NXP Semiconductors
User's guide

Document Number: PEXMVHBRIDGEPUG
Rev. 1.0, 2/2016

MVHBridge Programming Guide

Contents

1 General Info

This documentation introduces Processor Expert component named MVHBridge. This component supports and provides flexible software solution for these analog parts:

NXP MC33926: Single H-Bridge, Brushed DC Motor Driver, 5-28V, 5A, 20kHz

NXP MC33931: Single H-Bridge, Brushed DC Motor Driver, 5-28V, 5A, 11kHz

NXP MC33932: Dual H-Bridge, Stepper/Brushed DC Motor Driver, 5-28V, 5A, 11kHz

NXP MC34931(S): Single H-Bridge, Brushed DC Motor Driver, 5-36V, 5A, 11kHz/20kHz

NXP MC34932(S): Dual H-Bridge, Stepper/Brushed DC Motor Driver, 5-36V, 5A, 11kHz/20kHz

NXP offers Tower and Freedom board solutions based on these chips, namely TWR-MC-MVHB1EVB (for MC33926 and MC33932), FRDM-33931-EVB, FRDM-34931-EVB and FRDM-34931S-EVB boards. Detailed description can be found in related hardware user guides and datasheets.

2 Embedded Component Description

2.1 Component API

MVHBridge component provides API, which can be used for dynamic real-time configuration of device in user code. Available methods and events are listed under component selection Some of those methods/events are marked with ticks and other ones with crosses, it distinguishes which methods/events are supposed to be generated or not. You can change this setting in Processor Expert Inspector. Note that methods with grey text are always generated because they are needed for proper functionality. This forced behavior depends on various combinations of settings of component properties. For summarization of available API methods and events and their descriptions, see Table 1 MVHBridge Component API

Table 1

Method	Description
Init	Initializes the device. Allocates memory for the device data structure, sets HBridge device mode, etc. This method can be called only once. Before the second call of Init() the Deinit() must be called first. Components linked by HBridge (TimerUnitLDD, ADC_LDD) are not initialized here.
Deinit	This method deinitializes the device. Method Deinit of enable and disable pins (BitIO_LDD components) is called. This method does not influence linked timer device (TimerUnitLDD component). Channels of the timer used by the MVHBridge component remains functional.
SetMode	This method sets HBridge device mode using enable and disable pins. The method is available only when Enable and Disable Pins setting is enabled.
GetCurrentMeasurement	This method is intended for measuring of HBridge output current. The method is available only when Feedback Pin setting is enabled. This method returns raw measurement result from ADC. HBridge feedback provides groundreferenced 0.24% of the high side output current.
GetStatusFlag	This method returns fault status of HBridge device. Fault is detected when HBridge Status Flag pin goes to LOW. The method is available only when Status Flag Pin attribute is enabled.
ClearStatusFlag	This method clears fault flag of HBridge device by toggling of D1 pin. Method is available only when Status Flag Pin setting and Enable and Disable Pins setting is enabled.

InvertInputPins	This method inverts output value of IN pins. The method is available only when Input Invert setting is enabled and MC33926 HBridge model is used.
RotateProportional	This method starts rotation of brush motor. The method allows control of motor speed.
RotateFull	This method starts rotation of brush motor. The method is intended for state motor control (on/off).
SetRecirculation	This method sets recirculation of brush motor.
SetDirection	This method sets direction of brush motor.
SetFullStepSpeed	Set speed of fullstep mode. Unit is number of fullsteps per second. It is not allowed to change speed while motor is running.
SetMicroStepSpeed	Set speed of microstep mode. Unit is number of microsteps per second. Size of microstep depends on setting in Processor Expert (number of microsteps per fullstep). It is not allowed to change speed while motor is running.
MoveSteps	This method moves the motor by specified number of fullsteps. When the rotor is not at physical fullstep position then the method sets the nearest fullstep position without correction. Note that number of steps returned by method [GetFullStepPosition] are updated before they are executed. For example an user calls the method [MoveSteps] with parameter Steps equal to 100. Certain number of these steps are counted before they are physically executed (for example 64 steps when "OutputControl" property is set to PWM and FTM device is used).
MoveMicroSteps	Moves motor by specified number of microsteps. When the rotor is not at physical microstep position then the method sets nearest microstep without correction. For example the size is initialized to 32 microsteps per one fullstep and the motor executed three microsteps. Then user changes microstep size to 2 microsteps per one fullstep and starts motor movement (previous three microsteps are not visible).
MoveContinual	Moves motor continually in fullstep mode. You can stop motor by calling [StopContinualMovement] method. When rotor is not at physical fullstep position then the method sets nearest fullstep without correction. This method is not available when acceleration ramp is used.
MoveMicroContinual	This method moves motor continually in microstep mode. You can stop motor by calling [StopContinualMovement] method. When the rotor is not at physical microstep position then the method sets nearest microstep without correction. For example the size is initialized to 32 microsteps per one fullstep and the motor executed three microsteps. Then user changes microstep size to 2 microsteps per one fullstep and starts motor movement (previous three microsteps are not visible). This method is not available when the acceleration ramp is used.
StopContinualMovement	This method is intended to stop continual movement of stepper motor. The method does not stop motor immediately, motor can execute several steps. In microstep mode the motor does not have to stop at fullstep position (can stop anywhere). This method is not available when acceleration ramp is used or method [MoveContinual] or [MoveMicroContinual] is not enabled.
GetMotorStatus	This method returns status of stepper motor control. Possible values are defined in TMotorStatus enumeration in header file.
AlignRotor	Align the rotor to fullstep position. The method executes 4 fullsteps forward (one electrical revolution) at minimum speed (see "component_name"_MIN_FULLSTEP_SPEED constant). These steps are not counted to the number of fullsteps. This method is not available when Motor Control Mode is set to Microstep.

SetMicroStepSize	This method serves to change size of microstep. Note that the size of microstep is initialized to value set in Processor Expert property "Microsteps per Step". The motor must not be running when you call this method. The method is available only when microstepping is enabled.
GetFullStepPosition	This method returns current fullstep position. Position is set to zero when initialization of HBridge component occurs. It can be reset by method [ResetFullStepPosition].
ResetFullStepPosition	This method sets counter of fullsteps to zero.
DisableMotor	The method sets IN pins output value to LOW. The method can be used to stop the stepper motor. Output value of the pins are not changed immediately, because the counter registers are updated after the counter overflows (at the beginning of the next counter period). Note that default behavior of the motor control is to hold position when a movement is completed.

2.2 Events

OnStatusFlagA - This event is called when status flag signal goes to LOW on Bridge A. The handler is available only when Status Flag Pin setting is enabled for interface A.

*ANSIC prototype:*void OnStatusFlagA(LDD_TUserData UserDataPtr)
UserDataPtr:LDD_TUserData - Pointer to the user data. The pointer passed as the parameter of Init method.

OnStatusFlagB - This event is called when status flag signal goes to LOW on Bridge B. The handler is available only when Status Flag Pin setting is enabled for interface B and dual H-Bridge models is used.

*ANSIC prototype:*void OnStatusFlagB(LDD_TUserData UserDataPtr)
UserDataPtr:LDD_TUserData - Pointer to the user data. The pointer passed as the parameter of Init method.

OnActionComplete - This event is called when the motor reaches desired number of steps or the motor is stopped by method [StopContinualMovement]. The handler is available only for stepper motor.

*ANSIC prototype:*void OnActionComplete(LDD_TUserData UserDataPtr)
UserDataPtr:LDD_TUserData - Pointer to the user data. The pointer passed as the parameter of Init method.

2.3 Methods

Init - Initializes the device. Allocates memory for the device data structure, sets H-Bridge device mode, etc. This method can be called only once. Before the second call of Init() the Deinit() must be called first. Components linked by H-Bridge (TimerUnitLDD, ADC_LDD) are not initialized here.

*ANSIC prototype:*Init(LDD_TUserData *UserDataPtr)
UserDataPtr: Pointer to LDD_TUserData - Pointer to the user data. This pointer will be passed as an event or callback parameter.

Return value:void - Device data structure pointer.

Deinit - This method deinitializes the device. Method Deinit of enable and disable pins (BitIO_LDD components) is called. This method does not influence linked timer device (TimerUnitLDD component). Channels of the timer used by the MVHBridge component remains functional.

ANSIC prototype: void Deinit()
Return value:void -

SetMode - This method sets H-Bridge device mode using enable and disable pins. The method is available only when Enable and Disable Pins setting is enabled.

ANSIC prototype: LDD_TError SetMode(THBrMode Mode,THBridge Bridge)

Mode:THBrMode - Desired H-Bridge mode, possible values are defined in enumeration THBrMode in header file.

Bridge:THBridge - Selection of H-Bridge interface. Only hbBRIDGE_A value is correct when single H-Bridge model is used.

Return value:LDD_TError - Error code.

ERR_OK : No problem detected

ERR_PARAM_VALUE : Invalid value of parameter Bridge or Mode.

GetCurrentMeasurement - This method is intended for measuring of H-Bridge output current. The method is available only when Feedback Pin setting is enabled. This method returns raw measurement result from ADC. H-Bridge feedback provides ground-referenced 0.24% of the high side output current.

ANSIC prototype: LDD_TError GetCurrentMeasurement(THBridge Bridge,uint16_t *BufferPtr)

Bridge:THBridge - Selection of H-Bridge interface. Only hbBRIDGE_A value is correct when single H-Bridge model is used.

BufferPtr: Pointer to uint16_t - Array with one item for measurement result. Current must be count from BufferPtr. Formula for I=f(BufferPtr)[mA] is: I = (BufferPtr * ADC_REF * 1000) / (65535 * R_FB * 0.24%),

where R_FB is Resistor on Feedback pin and ADC_REF is reference Voltage on ADC converter.

Return value:LDD_TError - Error code.

ERR_OK : No problem detected

ERR_PARAM_VALUE : Invalid value of parameter Bridge.

Other error codes are defined by ADC_LDD. For more details see the ADC_LDD documentation.

GetStatusFlag - This method returns fault status of H-Bridge device. Fault is detected when H-Bridge Status Flag pin goes to LOW. The method is available only when Status Flag Pin attribute is enabled.

ANSIC prototype: LDD_TError GetStatusFlag(THBridge Bridge,bool *Fault)

Bridge:THBridge - Selection of H-Bridge interface. Only hbBRIDGE_A value is correct when single H-Bridge model is used.

Fault: Pointer to bool - Here will be stored value of H-Bridge current status. It is set to TRUE when a fault occurred, else set to FALSE.

Return value:LDD_TError - Error code.

ERR_OK : No problem detected

ERR_PARAM_VALUE : Invalid value of parameter Bridge.

ClearStatusFlag - This method clears fault flag of H-Bridge device by toggling of D1 pin. Method is available only when Status Flag Pin setting and Enable and Disable Pins setting is enabled.

ANSIC prototype: LDD_TError ClearStatusFlag(THBridge Bridge)

Bridge:THBridge - Selection of H-Bridge interface. Only hbBRIDGE_A value is correct when single H-Bridge model is used.

Return value:LDD_TError - Error code.

ERR_OK : No problem detected

ERR_PARAM_VALUE : Invalid value of Bridge parameter.

InvertInputPins - This method inverts output value of IN pins. The method is available only when Input Invert setting is enabled and MC33926 H-Bridge model is used.

ANSIC prototype: void InvertInputPins(bool Invert)

Invert:bool - TRUE for inverting value on IN pins. Put FALSE when you do not want to invert value on pins.

Return value:void -

RotateProportional - This method starts rotation of brush motor. The method allows control of motor speed.

ANSIC prototype: LDD_TError RotateProportional(uint8_t PWMDuty,THBridge Bridge)

PWMDuty:uint8_t - Value of PWM duty. Value have to be in range 0,.., 100.

Bridge:THBridge - Selection of H-Bridge interface. Only hbBRIDGE_A value is correct when single H-Bridge model is used.

Return value:LDD_TError - Error code.

ERR_OK : No problem detected

ERR_PARAM_VALUE : Invalid value of parameter PWMDuty or parameter Bridge.

Other error codes are defined by TimerUnit_LDD. For more details see the TimerUnit_LDD documentation.

RotateFull - This method starts rotation of brush motor. The method is intended for state motor control (on/off).

ANSIC prototype: LDD_TError RotateFull(bool Rotate,THBridge Bridge)

Rotate:bool - TRUE for rotation, FALSE for stop.

Bridge:THBridge - Selection of H-Bridge interface. Only hbBRIDGE_A value is correct when single H-Bridge model is used.

Return value:LDD_TError - Error code.

ERR_OK : No problem detected

ERR_PARAM_VALUE : Invalid value of parameter Bridge.

SetRecirculation - This method sets recirculation of brush motor.

ANSIC prototype: LDD_TError SetRecirculation(bool HSide,THBridge Bridge)

HSide:bool - When the parameter is TRUE high-Side Recirculation is set (IN1 and IN2 are set to HIGH or IN3 and IN4 are set to HIGH). When the parameter Low-Side Recirculation is set (IN1 and IN2 are set to LOW or IN3 and IN4 are set to LOW).

Bridge:THBridge - Selection of H-Bridge interface. Only hbBRIDGE_A value is correct when single H-Bridge model is used.

Return value:LDD_TError - Error code.

ERR_OK : No problem detected

ERR_PARAM_VALUE : Invalid value of paramter Bridge.

Other error codes are defined by TimerUnit_LDD. For more details see the TimerUnit_LDD documentation.

SetDirection - This method sets direction of brush motor.

ANSIC prototype: LDD_TError SetDirection(bool Forward,THBridge Bridge)

Forward:bool - Motor direction.

Bridge:THBridge - Selection of H-Bridge interface. Only hbBRIDGE_A value is correct when single H-Bridge model is used.

Return value:LDD_TError - Error code.

ERR_OK : No problem detected

ERR_PARAM_VALUE : Invalid value of parameter Bridge.

SetFullStepSpeed - Set speed of full-step mode. Unit is number of full-steps per second. It is not allowed to change speed while motor is running.

ANSIC prototype: LDD_TError SetFullStepSpeed(uint32_t StepsSec)

StepsSec:uint32_t - Motor speed in number of full-steps per second. Minimal and maximal speed is defined by constants "component_name"_MIN_FULLSTEP_SPEED and "component_name"_MAX_FULLSTEP_SPEED placed in header file.

Return value:LDD_TError - Error code.

ERR_OK : No problem detected

ERR_BUSY : Motor is running.

ERR_PARAM_VALUE : Invalid value of parameter StepsSec.

Other error codes are defined by TimerUnit_LDD. For more details see the TimerUnit_LDD documentation.

SetMicroStepSpeed - Set speed of micro-step mode. Unit is number of micro-steps per second. Size of micro-step depends on setting in Processor Expert (number of micro-steps per full-step). It is not allowed to change speed while motor is running.

ANSIC prototype: LDD_TError SetMicroStepSpeed(uint32_t MicroStepsSec)

MicroStepsSec:uint32_t - Motor speed in number of micro-steps per second. Minimal and maximal speed is defined by constants "component_name"_MIN_MICROSTEP_SPEED, "component_name"_MAX_MICROSTEP_SPEED.

Return value:LDD_TError - Error code.

ERR_OK : No problem detected

ERR_BUSY : Motor is running.

ERR_PARAM_VALUE : Invalid value of parameter MicroStepsSec.

Other error codes are defined by TimerUnit_LDD. For more details see the TimerUnit_LDD documentation.

MoveSteps - This method moves the motor by specified number of full-steps. When the rotor is not at physical full-step position then the method sets the nearest full-step position without correction. Note that number of steps returned by method [GetFullStepPosition] are updated before they are executed. For example an user calls the method [MoveSteps] with parameter Steps equal to 100. Certain number of these steps are counted before they are physically executed (for example 64 steps when "OutputControl" property is set to PWM and FTM device is used).

ANSIC prototype: LDD_TError MoveSteps(bool Forward,uint32_t Steps)

Forward:bool - Motor direction.

Steps:uint32_t - Number of full-steps to be performed. Maximal value is 100 000 000.

Return value:LDD_TError - Error code.

ERR_OK : No problem detected

ERR_BUSY : Motor is running.

ERR_PARAM_VALUE : Invalid number of steps.

ERR_FAILED : Invalid value of TimerUnit_LDD input frequency (see setting Counter frequency of TimerUnit_LDD component).

Other error codes are defined by TimerUnit_LDD. For more details see the TimerUnit_LDD documentation.

MoveMicroSteps - Moves motor by specified number of micro-steps. When the rotor is not at physical micro-step position then the method sets nearest micro-step without correction. For example the size is initialized to 32 micro-steps per one full-step and the motor executed three micro-steps. Then user changes micro-step size to 2 micro-steps per one full-step and starts motor movement (previous three micro-steps are not visible).

ANSIC prototype: LDD_TError MoveMicroSteps(bool Forward,uint32_t MicroSteps)

Forward:bool - Motor direction.

MicroSteps:uint32_t - Number of micro-steps to be performed. Maximal value is 100 000 000.

Return value:LDD_TError - Error code.

ERR_OK : No problem detected

ERR_BUSY : Motor is running.

ERR_PARAM_VALUE : Invalid number of steps.

ERR_FAILED : Invalid value of TimerUnit_LDD input frequency (see setting Counter frequency of TimerUnit_LDD component).

Other error codes are defined by TimerUnit_LDD. For more details see the TimerUnit_LDD documentation.

MoveContinual - Moves motor continually in full-step mode. You can stop motor by calling [StopContinualMovement] method. When rotor is not at physical full-step position then the method sets nearest full-step without correction. This method is not available when acceleration ramp is used.

ANSIC prototype: LDD_TError MoveContinual(bool Forward)

Forward:bool - Motor direction.

Return value:LDD_TError - Error code.

ERR_OK : No problem detected

ERR_BUSY : Motor is running.

ERR_FAILED : Invalid value of TimerUnit_LDD input frequency (see setting Counter frequency of TimerUnit_LDD component).

Other error codes are defined by TimerUnit_LDD. For more details see the TimerUnit_LDD documentation.

MoveMicroContinual - This method moves motor continually in micro-step mode. You can stop motor by calling [StopContinualMovement] method. When the rotor is not at physical micro-step position then the method sets nearest micro-step without correction. For example the size is initialized to 32 micro-steps per one full-step and the motor executed three micro-steps. Then user changes micro-step size to 2 micro-steps per one full-step and starts motor movement (previous three micro-steps are not visible). This method is not available when the acceleration ramp is used.

ANSIC prototype: LDD_TError MoveMicroContinual(bool Forward)

Forward:bool - Motor direction.

Return value:LDD_TError - Error code.

ERR_OK : No problem detected

ERR_BUSY : Motor is running.

ERR_FAILED : Invalid value of TimerUnit_LDD input frequency (see setting Counter frequency of TimerUnit_LDD component).

Other error codes are defined by TimerUnit_LDD. For more details see the TimerUnit_LDD documentation.

StopContinualMovement - This method is intended to stop continual movement of stepper motor. The method does not stop motor immediately, motor can execute several steps. In micro-step mode the motor does not have to stop at full-step position (can stop anywhere). This method is not available when acceleration ramp is used or method [MoveContinual] or [MoveMicroContinual] is not enabled.

ANSIC prototype: LDD_TError StopContinualMovement()

Return value:LDD_TError - Error code.

ERR_OK : No problem detected

ERR_FAILED : Motor is not running or not running in continuous mode.

GetMotorStatus - This method returns status of stepper motor control. Possible values are defined in TMotorStatus enumeration in header file.

ANSIC prototype: TMotorStatus GetMotorStatus()

Return value:TMotorStatus - Motor status.

AlignRotor - Align the rotor to full-step position. The method executes 4 full-steps forward (one electrical revolution) at minimum speed (see "component_name"_MIN_FULLSTEP_SPEED constant). These steps are not counted to the number of full-steps. This method is not available when Motor Control Mode is set to Micro-step.

ANSIC prototype: LDD_TError AlignRotor()

Return value:LDD_TError - Error code.

ERR_OK : No problem detected

ERR_BUSY : Motor is running.

ERR_FAILED : Invalid value of TimerUnit_LDD input frequency (see setting Counter frequency of TimerUnit_LDD component).

Other error codes are defined by TimerUnit_LDD. For more details see the TimerUnit_LDD documentation.

SetMicroStepSize - This method serves to change size of micro-step. Note that the size of micro-step is initialized to value set in Processor Expert property "Micro-steps per Step". The motor must not be running when you call this method. The method is available only when micro-stepping is enabled.

ANSIC prototype: LDD_TError SetMicroStepSize(uint8_t Size)

Size:uint8_t - Number of micro-steps per one full-step. Possible values are 2, 4, 8, 16, 32. For example put 16 to set 16 micro-steps per one full-step.

Return value:LDD_TError - Error code.
ERR_OK : No problem detected
ERR_BUSY : Motor is running.
ERR_PARAM_VALUE : Invalid value of parameter Size.

GetFullStepPosition - This method returns current full-step position. Position is set to zero when initialization of H-Bridge component occurs. It can be reset by method [ResetFullStepPosition].

ANSIC prototype: int32_t GetFullStepPosition()
Return value:int32_t - Current position of rotor in number of full-steps taken from initial position.

ResetFullStepPosition - This method sets counter of full-steps to zero.

ANSIC prototype: LDD_TError ResetFullStepPosition()
Return value:LDD_TError - Error code.
ERR_OK : No problem detected
ERR_BUSY : Motor is running.

DisableMotor - The method sets IN pins output value to LOW. The method can be used to stop the stepper motor. Output value of the pins are not changed immediately, because the counter registers are updated after the counter overflows (at the beginning of the next counter period). Note that default behavior of the motor control is to hold position when a movement is completed.

ANSIC prototype: LDD_TError DisableMotor()
Return value:LDD_TError - Error code. Error code values are defined by TimerUnit_LDD. For more details see the TimerUnit_LDD documentation.

2.4 Properties

Component Name - Name of the component.

H-Bridge Model - H-Bridge model.

Motor Control - Select type of motor you want to use. DC brush motor is controlled by two input pins. Stepper motor is controlled by 4 input pins and can be used only with dual H-Bridge model. Custom control is without motor settings.

Timer Settings - Timer Settings property. Select TimerUnit_LDD component and select Timer device.

The following items are available only if the group is enabled (the value is "Enabled"):

Timer Component - Reference to TimerUnit_LDD that serves for timing of motor control.

Timer Device - Name of the counter used by TimerUnit_LDD component.

Stepper Motor - Stepper motor settings.

Output Control - Stepper control method.

There are 2 options:

PWM: All four IN pins are controlled by PWM signal (TimerUnit_LDD channels).
GPIO: All four IN pins are controlled by GPIO (BitIO_LDD).

Manual Timer setting - Setting "Counter frequency" of linked TimerUnit_LDD component is automaticaly set when "Manual timer setting" is set to "Disable". You can change the timer frequency when you enable the manual setting. Please note that list of frequencies (2 values of frequency, first is for full-stepping and second for micro-stepping) must be set when the "Motor Control Mode" is set to "Full-step and Micro-step". Fixed value (1 value of frequency) is required in other cases. You also must set correct values of frequency. Minimal frequency for full-stepping is 131 072 Hz and maximal frequency is 1 MHz. Minimal frequency for micro-stepping is 1.2 MHz and maximal value is 10 MHz.

There are 2 options:

Enabled
Disabled

Motor Control Mode - Stepper motor control mode. Micro-stepping is available only when output control is set to "PWM".

Full-step Configuration - Configuration of full-stepping.

Initial Speed - Initial motor speed in full-step mode. The speed can be changed later in C code. Unit is number of full-steps per second.

Acceleration Ramp - Fluent acceleration to desired speed and decelaration to zero. Put 0 value to disable ramp. Unit is full-steps per second per second.

Micro-step Configuration - Configuration of micro-stepping.

PWM Frequency - PWM frequency for micro-stepping in kHz. Maximum value depends on used H-Bridge model. 20 kHz is max. when MC34931S or MC34932S models are selected in "H-Bridge Model". Other dual H-Bridge models have max. 11 kHz.

Micro-steps per Step - Number of micro-steps per one full-step. The size can be changed later in C code.

There are 2 options:

2 Micro-steps

4 Micro-steps

Initial Speed - Initial motor speed in micro-step mode. The speed can be changed later in C code. Unit is number of micro-steps per second. Size of micro-step depends on property "Micro-steps per step".

Acceleration Ramp - Fluent acceleration to desired speed and decelaration to zero. Put 0 value to disable ramp. Unit is micro-steps per second per second. Size of micro-step depends on Micro-steps per Step setting.

H-Bridge A MCU Interface - Configuration of H-Bridge interface to MCU. If the H-Bridge model has two independent interfaces (dual H-Bridge model) then this is interface A.

The following items are available only if the group is enabled (the value is "Enabled"):

DC brush - Configuration of DC Brush motor.

Control Mode - Motor can be controlled by PWM signal from linked TimerUnit_LDD (speed control) or by GPIO pins (state control). In state control you can only switch the motor on or off.

PWM Frequency - PWM frequency for speed motor control. Maximum value depends on selected H-Bridge model (11 kHz is limit of MC33931, MC34931, MC33932, MC34932; 20 kHz is limit of MC33926, MC34931S, MC34932S). It is necessary to type here both a value and an unit.

Direction Control - Setting of direction in which you can change speed. Available only when Speed control is set.

Init. Direction - Initial direction of brushed DC motor. Forward means that IN1 is high and IN2 low. Reverse means IN1 is low and IN2 high. Direction can be changed later in C code.

There are 2 options:

Forward: Initial direction is forward.

Reverse: Initial direction is reverse.

Device Mode - Selection of H-Bridge operating mode.

There are 3 options:

Normal Mode: H-Bridge is fully operational.

Sleep Mode: H-Bridge is not operational and saves energy (consuming max 50uA).

StandBy Mode: H-Bridge is operational, output OUT1 and OUT2 are tri-stated (consuming max 20 mA).

Device Settings A - H-Bridge pins configuration. If the H-Bridge model has two independent interfaces (dual H-Bridge model) then this is interface A.

Enable and Disable Pins - Settings of enable and disable pins.

The following items are available only if the group is enabled (the value is "Enabled"):

Pin for D1 - Disable input 1 pin for stand by mode.

Pin for D2 - Disable input 2 pin for stand by mode.

Pin for EN/D2bar - Enable pin for sleep mode.

Pin for En - Enable pin for sleep mode.

Slew Rate - Pin is used to select fast or slow Slew rate.

The following items are available only if the group is enabled (the value is "Enabled"):

Pin for SLEW - Selection of pin for Slew rate. Slew rate is selected automatically according to value of PWM Frequency property (fast slew rate for more than 11 kHz, else slow slew rate).

Input Invert - Input invert pin. Default condition is not inverted.

The following items are available only if the group is enabled (the value is "Enabled"):

Pin for INV - Selection of Input invert pin. Input pins are not inverted by default.

Input Control Pins - Control mode for IN1 and IN2 pin. Attribute can be changed only when "Custom" option is selected in "Motor Control" attribute.

Pin for IN1 - IN1 pin for control of OUT1.

Pin for IN2 - IN2 pin for control of OUT2.

Feedback Pin - Sensing of current via AD converter.

The following items are available only if the group is enabled (the value is "Enabled"):

ADC Component - Select linked ADC_LDD you want to use.

ADC Device - AD converter device used by linked ADC_LDD.

ADC Pin - Select pin for current sensing via AD converter.

There are 2 options:

Enabled

Disabled

ADC Conversion Time - Time for one AD conversion. It is necessary to type here both a value and an unit.

Status Flag Pin - Status flag for H-Bridge fault detection.

The following items are available only if the group is enabled (the value is "Enabled"):

Pin for Status Flag - Select pin you want to use for status flag.

There are 2 options:

Enabled

Disabled

H-Bridge B MCU Interface - Configuration of H-Bridge interface to MCU. If the H-Bridge model has two independent interfaces (dual H-Bridge model) then this is interface B.

The following items are available only if the group is enabled (the value is "Enabled"):

DC brush - Configuration of DC Brush motor.

Control Mode - Motor can be controlled by PWM signal from linked TimerUnit_LDD (speed control) or by GPIO pins (state control). In state control you can only switch the motor on or off.

PWM Frequency - PWM frequency for speed motor control. Maximum value depends on selected H-Bridge model (11 kHz is limit of MC33931, MC34931, MC33932, MC34932; 20 kHz is limit of MC33926, MC34931S, MC34932S). It is necessary to type here both a value and an unit.

Direction Control - Setting of direction in which you can change speed. Available only when Speed control is set.

Init. Direction - Initial direction of brushed DC motor. Forward means that IN3 is high and IN4 low. Reverse means IN3 is low and IN4 high. Direction can be changed later in C code.

There are 2 options:

Forward: Initial direction is forward.

Reverse: Initial direction is reverse.

Device Mode - Selection of H-Bridge operating mode.

There are 3 options:

Normal Mode: H-Bridge is fully operational.

Sleep Mode: H-Bridge is not operational and saves energy (consuming max 50uA).

StandBy Mode: H-Bridge is operational, output OUT1 and OUT2 are tri-stated (consuming max 20 mA).

Device Settings B - H-Bridge pins configuration. If the H-Bridge model has two independent interfaces (dual H-Bridge model) then this is interface B.

Enable and Disable Pins - Settings of enable and disable pins.

The following items are available only if the group is enabled (the value is "Enabled"):

Pin for D3 - Disable input 3 pin for stand by mode.

Pin for EN/D4bar - Enable pin for sleep mode.

Input Control Pins - Control mode for IN3 and IN4 pin. Attribute can be changed only when "Custom" option is selected in "Motor Control" attribute.

Pin for IN3 - IN3 pin for control of OUT3

Pin for IN4 - IN4 pin for control of OUT4

Feedback Pin - Sensing of current via AD converter.

The following items are available only if the group is enabled (the value is "Enabled"):

ADC Component - Select linked ADC_LDD you want to use.

ADC Device - AD converter device used by linked ADC_LDD.

ADC Pin - Select pin for current sensing via AD converter.

There are 2 options:

Enabled

Disabled

ADC Conversion Time - Time for one AD conversion. It is necessary to type here both a value and an unit.

Status Flag Pin - Status flag for H-Bridge fault detection.

The following items are available only if the group is enabled (the value is "Enabled"):

Pin for Status Flag - Select pin you want to use for status flag.

There are 2 options:

Enabled

Disabled

Auto Initialization - Automated initialization of the component. The component Init method is automatically called from CPU component initialization function PE_low_level_init().

There are 2 options:

yes

no

3 Typical Usage

Examples of typical settings and usage of MVHBridge component.

The state control of DC brushed motor

Required component setup:

Motor Control Mode: Brushed

Device Mode: Normal Mode

Set properties under group *H-Bridge A MCU Interface*. Set also *H-Bridge B MCU Interface* when you use dual H-Bridge model:

Control Mode: State Control

Direction Control: select Bidirectional if you want to change the motor direction in runtime (not necessary here).

Set Enable and Disable Pins and Input Control Pins according to used MCU.

Feedback Pin and Status Flag Pin are not necessary (you can disable these pins).

Auto Initialization: no

Methods: Init, RotateFull,

Note: "MVH1" is name of MVHBridge component.

Content of Main.c:

Listing 1: Source code

```
void main(void)
{
  /* Variable for saving error code. */
  LDD_TError error = ERR_OK;

  ...

  /* Initialization of MVHBridge component. It is possible to pass pointer
   to
   * your own data, which is then stored in device data structure. */
  MVH1_Init(&UserData);

   * placed in header file (here MVH1.h). */
  error = MVH1_RotateFull(TRUE, hbBRIDGE_B);
  if (error != ERR_OK) {
    /* Handle error. */
  }

  /* Stop motor */
  error = MVH1_RotateFull(FALSE, hbBRIDGE_B);
  if (error != ERR_OK) {
    /* Handle error. */
  }
  ...
}
```

The proportional control of DC brushed motor

Required component setup:

Motor Control Mode: Brushed

Set properties under group *H-Bridge A MCU Interface*. Set also *H-Bridge B MCU Interface* when you use dual H-Bridge model

Device Mode: Normal Mode
Control Mode: Speed Control
Direction Control: Bidirectional
Set Enable and Disable Pins and Input Control Pins according to used MCU.
Feedback Pin and Status Flag Pin are not necessary (you can disable these pins).

Auto Initialization: yes

Methods: Init, SetDirection, RotateProportional

Note: example also shows how to change the motor direction. The method "Init" is called from "PE_low_level_init" function automatically due to auto initialization.

Content of Main.c:

Listing 2: Source code

```
void main(void)
{
  /* Variable for saving error code. */
  LDD_TError error = ERR_OK;

  ...

  /* Set forward direction of a motor on the interface A.*/
  error = MVH1_SetDirection(TRUE, hbBRIDGE_A);
  if (error != ERR_OK) {
    /* Handle error. */
  }
```

```
  /* Run the motor in forward direction. */
  error = MVH1_RotateProportional(50, hbBRIDGE_A);
  if (error != ERR_OK) {
    /* Handle error. */
  }

  /* Stop the motor. */
  error = MVH1_RotateProportional(0, hbBRIDGE_A);
  if (error != ERR_OK) {
    /* Handle error. */
  }

  /* Set reverse direction of the motor on interface A.*/
  error = MVH1_SetDirection(FALSE, hbBRIDGE_A);
  if (error != ERR_OK) {
    /* Handle error. */
  }
  /* Run the motor in reverse direction. */
  error = MVH1_RotateProportional(50, hbBRIDGE_A);
  if (error != ERR_OK) {
    /* Handle error. */
  }
  ...
}
```

High/Low Side Recirculacion of DC brushed motor

Required component setup:

Motor Control Mode: Brushed

Device Mode: Normal Mode

Set properties under group *H-Bridge A MCU Interface*. Set also *H-Bridge B MCU Interface* when you use dual H-Bridge model:

Control Mode: Speed Control

Direction Control: select Bidirectional if you want to change the motor direction in runtime (not necessary here).

Set Enable and Disable Pins and Input Control Pins according to used MCU.

Feedback Pin and Status Flag Pin are not necessary (you can disable these pins).

Auto Initialization: yes

Methods: Init, SetRecirculation, RotateProportional

Note: the example also shows how to change the motor direction.

Content of Main.c:

Listing 3: Source code

```
void main(void)
{
  /* Variable for saving error code. */
  LDD_TError error = ERR_OK;

  ...

  /* Run a motor. */
  error = MVH1_RotateProportional(100, hbBRIDGE_A);
  if (error != ERR_OK) {
    /* Handle error. */
  }
  /* Set High side recirculation. */
  error = MVH1_SetRecirculation(TRUE, hbBRIDGE_A);
```

```
  if (error != ERR_OK) {
    /* Handle error. */
  }

  /* Run a motor. */
  error = MVH1_RotateProportional(100, hbBRIDGE_A);
  if (error != ERR_OK) {
    /* Handle error. */
  }
  /* Set Low side recirculation. */
  error = MVH1_SetRecirculation(FALSE, hbBRIDGE_A);
  if (error != ERR_OK) {
    /* Handle error. */
  }
  ...
}
```

Stepper motor control

This example demonstrates usage of full-step and micro-step mode.

Required component setup:

H-Bridge Model: MC33932 or MC34932

Motor Control Mode: Stepper

Timer Device: you must use FTM device (not TPM), beacuse we want to switch between full-step and micro-step in run-time (available only when FTM is used).

Output Control: PWM

Motor Control Mode: Full-step and Micro-step.

Initial Speed: for example 20 (full-step)

Acceleraion Ramp: 0 (full-step)

Micro-steps per Step: for example 2 Micro-steps

Initial Speed: 30 (micro-step)

Acceleraion Ramp: 0 (micro-step)

Set properties under groups *H-Bridge A MCU Interface* and *H-Bridge B MCU Interface*:

Device Mode: Normal Mode

Set Enable and Disable Pins and Input Control Pins according to used MCU.

Feedback Pin and Status Flag Pin are not necessary (you can disable these pins).

Auto Initialization: yes

Methods: Init, SetMode, AlignRotor, GetMotorStatus, SetFullStepSpeed, SetFullStepSpeed, MoveSteps, MoveMicroSteps,

Note that name of the MVHBridge component is MVH1 (this shortcut is used as prefix in MVHBridge methods).

Content of Main.c ():

Listing 4: Source code

```
void main(void)
{
  ...

  /* Align the rotor to the full-step position (4 full-steps in forward
    direction). */
  if (MVH1_AlignRotor() != ERR_OK) {
    /* Handle error. */
  }

  while (MVH1_GetMotorStatus() == msRUNNING) {
```

```
    /* Wait until the motor stops. */
  }

  /* Check possible error. */
  if (MVH1_GetMotorStatus() == msERROR) {
    /* Handle error. */
  }

  /* Change full-stepping speed to 50 full-steps per second. */
  if (MVH1_SetFullStepSpeed(50) != ERR_OK) {
    /* Handle error. */
  }

  /* Execute 25 full-steps in forward direction (second parametr is TRUE).
   * For more information about motor direction see H-Bridge component user
   guide. */
  if (MVH1_MoveSteps(TRUE, 25) != ERR_OK) {
    /* Handle error. */
  }

  while (MVH1_GetMotorStatus() == msRUNNING) {
    /* Wait until the motor stops. */
  }

  /* Check possible error. */
  if (MVH1_GetMotorStatus() == msERROR) {
    /* Handle error. */
  }

  /* Change full-stepping speed to 40 micro-steps per second. Size of micro-
   step
   * depends on Micro-steps per Step setting. */
  if (MVH1_SetMicroStepSpeed(40) != ERR_OK) {
    /* Handle error. */
  }

  /* Execute 10 micro-steps in forward direction. */
  if (MVH1_MoveMicroSteps(TRUE, 10) != ERR_OK) {
    /* Handle error. */
  }
  ...
}
```

Stepper motor control with acceleration ramp

This example demonstrates usage of full-step and micro-step mode with enabled acceleration and deceleration ramp.

Required component setup:

H-Bridge Model: MC33932 or MC34932

Motor Control Mode: Stepper

Timer Device: you must use FTM device (not TPM), beacuse we want to switch between full-step and micro-step in run-time (available only when FTM is used).

Output Control: PWM

Motor Control Mode: Full-step and Micro-step.

Initial Speed: for example 200 (full-step)

Acceleraion Ramp: for example 100 (full-step)

Micro-steps per Step: for example 2 Micro-steps

Initial Speed: for example 300 (micro-step)

Acceleraion Ramp: for example 150 (micro-step)

Set properties under groups *H-Bridge A MCU Interface* and *H-Bridge B MCU Interface*:

Device Mode: Normal Mode

Set Enable and Disable Pins and Input Control Pins according to used MCU.

Feedback Pin and Status Flag Pin are not necessary (you can disable these pins).

Auto Initialization: yes

Methods: Init, SetMode, AlignRotor, GetMotorStatus MoveSteps MoveMicroSteps

Events: OnActionComplete

Note that name of the MVHBridge component is MVH1 (this shortcut is used as prefix in MVHBridge methods).

Content of Main.c ():

Listing 5: Source code

```
void main(void)
{
  /* Error code. */
  LDD_TError Error = ERR_OK;

  ...

  /* Note: OnActionComplete Event handler will be called when alignment
   * of rotor is completed. */
  if ((Error = MVH1_AlignRotor()) != ERR_OK) {
    /* Handle error. */
  }
  ...
}
```

Content of Events.c ():

Listing 6: Source code

```
typedef enum {
  /* 1st test case: test of full-step mode with acceleration and
    deceleration ramp. */
  stcFULLSTEP_RAMP = 0,

  /* 2nd test case: test of micro-step mode with the ramp. */
  stcMICROSTEP_RAMP,

  /* All test case are completed in this state. */
  stcTESTS_COMPLETE,
} TStepperTestCase;

void MVH1_OnActionComplete(LDD_TUserData *UserDataPtr)
{
  /* Error code. */
  LDD_TError Error = ERR_OK;
  static TStepperTestCase TestCase = stcFULLSTEP_RAMP;

  /* Check if an error occurred during previous motor movement. */
  if (MVH1_GetMotorStatus() == msERROR) {
    /* Handle error. */
  }

  switch (TestCase) {
    case stcFULLSTEP_RAMP:
      /* Run the motor in forward direction. The full-step mode is used and
    number
```

```
        * of full-steps are 200 (1 mechanical revolution). */
       Error = MVH1_MoveSteps(TRUE, 200);
       if (Error != ERR_OK) {
         /* Handle error. */
       }
       /* Switch to next test case. */
       TestCase = stcMICROSTEP_RAMP;
       break;

     case stcMICROSTEP_RAMP:
       /* Run the motor in forward direction. The micro-step mode is used and
     number
        * of micro-steps are 400 (1 mechanical revolution). */
       Error = MVH1_MoveMicroSteps(TRUE, 400);
       if (Error != ERR_OK) {
         /* Handle error. */
       }
       TestCase = stcTESTS_COMPLETE;
       break;

     case stcTESTS_COMPLETE:
       /* Do nothing. */
     default:
       break;
   }
}
```

Custom mode

The MVHBridge component does not offer methods to control a motor in this mode. User must implement control logic. This example shows control of two DC brushed motors.

Required component setup:

H-Bridge Model: here is used dual H-Bridge model (MC33932 or MC34932), but you can use any H-Bridge model.

Device Mode: Normal Mode

Set properties under group *H-Bridge A MCU Interface*:

Input Control Pins: Two GPIO Pins

Set Enable and Disable Pins and Input Control Pins according to used MCU.

Feedback Pin and Status Flag Pin are not necessary (you can disable these pins).

Set properties under group *H-Bridge B MCU Interface*:

Input Control Pins: Two PWM Pins

Set Enable and Disable Pins and Input Control Pins according to used MCU.

Feedback Pin and Status Flag Pin are not necessary (you can disable these pins).

Auto Initialization: yes

Methods: Init, SetMode

Required setup of TimerUnit_LDD component used by MVHBridge:

Period: 11 kHz

Set these properties on all channels used by MVHBridge (channel 0 and 1 here):

Offset: 0

Output on compare: Clear

Output on overrun: Set

Methods: Init, SetOffsetTicks

Content of Main.c:

Listing 7: Source code

```
void main(void)
{
  /* Variable for saving error code. */
  LDD_TError error = ERR_OK;

  ...

  /* Control of H-Bridge interface A - two GPIO pins (two BitIO_LDD
   components). */
  IN1BitIO1_SetVal(PE_LDD_GetDeviceStructure(PE_LDD_COMPONENT_IN1BitIO1_ID))
   ;
  IN2BitIO1_ClrVal(PE_LDD_GetDeviceStructure(PE_LDD_COMPONENT_IN2BitIO1_ID))
   ;

  /* Control of H-Bridge interface B - two PWM pins (two channels of
   TimerUnit_LDD
   * component). Channels 0 (pin IN3) and 1 (pin IN4) of TimerUnit_LDD
   component
   * are used by H-Bridge. Set 0% PWM duty on channel 0 and 50% duty on
   channel 1. */
  Error = TU1_SetOffsetTicks(PE_LDD_GetDeviceStructure(
   PE_LDD_COMPONENT_TU1_ID), 0, 0);
  if (Error != ERR_OK) {
    /* Handle error. */
  }
  Error = TU1_SetOffsetTicks(PE_LDD_GetDeviceStructure(
   PE_LDD_COMPONENT_TU1_ID), 0, TU1_PERIOD_TICKS / 2);
  if (Error != ERR_OK) {
    /* Handle error. */
  }

  ...
}
```

4 User Types

***ComponentName*_THBrMode** = enum { hbmNORMAL, hbmSLEEP, hbmSTANDBY}

***ComponentName*_THBridge** = enum { hbBRIDGE_A, hbBRIDGE_B} *Type*

***ComponentName*_TMotorStatus** = enum { msRUNNING, msSTOP, msERROR} *Type*

How to Reach Us:

Home Page:
NXP.com

Web Support:
http://www.nxp.com/support

Document Number: PEXMVHBRIDGEPUG
Rev. 1.0
2/2016