

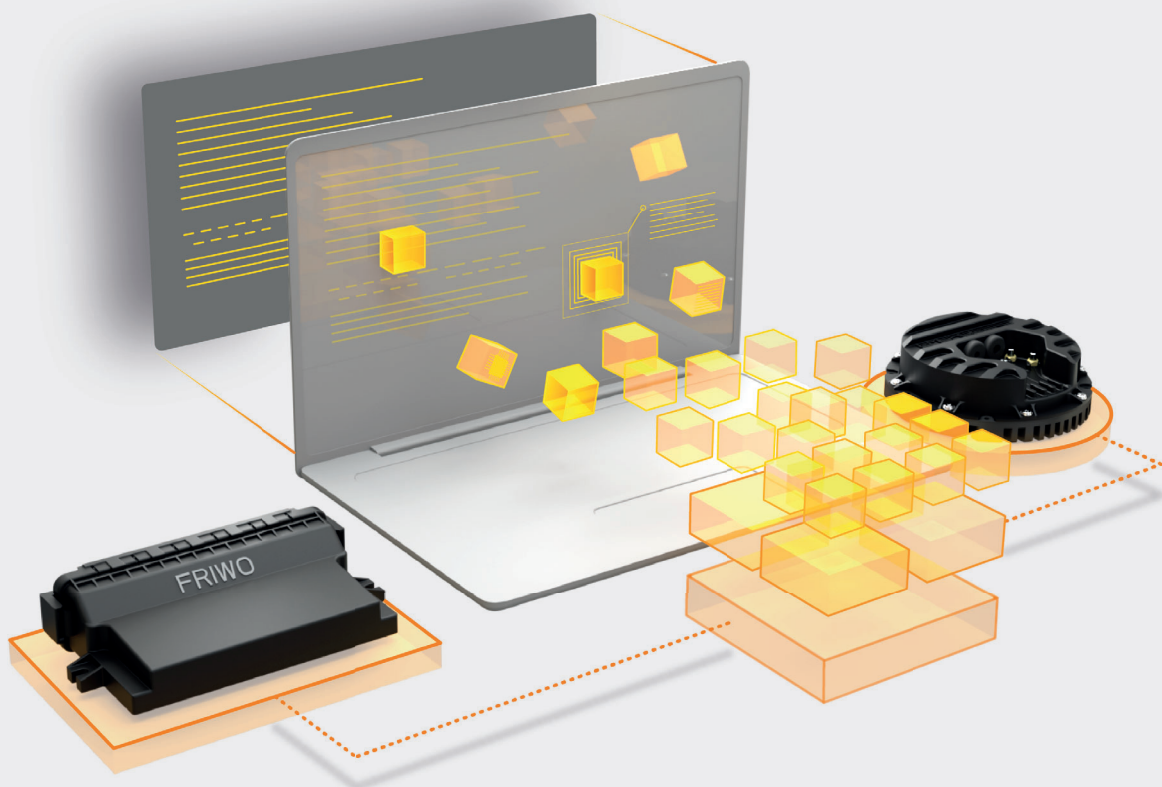


## SOFTWARE DEVELOPMENT KIT

End-to-end Development Environment Setup Solution

# APPLICATION GUIDE – HILL ASSIST

Version 2.0



## ABOUT

The **FRIWO SDK** enables the user to integrate own functionalities into a fully developed software environment for FRIWO motor controllers.

This document demonstrates how to implement a **Hill Assist** function by creating a customized TRQ\_DES-module. Basically, the Hill Assist prevents the vehicle from rolling backwards for a parameterizable time when starting on hill and releasing the brake pedal.

This guide gives a step-by-step overview of the implementation workflow from project creation to flashing and testing of the generated firmware on hard-

ware. Therefore, it assumes the **FRIWO SDK Tool Environment** to be set up already. For a guidance of the basic setup please refer to our **Quickstart Guide**.

<https://friwo.link/ag/quickstart-guide>



If you need a detailed description of the variable naming scheme, have a look at the **Software Manual**.

<https://friwo.link/ag/manual>



## SETUP REQUIREMENTS

Mechanical braking system must be available with analog output (0...3.3V), which is connected to the motor controller Analog Interface 2. Throttle pedal must be available with analog output (0...3.3V) connected to the motor controller Analog Interface 1.

For wiring information about our Motor Control Unit, see our **MCU data sheet**.

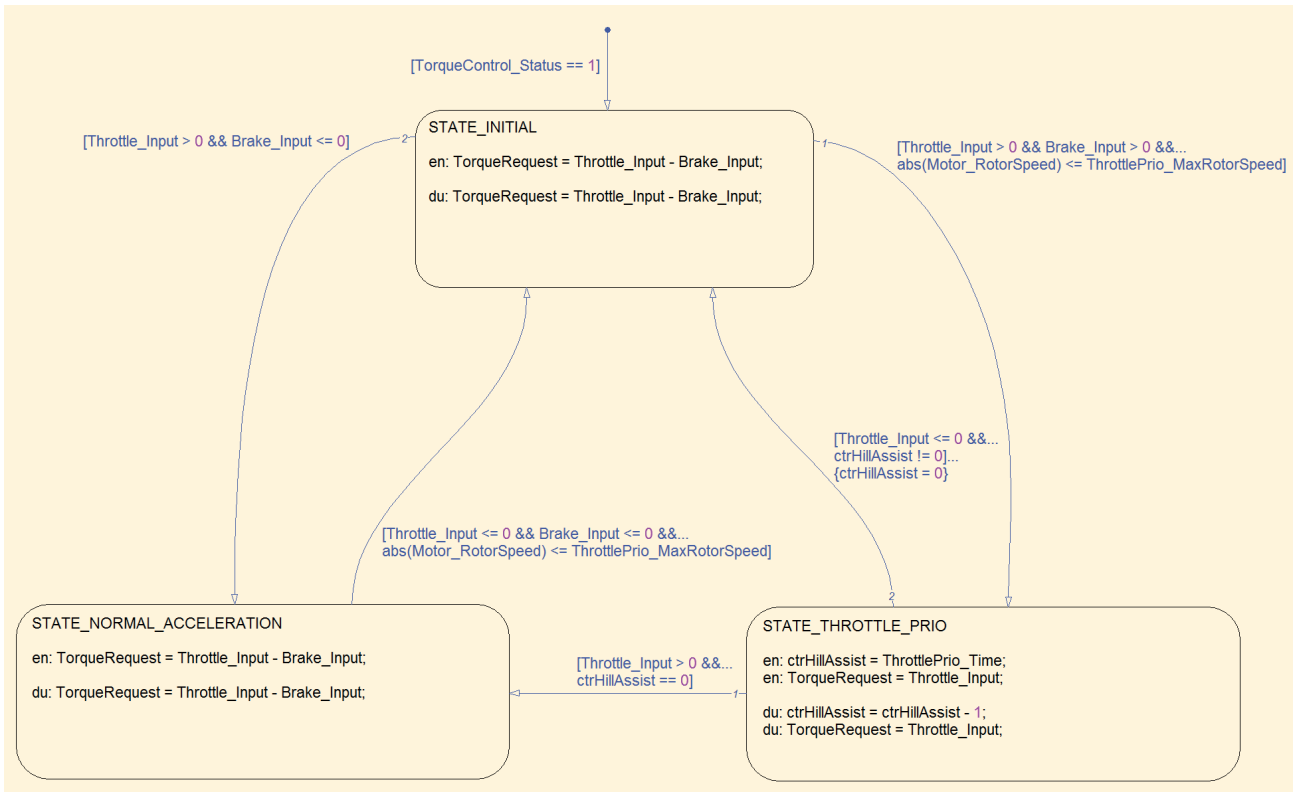
<https://friwo.link/ag/mcu>



## DESCRIPTION OF HILL ASSIST

For the Hill Assist application we use the throttle and brake signals from the analog channels, as well as the motor rotor speed as input signals to calculate the actual desired torque.

The Hill Assist is realized as a state machine which utilizes three states, illustrated in the following state-flow chart:



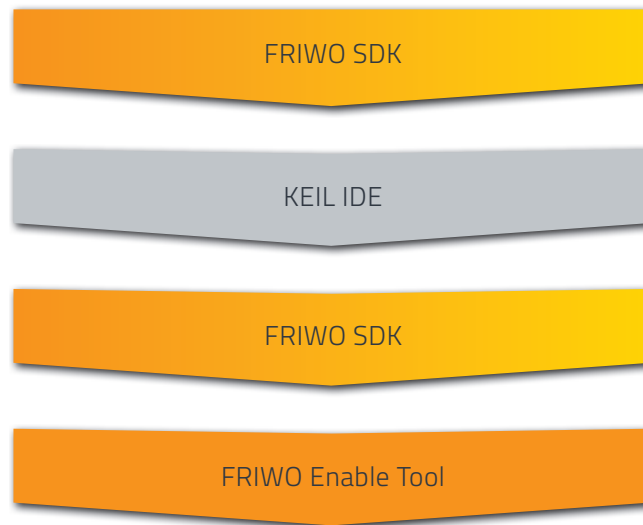
As soon as the torque control is activated, the state machine enters the state "STATE\_INITIAL". Now it depends, whether the driver is requesting a positive torque with brake released or with brake held. In the first case, the state machine jumps into state "STATE\_NORMAL\_ACCELERATION", where both throttle and brake are handled equally.

As soon as the counter ctrHillAssist reaches zero, meaning that the specified time has elapsed, the state machine jumps into state "STATE\_NORMAL\_ACCELERATION". If the time hasn't elapsed yet when throttle is released again, however, the state machine jumps back from state "STATE\_THROTTLE\_PRIO" to "STATE\_INITIAL".

If the driver wants to accelerate on hill, holding the brake while the motor rotor speed is below the specified threshold ThrottlePrio\_MaxRotorSpeed, the state machine jumps into the state "STATE\_THROTTLE\_PRIO". Here the throttle signal gets prioritized over the brake signal, so the requested torque is as high making acceleration on hill possible. The state machine remains in this state for a specified time value ThrottlePrio\_Time.

Remaining in state "STATE\_NORMAL\_ACCELERATION", the state machine enters state "STATE\_INITIAL", if throttle and brake both equal to zero and the rotor speed is below the specified threshold.

## THE BASIC WORKFLOW OF THE FRIWO SDK



We start, by creating a new project inside the **FRIWO SDK**. All necessary basic software gets pulled from the FRIWO Servers and made ready for usage.

In order to write your software, you need an IDE. We recommend **Keil IDE** or Visual Studio Professional.

After the software is written inside the IDE, we switch back to the **FRIWO SDK**. We start the compilation process and receive the customized firmware inside our workspace folder.

With the **FRIWO Enable Tool**, we can flash the customized firmware on the motor controller.

## CREATE A NEW PROJECT

To get the necessary source files for the TRQ\_DES-module a new project is created using the following steps:

- Open FRIWO SDK
- In main view press **Select Project**
- **Name** the project (i.e. HillAssist)
- Choose your **workspace folder** path (default: C:\Users\USERNAME\Documents\SDK\_Workspace)
- Choose framework **MCU FRIWO Standard V1.1**
- Press **Create**

Next, we define the module to be customized.

- In main view press **Select Module**
- Select module **TRQ\_DES** to be customized
- Confirm the dialog window
- Press **OK**

A new project folder is created in your workspace folder. The project's subfolder `.module_TRQ_DES` contains the c-files `TRQ_DES_custom.c` and `TRQ_DES_custom.h`, as well as a variable description file `TRQ_DES_variables.xml` and the header file `trqdesApi.h`.

## PREPARATION

Before we start programming the Hill Assist functionality, there has to be done some preparations of the source files of the TRQ\_DES-module. This is done by using the ANSI C IDE of your choice. The following procedure is explained using Keil  $\mu$ Vision4 IDE.

For the first step we are having a look at the header-file `trqdesApi.h`.

- Open **Keil  $\mu$ Vision4 IDE**
- Select **File->Open**
- Navigate to the project's workspace and there inside the subfolder `.\module_TRQ_DES`
- Select **trqdesApi.h**
- Press **Open**

The header-file `trqdesApi.h` describes the API of your custom module to the rest of the application software. It lists some type definitions as well as all Get-/Set-Functions, which can be used to access defined variables:

```
/*~~~~~*/
/* PUBLIC FUNCTION PROTOTYPES */
/*~~~~~*/

/* Function prototypes to GET variables from other modules */
Float32 trqdesApi_Get_APP_Brake_Signal_Channel(void);
Float32 trqdesApi_Get_APP_Reverse_Gear_Signal_Channel(void);
Float32 trqdesApi_Get_APP_Throttle_Signal_Channel(void);
Float32 trqdesApi_Get_AIN1_Throttle(void);
Float32 trqdesApi_Get_AIN2_Throttle(void);
Float32 trqdesApi_Get_CAN_EXT_Reverse_Gear(void);
Float32 trqdesApi_Get_CAN_EXT_Torque_Request(void);
Float32 trqdesApi_Get_DIN_DIN1_Signal(void);
Float32 trqdesApi_Get_DIN_DIN2_Signal(void);
Float32 trqdesApi_Get_PWM_Throttle(void);
Float32 trqdesApi_Get_INFO_Rotor_Speed(void);
Float32 trqdesApi_Get_APP_Dispatch_Mode(void);
Float32 trqdesApi_Get_SM_OUT_SYS_Trq_Control(void);
Float32 trqdesApi_Get_IHS_Vibration_Detected(void);

/* Function prototypes to SET variables for other modules */
void trqdesApi_Set_TRQ_DES_Driver_Throttle(Float32);
void trqdesApi_Set_TRQ_DES_Driver_Brake(Float32);
void trqdesApi_Set_TRQ_DES_Driver_Reverse_Gear(Float32);
void trqdesApi_Set_TRQ_DES_Trq_Req_Rel(Float32);
```

**Note:** The header-file should not be changed.

In order to get the analog signal for throttle calculation, the following syntax has to be used in your custom code:

```
userVariable = trqdesApi_Get_AIN1_Throttle();
```

Similar procedure has to be done when an interface variable is set:

```
trqdesApi_Set_TRQ_DES_Driver_Throttle(userVariable);
```

Next, we open the c-file TRQ\_DES\_custom.c:

- Select **File->Open**
- Navigate to the project's subfolder .\module\_TRQ\_DES
- Select **TRQ\_DES\_custom.c**
- Press **Open**

The file template is divided into code sections and already contains all basic include-Statements as well as the public function TRQ\_DES\_custom() which are needed to be integrated into the application software framework.

As described before, the Hill Assist will be implemented using the three states STATE\_INITIAL, STATE\_NORMAL\_ACCELERATION and STATE\_THROTTLE\_PRIO. Therefore, in section `/* PRIVATE TYPEDEF */` we define the datatype hillAssistState\_TypeDef as enumeration including the mentioned states:

```
/*~~~~~*/
/* PRIVATE TYPEDEF */
/*~~~~~*/

/**
 * @brief Define different states for hill-assist state machine as enumeration.
 */
typedef enum
{
    STATE_INITIAL, /**< @brief Initial state after start-up; decides whether
    to accelerate in assisted or normal mode depending on throttle/brake input
    and rotor speed. */
    STATE_THROTTLE_PRIO, /**< @brief Priorizes throttle over brake input for
    a certain time (counter) in order to generate enough torque for
    acceleration on hill. */
    STATE_NORMAL_ACCELERATION /**< @brief Add up both throttle and brake
    inputs to calculate desired torque. State is left, if both inputs equal
    zero and rotor speed is below threshold. */
}hillAssistState_TypeDef;
```

In section `/* PUBLIC VARIABLES */` we define global variables which are accessible by Enable Tool and can be divided into two types:

- **Display** variables: Read only access (`EMERGE_DISP_RAM`)
- **Calibration** variables: Read and write access (`EMERGE_NV_RAM_PAGE1`)

The differentiation is done by using the keyword `__attribute__` to provide the defined variables with the property to be stored in a specific RAM page or section.

- Define the following custom display variables using syntax

```
__attribute__((section("EMERGE_DISP_RAM")))
```

```

/*~~~~~*/
/* PUBLIC VARIABLES */
/*~~~~~*/

/**
 * Define variables to be displayed in FRIWO EnableTool Application.
 * Section EMERGE_DISP_RAM: Application data which will be read from Flash.
 * All data types which can be chosen: Int8, Int16, Int32, UInt8, UInt16, UInt32,
 Bool and Float32.
 */
__attribute__((section("EMERGE_DISP_RAM")))
volatile Float32 TRQ_DES_Throttle_Input; /*
    Description: Throttle signal value after selection of input channel [%] */

__attribute__((section("EMERGE_DISP_RAM")))
volatile Float32 TRQ_DES_Brake_Input; /*
    Description: Brake signal value after selection of input channel [%] */

__attribute__((section("EMERGE_DISP_RAM")))
volatile Float32 TRQ_DES_ReverseGear_Input; /*
    Description: Shows if reverse gear is selected after selection of input
channel; 0 = forward gear selected; 1 = reverse gear selected */

__attribute__((section("EMERGE_DISP_RAM")))
volatile Float32 TRQ_DES_TorqueRequest; /*
    Description: Shows the desired torque request returned to trqdesApi [%]; */

__attribute__((section("EMERGE_DISP_RAM")))
volatile UInt8 TRQ_DES_TorqueRequest_UpperLim; /*
    Description: Shows if desired torque has reached the upper bound of
allowed operational range */

__attribute__((section("EMERGE_DISP_RAM")))
volatile UInt8 TRQ_DES_TorqueRequest_LowerLim; /*
    Description: Shows if desired torque has reached the lower bound of
allowed operational range */

__attribute__((section("EMERGE_DISP_RAM")))
volatile UInt8 TRQ_DES_HillAssist_State; /*
    Description: Shows the actual state of hill-assist algorithm;
0 = Initial state; 1 = Throttle prioritization;
2 = Accelerate without prioritization; */

__attribute__((section("EMERGE_DISP_RAM")))
volatile UInt16 TRQ_DES_HillAssist_ValCounter; /*
    Description: Shows the actual value of hill-assist counter to prioritize
throttle */

```



- Define the following custom calibration variables using syntax

```
__attribute__((section("EMERGE_NV_RAM_PAGE1")))
```

```
/**
 * Define variables to be calibrated with FRIWO EnableTool Application.
 * Section EMERGE_NV_RAM_PAGE1: Standart application data which will be stored in
 flash when writing a snapshot.
 * All data types which can be chosen: Int8, Int16, Int32, UInt8, UInt16, UInt32,
 Bool and Float32.
 */
__attribute__((section("EMERGE_NV_RAM_PAGE1")))
volatile UInt8 TRQ_DES_C_ReverseGear_TestInput = 0u; /*
    Description: Test parameter for manual input of reverse gear signal [-];
    Limits: 0...1 */

__attribute__((section("EMERGE_NV_RAM_PAGE1")))
volatile UInt16 TRQ_DES_C_ThrottlePriorization_Time = 10000u; /*
    Description: Parameter for time during which throttle will be prioritized
 when both brake and throttle
 pedal are used in parallel [ms]; Limits: 0...65535 */

__attribute__((section("EMERGE_NV_RAM_PAGE1")))
volatile Float32 TRQ_DES_C_ThrottlePriorization_MaxRotorSpeed = 2.F; /*
    Description: Parameter for maximum rotor speed to prioritize throttle when
 both brake and throttle
 pedal are used in parallel [1/s]; Limits: -1...2000 */
```

Next, we define a private function for saturation of Float32 signals. This function should receive the lower and upper bound value as well as the signal to be saturated itself. The return value is the saturated signal.

- Declare the function prototype *sigSaturation* in section */\* PRIVATE FUNCTION PROTOTYPES \*/*:

```
/*~~~~~*/
/* PRIVATE FUNCTION PROTOTYPES */
/*~~~~~*/

/* Saturation function to saturate intermediate results and return parameters */
Float32 sigSaturation(Float32, Float32, Float32);
```

- Define the private function `sigSaturation` in section `/* PRIVATE FUNCTIONS */`:

```
/*~~~~~*/
/* PRIVATE FUNCTIONS */
/*~~~~~*/

/**
 * @brief Saturate input signal by the transferred lower and upper limits.
 * @param sigLowerLimit: Lower limit of input signal used for saturation.
 * @param sigUpperLimit: Upper limit of input signal used for saturation.
 * @param sigInput: Input signal to be saturated.
 * @return saturated value of input.
 */
Float32 sigSaturation(Float32 sigLowerLimit, Float32 sigUpperLimit, Float32
sigInput){
    Float32 sigOutput = 0.F;

    if (sigInput > sigUpperLimit) {
        sigOutput = sigUpperLimit;
    }
    else {
        if (sigInput < sigLowerLimit) {
            sigOutput = sigLowerLimit;
        }
        else {
            sigOutput = sigInput;
        }
    }
    return sigOutput;
}
```

## IMPLEMENT HILL ASSIST

At this point we're done with preparation. Now we can stick to the public function `TRQ_DES_custom()` and focus on the Hill Assist itself. First we need to get all relevant variables from the API as input for our module. Required inputs for the Hill Assist application are: Analog Inputs 1 and 2, Motor Rotor Speed and the State of Torque Control. Additionally, the state and counter variables have to be defined, which describe the procedure of the state machine. These variables are set to a default value.

- Get all relevant variables from API using the Get-functions specified in `trqdesApi.h` and store them in locally defined variables:

```
/*~~~~~*/
/* PUBLIC FUNCTIONS */
/*~~~~~*/

/**
 * @brief Public function for desired torque calculation which can be customized.
 *
 * This function is the first module of torque calculation and strategy, which is
 * called in basic firmware.
 * In this example, the function realizes a hill-assist functionality which can be
 * used in eScooter applications.
 * Necessary input values such as analog input signals from throttle and brake
 * pedal are received from trqdesApi through
 * Get functions. Additionally, there are four output signals which have to be set
 * since they are cross connected to
 * other modules in basic firmware.
 * @see trqdesApi.h file for more details about available Get and Set functions.
 */
void TRQ_DES_custom(void){

    /* Get desired signals from trqdesApi and store them in locally defined
    variables. */
    Float32 AnalogInput1_Signal = trqdesApi_Get_AIN1_Throttle();
    Float32 AnalogInput2_Signal = trqdesApi_Get_AIN2_Throttle();
    Float32 Motor_RotorSpeed = trqdesApi_Get_INFO_Rotor_Speed();
    Float32 TorqueControl_Status = trqdesApi_Get_SM_OUT_SYS_Trq_Control();
```

- Define local static variables for the state and counter value of the Hill Assist using the predefined enumeration datatype `hillAssistState_TypeDef`:

```
/* Define state and counter variables which show the current state of hill-
assist state machine. */
static hillAssistState_TypeDef stateHillAssist = STATE_INITIAL;
static UInt16 ctrHillAssist = 0u;
```

To be sure that the input values do not exceed certain levels, we use the private saturation function *sigSaturation()* to limit them accordingly. Within this scope we store the limited analog signal values to the global variables *TRQ\_DES\_Throttle\_Input* and *TRQ\_DES\_Brake\_Input*.

- Saturate the input values using private function *sigSaturation()*:

```
/* Saturate the input signals to the desired range */
TRQ_DES_Throttle_Input = sigSaturation(0.F, 100.F, AnalogInput1_Signal);
TRQ_DES_Brake_Input = sigSaturation(0.F, 100.F, AnalogInput2_Signal);
Motor_RotorSpeed = sigSaturation(-2400.F, 2400.F, Motor_RotorSpeed);
TorqueControl_Status = sigSaturation(0.F, 1.F, TorqueControl_Status);
TRQ_DES_ReverseGear_Input = sigSaturation(0.F, 1.F, (Float32)TRQ_DES_C_ReverseGear_TestInput);
```

Now the different states of the Hill Assist can be written down as illustrated in figure 1. We use a switch-case-statement to distinguish between the three different states. The state transitions are represented by if-else-statements within the case bodies. The torque request is calculated by throttle and brake input depending on the respective conditions, while the result is stored in the globally defined variable *TRQ\_DES\_TorqueRequest*.

As already mentioned in the description of the hill assist, we need the calibration parameter *TRQ\_DES\_C\_ThrottlePriorization\_MaxRotorSpeed* which implements the maximum rotor speed threshold when to jump to "STATE\_THROTTLE\_PRIO". Additionally, the remaining time for throttle priorization has to be calibrated by the parameter *TRQ\_DES\_C\_ThrottlePriorization\_Time*, which sets the reload value of the Hill Assist counter. Since the custom *TRQ\_DES* module is executed every 1ms, the time value must be specified in milliseconds.

Both parameters have been defined already in the previous section of this guide and must now be handled accordingly. For safety reasons the Hill Assist is only executed, if the status of torque control is active. Otherwise, torque request will be set to zero and state/counter variables will be set to their initial values.

- Implement state-machine of Hill Assist to be executed depending on torque control status:

```

/* Make sure that system is ready and torque control is active by checking the status flag for
torque control */
if(TorqueControl_Status == 1.F) {
  /* Implement the state-machine for the hill-assist using the three states STATE_INITIAL,
  STATE_THROTTLE_PRIO and STATE_NORMAL_ACCELERATION */
  switch(stateHillAssist) {
    case STATE_INITIAL: {
      /* When accelerating while brake is pulled and rotor speed is below threshold,
      reload hill-assist counter and jump to STATE_THROTTLE_PRIO */
      if (TRQ_DES_Throttle_Input > 0.F && TRQ_DES_Brake_Input > 0.F &&
      abs(Motor_RotorSpeed) <= TRQ_DES_C_ThrottlePriorization_MaxRotorSpeed) {
        TRQ_DES_TorqueRequest = TRQ_DES_Throttle_Input;
        stateHillAssist = STATE_THROTTLE_PRIO;
        TRQ_DES_HillAssist_State = (UInt8)stateHillAssist;
        ctrHillAssist = TRQ_DES_C_ThrottlePriorization_Time;
      }
      /* When accelerating without holding brake there is no prioritization of
      throttle input and jump to STATE_NORMAL_ACCELERATION */
      else if (TRQ_DES_Throttle_Input > 0.F && TRQ_DES_Brake_Input <= 0.F) {
        TRQ_DES_TorqueRequest = TRQ_DES_Throttle_Input - TRQ_DES_Brake_Input;
        stateHillAssist = STATE_NORMAL_ACCELERATION;
        TRQ_DES_HillAssist_State = (UInt8)stateHillAssist;
      }
      /* When there is no acceleration stay in STATE_INITIAL */
      else {
        TRQ_DES_TorqueRequest = TRQ_DES_Throttle_Input - TRQ_DES_Brake_Input;
        TRQ_DES_HillAssist_State = (UInt8)stateHillAssist;
      }
      break;
    }
    case STATE_THROTTLE_PRIO: {
      ctrHillAssist--;
      /* If still accelerating when counter reaches zero jump to
      STATE_NORMAL_ACCELERATION */
      if (TRQ_DES_Throttle_Input > 0.F && ctrHillAssist == 0u) {
        TRQ_DES_TorqueRequest = TRQ_DES_Throttle_Input - TRQ_DES_Brake_Input;
        stateHillAssist = STATE_NORMAL_ACCELERATION;
        TRQ_DES_HillAssist_State = (UInt8)stateHillAssist;
      }
      /* If throttle is released while counter still running reset counter and jump
      back to STATE_INITIAL */
      else if (TRQ_DES_Throttle_Input <= 0.F && ctrHillAssist != 0u) {
        ctrHillAssist = 0u;
        TRQ_DES_TorqueRequest = TRQ_DES_Throttle_Input - TRQ_DES_Brake_Input;
        stateHillAssist = STATE_INITIAL;
        TRQ_DES_HillAssist_State = (UInt8)stateHillAssist;
      }
      /* As long as counter has not reached zero prioritize throttle over brake input */
      else {
        TRQ_DES_TorqueRequest = TRQ_DES_Throttle_Input;
      }
      break;
    }
    case STATE_NORMAL_ACCELERATION: {
      /* If both pedals throttle and brake are released and rotor speed is below
      threshold, go back to STATE_INITIAL to reset hill-assist state machine */
      if (TRQ_DES_Throttle_Input <= 0.F && TRQ_DES_Brake_Input <= 0.F &&
      abs(Motor_RotorSpeed) <= TRQ_DES_C_ThrottlePriorization_MaxRotorSpeed) {
        TRQ_DES_TorqueRequest = TRQ_DES_Throttle_Input - TRQ_DES_Brake_Input;
        stateHillAssist = STATE_INITIAL;
        TRQ_DES_HillAssist_State = (UInt8)stateHillAssist;
      }
      /* Normal acceleration without prioritization after hill-assist counter has reached
      zero */
      else {
        TRQ_DES_TorqueRequest = TRQ_DES_Throttle_Input - TRQ_DES_Brake_Input;
      }
      break;
    }
  }
}
else {
  stateHillAssist = STATE_INITIAL;
  ctrHillAssist = 0u;
  TRQ_DES_TorqueRequest = 0.F;
}

```

- Visualize the actual state and counter value by writing them to the previously defined global variables which are accessed by FRIWO Enable Tool:

```
/* Show current state and counter value of hill-assist state machine */  
TRQ_DES_HillAssist_ValCounter = ctrHillAssist;  
TRQ_DES_HillAssist_State = (UInt8)stateHillAssist;
```

<https://friwo.link/ag/variable-description>

As also described in the [variable description document](#), the allowed range of the desired torque request is within -100% and 100%. Thus, the variable *TRQ\_DES\_TorqueRequest* has to be saturated to these boundaries. Additionally, we need to do some torque coordination depending on rotor speed and reverse gear selection since torque request can also be negative for regenerative braking.



Two different issues have to be considered:

1. The torque calculation up to this point is done only for the forward gear.
2. If motor is in standstill and we keep holding the brake: A negative torque is generated by the brake input, thus motor would start rotating backwards if there was no mechanical brake torque.

As a result of the first issue, we need to process the parameter *TRQ\_DES\_C\_ReverseGear\_TestInput* to distinguish between driving directions and invert torque request respectively.

Addressing issue number two we define mechanical rotor speed thresholds in forward and backward direction (-0.5...0.5 1/s), up to which no negative torque will be generated.

We use the private function `sigSaturation()` to limit the calculated torque request to the respective limits.

- Saturate the desired torque request `TRQ_DES_TorqueRequest` depending on reverse gear selection and motor rotor speed:

```

/**
 * Saturation of relative torque request depending on rotor speed and driving direction.
 * If motor rotor speed exceeds a certain level, regenerative braking is enabled.
 * Otherwise the motor rotor is just spinning out, without regenerative torque.
 */
if(Motor_RotorSpeed >= 0.5F) {
    /* Rotor spins in forward direction */
    if(TRQ_DES_ReverseGear_Input == 0) {
        /* Positive and negative torque possible for regenerative braking */
        TRQ_DES_TorqueRequest = sigSaturation(-100.F, 100.F, TRQ_DES_TorqueRequest);
    }
    else {
        /* Only negative torque if reverse direction selected and rotor spins forward */
        TRQ_DES_TorqueRequest = sigSaturation(-100.F, 0.F, TRQ_DES_TorqueRequest);
    }
}
else if(Motor_RotorSpeed <= -0.5F) {
    /* Rotor spins in negative direction */
    if(TRQ_DES_ReverseGear_Input == 0) {
        /* Only positive torque possible */
        TRQ_DES_TorqueRequest = sigSaturation(0.F, 100.F, TRQ_DES_TorqueRequest);
    }
    else {
        /* Only negative torque if reverse direction selected and rotor spins forward */
        TRQ_DES_TorqueRequest = sigSaturation(-100.F, 100.F, -TRQ_DES_TorqueRequest);
    }
}
else {
    /* Rotor is at standstill or close to it */
    TRQ_DES_TorqueRequest = sigSaturation(0.F, 100.F, TRQ_DES_TorqueRequest);
    if(TRQ_DES_ReverseGear_Input == 1) {
        /* Only negative torque if reverse direction selected */
        TRQ_DES_TorqueRequest = -TRQ_DES_TorqueRequest;
    }
}
}

```

Finally, we need to set the module's outputs using the `trqdesApi` as described above in section "Preparation". Even if we solely focused on the calculation of the desired torque request within this application, we need to return also the remaining outputs since they are cross connected to other modules and crucial for basic firmware execution. These are the throttle and brake input signals as well as the reverse gear input (see also **module description**).



<https://friwo.link/ag/module-description>

- Set module's outputs using the Set-functions of the `trqdesApi` to return the calculated values:

```

/**
 * Return calculated set values as module outputs to trqdesApi.
 * Besides desired torque TRQ_DES_TorqueRequest the following cross connections for state
 * management and system startup must be considered and set as well:
 * TRQ_DES_Driver_Throttle, TRQ_DES_Driver_Brake and TRQ_DES_Reverse_Gear
 */
trqdesApi_Set_TRQ_DES_Driver_Throttle(TRQ_DES_Throttle_Input);
trqdesApi_Set_TRQ_DES_Driver_Brake(TRQ_DES_Brake_Input);
trqdesApi_Set_TRQ_DES_Driver_Reverse_Gear(TRQ_DES_ReverseGear_Input);
trqdesApi_Set_TRQ_DES_Trq_Req_Rel(TRQ_DES_TorqueRequest);

```

- In Keil menu bar select **File->Save all** to save your file inside the project workspace

## VARIABLE DESCRIPTION

To display the previously defined global variables in FRIWO Enable Tool, we need to describe these variables in the variable description file TRQ\_DES\_variables.xml. This file can also be found in the workspace folder.

- Navigate to the project's subfolder .\module\_TRQ\_DES
- Open TRQ\_DES\_variables.xml with a text editor of your choice
- Add all global variables defined at the beginning of the source file TRQ\_DES\_custom.c (see also section "Preparation") using the following format for each:

```
<ddObj Name="TRQ_DES_Throttle_Input" Kind="Variable">
  <ddProperty Name="Description">Throttle input value after selection of input
    channel [%]; Limits: 0...100</ddProperty>
  <ddProperty Name="Type">Float32</ddProperty>
  <ddProperty Name="Scaling">./LocalScaling</ddProperty>
  <ddProperty Name="Value"></ddProperty>
  <ddProperty Name="Min">0</ddProperty>
  <ddProperty Name="Max">100</ddProperty>
  <ddProperty Name="Address"></ddProperty>
  <ddObj Name="LocalScaling" Kind="Scaling">
    <ddProperty Name="LSB">1</ddProperty>
    <ddProperty Name="Unit">s</ddProperty>
  </ddObj>
</ddObj>
```

**Note:** Make sure that the datatypes defined in the xml-file match with the variable declarations in your c-file. Otherwise data is not processed correctly.



## CREATING THE NEW FIRMWARE


At this point we have successfully prepared the c- and variable description files for implementation of the Hill Assist. In the following we use the FRIWO SDK to compile our individual module TRQ\_DES\_custom and integrate it into the basic software framework.

- Open FRIWO\_SDK (if not already open)
- In main view press **Select Project**
- Press **Load Project**
- **Navigate** to the project folder in your SDK workspace (i.e. .\SDK\_Workspace\HillAssist)
- Select the project file (i.e. HillAssist.sdkproj) and press **open**

Now the default project settings are loaded. We can check if the right module (TRQ\_DES\_custom) is selected by pressing **Select Module** in the main view.

- Press **Compile** to build the firmware

The compile process is finished if the process bar is fully loaded and the status shows "Finished – Click here to view your firmware!". By clicking on the text, you are navigated directly to the output build folder of the firmware file (\*.eef). This folder is generated inside the project path (i.e. .\YourProjectName\FirmwareRelease\RELEASEID)

If an error occurs during the compilation process, refer to **section „Troubleshooting“ in our Manual** using the indicated error code. <https://friwo.link/ag/manual> 

## UPDATE MOTOR CONTROLLER

For updating the Motor Controller with the generated firmware we have to switch from FRIWO SDK to FRIWO

 **Enable Tool.** <https://friwo.link/ag/enable-tool>

Please refer to **Enable Tool Manual** for further information about updating the Motor Controller. 

<https://friwo.link/ag/et-manual>

# CALIBRATE HILL ASSIST

At first, make sure the following requirements are given:

- System is in safe conditions, i.e. motor is in standstill
- FRIWO EnableTool is running
- Motor Controller with customized Hill Assist firmware is connected via USB
- Correct .xml-file for variable configuration is loaded in FRIWO EnableTool
- Throttle pedal is connected to AIN1-Interface
- Brake pedal is connected to AIN2-Interface

Now we have a look at the SDK setup to make sure, that our customized module is executed by basic firmware.

- In FRIWO EnableTool variable list on the right select the **SDK** dropdown
- Set SDK\_C\_TRQDES\_Custom\_Module\_Enable to 1 to activate the TRQ\_DES\_custom module

SDK	
SDK_C_CAN_Custom_Timeout_Enable	0
SDK_C_TRQDES_Custom_Module_Enable	1
SDK_CAN_Custom_Timeout_Enable	0
SDK_TRQDES_Custom_Module_Enable	1



**Note:** Activation/Deactivation of SDK modules is only possible, if the motor is in standstill.

The display variable SDK\_TRQDES\_Custom\_Module\_Enable = 1 indicates that the TRQ\_DES\_custom module is executed on the Motor Controller.

- In variable list on the right select the **TRQ\_DES\_custom** dropdown to show the previously defined variables:



TRQ_DES_custom	
TRQ_DES_Brake_Input	0
TRQ_DES_C_ReverseGear_TestInput	0
TRQ_DES_C_ThrottlePriorization_MaxRotorSpeed	2
TRQ_DES_C_ThrottlePriorization_Time	10000
TRQ_DES_HillAssist_State	0
TRQ_DES_HillAssist_ValCounter	0
TRQ_DES_ReverseGear_Input	0
TRQ_DES_Throttle_Input	0
TRQ_DES_TorqueRequest	0
TRQ_DES_TorqueRequest_LowerLim	0
TRQ_DES_TorqueRequest_UpperLim	0

- Double click the display variables in order to move them to the logging window on the left:

ECU	Live View																						
																							
Firmware: 100999387 OEM ID: 0 Client ID: 0 Project ID: 0	<table border="1"> <tr><td>INFO</td><td></td></tr> <tr><td>INFO_Rotor_Speed</td><td>0</td></tr> <tr><td>SDK</td><td></td></tr> <tr><td>SDK_TRQDES_Custom_Module_Enable</td><td>1</td></tr> <tr><td>TRQ_DES_custom</td><td></td></tr> <tr><td>TRQ_DES_Brake_Input</td><td>0</td></tr> <tr><td>TRQ_DES_HillAssist_State</td><td>0</td></tr> <tr><td>TRQ_DES_HillAssist_ValCounter</td><td>0</td></tr> <tr><td>TRQ_DES_ReverseGear_Input</td><td>0</td></tr> <tr><td>TRQ_DES_Throttle_Input</td><td>0</td></tr> <tr><td>TRQ_DES_TorqueRequest</td><td>0</td></tr> </table>	INFO		INFO_Rotor_Speed	0	SDK		SDK_TRQDES_Custom_Module_Enable	1	TRQ_DES_custom		TRQ_DES_Brake_Input	0	TRQ_DES_HillAssist_State	0	TRQ_DES_HillAssist_ValCounter	0	TRQ_DES_ReverseGear_Input	0	TRQ_DES_Throttle_Input	0	TRQ_DES_TorqueRequest	0
INFO																							
INFO_Rotor_Speed	0																						
SDK																							
SDK_TRQDES_Custom_Module_Enable	1																						
TRQ_DES_custom																							
TRQ_DES_Brake_Input	0																						
TRQ_DES_HillAssist_State	0																						
TRQ_DES_HillAssist_ValCounter	0																						
TRQ_DES_ReverseGear_Input	0																						
TRQ_DES_Throttle_Input	0																						
TRQ_DES_TorqueRequest	0																						

In initial operation, when torque control is deactivated, the Hill Assist is not executed, thus *TRQ\_DES\_HillAssist\_State* indicates state 0, which relates to STATE\_INITIAL. This status holds also, if torque control is activated and no torque is requested.

- Simultaneously hold the brake pedal and accelerate while watching the two variables *TRQ\_DES\_HillAssist\_State* and *TRQ\_DES\_HillAssist\_ValCounter*

ECU	Live View																						
																							
Firmware: 100999387 OEM ID: 0 Client ID: 0 Project ID: 0	<table border="1"> <tr><td>INFO</td><td></td></tr> <tr><td>INFO_Rotor_Speed</td><td>0</td></tr> <tr><td>SDK</td><td></td></tr> <tr><td>SDK_TRQDES_Custom_Module_Enable</td><td>1</td></tr> <tr><td>TRQ_DES_custom</td><td></td></tr> <tr><td>TRQ_DES_Brake_Input</td><td>1.423</td></tr> <tr style="border: 2px solid red;"><td>TRQ_DES_HillAssist_State</td><td>1</td></tr> <tr style="border: 2px solid red;"><td>TRQ_DES_HillAssist_ValCounter</td><td>6669</td></tr> <tr><td>TRQ_DES_ReverseGear_Input</td><td>0</td></tr> <tr><td>TRQ_DES_Throttle_Input</td><td>22.215</td></tr> <tr><td>TRQ_DES_TorqueRequest</td><td>22.215</td></tr> </table>	INFO		INFO_Rotor_Speed	0	SDK		SDK_TRQDES_Custom_Module_Enable	1	TRQ_DES_custom		TRQ_DES_Brake_Input	1.423	TRQ_DES_HillAssist_State	1	TRQ_DES_HillAssist_ValCounter	6669	TRQ_DES_ReverseGear_Input	0	TRQ_DES_Throttle_Input	22.215	TRQ_DES_TorqueRequest	22.215
INFO																							
INFO_Rotor_Speed	0																						
SDK																							
SDK_TRQDES_Custom_Module_Enable	1																						
TRQ_DES_custom																							
TRQ_DES_Brake_Input	1.423																						
TRQ_DES_HillAssist_State	1																						
TRQ_DES_HillAssist_ValCounter	6669																						
TRQ_DES_ReverseGear_Input	0																						
TRQ_DES_Throttle_Input	22.215																						
TRQ_DES_TorqueRequest	22.215																						

The state machine jumps into state `STATE_THROTTLE_PRIO`, indicated by `TRQ_DES_HillAssist_State = 1` as long as mechanical rotor speed is below the threshold `TRQ_DES_C_ThrottlePriorization_MaxRotorSpeed`. During this time, the requested torque `TRQ_DES_TorqueRequest` equals the signal `TRQ_DES_Throttle_Input`, which is prioritized.

The variable `TRQ_DES_HillAssist_ValCounter` shows the actual value of the Hill Assist counter which is reloaded by the calibratable value `TRQ_DES_C_ThrottlePriorization_Time` when entering the state `STATE_THROTTLE_PRIO` and counts down every time the `TRQ_DES_custom` module is called.

If this counter reaches zero, the state machine jumps into state `STATE_NORMAL_ACCELERATION`, indicated by `TRQ_DES_HillAssist_State = 2`.

If this condition holds, both signals `TRQ_DES_Throttle_Input` and `TRQ_DES_Brake_Input` are simply subtracted in this application. In order to get back into `STATE_INITIAL`, both the throttle and brake pedal must be released.

Have fun programming!

## Feedback

We are working very hard to improve our products and therefore **feedback** is indispensable! Please send us your valuable feedback as contact form or via Mail to [feedback@friwo.com](mailto:feedback@friwo.com)



<https://friwo.link/ag/feedback>

