STM32 Target

WORKSHOP HANCODER STM32 TARGET

Contents

VERSIONING

Nr	Date	Person	Changes	Status
0.1	01-12-2014	JAD van Kolfschoten	First version	Concept
0.2	04-12-2014	JAD van Kolfschoten	Added the optional extra. Renewed some screenshots	Concept
0.3	14-01-2016	JAD van Kolfschoten	Updated to use with USB Bootloader	Concept
0.4	17-10-2016	G Jansma	Updated guide for HANcoder 0.4 blockset	Concept
0.5	23-11-2016	JAD van Kolfschoten	Updated guide for HANcoder 0.5 blockset	Final

INTRODUCTION

This document will describe the steps to form a model with which to control a motor via a potentiometer and read the values with HANtune.

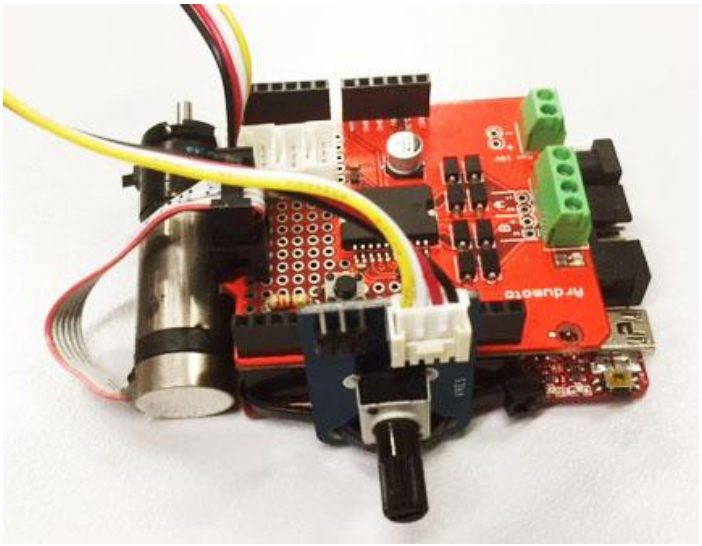


figure 0-1 The workshop setup

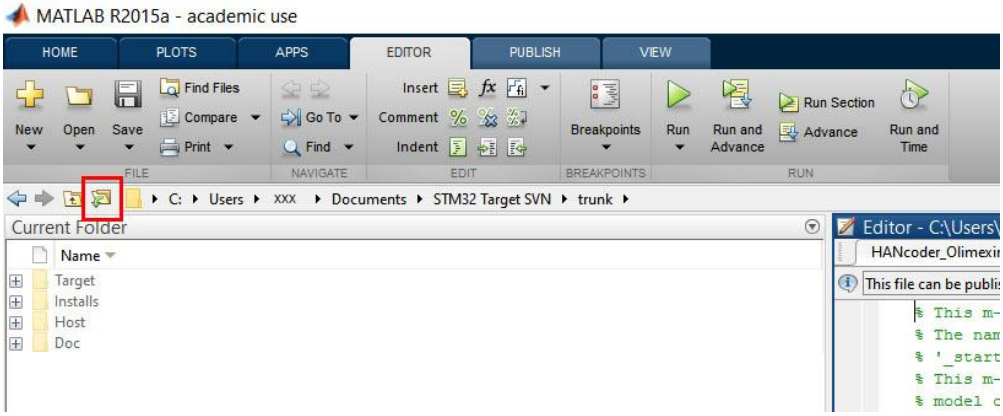
For this workshop we will be using an Olimexino as microcontroller board. On top of the Olimexino is a shield with a motor driver or H-bridge, the Ardumoto. The H-bridge is controlled with a PWM signal for the speed on pin D3 and a digital signal for controlling the direction on pin D12. A small DC motor which has 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. Next to that there is a potentiometer attached to the Olimexino on pin A3. The potentiometer will serve as a setpoint for the motor speed. The further the potentiometer is turned the faster the motor should turn.

Before being able to start with this process it is necessary to install certain programs such as MATLAB. These steps are described in 'Getting Started Guide Olimexino.docx'.

It is advised to use MATLAB 2013b or later for this workshop.

1 EXPLANATION OF THE BASE MODEL

Open the Simulink model by starting MATLAB and navigating to the HANcoder directory where the Simulink models are located, the 'Target' directory. By double-clicking the provided model file, i.e. 'HANcoder_Olimexino.slx', from within MATLAB the model is started.



The right directory can easily be found with the **browse for folder** button.

Figure 1-1 Browse for folder button Matlab

Make sure that the **Target folder** is selected. Otherwise Matlab will have trouble finding the necessary files to build HANcoder models.

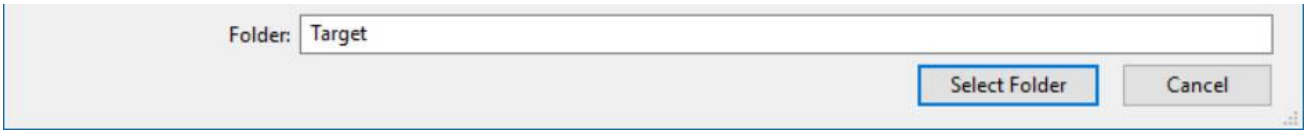


Figure 1-2 select target folder

NOTE: Matlab versions prior to the 2013b release cannot open the .slx files and instead the .mdl files for these earlier versions can be found in the zip file: **Template models for older versions.zip**.

If the default project is used, the following screen will appear:

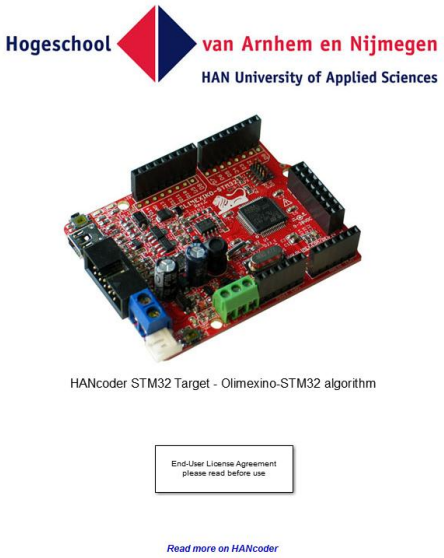


Figure 1-3 View when opening the model

Nothing in this part of the model can be changed, otherwise the code generation will no longer work!

When double-clicked on the picture of the controller, the content of this block is shown:

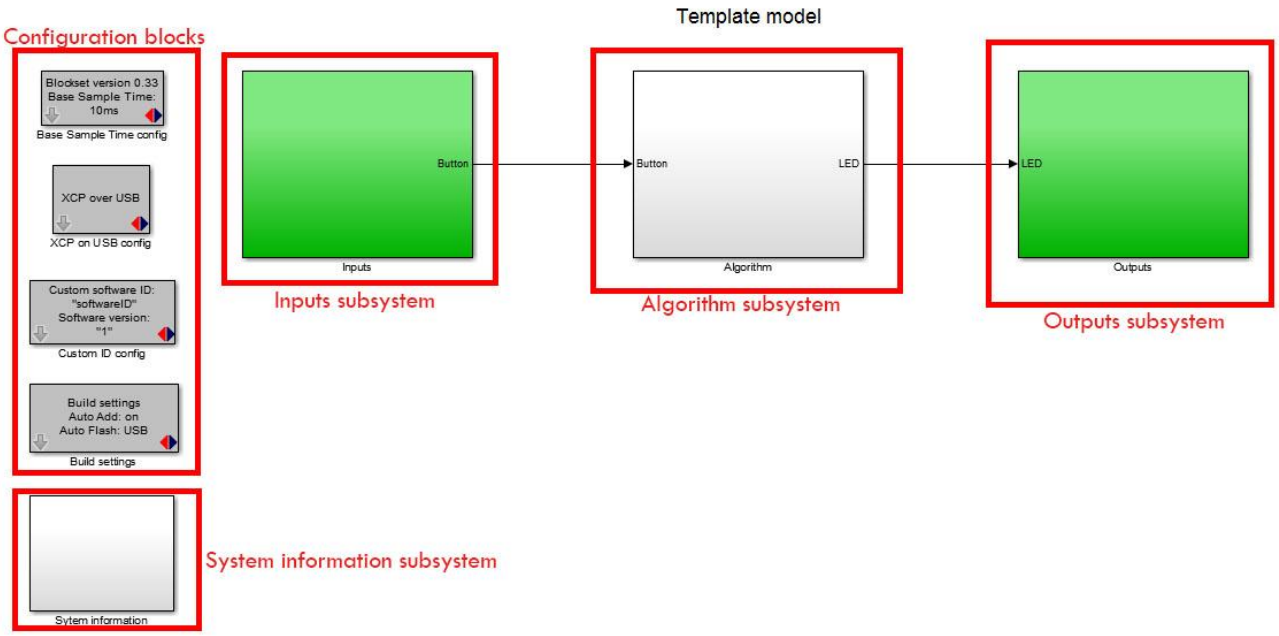


Figure 1-4 Template model delivered with blockset

Figure 3-2 shows the template model. This is the base of every new project. This simple project makes the green LED blink at 5Hz when the button on the side of the board isn't pressed and at 10Hz when it is. It is advised to keep this functionality in your project so you can always check if the software is still responsive.

The template model consists of:

- The inputs subsystem

- The algorithm subsystem
- The outputs subsystem
- The configuration blocks
- The system Information subsystem

The inputs subsystem

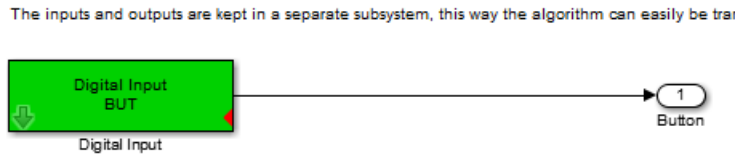


Figure 1-5 Inside the Inputs subsystem

The inputs of the model are placed inside the Inputs subsystem. The inputs are connected with the Algorithm through Outport blocks (the block with 'Button' below it in the figure above).

The algorithm subsystem

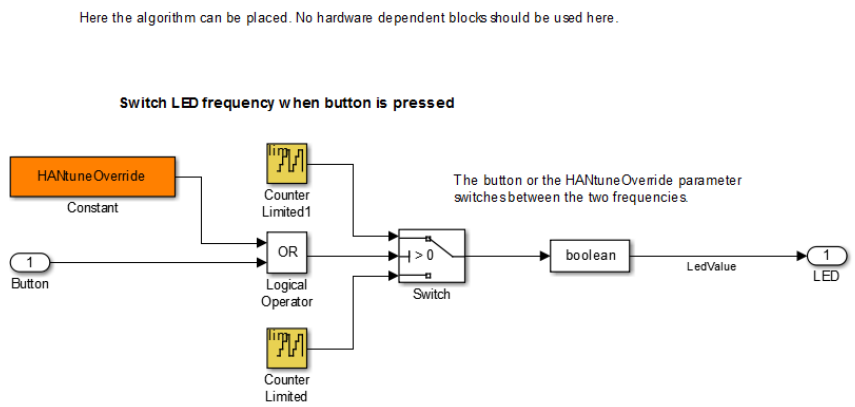


Figure 1-6 The algorithm subsystem template model

In the algorithm subsystem the functionality is placed. The subsystem is connected with the inputs and outputs through In- and Outport blocks. The functionality can easily be transferred to another hardware platform because there are no hardware dependent in- or outputs in this part, it is recommended to work the same way in your own projects.

The outputs subsystem

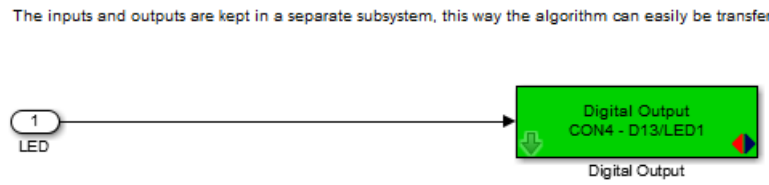


Figure 1-7 The outputs subsystem template model

In here the outputs of your model are located.

The configuration blocks

These blocks are the settings of the model.
The Base Sample Time determines the interval at which the model is run. You can add blocks to the model which run at lower rates but not faster. In the template model the base sample time is 10ms so the model can run at a 100Hz.

The XCP on USB config block configures the Olimexino to communicate to HANtune over USB.

The custom software ID block lets you choose a name and version number for your model, this will be used when connecting with HANtune.

With the Build settings you can let HANcoder automatically add all signals and parameters in the workspace of the project. This is important to communicate with HANtune.
The Auto Flash function flashes the software automatically after a successful build.

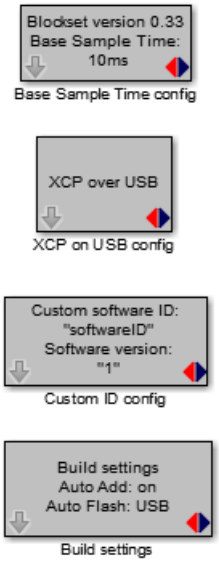


Figure 1-8 Configuration blocks template model

The System Information Subsystem

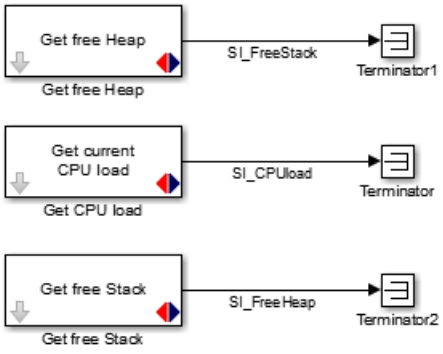


Figure 1-9 The System Information subsystem template model

The System Information Subsystem gives you information about the Load, Heap and Stack of the system once the model is running. This way you can monitor how many resources your software program uses. The Signals are already defined by HANcoder and will be visible in HANtune.

2 ADDING THE BLOCKS TO CONTROL THE MOTOR

To add a block for the Olimexino click on the 'Library browser' button in the toolbar of the model:

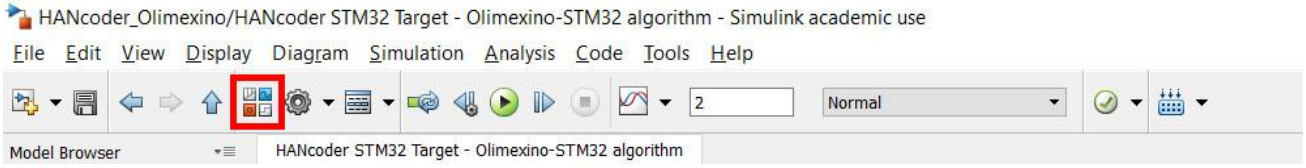


Figure 2-1 Simulink Library button

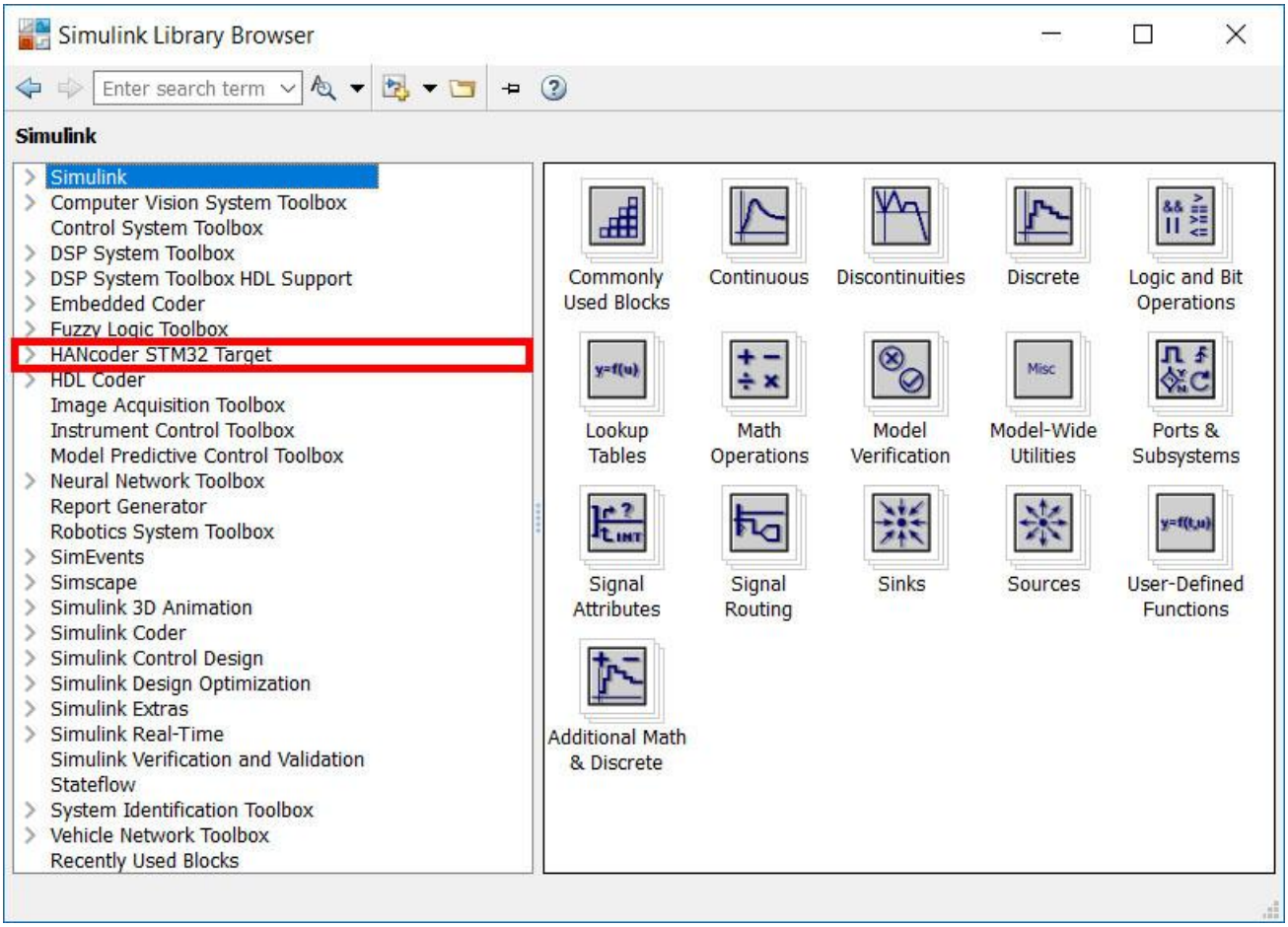


Figure 2-2 Simulink library

Open the 'HANcoder STM32 Target' toolbox by clicking on it.

If the library isn't visible try refreshing the library tree view by pressing F5. If there is a Pop up bar that says 'Some libraries are missing repository information. Fix' Click on Fix. Turn on 'Generate repositories in memory' in the pop up.

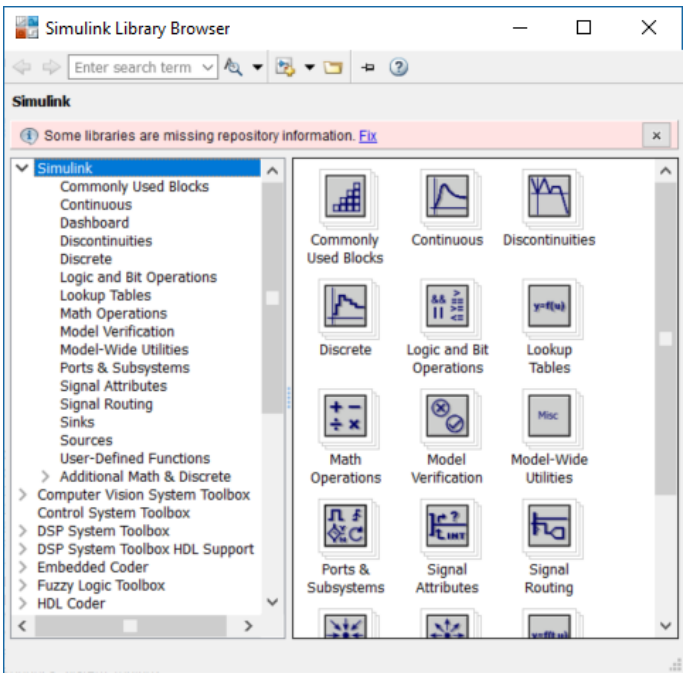


Figure 2-3 Library pop up

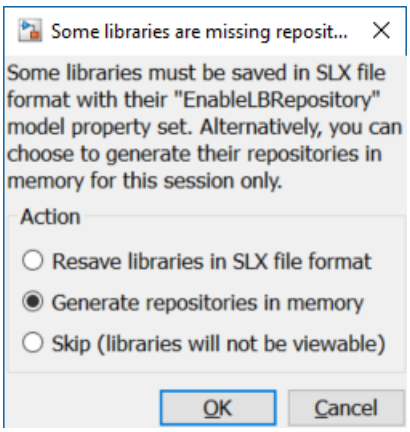


Figure 2-4 library Fix pop up

When the HANcoder STM32 Target expands the HANcoder blocks for the compatible STM32 development boards are visible.

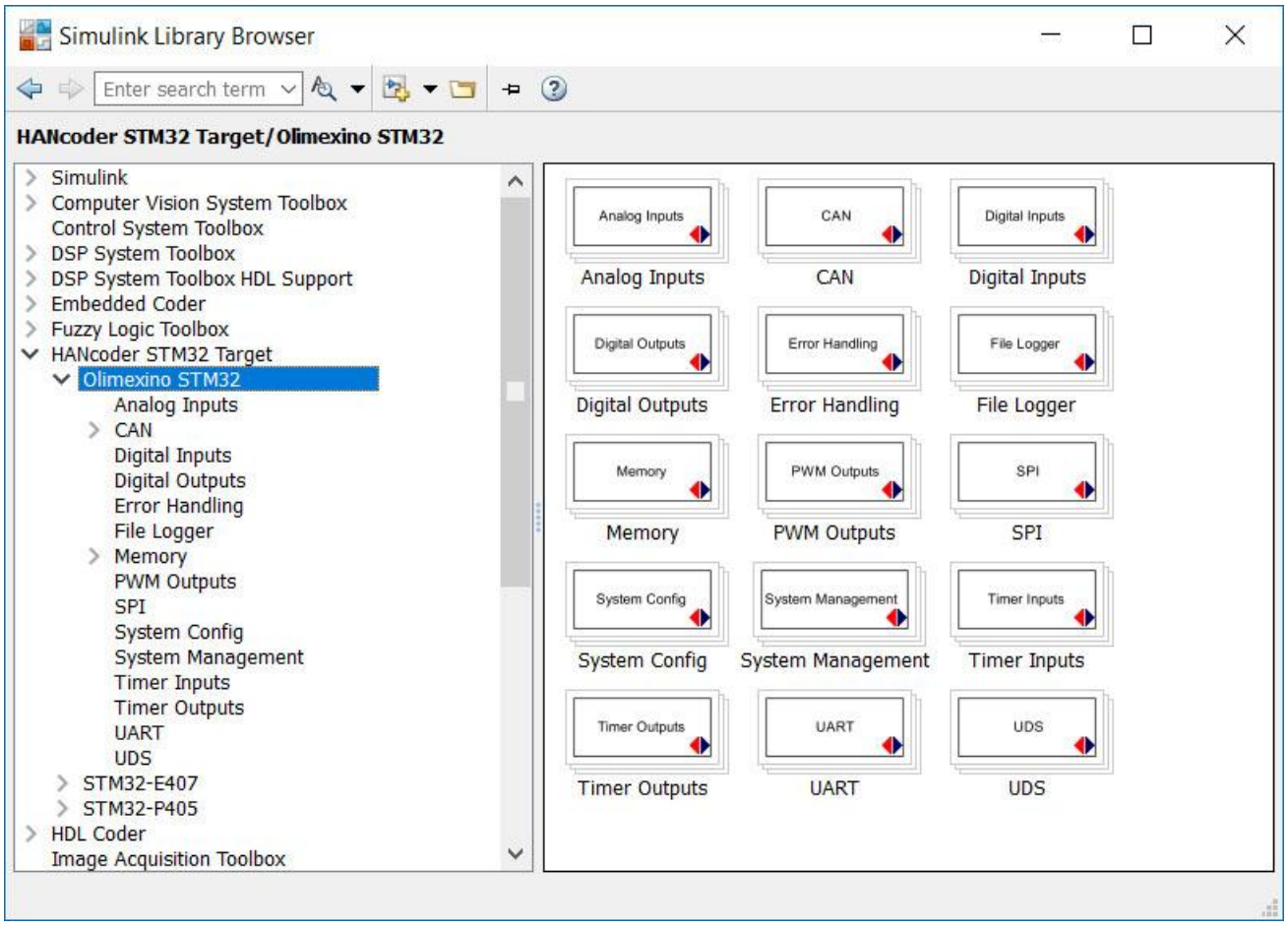


Figure 2-5 Simulink HANcoder STM32 Target Library

To the right, the content of the toolbox is placed. This content is sorted in a tree like view for easy access. Use the help files to determine the proper use of a block. These are accessible by right clicking on a block and then selecting “Help”.

When the library is opened, go to **Analog Inputs**.

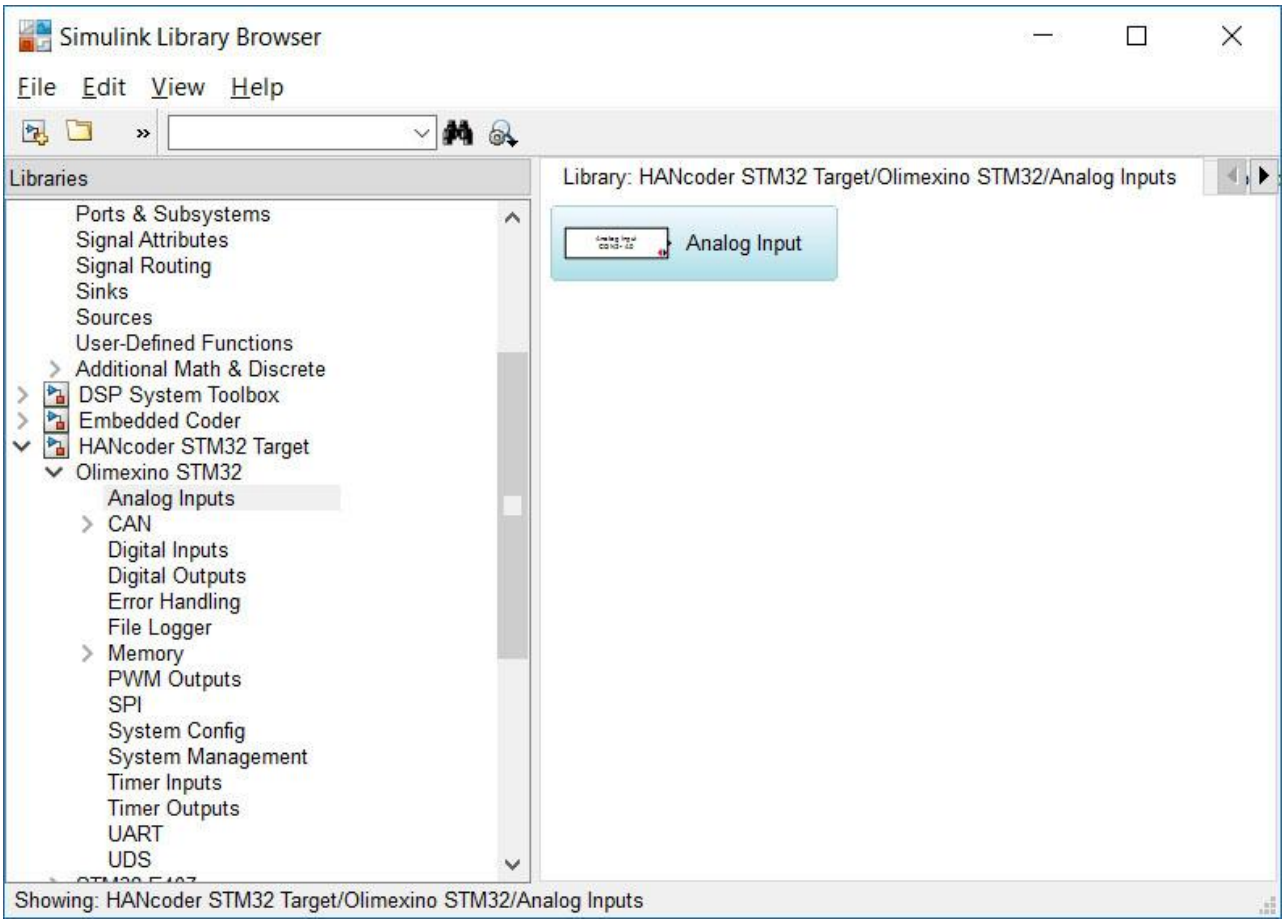


figure 2-6 HANcoder STM32 Olimexino Library: analog inputs

Click and hold the block called **Analog Input** and drag it into the model. Now go to **PWM Outputs** in the library and do the same for the **PWM Set Duty Cycle** and **PWM Init** block. You should end up with the following model:

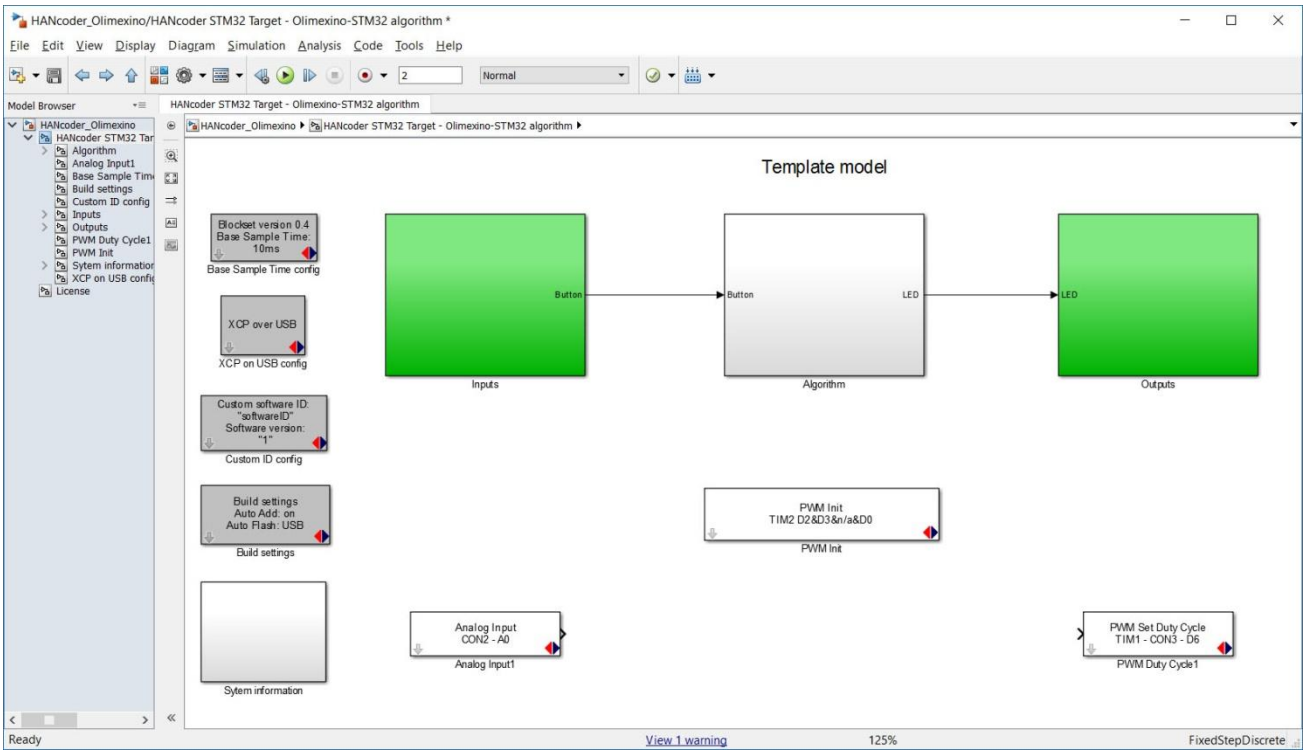


figure 2-7 Model with PWM and analog input

First we will set up the blocks for the right pin on the Olimexino. We start with the settings of the analog input. Open the settings dialog by double clicking on the block and select pin: **CON2 – A3** from the dropdown menu.

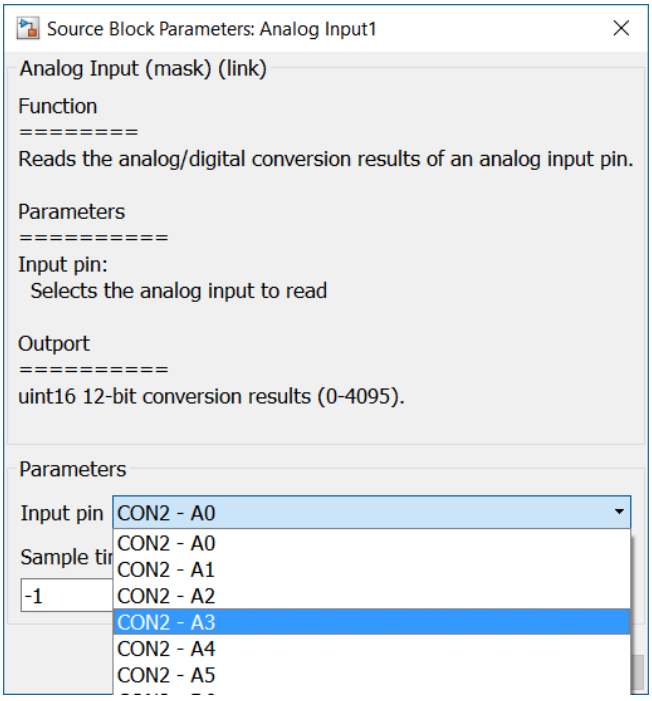


figure 2-8 Analog input parameter: CON2 - A3

It is not necessary to change the sample time. Note that the output is 0-4095. Save the settings by clicking **OK**.

Next up is the PWM Init block, double click the block to adapt the settings. scroll down until you will see the following screen:

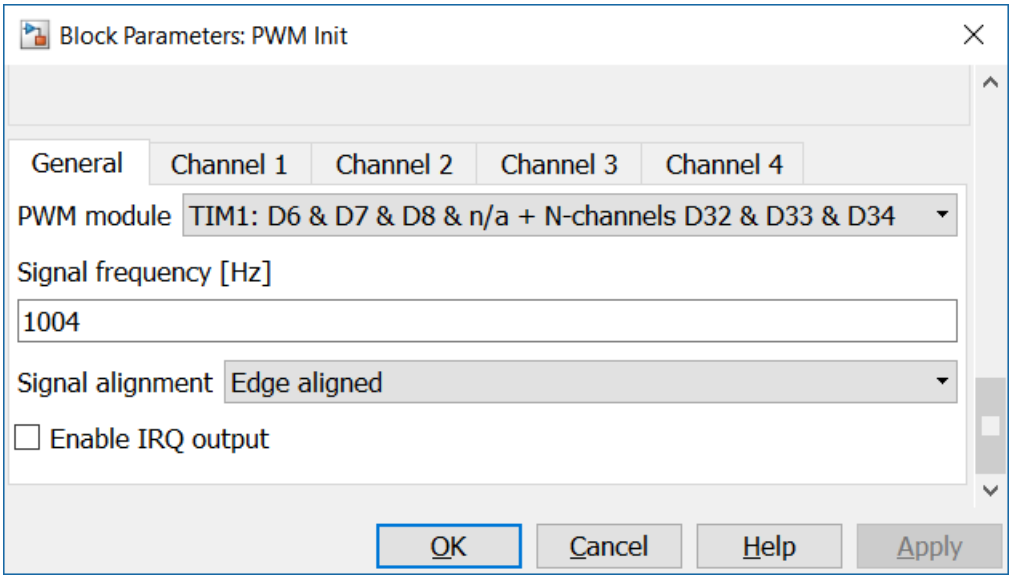


figure 2-9 PWM init Parameters

Note that the input is 0-1023.

The Ardumoto is connected to pin **D3** on the Olimexino so we have to choose **TIM2: D2 & D3 & n/a & D0** In order to turn on only that specific pin go to the tab of Channel 2 and select: **Generate PWM signal on channel 2** in the dropdown menu **PWM module** the pins are given in order of channel number; D2 is channel 1, D3 is channel 2, channel 3 is not available and D0 is channel 4

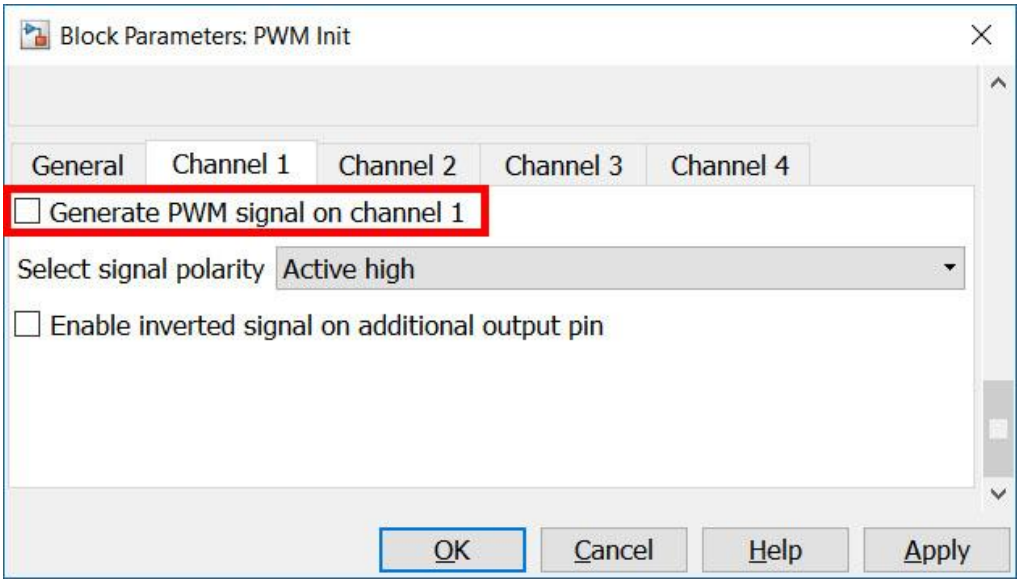


figure 2-10 Turn off PWM signal generation channels 1,3,4

Further we have to select a frequency for the PWM signal to drive the motor, a recommended frequency would be 10000Hz. Go back to the 'General' tab and replace 1004 by 10000. Close the dialog by clicking OK.

Up next is the PWM Duty Cycle block. Set the parameter **Output pin** to **TIM2 – CON3 – D3**

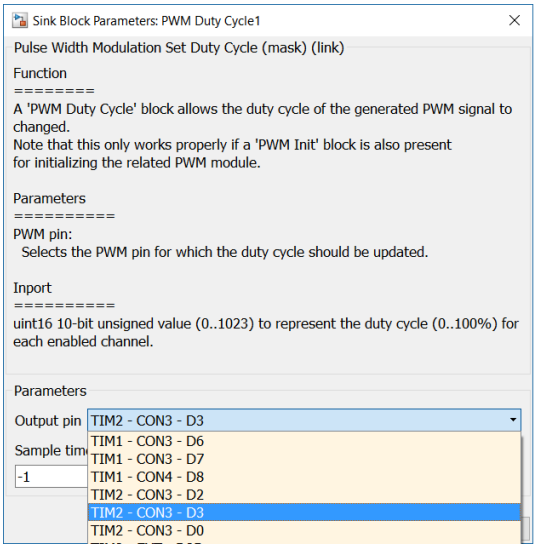


figure 2-11 Parameter setting PWM Duty Cycle

Next we will connect the two blocks with each other, however we have seen that the Analog input has a range of 0-4095 (0 to 3,3V) and the PWM output has a range of 0-1023 (0-100%). The value of the analog input is 4 times higher than the input necessary for the PWM Output. Therefore we need to divide the value of the analog input by 4, for this we add a gain block, this can be found in the Simulink library under Simulink -> Math Operations. Open the library, as previously described, and find the 'Gain' block and drag it into the model. Put it between the Analog Input block and the PWM Outputs block and connect the blocks by clicking and holding the left mouse button and drag the output of one block to the input of the other block. If the blocks are not properly connected the line will be red, the line becomes black when the blocks are properly connected.

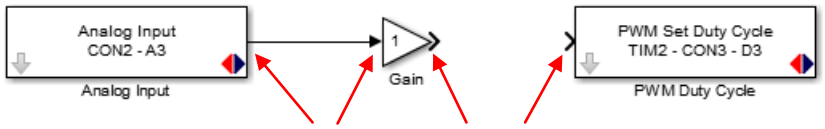
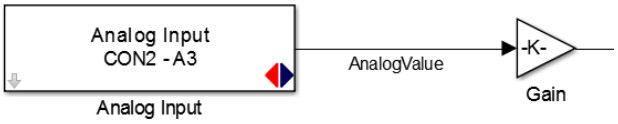


figure 2-12 connect the blocks

When the blocks are connected double click the gain and enter the value 0.25 or 1/4 for Gain. Go to the Signal Attributes tab and change the Output data type: from **Inherit: Inherit via internal rule** to **uint16**. When the output data type is set to 'Inherit: Inherit via internal rule' Simulink will choose calculate the appropriate data type itself. Because a fraction is used, Simulink will most probably use a fixed point data type. The input of the PWM Outputs block however needs to be an unsigned 16 bit unsigned integer(uint16), therefore we choose this manually. Close the dialog by clicking OK. Note that if the block is too small to show the parameter it will be replaced by -K-.

For monitoring the value of the potentiometer a name must be given to this signal. Double click the line going from the Analog Input to the Gain block and give it a name by typing. Do not press enter but simply click somewhere else in the model when done.

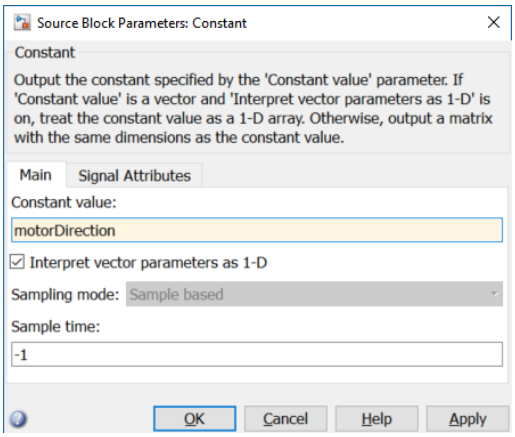


Now it is time to set up the model to control the motor direction.

Open the Simulink library browser by clicking the **library browser button** and go to the Olimexino STM32 in the 'HANcoderSTM32 Target' library. Drag a **Digital Output block** from the library to the model. Double click the block and select pin: **CON4 - D12**. Leave the configuration on **Push pull.** Close the dialog by clicking OK.

Next go back to the Simulink library browser and navigate to the Simulink standard library (the first library in the list). Drag a **constant** block into the model from the Simulink->Sources tab.

Connect the constant block with the Digital Outputs block. Double click the constant block and enter 'motorDirection' in the **Constant value** box. Set the sample time to -1. In the **Signal Attributes** tab set the output data type to **Boolean** and click **OK**



We are now ready to generate the code and test the program on the Olimexino. To start the code generation, click the **Build model** button in the toolbar of the model or press **ctrl+b**.

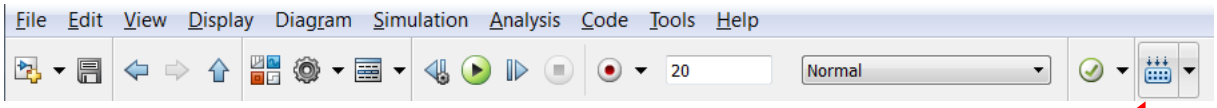
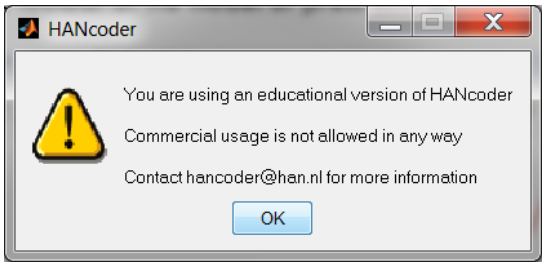
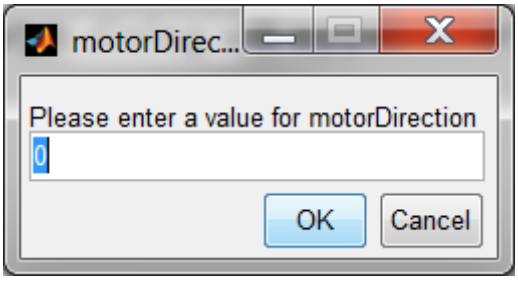


figure 2-13 The Build model button

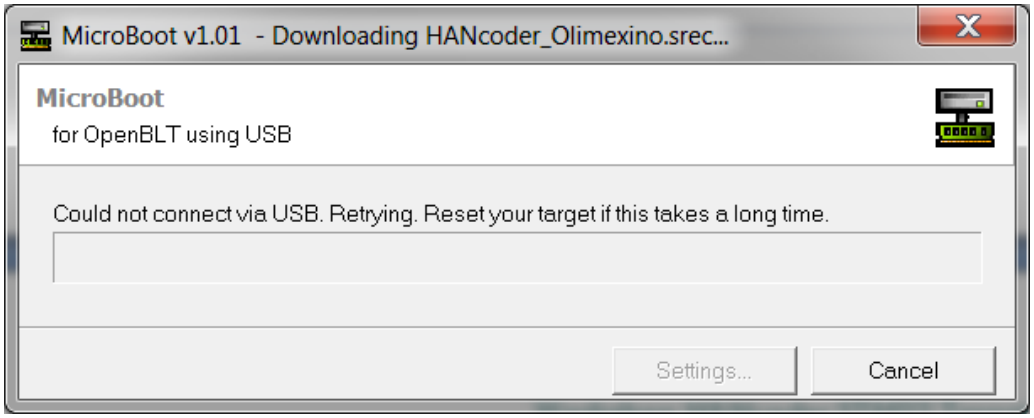


A warning will appear to indicate this software is only meant for use in non-commercial projects. Click OK to proceed.

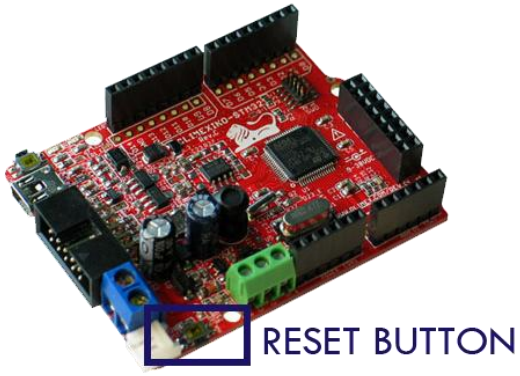
Next a pop-up will appear because the value of the constant motorDirection is still unknown. Leave the value '0' and click OK to continue. This will add the parameter to the workspace in Matlab. See the next step down below.



If the program was build successfully by Matlab, the program will be flashed automatically because of the **Build settings block** in your model. This means the flashing program, MicroBoot, will be started automatically with the right settings:



Press the reset button to start the flash procedure and check if the system works.



3 CONNECT WITH HANTUNE

Start HANtune by double clicking HANtune.exe.

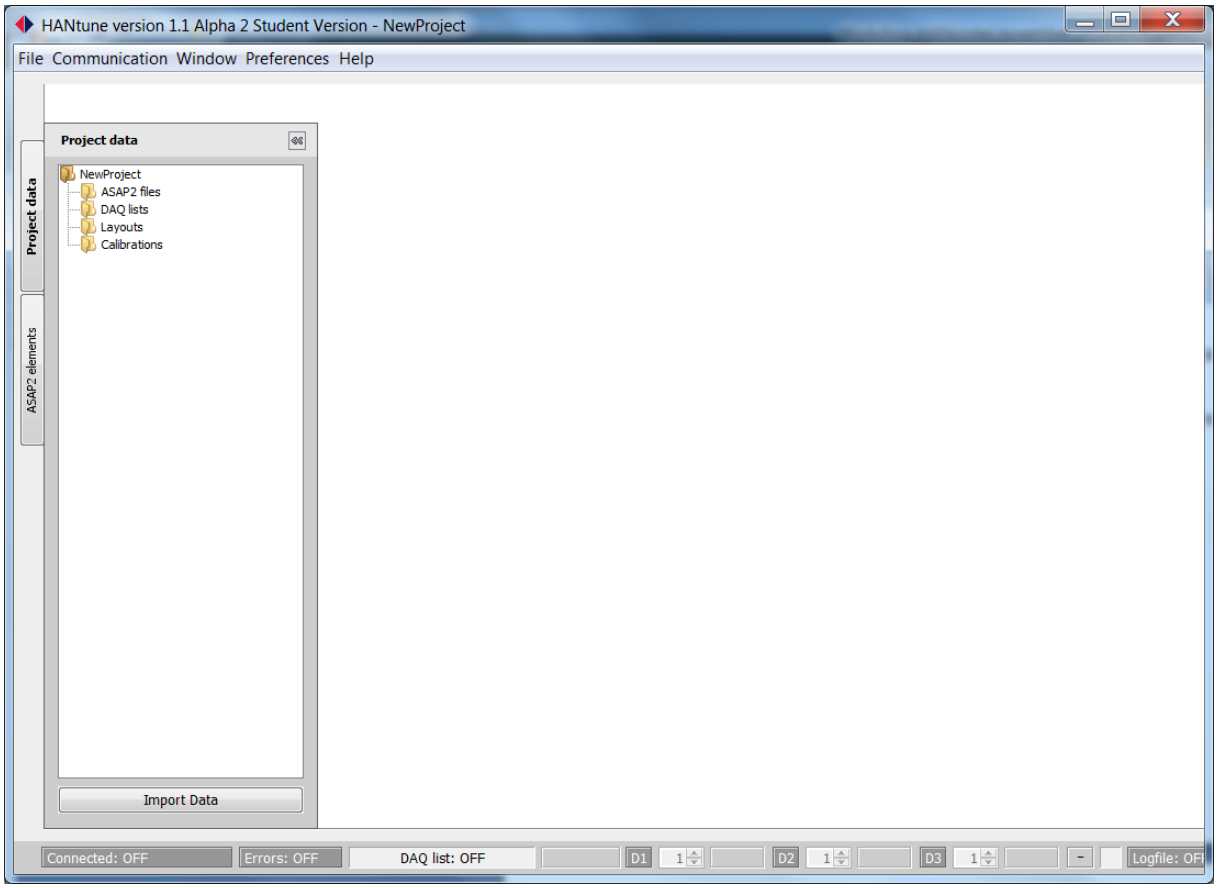


figure 3-1 HANtune

On the left in the tab **Project data** right click on the folder ASAP2 files and click 'Add ASAP2 file'. Choose the HANcoder_Olimexino.a2l file located in the same folder as the model file: 'Target' and click open.

The .a2l has been added to the project and appears in the window. Double click the file to load it, it will become bold when loaded.

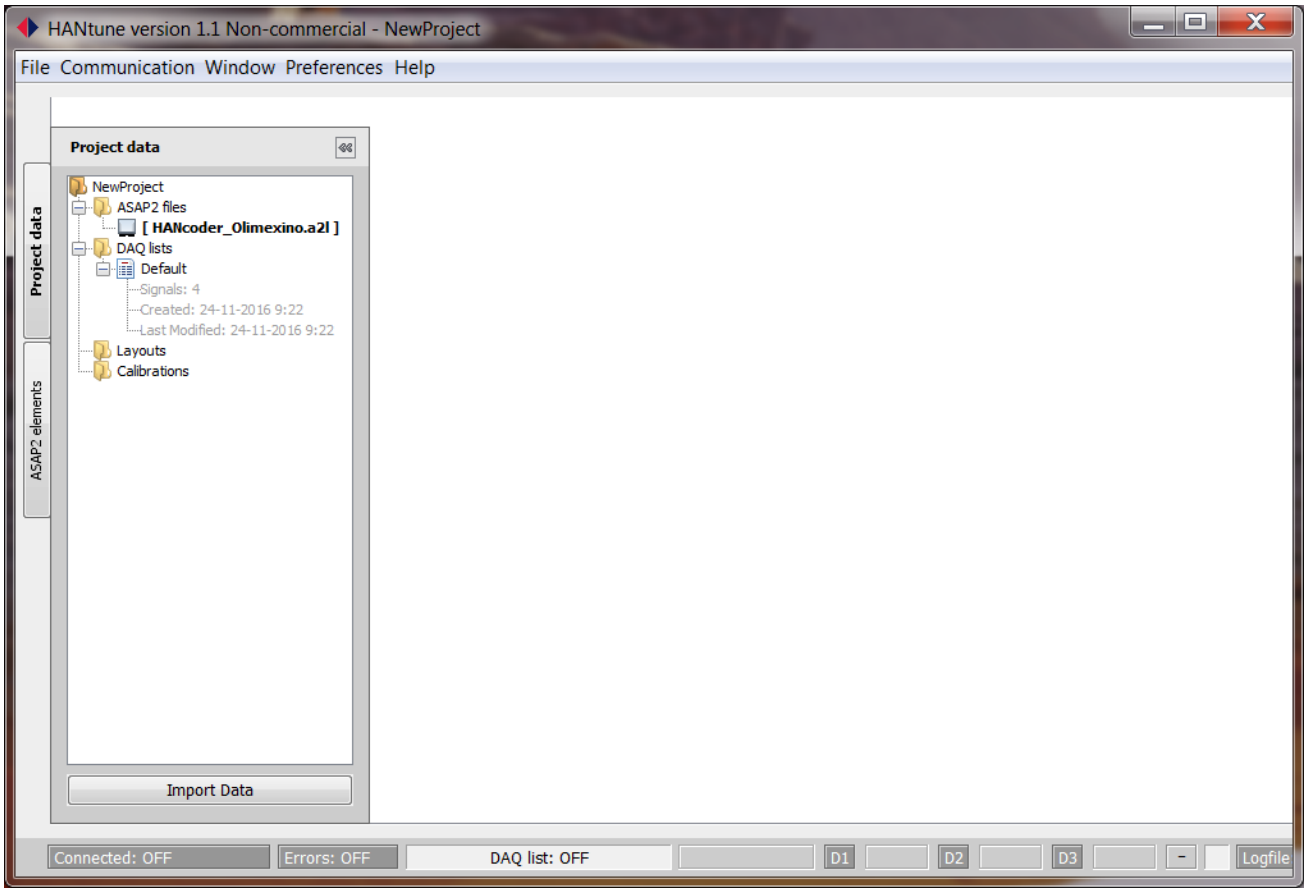


figure 3-2 HANtune ASAP2 file

Next it is necessary to create a Layout, this is done in a similar way as adding the ASAP2 file. Right click the Layouts folder and select 'New Layout'. Give the new layout a name and load it by right clicking on it and selecting 'Load layout' or by double clicking it.

Now we can start adding viewers to the layout. In the tab ASAP2 Elements located on the left on the screen all the signals and parameters are listed.

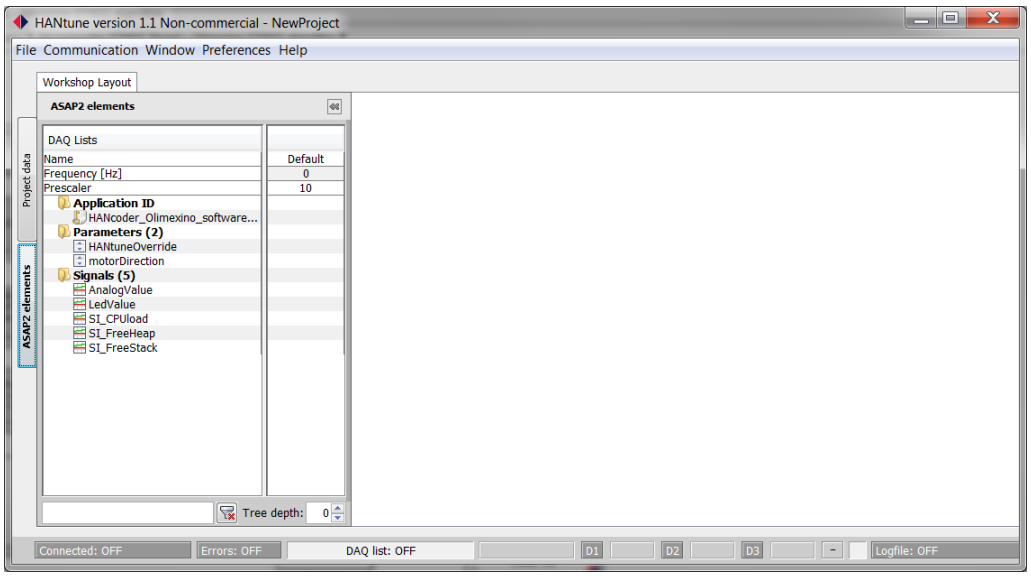


figure 3-3 Parameters and signal in HANtune

In order to add a viewer, right click on a signal and select the viewer of choice. In this example we will use the 'GaugeViewer' for the AnalogValue. The viewer is now automatically added to the layout. Because the data type is an unsigned 16-bit integer the viewer will automatically set its range to the minimum and maximum value of the data type, in this case 0 to 65535. The range of the viewer can be changed by right-clicking on it and selecting 'Modify display range'.

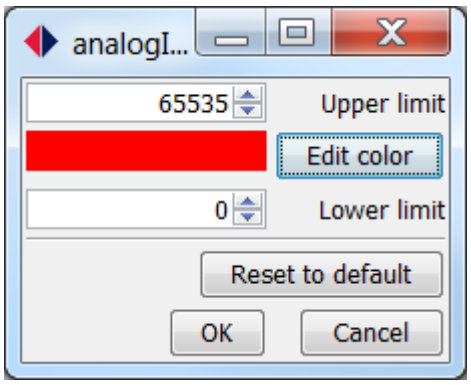


figure 3-4 settings gauge viewer

A dialog appears where the upper and lower limit of the GaugeViewer can be set. We know the minimum and maximum value of the Analog Input block are 0 and 4095 (from the description in the settings dialog) so we will set the maximum value to 4095.

With the 'Edit color' button you can change the color of the needle of the dial. Click OK to close.

Next we will add a MultiEditor by right-clicking on the parameter motorDirection and choosing 'MultiEditor'. The MultiEditor will appear in the screen on top of the GaugeViewer. It can easily be relocated to the desired position.

We now have finished our (very basic) layout and we are almost ready to connect to the Olimexino. Because HANtune can connect via CAN, USB and Ethernet (not for Olimexino) we first have to set up the right interface in HANtune. Go to **Communication → Communication Settings** and select **XCP on USB/UART** as communication driver from the drop down menu.

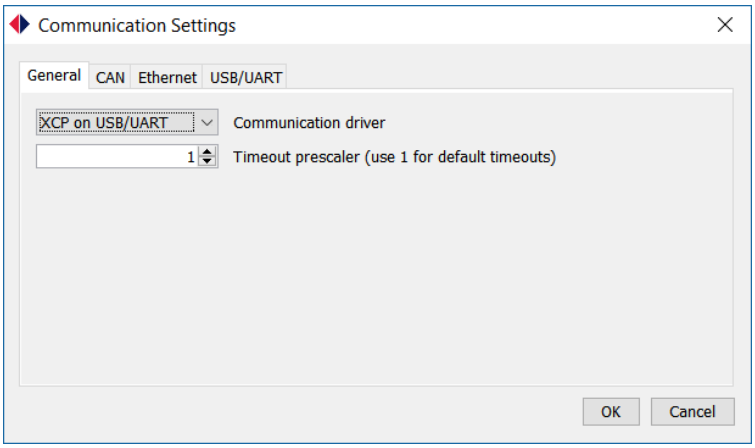


figure 3-5 HANtune communication settings

HANtune uses a virtual COM-port to connect to the Olimexino. To find out which COM-port is connected to your Olimexino, follow the following steps:

- Type **device manager** (dutch: apparaat beheer) in the search section of the startmenu and press enter.

- Expand **Ports (COM & LPT)** and search for **STMicroelectronics Virtual COM Port**

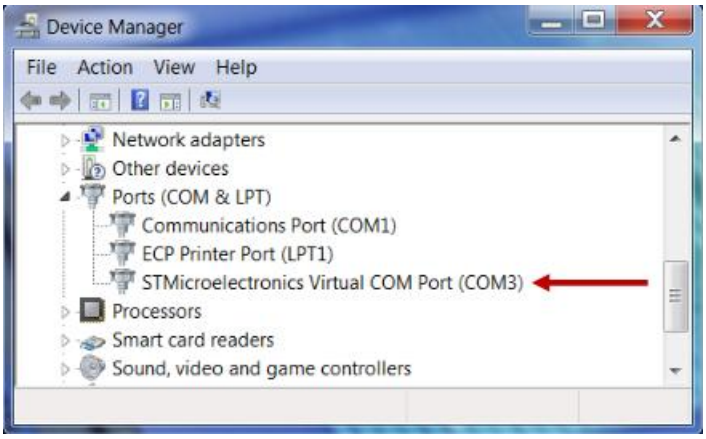


figure 3-6 Device Manager

Go back to HANtune and click on the tab UART (in communication settings) to set the right COM-port in the drop down menu UART port.

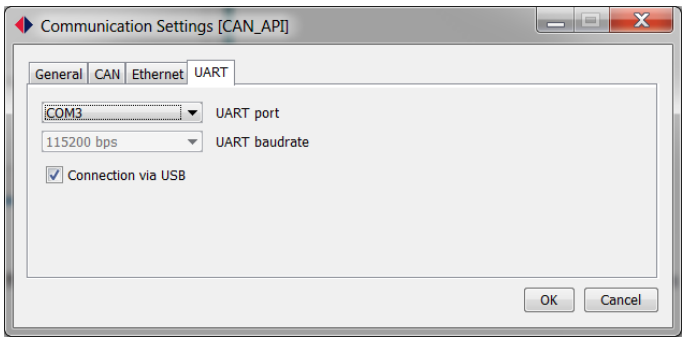


figure 3-7 Communication settings UART

We are now ready to connect to the Olimexino, this can be done in three different ways:

1. Press F5
2. Go to Communication -> Connect to XCP device in the menu bar
3. Click the connection indicator in the lower left corner of HANtune

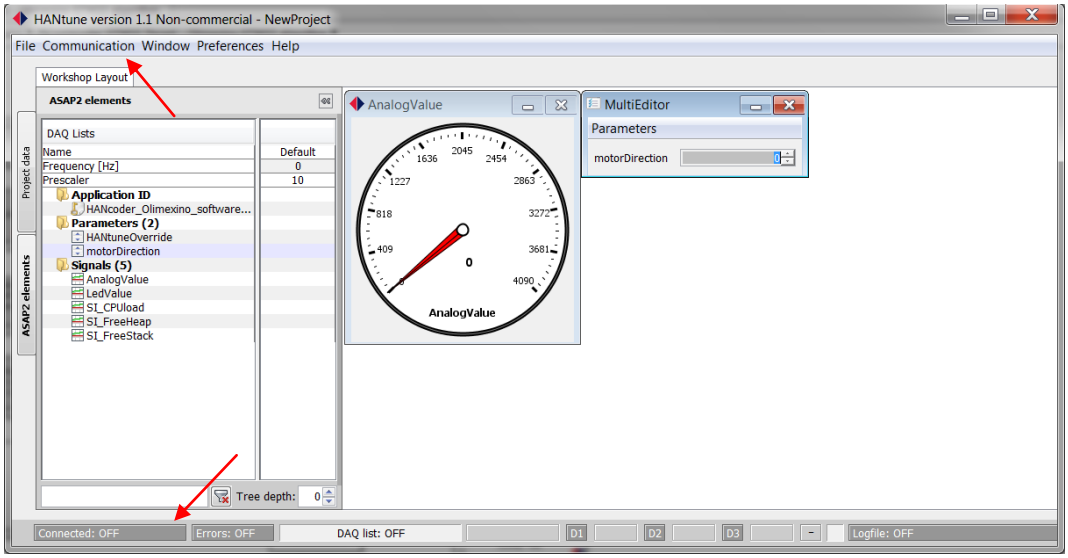


figure 3-8 connecting to the hardware (F5)

The connection dialog will now appear:

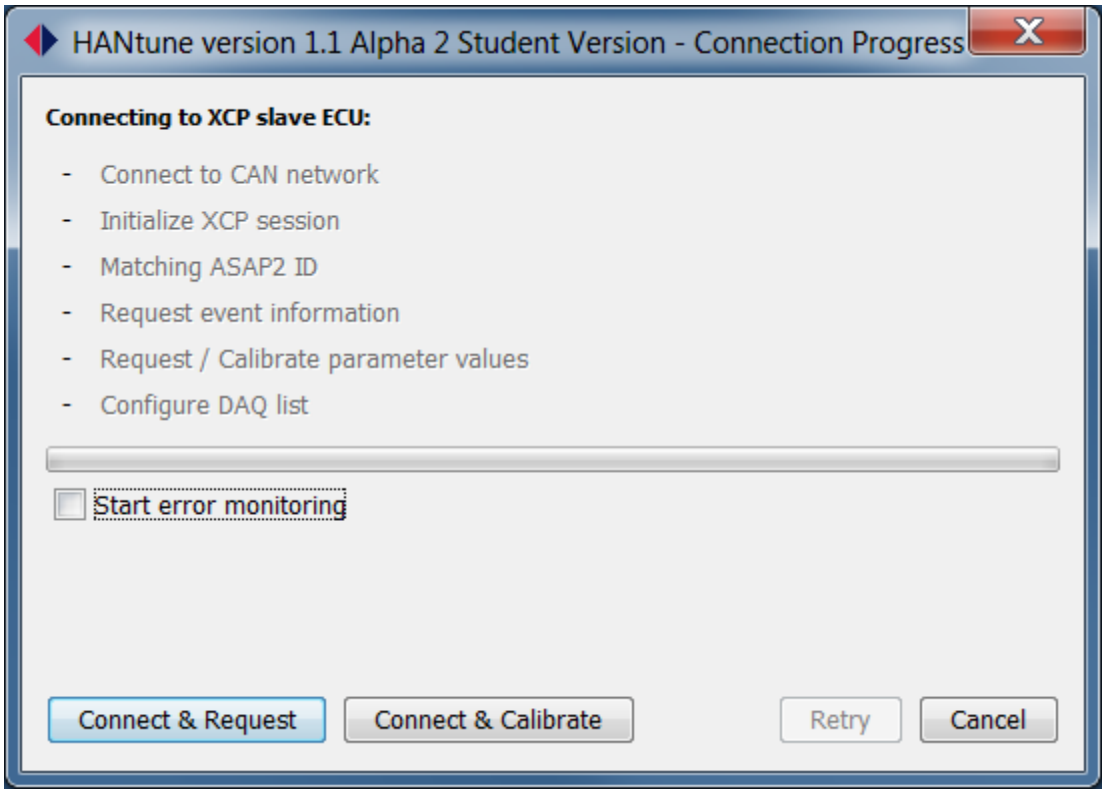


figure 3-9 Connection progress

By clicking 'Connect & Request' HANtune will connect to the Olimexino and it will request the values of the parameter, in this case only the value of 'motorDirection'.

By clicking 'Connect & Calibrate' HANtune will connect to the Olimexino and it will send the values of the parameters in the editors to the Olimexino.

It is usual to use 'Connect & Request' so we will use this option here as well.

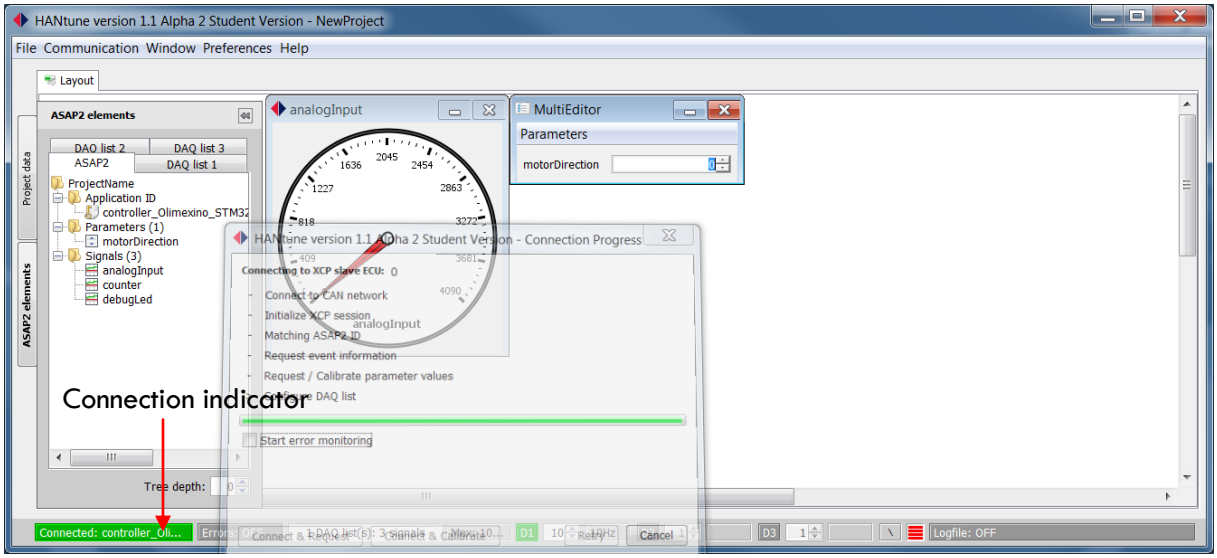


figure 3-10 Connection with target

As soon as HANtune has a connection the connection dialog will disappear and the connection indicator will turn green.

Now when you turn the knob on the potentiometer you can see the value of the analog input in HANtune. And when toggling the value of the motorDirection you can change the turning direction of the motor.

In the next section we will add a block to read the motor position.

The motor is connected to an encoder. In the blockset a special block is present to directly read from an encoder. Go to **HANcoder STM32 Target → Olimexino STM32 → Timer Inputs** in the '**Simulink Library Browser**' and drag the '**Quadrature Encoder Get**' block to the model. Also drag a **Terminator** block from **Simulink → Sinks** and a **Discrete Derivative** from **Simulink → Discrete** into the model.

figure 3-11 Quadrature Encoder Get, Discrete Derivative and terminator Block

Because the standard settings are already right there is no need to set up the Quadrature Encoder Get block. Double-click the Discrete Derivative block and change the Gain value to 0.01, this is done because the encoder gives around 14000 pulses per rotation and the speed would be very high with a gain value of 1.

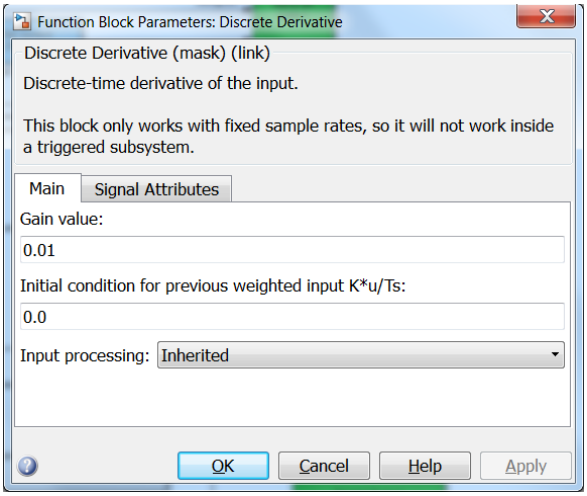


figure 3-12 Discrete Derivative parameters

In the Signal Attributes tab; change the Output Data Type to single because else MATLAB will automatically determine the 'appropriate' data type. Close the window by clicking 'OK'.

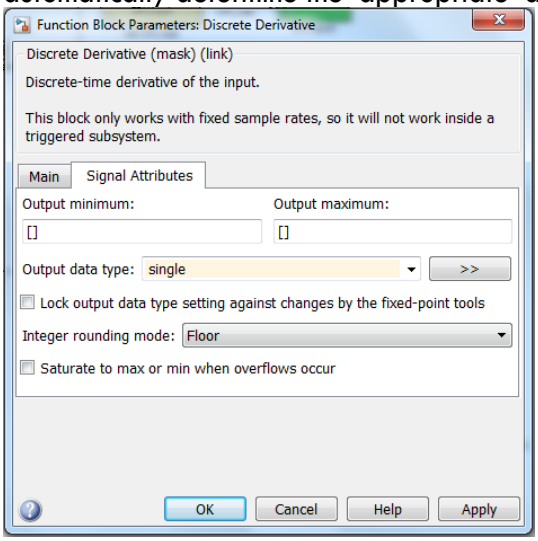


figure 3-13 Discrete Derivative: Signal attributes parameters

Now connect the output of the 'Quadrature Encoder Get' to the input of the 'Discrete Derivative' and connect the output of the 'Discrete Derivative' to the terminator.

Name the output signal of the Quadrature Encoder Get 'motorPosition' and the output of the Discrete Derivative 'motorSpeed'. Naming a signal is done by double clicking on the line representing the signal and typing the desired name.

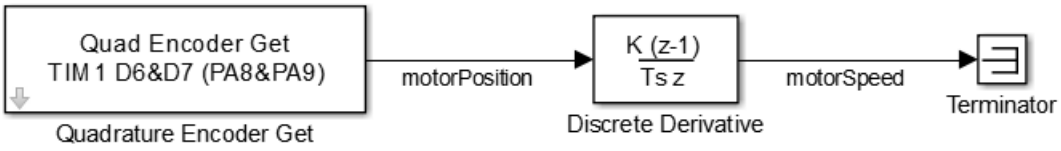


figure 3-14 Motor Position Model

Rebuild the model by clicking the **Incremental Build button** or pressing **Ctrl+b**. Once the build procedure is complete go to HANtune and disconnect from the Olimexino. To disconnect, click on the connection indicator or press **F5**. The connection indicator will become gray.

Now restart the Olimexino by pressing the reset button. The flashing should start automatically. Check if the motor still works.

In HANtune we will need to reload the ASAP2 file. This is done by right-clicking the ASAP2 file in the Project data tab and selecting '(un)Load file' and then right-clicking it again to load it by selecting 'Load file'.

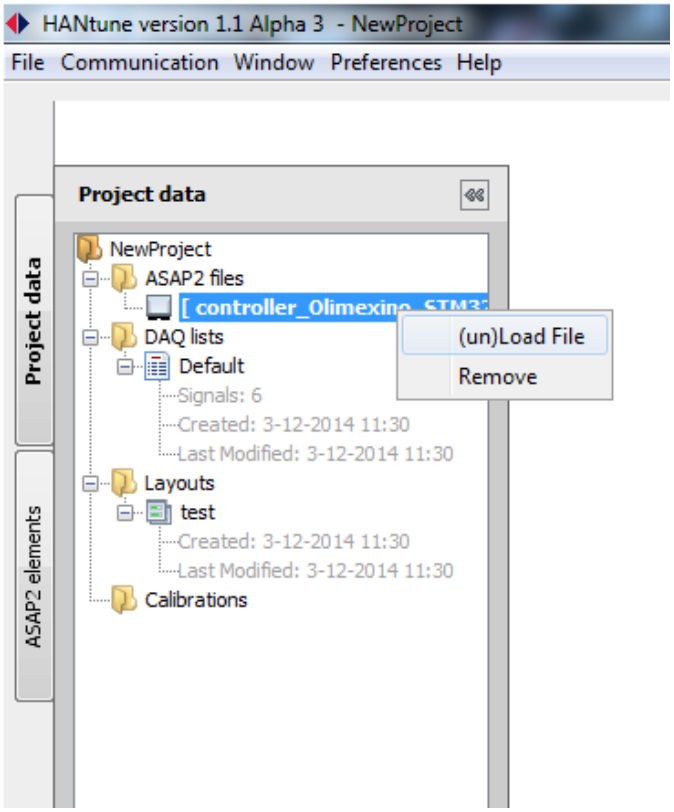


figure 3-15 load new ASAP2

Connect to the Olimexino by pressing F5 and select 'Connect & Request'.

Go to the ASAP2 elements tab and add a digital viewer for motorPosition and motorSpeed by right clicking on the signal name in the treeview and selecting DigitalViewer.

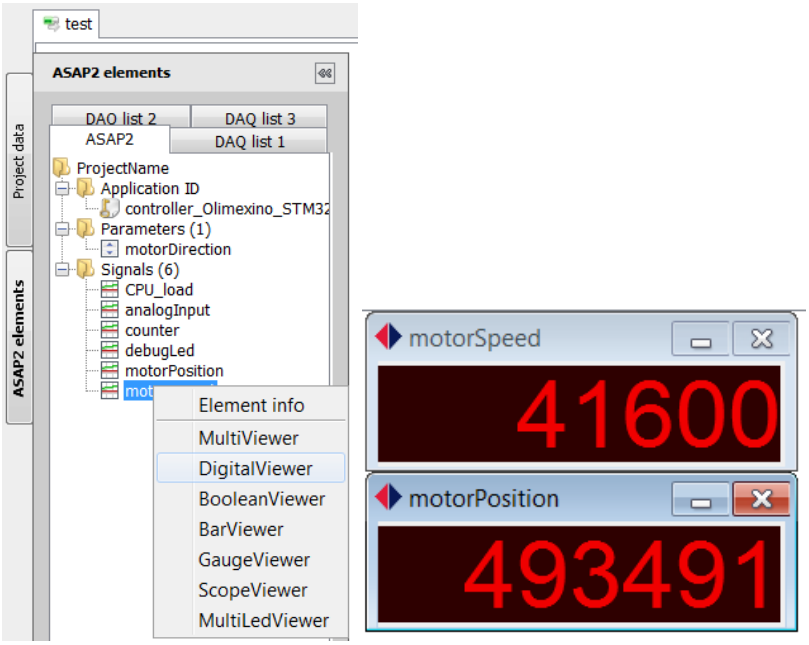
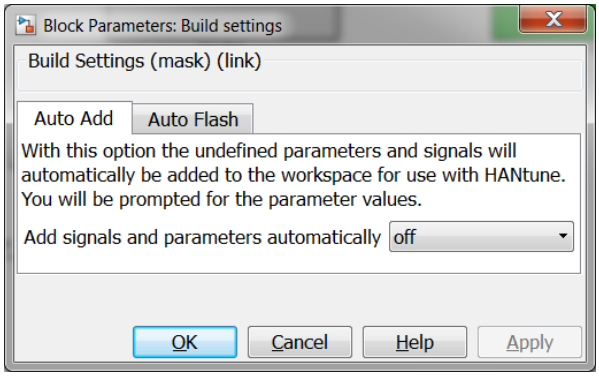


figure 3-16 Inserting signal to dashboard

Test your program by turning the potentiometer.

3.1 Working with an m-script

Up until now all the signals and parameters were automatically added in HANtune. This is done by the Auto Add function located in the Build Settings block. When this function is turned on signals with a name and parameters in constant blocks will automatically be added to the workspace so that they can be used in HANtune. In large projects it might be desirable to have more control over which signals and parameters end up in HANtune. To do this the Auto Add function has to be turned off.



Double click the **Build settings** block and turn off the Auto Add function.

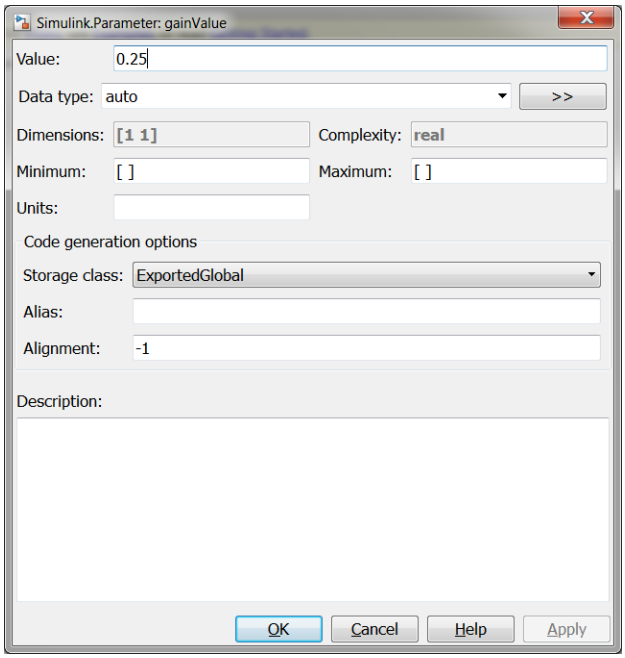
Next we will introduce a parameter which will be editable in HANtune. Double click the **Gain** block and change 0.25 or ¼ to gainValue. Press OK to close the dialog.

Try to build the model by pressing the incremental build button or ctrl+b. An error will appear stating that the 'gainValue' is not defined.

To define the parameter go to the Matlab Command window and type:

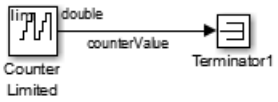
gainValue = Simulink.Parameter;

It will now show up in the workspace. Double click on the parameter name in the workspace to see its attributes:



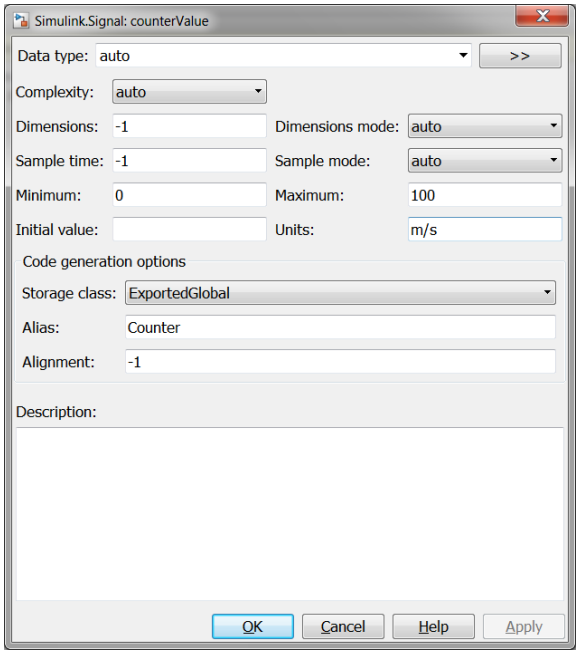
The **Value** field is still empty []. Enter the value 0.25, without the brackets. Another important setting is the **Storage class**, this should be set to Exported Global in order to be able to edit it in HANtune. Click OK to exit.

Next we will add another signal, a line in Simulink, for viewing in HANtune. Open the library browser again and add a **Counter Limited** block from the **Simulink →Sources** and a **Terminator** from the **Sinks** to the model. Connect the blocks and name the signal by double clicking it and typing: 'counterValue'. Double click the Counter Limited block and set the Upper Limit to 100 and the sample time to 0.1.



Now we must define the signal in the command window:

counterValue = Simulink.Signal;

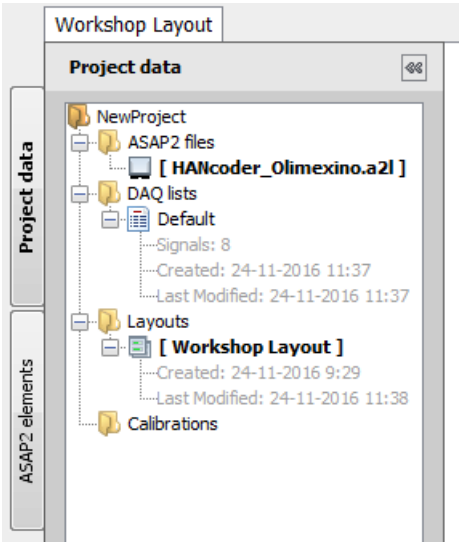


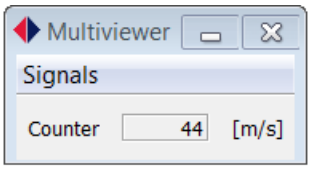
Double click the newly created signal in the workspace and set it up as shown. Note that only the **Storage Class** must be changed to make the signal visible in HANtune. The other changes are optional. Press OK to close and build the model once again by pressing Ctrl+b or the incremental build button.

Flash the controller by pressing the reset button.

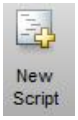
Go to HANtune and reload the a2l file by double clicking it twice in the project data tab. After this reload the Layout by double clicking it.

Now go to the ASAP2 elements tab and add a GaugeViewer for the newly added signal. Note that the name in HANtune is now Counter instead of counterValue. HANtune will automatically set the upper limit of the Counter to 100, just as defined in Matlab. If you add a MultiViewer for this signal as well you will see the Units.





Instead of doing this by hand every time, it would be easier to have a script to do this for you. To create such a script click the New Script button in Maltab.



Type (or copy) the following text:

```
%% This script defines the parameters and signals for the HANcoder
workshop
% Defining the parameter gainValue
gainValue = Simulink.Parameter;
gainValue.Value = 0.25;

% Defining the signal counterValue
counterValue = Simulink.Signal;
counterValue.StorageClass = 'ExportedGlobal';
counterValue.Min = 0;
counterValue.Max = 100;
counterValue.DocUnits = 'm/s';
counterValue.CoderInfo.Alias = 'Counter';
```

Save the script under any name you wish. This script defines gainValue and counterValue with exactly the same settings as previously done. Next time the model will be used the script can simply be run to define these parameters.

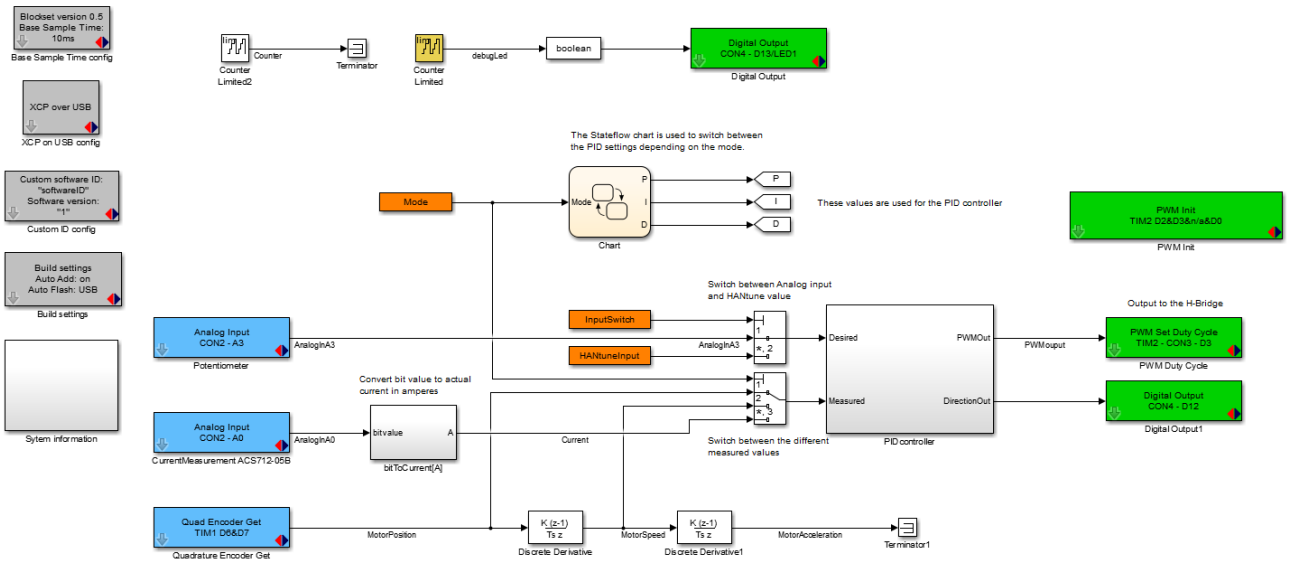
This concludes this part of the workshop.

4 OPTIONAL EXTRA

A control model for the workshop set-up can be found in the Target directory. This model can control the electric motor's position or speed. A torque control loop is also added for illustration but this requires a current sensor which is not present in the workshop set-up.

For convenience the model is already placed in the HANcoder STM32 Target folder of the workshop, together with the HANtune project(.hml) file.

Open the model by double clicking the DemoMotorController_Olimexino_STM32.slx file.



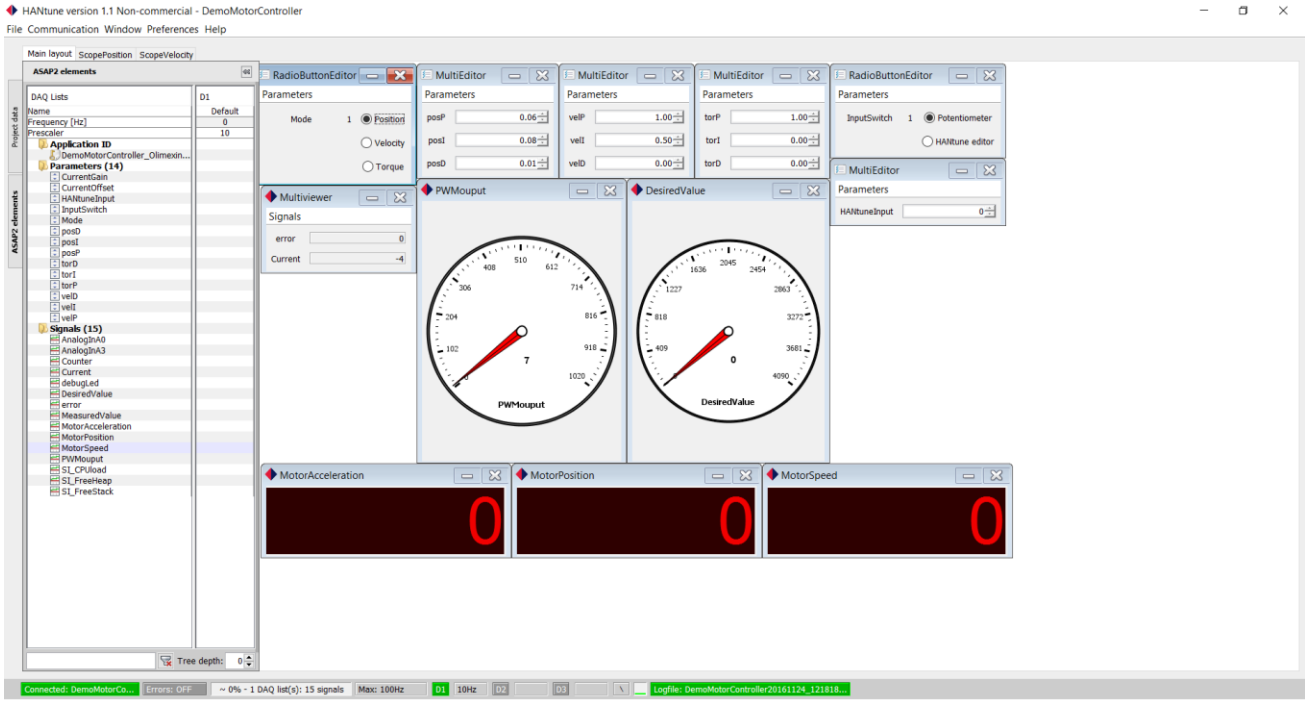
The model has the Analog input to read the potentiometer's position, the quadrature encoder input to measure its position, the PWM output to control the motor speed and the digital output to change the direction. The only things that are different are the addition of a Stateflow chart, a PID controller subsystem and an analog input to be able to read a current sensor. The model will not be described any further in this document because comments are present in the model itself.

Build the model by pressing the 'incremental build' button or pressing 'Ctrl+B'.

It is now time to tune the PID control loop with HANtune. Open the project 'DemoMotorController.hml' with HANtune by selecting 'Open project' from the 'File' menu or by pressing 'Ctrl+O'.

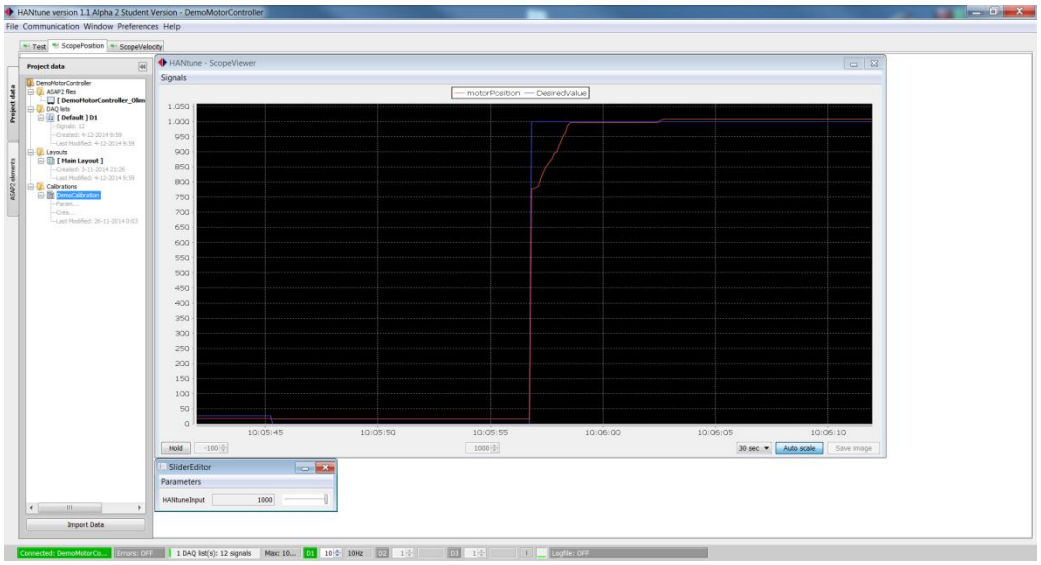
After opening the project load the a2l file and layout by double clicking on them. The last step is to connect to the Olimexino by pressing 'F5' and selecting 'Connect & Calibrate'. (The connection settings are already stored in this project)

The following should be visible now:



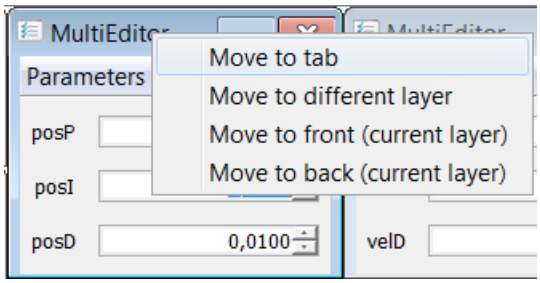
On the top left you can switch between the different control modes. In the top a MultiEditor is located for each control mode, here the values of the P,I and D actions can be tuned. On the top right the control for switching between the potentiometer and HANtune is located. Here you can choose to use the potentiometer as input for the PID controller or the value HANtuneInput from the MultiEditor. Change the input to HANtune and test it by changing the value in HANtuneInput.

Next click on the ScopePosition tab located on the top of the HANtune window. Here you can find a scope with the motor position and the desired position.



Change the value of HANtuneInput and check the response in the scope. Using the graph of the step response tune the PID controller so it gives a better performance.

You can add another MultiEditor to this tab with the PID settings for the position control. You can also move the MultiEditor from the Test tab by right clicking on its title bar and selecting ‘Move to tab’. It will then appear on the same location in the selected tab.



When you have found good settings for the PID values save an image of the step response by clicking 'Save Image' on the lower right of the ScopeViewer with the motorPosition and DesiredValue.

You can save the settings by creating a calibration. To create a calibration go to the Project data tab, this is where the ASAP2 file and Layout is also located. Right-click on the folder 'Calibrations' and click 'New calibration', select any name you like. The settings of all the Editors which are visible in HANtune are now saved. You can always switch back to this calibration by right-clicking it and selecting 'Load Calibration'. It is also possible to update the calibration when better settings are found by selecting 'Update Calibration'.

The final step of this workshop is to make a log file with HANtune and opening it in Microsoft Excel. First go to Preferences in the menu bar and click 'User Preferences'. Here you can select the location where to place the log file. You are free to change it to any location or just leave it as it is. To start a log file go to Communication and select 'Enable Datalogging' or click the log file indicator in the status bar. The logfile indicator should become green. Stopping the data logging can be done in the same way as starting it. The log file is saved as a comma separated values file (.csv) This file can be opened easily with Microsoft Excel or with MATLAB by the 'csvread' or 'uiimport' functions.