NXP Semiconductors
User's guide

Document Number: PEXVALVECONTROLLERPUG
Rev. 1.0, 2/2016

Valve Controller Programming Guide

Contents

1 General Info

This documentation introduces Processor Expert component named MC34ValveController. This component supports and provides flexible software solution for these analog parts:

NXP MC34SB0410: Quad Valve Controller System On Chip

NXP MC34SB0800: Octal Valve Controller System On Chip

NXP offers tower board solutions based on these chips, namely TWR-MC34SB0410 and TWR-MC34SB0800 boards. Detailed description can be found in related hardware user guides and datasheets.

2 Embedded Component Description

2.1 Component API

MC34ValveController 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 MC34ValveController Component API

Table 1

Method	Description
Init	Initializes the device with predefined values.
Deinit	Deinitializes the device. It sets reset pin (RSTB) to LOW. Valve controller consequently clears all registers of valve controller device.
WriteRegister	This method writes a value to selected SI register. It allocates SPI bus and calls internal function VC_write_register.
ReadRegister	This method reads a value from selected SO register. It allocates SPI bus and calls internal function VC_read_register.
GetControllerStatus	Gets selected status information. It reads content of two selected device registers and returns them. Then you can check possible faults.
ClearDriverFault	Clears selected fault flags. This method handles only faults related to driver modules (all lowside drivers, highside driver, pump motor predriver). It is not intended to clear faults of supervision module (i.e. RST_WD, RST_ALU, RST_EXT, RST_CLK, VINT_UV, VCC5_UV, DOSV_UV, OT, GND_LOSS, VPWR_UV, VPWR_OV).
SetDriverState	This method sets selected driver output value. It handles driver either by SPI bit (SPI control mode) or directly by output of the MCU (Direct control mode). In case of PWM control, "dsON" stands for predefined PWM duty, "dsOFF" means 0 percent duty.
SetPDPWMDuty	This method sets PWM duty cycle for pump motor predriver. It is available only when property "Input Control" of pump motor predriver is set to "PWM".
SetLSDPWMDuty	This method sets PWM duty cycle for selected lowside driver (LSD) for inductive loads. An error is returned when the selected LSD is not in PWM mode. It also reports an error when the PWM duty converted to target current is above limit (see property "Maximum Current").

SetLSDPWMFrequency	This method sets PWM frequency for selected lowside driver (LSD) for inductive loads.
SetLSDCurrent	This method sets current target for selected lowside driver for inductive loads (LSD). This method is blocking. When the lowside driver is in PWM mode the software PI regulation is utilized to reach current target. An error is returned when the current target is above limit (see property "Maximum Current").
SetPIRegulator	This method sets parameters of PI regulator.
GetLSDPWMDuty	This method returns PWM duty cycle for selected lowside driver for inductive loads (LSD). It can be used only when selected LSD is in current regulation mode.
GetLSDCurrent	This method returns current value for selected lowside driver for inductive loads. It can be used only when selected LSD is in PWMed mode.
GetADCValue	This method gets and interprets selected ADC value from a valve controller register.
FeedWatchdog	This method handles watchdog of valve controller. It sends MCU monitoring result computed for LFSR output received from the device.
GetLSDMode	This method returns mode for selected lowside driver for inductive loads (LSD).
GetResetPinVal	This method returns value of reset pin. When the pin is LOW the valve controller is in fault state. You can use method Init for recovery.
FlutterCurrent	This method checks whether to adjust the current of lowside driver and sets new current target to create sinusiodal current curve. Call this method as often as possible.

2.2 Events

There are no Events in this component

2.3 Methods

Init -Initializes the device with predefined values.

ANSIC prototype: TError Init(void)

Return value:TError - error code ERR_OK - success ERR_RST_EXT - RST pin level is low errors related to WriteRegister

Deinit -Deinitializes the device. It sets reset pin (RSTB) to LOW. Valve controller consequently clears all registers of valve controller device.

ANSIC prototype: void Deinit(void)

WriteRegister -This method writes a value to selected SI register. It allocates SPI bus and calls internal function VC_write_register.

ANSIC prototype: TError WriteRegister(uint8_t RegNum,uint16_t RegVal)

uint8_t :RegNum- Identifier of register to be written. It corresponds to message ID (MSG_ID) defined in valve controller datasheet. You can use macros defined in MC34SB0410.h or MC34SB0800.h according to valve controller model. Possible values for MC34SB0410 are in range [0 - 16] except reserved registers [2]. Possible values for MC34SB0800 are in range [0 - 26] except reserved registers [2, 19-23].

uint16_t :RegVal- New value of selected register.

Return value:TError - error code ERR_OK - success ERR_PARAM_VALUE - invalid parameter value ERR_BUS_OFF - SPI bus not available ERR_PARITY - wrong parity of send or received data errors related to internal functions responsible for SPI communication (for more details see PE_Error.h)

ReadRegister -This method reads a value from selected SO register. It allocates SPI bus and calls internal function VC_read_register.

ANSIC prototype: TError ReadRegister(uint8_t RegNum,uint16_t *RegValPtr)

uint8_t :RegNum- Identifier of register to be written. It corresponds to message ID (MSG_ID) defined in valve controller datasheet. You can use macros defined in MC34SB0410.h or MC34SB0800.h according to valve controller model. Possible values for MC34SB0410 are in range [0 - 16] except reserved registers [2]. Possible values for MC34SB0800 are in range [0 - 26] except reserved registers [2, 19-23].

uint16_t : Pointer to RegValPtr- pointer to memory where content of selected 16 bit register is stored.

Return value:TError - error code ERR_OK - success ERR_PARAM_VALUE - invalid parameter value ERR_PARAM_ADDRESS - invalid parameter address (null pointer) ERR_BUS_OFF - SPI bus not available ERR_PARITY - wrong parity of send or received data errors related to internal functions responsible for SPI communication (for more details see PE_Error.h)

GetControllerStatus -Gets selected status information. It reads content of two selected device registers and returns them. Then you can check possible faults.

ANSIC prototype: TError GetControllerStatus(TControllerStatus StatusSelection,uint16_t *StatusDataPtr)

TControllerStatus :StatusSelection- Type of status information to be read (fault status, version of device, etc.). Common inputs csDEVICE_INFO [M0R] version, [M1R] manufacturing data Specific inputs for MC34SB0410 - csSUPERVISION [M0R] RST_EXT, RST_CLK, VINT_UV, VCC5_UV, DOSV_UV, [M3R] OTW, GND_LOSS, VPWR_UV, VPWR_OV csPD [M3R] PD_OT [M4R] PD_OC csLS12 [M6R] VDS_LD1, LD1_OT, LD1_OP, LD1_OC [M7R] VDS_LD2, LD2_OT, LD2_OP, LD2_OC csLSD_CRERR [M5R] LSD1-4_CRER csLSD12 [M10-13R] VDS_LSD1-4, LSD1-4_OT, LSD1-4_OP, LSD1-4_OC csLSD34 Specific inputs for MC34SB0800 - csSUPERVISION [M0R] RST_WD, RST_ALU, RST_EXT, RST_CLK, VINT_UV, VCC5_UV, DOSV_UV, [M3R] OT, GND_LOSS, VPWR_UV, VPWR_OV csPD_HD [M5R] PD_OC, HD_OC [M18R] HD_LKG csLS [M6R] VDS_LD, LD_OT, LD_OP, LD_OC csLSD_CRERR [M8R] LSD1-4_CRER csLSD12 [M10-17R] VDS_LSD1-8, LSD1-8_OT, LSD1-8_OP, LSD1-8_OC csLSD34 csLSD56 csLSD78 csHS [M24R] VDS_HS, HS_OT, HS_OP, HS_OC

uint16_t : Pointer to StatusDataPtr- Two element array to store selected status information (i.e. Content of two selected device registers)

Return value:TError - error code ERR_OK - success ERR_PARAM_VALUE - invalid parameter value ERR_PARAM_ADDRESS - invalid parameter address (null pointer) errors related to ReadRegister

ClearDriverFault -Clears selected fault flags. This method handles only faults related to driver modules (all low-side drivers, high-side driver, pump motor pre-driver). It is not intended to clear faults of supervision module (i.e. RST_WD, RST_ALU, RST_EXT, RST_CLK, VINT_UV, VCC5_UV, DOSV_UV, OT, GND_LOSS, VPWR_UV, VPWR_OV).

ANSIC prototype: TError ClearDriverFault(TFaultSelection FaultSelection)

TFaultSelection :FaultSelection- Selection of fault to be cleared. It is not allowed to combine faults. If you want clear all faults, pass fsALL value. Common inputs - fsLSD, fsPD, fsALL Specific inputs for MC34SB0410 - fsLD1, fsLD2 Specific inputs for MC34SB0800 - fsLD, fsHD, fsHS

Return value:TError - error code ERR_OK - success ERR_PARAM_VALUE - invalid parameter value errors related to WriteRegister

SetDriverState -This method sets selected driver output value. It handles driver either by SPI bit (SPI control mode) or directly by output of the MCU (Direct control mode). In case of PWM control, "dsON" stands for predefined PWM duty, "dsOFF" means 0 percent duty.

ANSIC prototype: TError SetDriverState(TDriverSelection Driver,TDriverState State)

TDriverSelection :Driver- Selection of valve controller driver. Common inputs - dsPD Specific inputs for MC34SB0410 - dsLD1, dsLD2 Specific inputs for MC34SB0800 - dsHD, dsHS, dsLD

TDriverState :State- New output value of the selected driver. Possible values are dsON and dsOFF.

Return value:TError - error code ERR_OK - success ERR_PARAM_VALUE - invalid parameter value ERR_PARAM_ADDRESS - invalid parameter address (null pointer) ERR_PD_DIS - PD disabled in properties errors related to WriteRegister

SetPDPWMDuty -This method sets PWM duty cycle for pump motor pre-driver. It is available only when property "Input Control" of pump motor pre-driver is set to "PWM".

ANSIC prototype: TError SetPDPWMDuty(uint8_t Duty)

uint8_t :Duty- PWM duty cycle. Zero value stands for 0 percent and value 255 represents 100 percent duty cycle.

Return value:TError - error code ERR_OK - success errors related to SetRatio16 of PWM_LDD component

SetLSDPWMDuty -This method sets PWM duty cycle for selected low-side driver (LSD) for inductive loads. An error is returned when the selected LSD is not in PWM mode. It also reports an error when the PWM duty converted to target current is above limit (see property "Maximum Current").

ANSIC prototype: TError SetLSDPWMDuty(uint8_t LSD,uint8_t Duty)

uint8_t :LSD- Selection of low-side driver for inductive loads. Possible values are in range [1 - 4] for MC34SB0410 and in range [1 - 8] for MC34SB0800.

uint8_t :Duty- PWM duty cycle. Zero value stands for 0 percent and value 255 represents 100 percent duty cycle.

Return value:TError - error code ERR_OK - success ERR_PARAM_VALUE - invalid parameter value ERR_PARAM_RANGE - invalid parameter range ERR_LSD_MODE - invalid LSD mode errors related to WriteRegister

SetLSDPWMFrequency -This method sets PWM frequency for selected low-side driver (LSD) for inductive loads.

ANSIC prototype: TError SetLSDPWMFrequency(TLSDSelection LSDGroup,TLSDFrequency PWMFreq)

TLSDSelection :LSDGroup- Select LSD group to apply PWM frequency change. Always four LSD have common PWM frequency setting. Common inputs - lsLSD14 Specific inputs for MC34SB0800 - lsLSD58

TLSDFrequency :PWMFreq- PWM frequency. For possible values see definition of TLSDFrequency enumeration.

Return value:TError - error code ERR_OK - success ERR_PARAM_VALUE - invalid parameter value errors related to WriteRegister

SetLSDCurrent -This method sets current target for selected low-side driver for inductive loads (LSD). This method is blocking. When the low-side driver is in PWM mode the software PI regulation is utilized to reach current target. An error is returned when the current target is above limit (see property "Maximum Current").

ANSIC prototype: TError SetLSDCurrent(uint8_t LSD,uint16_t Current)

uint8_t :LSD- Selection of low-side driver for inductive loads. For example if you want to control LSD 1, put value 1.

uint16_t :Current- Current target in mili amps. Minimum is 0, maximum 2250 and step is 2.2 mili amps. This value is scaled to 10 bits and stored into register (rounding to the nearest available value).

Return value:TError - error code ERR_OK - success ERR_PARAM_VALUE - invalid parameter value ERR_PARAM_RANGE - invalid parameter range ERR_MAX_CURRENT - the current target is above limit (see property "Maximum Current") ERR_LSD_MODE - invalid LSD mode errors related to WriteRegister

SetPIRegulator -This method sets parameters of PI regulator.

ANSIC prototype: TError SetPIRegulator(TPCharacteristic PChar,TICharacteristic IChar,TIntegratorLimit Limit)

TPCharacteristic :PChar- Proportional characteristic of the PI regulator. For possible values see definition of TPCharacteristic enumeration.

TICharacteristic :IChar- Integration characteristic of the PI regulator. For possible values see definition of TICharacteristic enumeration.

TIntegratorLimit :Limit- Limit of the integrator. Possible values are ilLOW (0x03FF) and ilHIGH (0x07FF).

Return value:TError - error code ERR_OK - success ERR_PARAM_VALUE - invalid parameter value errors related to WriteRegister

GetLSDPWMDuty -This method returns PWM duty cycle for selected low-side driver for inductive loads (LSD). It can be used only when selected LSD is in current regulation mode.

ANSIC prototype: TError GetLSDPWMDuty(uint8_t LSD,uint8_t *DutyPtr)

uint8_t :LSD- Selection of low-side driver for inductive loads. Possible values are in range [1 - 4].

uint8_t : Pointer to DutyPtr- Pointer to memory where PWM duty value of selected driver is stored. Note that resulting value is in range [0 - 255] for minimum respective maximum duty.

Return value:TError - error code ERR_OK - success ERR_PARAM_VALUE - invalid parameter value ERR_PARAM_ADDRESS - invalid parameter address (null pointer) ERR_LSD_MODE - invalid LSD mode errors related to ReadRegister

GetLSDCurrent -This method returns current value for selected low-side driver for inductive loads. It can be used only when selected LSD is in PWMed mode.

ANSIC prototype: TError GetLSDCurrent(uint8_t LSD,uint16_t *CurrentPtr)

uint8_t :LSD- Selection of low-side driver for inductive loads. Possible values are in range [1 - 4].

uint16_t : Pointer to CurrentPtr- Pointer to memory where current value of selected driver is stored. Only lower 10 bits are used. Note that returned value is in range 0 - 4500 [mili Amps].

Return value:TError - error code ERR_OK - success ERR_PARAM_RANGE - invalid parameter range ERR_PARAM_ADDRESS - invalid parameter address (null pointer) ERR_LSD_MODE - invalid LSD mode errors related to ReadRegister

GetADCValue -This method gets and interprets selected ADC value from a valve controller register.

ANSIC prototype: TError GetADCValue(TADCSelection Selection,uint16_t *Result)

TADCSelection :Selection- Selection of measured value. Common inputs - asDIE_TEMP, asVINT_A, asVINT_D, asVPRE10, asVPRE12, asADIN1, asADIN2, asADIN3 Specific inputs for MC34SB0410 - asVGS_PD (voltage gate-source on PD) Specific inputs for MC34SB0800 - asVCP_VPWR (difference between Vcp and Vpwr)

uint16_t : Pointer to Result- Result of ADC measurement. (temperature [deg. of C.] or voltage [mV] scaled to internal reference VCC5)

Return value:TError - error code ERR_OK - success ERR_PARAM_VALUE - invalid parameter value errors related to ReadRegister

FeedWatchdog -This method handles watchdog of valve controller. It sends MCU monitoring result computed for LFSR output received from the device.

ANSIC prototype: TError FeedWatchdog(void)

Return value:TError - error code ERR_OK - success errors related to WriteRegister

GetLSDMode -This method returns mode for selected low-side driver for inductive loads (LSD).

ANSIC prototype: TLSDMode GetLSDMode(uint8_t LSD)

uint8_t :LSD- Selection of low-side driver for inductive loads. Possible values are in range [1 - 4].

Return value:TLSDMode - lsd mode Possible values lmPWM - driver is in PWM mode lmCR - driver is in current regulation mode

GetResetPinVal -This method returns value of reset pin. When the pin is LOW the valve controller is in fault state. You can use method Init for recovery.

ANSIC prototype: bool GetResetPinVal(void)

Return value:bool - level of RST pin (TRUE - high, FALSE - LOW)

FlutterCurrent -This method checks whether to adjust the current of low-side driver and sets new current target to create sinusiodal current curve. Call this method as often as possible.

ANSIC prototype: TError FlutterCurrent(void)

Return value:TError - error code ERR_OK - success errors related to WriteRegister

2.4 Properties

Component Name - Name of the component.

Valve Controller Model - Select valve controller model. Model selection affects which properties are available or hidden, enabled or disabled.

General Settings - Settings of valve controller.

SPI Link - Linked SPI_Device component.

Reset Pin Link - Link to a BitIO_LDD component.

Reset Pin - Select pin for RSTB. The pin is used for device reset and detection of fault state.

Discharge Slew Rate - Select slew rate used by pump motor pre-driver and high-side driver modules. When the power FET is switched off, the gate capacitance of the FET is discharged by a constant current, which is controlled fast or slow.

There are 2 options:

Slow: Slow slew rate (typ. 100 microA)

Fast: Fast slew rate (typ. 2 mA)

Clock Frequency - Select frequency of clock modules, i.e. the main supply clock CLK1 and auxiliary clock CLK2. Clock frequency is 14 MHz when "Fixed" option is selected. In other cases the frequency modulation is used. Two deviation frequencies are available to spread the oscillators energy over a wide frequency band.

There are 3 options:

Fixed

Modulated with 350 kHz: Deviation frequency is 350 kHz

Modulated with 700 kHz: Deviation frequency is 700 kHz

Internal Clock Monitoring - Set internal clock monitoring function. If disabled, it has no effect on functionality except clock monitoring function, because auxiliary clock CLK2 is deactivated. The main clock CLK1 remains active.

There are 2 options:

Enabled

Disabled

HS for Fail-safe Switch - Select initial state of high-side driver intended to control the fail-safe switch for the overall solenoid path.

There are 2 options:

On: Feature is on

Off: Feature is off

LSD for Inductive Loads - Settings of low-side drivers that control inductive loads.

Rise Time and Fall Time - Set rise time and fall time of low-side drivers.

There are 2 options:

Long: Long rise time (typ. 1.7 micro sec) and fall time (1.35 micro sec)

Short: Short rise time (typ. 0.5 micro sec) and fall time (1.0 micro sec)

Open Load Detection - Set low-side driver's sink current for open load detection.

There are 2 options:

Enabled

Disabled

PWM Frequency - Settings of output PWM frequency for low-side drivers.

Frequency of LSD 1 - 4 - Select output PWM frequency for low-side drivers 1 - 4.

There are 8 options:

3.0 kHz
3.2 kHz
3.4 kHz
3.6 kHz
3.9 kHz
4.2 kHz
4.5 kHz
5.0 kHz

Frequency of LSD 5 - 8 - Select output PWM frequency for low-side drivers 5 - 8

There are 8 options:

3.0 kHz
3.2 kHz
3.4 kHz
3.6 kHz
3.9 kHz
4.2 kHz
4.5 kHz
5.0 kHz

Current Regulation Mode - Settings of current regulation mode of low-side drivers. The load current is sensed by an internal low-side sense FET and digitized by an internal A/D converter. A digital current regulation circuitry compares the actual load current with the target current value and steers the duty cycle of the low-side power switch. The PI regulator is used.

PI Regulator - Settings of PI regulator used in current regulation mode.

P - Characteristic - Proportional characteristic of the PI regulator.

There are 15 options:

0.7812
0.8125
0.8438
0.875
0.9062
0.9375
0.9688
1
1.0312
1.0625
1.0938
1.125
1.1562
1.1875
1.2188

I - Characteristic - Integral characteristic of the PI regulator. Regulator stays idle until non-zero value is applied.

There are 8 options:

0.0312
0.0625
0.0938
0.125
0.1562
0.1875
0.25
0.3125

Integrator Limit - Set integrator limit. Possible values are 1023 (0x03FF) and 2047 (0x07FF).

There are 2 options:

1023: Limit is 1023 (0x03FF)

2047: Limit is 2047 (0x07FF)

Minimum PWM Duty - Select the minimum duty cycle of the low-side drivers 1 - 4 PWM outputs. This option affects time when the current meassurement occurs. Note that maximum duty cycle is 100%.

There are 4 options:

10%: Minimum PWM duty is 10%

3.12%: Minimum PWM duty is 3.12%

3.12% and 1.56%: Minimum PWM duty is 3.12% and duty 1.56% is forced every two cycles

3.12% and Skipping: Minimum PWM duty is 3.12% and skipping every two cycles

First PWM Duty - Select the first duty cycle of the low-side drivers 1 - 4 PWM outputs. The first duty cycle is either controlled by current or limited to a fixed duty cycle which a target current is transformed in.

There are 2 options:

Controlled by Current

Fixed Duty Cycle

Flutter Frequency Timer - Settings of a timer used by Flutter Frequency function.
The following items are available only if the group is enabled (the value is "Enabled"):

TimerInt_LDD Link - Link to a TimerInt_LDD component.

Timing Device - Name of a timing device used by TimerInt_LDD component.

LSD - Low-side driver for current regulated or PWMed valves.

Maximum Current - Select current limitation value in mA. This value is applied in component methods.

Control Mode - Set low-side driver mode. Either current regulation or PWM mode can be enabled in time. Current Regulation means that a digital current regulation circuitry compares the actual load current with the target current value and steers the duty cycle of the low-side power switch.

Target Current - Select target current in mA. Minimum value is 0 mA, maximum 2250.6 mA and step is 2.2 mA. Value is rounded to the nearest available value. In column "Details" you can see final value that is put into valve controller register.

Flutter Frequency - Flutter frequency settings.

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

Frequency - Frequency of sinusoidal current curve. When the Flutter Frequency function is utilized by two or more LSDs the frequency value is corrected. This corrections is needed, because all LSDs use one interrupt where the current is adjusted. So frequency of the interrupt must fit to all LSDs.

Points per Period - Number of points per period. One point correspond to a value of current.

There are 11 options:

8

12

16

20

24

28

32

36

40

44

48

Amplitude - Amplitude of sinusoidal curve in mili Amps.

PWM Duty - Select desired PWM duty cycle. Zero value stands for 0% and value 255 represents 100% duty cycle.

LSD - Low-side driver for current regulated or PWMed valves.

LSD - Low-side driver for current regulated or PWMed valves.

LSD - Low-side driver for current regulated or PWMed valves.

LSD - Low-side driver for current regulated or PWMed valves.

Maximum Current - Select current limitation value in mA. This value is applied in component methods.

PWM Duty - Select desired PWM duty cycle. Zero value stands for 0% and value 255 represents 100% duty cycle.

LSD - Low-side driver for current regulated or PWMed valves.

LSD - Low-side driver for current regulated or PWMed valves.

LSD - Low-side driver for current regulated or PWMed valves.

Pump Motor Pre-driver - Settings of DC motor pump module. Registers related to pump motor pre-driver are initialized to default values defined in datasheet when you select "Disabled". The following items are available only if the group is enabled (the value is "Enabled"):

Overcurrent Masking Time - Select masking time from direct input turn-on against malfunction on transient time. This masking time is used by overcurrent detection logic. T2 is typically 293 micro sec (see datasheet).

There are 2 options:

T2: Overcurrent masking time is equal to T2 (typ. 293 micro sec)

No Masking Time: No masking time

Overcurrent Masking Time - Select masking time from PDI turn-on against malfunction on transient time. This masking time is used by overcurrent detection logic. T1 is typically 18.2 micro sec (see datasheet).

There are 2 options:

Half of T1: Overcurrent masking time is equal to half of T2 (T1 is typ. 18.2 micro sec)

No Masking Time: No masking time

Overcurrent Filter Time - Select overcurrent filter timer of pump driver. The drain-source voltage of the FET on PD_G is checked if the high-side predriver is switched on. If the measured drain-source voltage exceeds the overcurrent voltage threshold, the output of the overcurrent comparator is enabled. If the output of the comparator is active longer than the defined filter time, the output PD_G is switched off. T1 is typically 18.2 micro sec and T2 is 293 micro sec (see datasheet).

There are 2 options:

T2: Overcurrent filter time is T2 (typ. 293 micro sec)

4 x T1: Overcurrent filter time is 4 x T1 (T1 is typ. 18.2 micro sec)

ControlMode - Select control mode. It is driven either directly by MCU pin ("Direct" mode) or through SPI interface ("SPI" mode). It is not possible to change the mode later when you select SPI mode due to selection of control pin.

Initial State - Select initial state of driver output.

There are 2 options:

On: Feature is on

Off: Feature is off

Input Control - Select control mode of driver input. Pump motor input pin can be controlled by GPIO or PWM port.

Control Pin Link - Link to a BitIO_LDD component.

Control Pin Link - Link to a PWM_LDD component.

Control Pin - Select pin to directly control the driver. This is the PDI pin when MC34SB0410 model is used or ADIN1 pin for MC34SB0800.

PWM Frequency - Select PWM frequency.

PWM Duty - Select PWM duty cycle. Zero value stands for 0 percent and value 255 represents 100 percent duty cycle.

Initial State - Select initial state of driver output.

There are 2 options:

On: Feature is on

Off: Feature is off

LD 1 for Resistive Charge - Settings of low-side driver 1 for general purpose.

There are 2 options:

On: Feature is on

Off: Feature is off

LD 2 for Resistive Charge - Settings of low-side driver 2 for general purpose. This driver is available only on MC34SB0410 device.

There are 2 options:

On: Feature is on

Off: Feature is off

HS for General Purpose - Select initial state of high-side driver for general purpose. It allows connecting and disconnecting loads like voltage dividers from the supply line, to reach low quiescent current of the total ECU or to driver small size relay driver.

There are 2 options:

On: Feature is on

Off: Feature is off

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 MC34ValveController component

Initialization of VAPS component.

Feeding watchdog.

Processing of VAPS status information.

Setting of driver state.

Setting of LSD current.

Setting of LSD PWM duty.

Setting of PD PWM duty.

Clearing of driver faults.

Measuring of die temperature and internal/external voltages.

Initialization of VAPS component.

This example shows how to handle device initialization when auto-initialization is disabled.

Required component setup and dependencies:

Properties: *Auto initialization*: Disabled

Methods: Init

Content of main.c:

Listing 1: Source code

```
void main(void)
{
  VC_TError Error;

  /* Calling Init method more then once in user code will restore initial
   Processor Expert setting. */

  Error = VC_Init();
  if (Error != ERR_OK) {
    /* Initialization was successful. */
  } else {
    /* Initialization was not successful. */
  }
}
```

Feeding watchdog.

This example shows how to feed watchdog to preserve communication between MCU and VAPS.

Note that this applies only for MC34SB0800 model.

Required component setup and dependencies:

Properties: *None*

Methods: FeedWatchdog

Content of main.c:

Listing 2: Source code

```
void main(void)
{
  VC_TError Error;

  /* Note that watchdog has to be called every 50ms. So user code has to
   count with that limitation. */

  while (1) {
    Error = VC1_FeedWatchdog();
    if (Error != ERR_OK) {
      /* Something went wrong. */
    }
  }
}
```

Processing of VAPS status information.

This example shows how to get and interpret VAPS status information.

Required component setup and dependencies:

Properties: *None*

Methods: GetControllerStatus

Content of main.c:

Listing 3: Source code

```
void main(void)
{
  VC_TError Error;
  uint16_t StatusData[2]; /* Note that StatusData is array of size 2. Every
   call of GetControllerStatus reads 2 related status registers. */
```

```
  /* Reads supervision data. */
  Error = VC1_GetControllerStatus(csSUPERVISION, StatusData);
  if (Error != ERR_OK) {
    /* Something went wrong. */
  }
  else { /* Interpretation of status data. */
    if (StatusData[0] & VC_M0R_RST_WD_MASK) { /* WD_RST (valid MR was not
    updated within t_wd) */
      /* Take some action. */
    }

    if (StatusData[0] & VC_M0R_RST_ALU_MASK) { /* WD_ALU (reset from
    monitoring module, ERR_CNT >= 101) */
      /* Take some action. */
    }

    if (StatusData[0] & VC_M0R_RST_EXT_MASK) { /* RST_EXT (reset from
    external pin (RSTB pin) */
      /* Take some action. */
    }

    if (StatusData[0] & VC_M0R_RST_CLK_MASK) { /* RST_CLK (SB0800 internal
    clock fault was detected) */
      /* Take some action. */
    }

    ...

  }

  /* The same procedure can be applied for the rest of status data. See
    GetControllerStatus description and MC34SB[0410/0800].h for other bit
    masks. */
}
```

Setting of driver state.

This example shows how to set selected driver state.

Required component setup and dependencies:

Properties: *None*

Methods: SetDriverState

Content of main.c:

Listing 4: Source code

```
void main(void)
{
  VC_TError Error;

  /* Sets PD ON. */
  /* Note that drivers which can be controlled by SPI or directly are
    handled the same way. */
  Error = VC1_SetDriverState(dsPD, dsON);
  if (Error != ERR_OK) {
    /* Something went wrong. */
  }

  ...
```

```
  /* Sets PD OFF. */
  Error = VC1_SetDriverState(dsPD, dsOFF);
  if (Error != ERR_OK) {
    /* Something went wrong. */
  }

  /* The same procedure can be applied for the rest of drivers (dsPD, dsLD
    [1,2], dsHS, dsHD), depends on the selected VAPS model. */
}
```

Setting of LSD current.

This example shows how to set LSD current.

Required component setup and dependencies:

Properties: *None*

Methods: SetLSDCurrent

Content of main.c:

Listing 5: Source code

```
void main(void)
{
  VC_TError Error;
  uint8_t Duty;

  if (VC1_GetLSDMode(1) != lmCR) {
    /* LSD is in PWM mode */
  }
  else {
    /* Sets current 500mA on LSD 1. */
    /* Note that this method returns error if LSD is in wrong mode or if
   desired current is above maximum current of selected LSD. */
    Error = VC1_SetLSDCurrent(1, 500);
    if (Error != ERR_OK) {
      /* Something went wrong. */
    }

    /* Waits until new current value is stabilized. */

    /* Reads corresponding duty value. */
    Error = VC1_GetLSDPWMDuty(1, &Duty);
    if (Error != ERR_OK) {
      /* Something went wrong. */
    }
  }

  /* The same procedure can be applied for the rest of LSD. */
  /* Note that only LSD 1-4 of both VAPS models are capable to work in lmCR
    (current regulation) mode. */
}
```

Setting of LSD PWM duty.

This example shows how to set LSD PWM duty.

Required component setup and dependencies:

Properties: *None*

Methods: SetLSDPWMDuty

Content of main.c:

Listing 6: Source code

```
void main(void)
{
  VC_TError Error;
  uint16_t Current;

  if (VC1_GetLSDMode(1) != lmPWM) {
    /* LSD is in CR mode */
  }
  else {
    /* Sets duty to 128 on LSD 1. */
    /* Note that this method returns error if LSD is in wrong mode or if
   duty is above maximum value. */
    Error = VC1_SetLSDPWMDuty(1, 128);
    if (Error != ERR_OK) {
      /* Something went wrong. */
    }

    /* Waits until new current value is stabilized. */

    /* Reads corresponding current value. */
    Error = VC1_GetLSDCurrent(1, &Current);
    if (Error != ERR_OK) {
      /* Something went wrong. */
    }
  }

  /* The same procedure can be applied for the rest of LSD. */
}
```

Setting of PD PWM duty.

This example shows how to set PD PWM duty.

Required component setup and dependencies:

Properties: *None*

Methods: SetPDPWMDuty

Content of main.c:

Listing 7: Source code

```
void main(void)
{
  VC_TError Error;

  /* Sets PD on. */
  Error = VC1_SetDriverState(dsPD, dsON);
  if (Error != ERR_OK) {
    /* Something went wrong. */
  }

  ...

  /* Changes PD PWM duty. */
  Error = VC1_SetPDPWMDuty(128);
  if (Error != ERR_OK) {
    /* Something went wrong. */
  }

  ...
```

```
  /* Sets PD OFF. */
  Error = VC1_SetDriverState(dsPD, dsOFF);
  if (Error != ERR_OK) {
    /* Something went wrong. */
  }
}
```

Clearing of driver faults.

This example shows how to clear faults of selected driver.

Required component setup and dependencies:

Properties: *None*

Methods: ClearDriverFault

Content of main.c:

Listing 8: Source code

```
void main(void)
{
  VC_TError Error;

  /* Clears faults of LSD drivers. */
  Error = VC1_ClearDriverFault(fsLSD);
  if (Error != ERR_OK) {
    /* Something went wrong. */
  }

  ...

  /* Clears faults of all drivers. */
  Error = VC1_ClearDriverFault(fsALL);
  if (Error != ERR_OK) {
    /* Something went wrong. */
  }

  /* Note that this cant be used to clear faults of supervision module.
   Those are cleared on read (see GetControllerStatus). */

}
```

Measuring of die temperature and internal/external voltages.

This example shows how to read measured values of VAPS.

Required component setup and dependencies:

Properties: *None*

Methods: GetADCValue

Content of main.c:

Listing 9: Source code

```
void main(void)
{
  VC_TError Error;
  uint16_t Result;

  /* Reads die temperature in degrees of Celsius. */
  Error = VC1_GetADCValue(asDIE_TEMP, &Result);
  if (Error != ERR_OK) {
    return Error;
```

```
  }

  ...

  /* Reads internal voltage in mV. See description of GetADCValue to get
    list of possible measured values. */
  Error = VC1_GetADCValue(asVINT_A, &Result);
  if (Error != ERR_OK) {
    return Error;
  }

  ...

  /* Reads external voltage in mV. */
  /* Note that ADIN1-3 input can be used to measure external voltage, but
    also those pins can be used to directly control some of drivers. */
  Error = VC1_GetADCValue(asADIN1, &Result);
  if (Error != ERR_OK) {
    return Error;
  }
}
```

4 User Types

***ComponentName*_TLSDMode** = enum { lmCR, lmPWM, lmUNDEFINED} *Mode of low-side driver for inductive loads.*

***ComponentName*_TADCSelection** = enum { asVINT_A, asVINT_D, asVPRE10, asVPRE12, asVGS_PD, asDIE_TEMP, asADIN1, asADIN2, asADIN3, asVCP_VPWR} *Selection of measured value from a valve controller register.*

***ComponentName*_TIntegratorLimit** = enum { ilLOW, ilHIGH} *Limit of the integrator used by LSD current regulator.*

***ComponentName*_TPCharacteristic** = enum { pch_1_0, pch_1_0312, pch_1_0625, pch_1_0938, pch_1_125, pch_1_1562, pch_1_1875, pch_1_2188, pch_0_9688, pch_0_9375, pch_0_9062, pch_0_875, pch_0_8438, pch_0_8125, pch_0_7812} *Proportional characteristic of the PI regulator for LSD.*

***ComponentName*_TICharacteristic** = enum { ich_0_125, ich_0_25, ich_0_1875, ich_0_1562, ich_0_3125, ich_0_0938, ich_0_0625, ich_0_0312} *Integration characteristic of the PI regulator for LSD.*

***ComponentName*_TLSDFrequency** = enum { lf_3_9_KHZ, lf_4_5_KHZ, lf_5_0_KHZ, lf_4_2_KHZ, lf_3_6_KHZ, lf_3_4_KHZ, lf_3_2_KHZ, lf_3_0_KHZ} *PWM Frequency values used for LSD.*

***ComponentName*_TLSDSelection** = enum { lsLSD14, lsLSD58} *LSD group selection. Always four LSDs have common PWM frequency setting.*

***ComponentName*_TControllerStatus** = enum { csSUPERVISION, csDEVICE, csPD, csLS, csLSD, csHS} *Type of status information to be read (fault status, version of device, etc.).*

***ComponentName*_TDriverState** = enum { dsOFF, dsON} *State of valve controller driver.*

***ComponentName*_TDriverSelection** = enum { dsPD, dsLD1, dsLD2, dsHD, dsHS, dsLD} *Selection of valve controller driver.*

***ComponentName*_TFaultSelection** = enum { fsLSD, fsPD, fsLD1, fsLD2, fsLD, fsHD, fsHS, fsALL} *Selection of driver fault. It is intended to clear faults.*

How to Reach Us:

Home Page:
NXP.com

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

Document Number: PEXVALVECONTROLLERPUG
Rev. 1.0
2/2016