NXP Semiconductors
User's guide

Document Number: PEXLVHBRIDGEPUG
Rev. 1.0, 2/2016

LVHBridge Programming Guide

Contents

1 General Info 2
2 Embedded Component Description 2
2.1 Component API 2
2.2 Events 4
2.3 Methods 4
2.4 Properties 8
3 Typical Usage 11
4 User Types 16

1 General Info

This documentation introduces LVHBridge Processor Expert component. LVHBridge is a software driver for DC brush motor and stepper motor control. This component creates a layer between hardware and user application and enables rapid application development by providing an easy to use interface. The component provides two sets of methods, one set per motor type. For DC brush motor control a single H-Bridge device is sufficient. The component utilizes to ways for DC motor control. The first way uses a timer device (TimerUnit_LDD component) which besides a simple on/off state switching also allows speed adjustment. The other way utilizes GPIO pins (BitIO_LDD components) which enable basic state control, switch a motor on or off. Stepper motors require a dual H-Bridge model to generate four output signals. The component provides both, full-stepping and micro-stepping control types, which enable precise motor control. This component supports and provides flexible software solution for these analog parts:

NXP MPC17C724: 0.4 A Dual H-Bridge Motor Driver

NXP MPC17510: 1.2 A 15 V H-Bridge Motor Driver

NXP MPC17511: 1.0 A 6.8 V H-Bridge Motor Driver

NXP MPC17529: 0.7 A Dual H-Bridge Motor Driver with 3.0V/5.0V Compatible Logic I/O

NXP MPC17531A: 0.7 A Dual H-Bridge Motor Driver with 3.0 V Compatible Logic I/O

NXP MPC17533: 0.7 A 6.8 V Dual H-Bridge Motor Driver

NXP MC34933: 1.4 A Dual H-Bridge Driver Compatible with 3.0 V Logic

NXP offers following board solutions based on these chips:

FRDM-17C724-EVB

FRDM-17510EJ-EVB

FRDM-17511EP-EVB

FRDM-17511EV-EVB

FRDM-17529EV-EVB

FRDM-17531EP-EVB

FRDM-17531EV-EVB

FRDM-17533EV-EVB

FRDM-34933EP-EVB

Detailed description can be found in related hardware user guides and datasheets.

2 Embedded Component Description

2.1 Component API

LVHBridge 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 LVHBridge 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. Components linked by HBridge

SetMode	This method sets HBridge device mode using enable pin.
SetGateDriver	This method controls Gate Driver Input (GIN) pin. It is available only for MPC17510 and MPC17511.
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) and is available only when property ”Control Mode” is set to ”State Control”.
SetTriState	This method sets output of specified HBridge to tristate (high impedance) using input control pins.
SetDirection	This method sets direction of brush motor. In forward direction the first IN pin is set to high (or PWM) and the second IN pin to low. In reverse direction the first IN is set to low and the second IN to high (or PWM). Change of the direction is applied when you start rotation (not when the motor is running).
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). Note that you must wait for completion of this action before you can run motor again (use method [GetMotorStatus] or event [OnActionComplete]).
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). Note that you must wait for completion of this action beforeyou can run motor again (use method [GetMotorStatus] or event [OnActionComplete]).
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. Note that you must wait for completion of this action before you can run motor again (use method [GetMotorStatus] or event [OnActionComplete]).
GetMotorStatus	This method returns status of stepper motor control. Possible values are defined in TMotorStatus enumeration in header file.
AlignRotor	Align rotor to a 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. Note that you must wait for completion of this action before you can run motor again (use method [GetMotorStatus] or event [OnActionComplete]).
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].
GetMicroStepPosition	This method returns current microstep position. Size of microstep depends on property "Microsteps per Step" in Processor Expert component GUI. 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

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.

*ANSI C prototype:*void OnActionComplete(LDD_TUserData *UserDataPtr)

LDD_TUserData : Pointer to UserDataPtr- 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. Components linked by H-Bridge (TimerUnitLDD) are not initialized here.

*ANSI C prototype:*void Init(LDD_TUserData *UserDataPtr)

LDD_TUserData : Pointer to UserDataPtr- Pointer to the user data. This pointer will be passed as an event or callback parameter.

SetMode -This method sets H-Bridge device mode using enable pin.

*ANSI C prototype:*void SetMode(bool Active) *bool :Active*- Desired H-Bridge mode. Put FALSE to set power save mode, TRUE for normal operational mode.

SetGateDriver -This method controls Gate Driver Input (GIN) pin. It is available only for MPC17510 and MPC17511.

*ANSI C prototype:*void SetGateDriver(bool OutputHigh)

bool :OutputHigh- TRUE to set GOUT pin to High, FALSE for Low.

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

*ANSI C prototype:*LDD_TError RotateProportional(uint8_t PWMDuty,THBridge Bridge)

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

THBridge :Bridge- Selection of H-Bridge interface. Only hbBRIDGE_1 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) and is available only when property "Control Mode" is set to "State Control".

*ANSI C prototype:*LDD_TError RotateFull(bool Rotate,THBridge Bridge)

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

THBridge :Bridge- Selection of H-Bridge interface. Only hbBRIDGE_1 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.

SetTriState -This method sets output of specified H-Bridge to tri-state (high impedance) using input control pins.

*ANSI C prototype:*LDD_TError SetTriState(THBridge Bridge)

THBridge :Bridge- Selection of H-Bridge interface. Only hbBRIDGE_1 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.

SetDirection -This method sets direction of brush motor. In forward direction the first IN pin is set to high (or PWM) and the second IN pin to low. In reverse direction the first IN is set to low and the second IN to high (or PWM). Change of the direction is applied when you start rotation (not when the motor is running).

*ANSI C prototype:*LDD_TError SetDirection(bool Forward,THBridge Bridge)

bool :Forward- Motor direction.

THBridge :Bridge- Selection of H-Bridge interface. Only hbBRIDGE_1 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.

*ANSI C prototype:*LDD_TError SetFullStepSpeed(uint16_t StepsSec)

uint16_t :StepsSec- 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.

> *ANSI C prototype:*LDD_TError SetMicroStepSpeed(uint16_t MicroStepsSec)
>
> *uint16_t :MicroStepsSec*- 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). Note that you must wait for completion of this action before you can run motor again (use method [GetMotorStatus] or event [OnActionComplete]).

> *ANSI C prototype:*LDD_TError MoveSteps(Boolean Forward,32bit unsigned Steps)
>
> *Boolean :Forward*- Motor direction.
>
> *32bit unsigned :Steps*- 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). Note that you must wait for completion of this action beforeyou can run motor again (use method [GetMotorStatus] or event [OnActionComplete]).

> *ANSI C prototype:*LDD_TError MoveMicroSteps(Boolean Forward,32bit unsigned MicroSteps)
>
> *Boolean :Forward*- Motor direction.
>
> *32bit unsigned :MicroSteps*- 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.

> *ANSI C prototype:*LDD_TError MoveContinual(bool Forward)
>
> *bool :Forward*- 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.

> *ANSI C prototype:*LDD_TError MoveMicroContinual(Boolean Forward)
>
> *Boolean :Forward*- 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. Note that you must wait for completion of this action before you can run motor again (use method [GetMotorStatus] or event [OnActionComplete]).

> *ANSI C prototype:*LDD_TError StopContinualMovement(void)
>
> *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.

> *ANSI C prototype:*TMotorStatus GetMotorStatus(void)
>
> *Return value:TMotorStatus* - Motor status.

AlignRotor -Align rotor to a 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. Note that you must wait for completion of this action before you can run motor again (use method [GetMotorStatus] or event [OnActionComplete]).

> *ANSI C prototype:*LDD_TError AlignRotor(void)
>
> *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.

> *ANSI C prototype:*LDD_TError SetMicroStepSize(uint8_t Size)
>
> *uint8_t :Size*- 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].

> *ANSI C prototype:*int32_t GetFullStepPosition(void)
>
> *Return value:int32_t* - Current position of rotor in number of full-steps taken from initial position.

GetMicroStepPosition -This method returns current micro-step position. Size of micro-step depends on property "Micro-steps per Step" in Processor Expert component GUI. Position is set to zero when initialization of H-Bridge component occurs. It can be reset by method [ResetFullStepPosition].

> *ANSI C prototype:*int32_t GetMicroStepPosition(void)

Return value:int32_t - Current position of rotor in number of micro-steps taken from initial position.

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

*ANSI C prototype:*LDD_TError ResetFullStepPosition(void)

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.

*ANSI C prototype:*LDD_TError DisableMotor(void)

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.

ActiveMode - Selection of H-Bridge operating mode.

There are 2 options:

yes

no

Enable Pins - The settings of H-Bridge enable pins.

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

Pin for PSAVE - Pin for logic input enable control of H-Bridges to save power (active-high).

PSAVE Linked - Inherited component for one-bit Input/Output to control PSAVE pin.

Pin for OE - Pin for logic output Enable control of H-Bridges (active-high).

OE Linked - Inherited component for one-bit Input/Output component to control OE pin.

Pin for EN - Enable control signal input pin (active-low).

EN Linked - Inherited component for one-bit Input/Output component to control EN pin.

Pin for GIN - Gate Driver Input pin to control GOUT pin (active-low). GIN is initialized to HIGH, so default value of GOUT is LOW.

GIN Linked - Inherited component for one-bit Input/Output component to control GIN pin.

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.

Timer Settings - The setings of timer devices, which are used to generate control signal. LVHBridge component sets automatically this item.

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

Primary Timer Component - Reference to TimerUnit_LDD component, which is used to generate signal for motor control. This timer is the source for global time base when property "Secondary Timer" is enabled.

Primary Timer Device - Name of a counter used by TimerUnit_LDD component. The counter must provide the source for the global time base (see datasheet of used MCU) if secondary timer is used (property "Secondary Timer" enabled).

Secondary Timer - Secondary timer can be used to generate signal for motor control. Enable the item when Input Control Pins aren't connected to one timer device. This timer is synchronized with Primary Timer Device. This setting is available only when property "Output Control" in "Stepper Motor" group is set to "PWM" or property "Control Mode" in "DC Brush" group is set to "Speed Control".

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

Secondary Timer Component - Reference to TimerUnit_LDD component, which is used to generate signal for motor control. This timer is synchronized with the global time base (Primary Timer Device).

Secondary Timer Device - Name of a 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 - The 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. For more information see component user guide.

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.

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

Acceleration - 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. Maximum value is 20 kHz.

Micro-steps per Step - Number of micro-steps per one full-step. The size can be changed later in C code. Note that micro-stepping table generated to C code contains only necessary number of items.

There are 5 options:

2 Micro-steps

:

32 Micro-steps

Speed - 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 - 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 1 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 1.

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 is 20 kHz.

Direction Control - Motor speed can be controlled in forward or reverse direction. You can set Bidirectional option to control speed in forward and reverse direction. This setting is available only when property "Control Mode" is set to "Speed control".

There are 3 options:

Bidirectional

Forward

Reverse

Init. Direction - Initial direction of brushed DC motor. Forward means that first IN pin is set to high and second IN pin to low. In reverse direction first IN pin is low and second IN high. Direction can be changed later in C code.

There are 2 options:

Forward

Reverse

Input Control Pins - Control mode for IN1 and IN2 pin. This attribute is set automatically by the component.

Pin for IN1 - Control signal input IN1 pin to control output OUT1.

IN1 Component - Inherited component for one-bit Input/Output component to control IN1 pin.

Pin for IN2 - Control signal input IN2 pin to control output OUT2.

IN2 Component - Inherited component for one-bit Input/Output component to control IN2 pin.

Input Control Pins - Control mode for IN1A and IN1B pin. This attribute is set automatically by the component.

Pin for IN1A - Control signal input IN1A pin to control output OUT1A.

IN1A Component - Inherited component for one-bit Input/Output component to control IN1A pin.

Pin for IN1B - Control signal input IN1B pin to control output OUT1B.

IN1B Component - Inherited component for one-bit Input/Output component to control IN1B pin.

H-Bridge 2 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 2.
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 is 20 kHz. This property is set automatically when "Control Mode" property in "H-Bridge 1 MCU Interface" group is "Speed Control".

Direction Control - Motor speed can be controlled in forward or reverse direction. You can set Bidirectional option to control speed in forward and reverse direction. This setting is available only when property "Control Mode" is set to "Speed control".

There are 3 options:

Bidirectional

Forward

Reverse

Init. Direction - Initial direction of brushed DC motor. Forward means that first IN pin is set to high and second IN pin to low. In reverse direction first IN pin is low and second IN high. Direction can be changed later in C code.

There are 2 options:

Forward

Reverse

Input Control Pins - Control mode for IN2A and IN2B pin. This attribute is set automatically by the component.

Pin for IN2A - Control signal input IN2A pin to control output OUT2A.

IN2A Component - Inherited component for one-bit Input/Output component to control IN2A pin.

Pin for IN2B - Control signal input IN2B pin to control output OUT2B.

IN2B Component - Inherited component for one-bit Input/Output component to control IN2B pin.

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

Primary Timer Allocator - The component is used to manage allocation of primary TimerUnit_LDD component channels.

Secondary Timer Allocator - The component is used to manage allocation of secondary TimerUnit_LDD component channels.

3 Typical Usage

Examples of typical settings and usage of LVHBridge component.

The state control of DC brushed motor

Required component setup:

Motor Control: Brushed

ActiveMode: yes

Set properties under group *H-Bridge 1 MCU Interface*. Set also *H-Bridge 2 MCU Interface* if you are using dual H-Bridge model:

Control Mode: State Control

Init. Direction: Forward

Auto Initialization: no

Methods: Init, RotateFull, SetDirection, SetTriState

Note: ”LVH1” is name of LVHBridge component.

Content of main.c:

Listing 1: Source code

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

  /* Initialization of LVHBridge component must be done maunually here,
   * because auto initialization is disabled. It is possible to pass pointer
   * to your own data (or NULL), which is then stored in device data
    structure. */
  LVH1_Init(&UserData);

  /* Run motor in forward direction. */
  if (LVH1_RotateFull(TRUE, hbBRIDGE_1) != ERR_OK) {
    /* Handle error. */
  }
  /* Note: insert here a waiting command to see the change. */

  /* Set H-Bridge output to tri-state. Motor should slowly stop. */
  if (LVH1_SetTriState(hbBRIDGE_1) != ERR_OK) {
    /* Handle error. */
  }
  /* Note: insert here a waiting command to see the change. */

  /* Change direction to "reverse". */
  if (LVH1_SetDirection(FALSE, hbBRIDGE_1) != ERR_OK) {
    /* Handle error. */
  }
  /* Run motor in reverse direction. */
  if (LVH1_RotateFull(TRUE, hbBRIDGE_1) != ERR_OK) {
    /* Handle error. */
  }
  /* Note: insert here a waiting command to see the change. */
  /* Stop motor. */
  if (LVH1_RotateFull(FALSE, hbBRIDGE_1) != ERR_OK) {
    /* Handle error. */
  }

  ...
}
```

Speed control of DC brushed motor

Required component setup:

Motor Control: Brushed

ActiveMode: yes

Set properties under group *H-Bridge 1 MCU Interface*. Set also *H-Bridge 2 MCU Interface* if you are using dual H-Bridge model:

Control Mode: Speed Control

Direction Control: Bidirectional

Init. Direction: Reverse

Auto Initialization: yes

Methods: Init, RotateProportional, SetDirection

Note: example also shows how to change 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)
{
  ...

  /* Run motor in reverse direction with PWM duty set to 75%. */
  if (LVH1_RotateProportional(75, hbBRIDGE_1) != ERR_OK) {
    /* Handle error. */
  }
  /* Note: insert here a waiting command to see the change. */

  /* Stop motor. */
  if (LVH1_RotateProportional(0, hbBRIDGE_1) != ERR_OK) {
    /* Handle error. */
  }
  /* Note: insert here a waiting command to see the change. */

  /* Set forward direction.*/
  if (LVH1_SetDirection(TRUE, hbBRIDGE_1) != ERR_OK) {
    /* Handle error. */
  }
  /* Run motor in forward direction with PWM duty set to 75%. */
  if (LVH1_RotateProportional(75, hbBRIDGE_1) != ERR_OK) {
    /* Handle error. */
  }

  ...
}
```

Stepper motor control

This example demonstrates usage of full-step and micro-step mode without acceleration ramp.

Required component setup:

H-Bridge Model: MC34933, MPC17529, MPC17C724, MPC17531A or MPC17533

ActiveMode: yes

Motor Control: Stepper

Primary Timer Device: Select Timer Device (TPM, FTM). Note that the counter must provide the source for the global time base (see datasheet of used MCU) if secondary timer is used (property ”Secondary Timer” enabled).

Secondary Timer: Enable Secondary Timer if H-Bridge input control pins are not connected to one timer device. You must set *Secondary Timer Device* when the Secondary Timer is enabled.

Output Control: PWM

Motor Control Mode: Full-step and Micro-step

Full-step Configuration

  Speed: for example 20

  Acceleration: 0

Micro-step Configuration

  PWM Frequency: for example 20 kHz

  Micro-steps per Step: for example 4 Micro-steps

  Speed: for example 50

  Acceleration: 0

Set properties under groups *H-Bridge 1 MCU Interface* and *H-Bridge 2 MCU Interface*:

Auto Initialization: yes

Methods: Init, SetMode, AlignRotor, GetMotorStatus, SetFullStepSpeed, SetMicroStepSpeed, MoveSteps, MoveMicroSteps, MoveMicroContinual, GetMicroStepPosition, StopContinualMovement

Micro-stepping uses a table with PWM duty cycle values corresponding to micro-steps. The table is defined in LVH1.c file (LVH1 is name of the LVHBridge component).

Content of main.c:

Listing 3: Source code

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

  /* Align the rotor to a full-step position (4 full-steps in forward
    direction). */
  if (LVH1_AlignRotor() != ERR_OK) {
    /* Handle error. */
  }
  while (LVH1_GetMotorStatus() == msRUNNING) {
    /* Wait until motor stops. */
  }
  /* Check possible error. */
  if (LVH1_GetMotorStatus() == msERROR) {
    /* Handle error. */
  }

  /* Change full-stepping speed to 50 full-steps per second. */
  if (LVH1_SetFullStepSpeed(50) != ERR_OK) {
    /* Handle error. */
  }

  /* Execute 25 full-steps in forward direction. */
  if (LVH1_MoveSteps(TRUE, 25) != ERR_OK) {
    /* Handle error. */
  }
  while (LVH1_GetMotorStatus() == msRUNNING) {
    /* Wait until motor stops. */
  }
  /* Check possible error. */
  if (LVH1_GetMotorStatus() == msERROR) {
    /* Handle error. */
  }

  /* Change micro-stepping speed to 100 micro-steps per second. */
  if (LVH1_SetMicroStepSpeed(100) != ERR_OK) {
    /* Handle error. */
  }

  /* Execute 100 micro-steps in reversed direction. */
  if (LVH1_MoveMicroSteps(FALSE, 100) != ERR_OK) {
    /* Handle error. */
  }
  while (LVH1_GetMotorStatus() == msRUNNING) {
    /* Wait until motor stops. */
  }
  /* Check possible error. */
  if (LVH1_GetMotorStatus() == msERROR) {
    /* Handle error. */
  }

  /* Run motor in continual mode in forward direction (i.e. until you
   * stop motor using method StopContinualMovement). */
  if (LVH1_MoveMicroContinual(TRUE) != ERR_OK) {
    /* Handle error. */
  }
  while (LVH1_GetMicroStepPosition() < 50) {
    /* Wait until motor executes at least 50 micro-steps. */
  }
  /* Stop motor. */
  if (LVH1_StopContinualMovement() != ERR_OK) {
    /* Handle error. */
  }
  while (LVH1_GetMotorStatus() == msRUNNING) {
    /* Wait until motor stops. */
  }
  /* Check possible error. */
  if (LVH1_GetMotorStatus() == msERROR) {
    /* Handle error. */
  }

  ...
}
```

Stepper motor control with acceleration ramp

This example demonstrates usage of full-step and micro-step mode with enabled acceleration ramp. Event handler OnActionComplete is used to detect movement completion instead of polling (i.e. calling method GetMotorStatus periodically).

Required component setup:

H-Bridge Model: MC34933, MPC17529, MPC17C724, MPC17531A or MPC17533

ActiveMode: yes

Motor Control: Stepper

Primary Timer Device: Select Timer Device (TPM, FTM). Note that the counter must provide the source for the global time base (see datasheet of used MCU) if secondary timer is used (property "Secondary Timer" enabled).

Secondary Timer: Enable Secondary Timer if H-Bridge input control pins are not connected to one timer device. You must set *Secondary Timer Device* when the Secondary Timer is enabled.

Output Control: PWM

Motor Control Mode: Full-step and Micro-step

Full-step Configuration

- *Speed*: for example 200
- *Acceleration*: 100

Micro-step Configuration

- *PWM Frequency*: for example 20 kHz
- *Micro-steps per Step*: for example 4 Micro-steps
- *Speed*: for example 400
- *Acceleration*: 200

Set properties under groups *H-Bridge 1 MCU Interface* and *H-Bridge 2 MCU Interface*:

Auto Initialization: yes

Methods: Init AlignRotor, GetMotorStatus, MoveSteps, MoveMicroSteps

Note that name of the LVHBridge component is LVH1 (this shortcut is used as prefix in LVHBridge methods).

Content of main.c ():

Listing 4: Source code

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

  /* Note: OnActionComplete Event handler will be called when alignment
   * of rotor is completed. */
  if (LVH1_AlignRotor() != ERR_OK) {
    /* Handle error. */
  }

  ...
}
```

Content of Events.c ():

Listing 5: Source code

```
...

/* This enumeration defines test cases executed in OnActionComplete event
    handler. */
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 LVH1_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 (LVH1_GetMotorStatus() == msERROR) {
    /* Handle error. */
  }

  switch (TestCase) {
    case stcFULLSTEP_RAMP:
      /* Switch to next test case. */
      TestCase = stcMICROSTEP_RAMP;

      /* Run motor in forward direction. The full-step mode is used and
   number
       * of full-steps are 200 (1 mechanical revolution). */
      if (LVH1_MoveSteps(TRUE, 200) != ERR_OK) {
        /* Handle error. */
      }

      break;

    case stcMICROSTEP_RAMP:
      /* Switch to next test case. */
      TestCase = stcTESTS_COMPLETE;

      /* Run motor in reversed direction. The micro-step mode is used and
   number
       * of micro-steps are 800 (1 mechanical revolution). */
      if (LVH1_MoveMicroSteps(FALSE, 800) != ERR_OK) {
        /* Handle error. */
      }
      break;

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

4 User Types

***ComponentName*_THBridge** = enum { hbBRIDGE_1, hbBRIDGE_2} *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: PEXLVHBRIDGEPUG
Rev. 1.0
2/2016