MCU Motor Studio - Firmware Application Programming Interface (API) Specification (TMPM4K)

Description

MCU Motor Studio - Firmware
Application Programming Interface (API) Specification
(TMPM4K).

Conventions used in this document

Numerical Values

Hexadecimal number: 0xABC or 0h12F
Decimal number: 123 or 0d123 (explicitly indicating the decimal numbers)
Binary number: 0b111

Signals

Active low signals are indicated by a _N at the end of the signal name. Example: RESET_N.
Assertion of a signal shall mean its activation (transition to the active state). Deassertion of a signal shall mean its deactivation (transition to the inactive state).
Bus signals are indicated by [x:y] at the end of signal name. Example: DATA [3:0] indicates a four-bit bus with the individual bus signals DATA [3], DATA [2], DATA [1] and DATA [0].

Registers

Register names are indicated by square brackets […]. Example: [ABCD].
Two or more of the same kind of registers, fields, and bit names are collectively referred to by using a numerical suffix n. Example: [XYZ1], [XYZ2] and [XYZ3] are collectively referred to as [XYZn].
The bit width of a register is expressed as [x:y] where x is the number of the most significant bit and y is the number of the least significant bit. Example: [XYZ][3:0] indicates a four bit-wide register named XYZ.
The configuration value of a register is expressed by either a hexadecimal number or a binary number. Example: [ABCD].EFG = 0x01 (hexadecimal), [XYZn].XY = 0b1 (binary).
The following definitions apply for Bytes and Words:

Byte	8 bits
Half Word	16 bits
Word	32 bits
Double Word	64 bits

Unless specified otherwise, registers support only word access.
Register which are indicated to be reserved must not be rewritten. The read value from reserved registers must not be used.

Properties of each bit in a register are expressed as follows:

R	Read only
W	Write only
W1C	Write 1 Clear; the corresponding bit is cleared (=0) when "1" is written to this bit.
W1S	Write 1 Set; the corresponding bit is set (=1) when "1" is written to this bit.
R/W	Read and Write are possible.
R/W0C	Read/Write 0 Clear
R/W1C	Read/Write 1 Clear
R/W1S	Read/Write 1 Set
RS/WC	Read Set/Write Clear; set after read operation, cleared after write operation.

Reading from register bits having a default value of "—" will result in an unknown value.
In case of write accesses to registers containing both read/write (R/W) and read-only (R) bits, the read-only bits shall be written with their default value. If this default is "—", follow the instructions of each register.
Reserved bits of Write-only (W) register should be written with their default value. If this default is "—", follow the instructions of each register.

Table of Contents

List of Figures

List of Tables

1. Introduction

The TOSHIBA 3-Phase Motor's Vector Control Solution has two main components:

- A highly scalable and fully configurable Motor Control Firmware designed for the TMPM4K series MCUs, featuring Field Oriented Control (FOC) of up to three motors.

- The "MCU Motor Studio" GUI Tool for Windows, utilized for parameter configuration, drive control and real-time logging of various motor parameters in a high-speed Digital Storage Oscilloscope fashion.

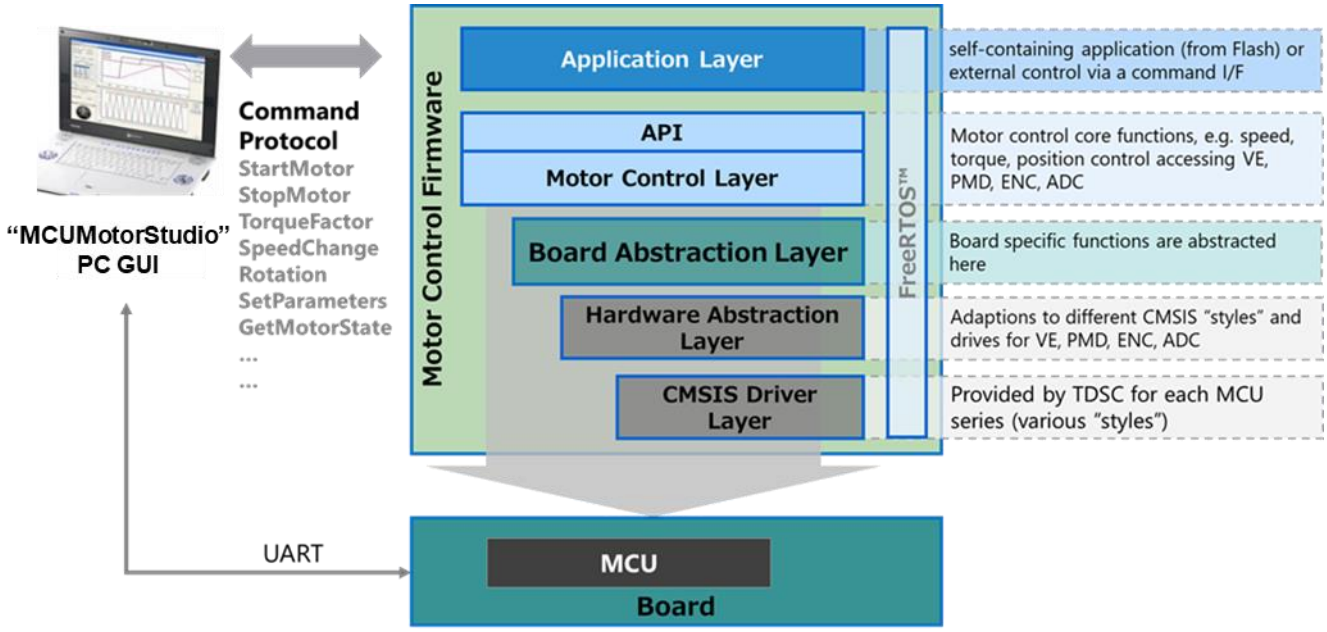

Figure 1: TOSHIBA 3-Phase Motor's Vector Control Solution

The main scope of this document is to describe the Application Programming Interface of the Motor Control Firmware, detailing the scope, parameters, usage and limitations.
All statements made in this guide are based on - "MCU Motor Studio" Firmware version 3.10 and "MCU Motor Studio PC Tool" version 4.1.0. Minor differences in the definitions are possible depending on the versions used.
The terms "Motor Control Firmware" and "the Firmware" as used throughout this document are referring to one and the same software entity and are fully interchangeable.

2. Design

Having scalability, extendibility and portability in mind a software design with multilayered structure was implemented. Every layer has its own responsibilities and provides services to the others, typically higher levels, via well-defined interfaces.
A top-level view of the firmware architecture is presented below:

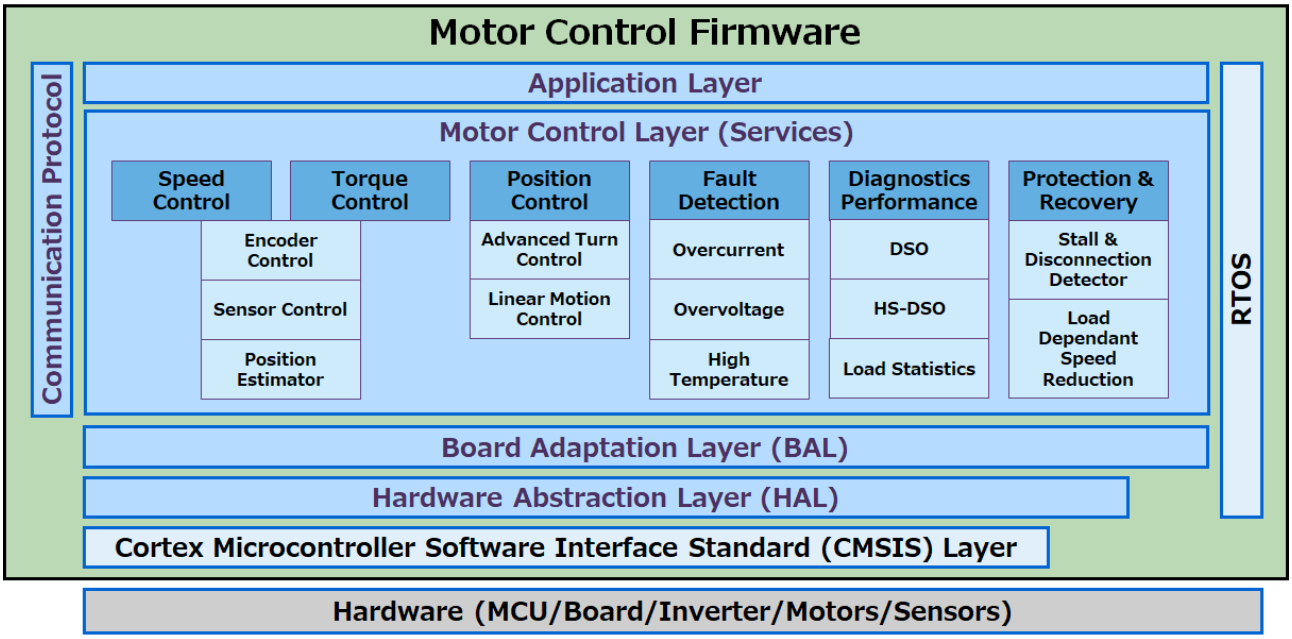

Figure 2: Motor Control Firmware Architecture

Application Layer – hosts the particular top-level user control implementation

Motor Control Layer – the core functionality for speed, torque and position control and various detection, protection and convenience services.

Board Adaptation Layer – integration and customization of the various platforms (boards) the firmware will run on.

Hardware Abstraction Layer – adapts to different versions (and implementations) of the CMSIS peripheral drivers available for the various MCUs, unifying their access methods and usage.

CMSIS Layer - includes the Cortex Microcontroller Software Interface Standard (CMSIS) compliant peripheral drivers (Device Peripheral Access Layer) dedicated for the particular MCU, adapting the differences in the type of peripherals, number of channels, etc.

FreeRTOS™– Resources management and prioritization in real time.

To ease and unify the interaction between subsystems, allowing simple exchange of modules and better testability, a well-defined & unified APIs for inter-layer communication were introduced. Those will be described with more details in the subsequent chapters.

3. Data Types & Constant Definitions

This chapter describes the data structure and global data variable utilized in the Motor control APIs.

3.1. Internal data structure

3.1.1. MotorParameters

Description - All parameters needed to describe a motor.			
File path		\Source\MotorControl\include\api.h	
Sl. No.	**Members**	**Unit**	**Description**
1.	PolePairs	NONE	Half of Poles
2.	Direction	NONE	MOTOR_CW_ONLY, MOTOR_CCW_ONLY, MOTOR_CW_CCW.
3.	Encoder	NONE	ENCODER_NO_ENCODER ENCODER_HALL_UVW ENCODER_HALL_UV ENCODER_INCENC_ABZ ENCODER_INCENC_AB ENCODER_SINGLE_PULSE
4.	EncoderUsage	NONE	ENCODER_USAGE_NONE ENCODER_USAGE_SPEED ENCODER_USAGE_EVENT ENCODER_USAGE_EVENT_IRQ
5.	EncoderResolution	NONE	Incremental rotary encoder counts per revolution
6.	EncoderMinPos	NONE	Minimal position for the linear motion in terms of encoder pulses
7.	EncoderStartPos	NONE	Start position for the linear motion in terms of encoder pulses
8.	EncoderMaxPos	NONE	Maximal position for the linear motion in terms of encoder pulses
9.	EncoderLMApproach	RPM	Absolute approach speed in RPM
10.	EncoderGearFactor	NONE	Gear factor
11.	EncoderReceiver	NONE	ENCODER_NO_RECEIVER, ENCODER_RESOLVER_RECEIVER, ENCODER_DIFFERENTIAL_RECEIVER
12.	EncoderReceiverMode	NONE	ENCODER_DIFF_D_HTL, ENCODER_DIFF_RS_422 ENCODER_DIFF_SE_HTL, ENCODER_DIFF_TTL
13.	MaxAngAcc	rad/sec^2	max angular velocity
14.	TorqueFactor	mNm/A	max torgue value * 10
15.	Resistance	mOhm	Winding resistance
16.	Inductance	uH	Winding inductance
17.	SpeedLimit	Hz or RPM	Limitation speed of motor

18.	SpeedChange	Hz or RPM	Max Forced speed of motor (no FOC)
19.	PositionDelay	ms	Time of Positioning Stage
20.	IqStart	mA	Start current (Iq)
21.	IdStart	mA	Start current (Id)
22.	IqLimit	mA	qaxis limitation current
23.	IdLimit	mA	daxis limitation current
24.	MotorId[MAX_LENGTH_MOTOR_ID]	NONE	name of the Motor device

3.1.2. PIControlSettings

Description - Settings for the PI Control Loop gain values			
File path		\Source\MotorControl\include\api.h	
Sl. No.	**Members**	**Unit**	**Description**
1.	PiIdKi	V/As	daxis current control Integral gain
2.	PiIdKp	V/A	daxis current control Proportional gain
3.	PiIqKi	V/As	qaxis current control Integral gain
4.	PiIqKp	V/A	qaxis current control Proportional gain
5.	PositionKi	Hz/Vs	Position estimation Integral gain (Ki) * 1000
6.	PositionKp	Hz/V	Position estimation Proportional gain (Kp) * 1000
7.	SpeedKi	mA/Hz*s	Speed Control Integral gain
8.	SpeedKp	mA/Hz	Speed Control Proportional gain

3.1.3. ChannelDependentValues

Description – Parameters that are channel dependent			
File path		\Source\MotorControl\include\api.h	
Sl. No.	**Members**	**Unit**	**Description**
1.	Deadtime	ns	Dead time of output switching polarity unit (steps of 100 ns)
2.	BootstrapDelay	ms	Bootstrap time
3.	SensitivityCurrentMeasure	mV/A	Input current(Iabc) at ADC =0xFFF0
4.	SensitivityVoltageMeasure	mV/V	Input voltage(Vdc) at ADC =0xFFF0
5.	GainCurrentMeasure	NONE	Gain setting for current measurement with 3 input lines (3-Shunt or 2Sensor)
6.	MeasurementType	NONE	SHUNT_TYPE_1 = 1,SHUNT_TYPE_3 = 3
7.	MeasurementDirection	NONE	Direction of the current measurement
8.	PolL	NONE	Active state for lower side FETs

9.	PolH	NONE	Active state for lower side FETs

3.1.4. SystemDependentValues

Description - Values that are dependent of the combination of Board and Motor			
File path		\Source\MotorControl\include\api.h	
Sl. No.	**Members**	**Unit**	**Description**
1.	PwmFrequency	Hz	PWM Frequency
2.	ShutdownMode	NONE	Shutdown Mode for the VE
3.	Braketime	ms	Time for braking when Short brake is selected
4.	Brakepercentage	ms	Percentage of braking mode when Short brake is selected
5.	RestartMode	NONE	Restart behavior
6.	StalldetectValue	NONE	vqi value for stall detection
7.	Overtemperature	Degree Celsius	Shutdown if temperature exceeds this temperature setting
8.	CanId	NONE	CAN ID of Board
9.	ExternalSpeedControl	NONE	ESC_NONE = 0, ESC_ADC = 1, ESC_PWM = 2)
10.	SwOvervoltage	V	Voltage for SW Overvoltage detection
11.	SwUndervoltage	V	Voltage for SW Under voltage detection
12.	SwOvercurrent	mA	Current for SW Overcurrent detection
13.	SpeedReduction	%	Percentage of IqLimitit when Speed Reduction shall be used

3.1.5. BoardInfoValues

Description - Values that give information about the board			
File path		\Source\MotorControl\include\api.h	
Sl. No.	**Members**	**Unit**	**Description**
1.	channels	NONE	Bit field for determine the available motor channels
2.	DSOSize	NONE	Number of Data Samples for DSO functionality
3.	BoardName[MAX_LENGTH_BOARD_NAME]	NONE	name of the board
4.	BoardNamePWR0[MAX_LENGTH_BOARD_NAME]	NONE	name of the power board channel 0
5.	BoardNamePWR1[MAX_LENGTH_BOARD_NAME]	NONE	name of the power board channel 1
6.	BoardRevision	NONE	revision of the board

3.1.6. MotorStateSettings

Description - Actual speed, direction, used current and used torque at the time of request			
File path		\Source\MotorControl\include\api.h	
Sl. No.	**Members**	**Unit**	**Description**
1.	ActualSpeed	RPM	Actual Rotation speed of Motor - negative is CCW
2.	TargetSpeed	RPM	Target Rotation speed of Motor - negative is CCW
3.	Current	mA	Current of motor
4.	Torque	Ncm	Torque of Motor
5.	Timestamp	Systick	systick counter of system

3.1.7. MotorExtendedStateSettings

Description - Actual control parameter value, target control value, used current and actual non-control value at the time of request			
File path		\Source\MotorControl\include\api.h	
Sl. No.	**Members**	**Unit**	**Description**
1.	ActualControlledParameter	RPM or Ncm	Actual Rotation speed or Torque of the Motor - negative speed is CCW
2.	TargetControlledParameter	RPM or Ncm	Target Rotation speed of Torque of the Motor - negative speed is CCW
3.	Current	mA	Current used to drive the motor
4.	NonControlParameter	Ncm or RPM	Current Torque or Speed of the Motor
5.	Timestamp	systicks	Systick counter of system
6.	ControlMethod	NONE	Idle, Control by Speed, Control by Torque

3.1.8. TurnStateSettings

Description – State values for Turn control.			
File path		\Source\MotorControl\include\api.h	
Sl. No.	**Members**	**Unit**	**Description**
1.	ActualCalculatedEncoderValue	RPM	Actual Rotation speed of Motor - negative is CCW
2.	CalculatedEncoderValue	lines	Calculated number of lines
3.	HallEncoderValue	pulses	Current number of HAL sensor pulses
4.	LineEncoderValue	lines	Current number of line encoder lines
5.	Timestamp	systicks	systick counter of system

3.1.9. ConfigDsoLogReq

Description – DSO configuration settings		
File path		\Source\MotorControl\include\api.h
Sl. No.	**Members**	**Description**
1.	motor_nr	Motor number
2.	selected_values	Bitfield with DSO_Selection enumerated values for selecting the signals to be logged
3.	trigger_on_value	Bitfield with DSO_Selection enumerated values for selecting the signals to trigger on
4.	trigger_mode	Bitfield with Trigger Mode enumerated values to select the trigger mode of the DSO
5.	trigger_level	Trigger level for the DSO
6.	spread_factor	Factor of which the logging is spread

3.1.10. DoTurnReq

Description - This structure contains all required information for a do turn request of a motor		
File path		\Source\MotorControl\include\api.h
Sl. No.	**Members**	**Description**
1.	motor_nr	Motor number:
2.	turns	Number of turns to perform
3.	additional_angle	Additional Angle in 1/10 degree in same direction as turns
4.	max_speed	Maximum rpm while turning

3.1.11. DoLinearMotionReq

Description - This structure contains all required information for a do linear motion request of a motor		
File path		\Source\MotorControl\include\api.h
Sl. No.	**Members**	**Description**
1.	motor_nr	Motor number:
2.	abs_postion	New absolute position in terms of encoder pulses
3.	max_speed	Maximum rpm while turning

4. Application Programming Interface

Every single API described in this chapter is fully implemented and tested for at least one family/family member and can be easily ported to any other.

If an API is specific to a family/family member this will be explicitly stated.

4.1. Hardware Abstraction Layer

4.1.1. ADC

All declarations and definitions are provided in **hal_adc.h**. The implementation is family specific and can be found in the corresponding family folder (M4KN) under **hal_adc.c**.

4.1.1.1. ADC Initialization

Synopsis	Initialize the ADC unit and channels, including clock gating, trigger mode, mode of operation and interrupt handling, that will be used for current measurement of the specified motor channel
Prototype	void HAL_ADC_Init(uint8_t channel_number, CURRENT_MEASUREMENT mesure-ment_type)
Parameters	Channel_number - the motor channel to be configured. Possible values are dependent on the number fi supported one. Currently these are limited to 0, 1 or 2
	mesurement_type – the sensor type used for the current measurement.
Return Value	None
Limitations	The assignment between ADC unit and motor channel is static.
Usage example(s)	HAL_ADC_Init(0, CURRENT_SHUNT_3);

4.1.1.2. DC Link Voltage Retrieval

Synopsis	Obtain the last successful/currently measured DC link voltage value for the motor channel specified
Prototype	uint16_t HAL_ADC_GetVDC(uint8_t channel_number)
Parameters	Channel_number - the motor channel to be configured. Possible values are dependent on the number fi supported one. Currently these are limited to 0, 1 or 2
Return Value	16-bit result of the measurement, that shall be scaled and converted to unit of Volts, considering the sensitivity of the power board circuitry.
Limitations	None

Usage example(s)	if(CHANNEL_IS_USED == ChannelUsageConfiguration[motor_nr])
	{
	voltage = HAL_ADC_GetVDC(motor_nr) * 10 * 5;
	voltage = * pow10(-ChannelValues[motor_nr].SensitivityVoltageMeasure.unit) / ChannelValues[motor_nr].SensitivityVoltageMeasure.value / 0xFFFU;

4.1.1.3. DC Link Voltage Protection

Synopsis	Configure the ADC compare values for automatic under and over voltage detection for the motor channel specified.
Prototype	void HAL_ADC_OverUndervoltageDetect(uint8_t channel_number)
Parameters	Channel_number - the motor channel to be configured. Possible values are dependent on the number fi supported one. Currently these are limited to 0, 1 or 2.
Return Value	None
Limitations	None
Usage example(s)	HAL_ADC_OverUndervoltageDetect(1);

4.1.2. Internal Amplifier/Comparator

This HAL module is only available for the M37X family. All declarations and definitions are provided in hal_amp_cmp.h. The implementation is family specific and can be found in the corresponding family folder (M4KN) under hal_amp_cmp.c.

4.1.2.1. Channel Configuration

Synopsis	Configure and enable the selected comparator channel, including the amplifier and its gain
Prototype	void HAL_AMPCMP_setup(M37x_AMPCMP_CHANNEL channel_number, M37x_GAIN gain)
Parameters	Channel_number - the comparator channel to be configured. Possible values are specified with the enumerated value M37x_AMPCMP_CHANNEL.
	gain – amplifier gain among the selectable factors, hardware implementation specific, 1.5x up to 10x for M370 for example.
Return Value	None
Limitations	None
Usage example(s)	/* Configure and enable Channel A with amplification factor of 10x */ HAL_AMPCMP_setup (CHANNEL_A, 0x7);

4.1.3. Clock Generation

All declarations and definitions are provided in hal_cg.h. The implementation is family specific and can be found in the corresponding family folder (M4KN) under hal_cg.c.

4.1.3.1. Middle Speed Clock Configuration A

Synopsis	Enable the middle speed clock(s) for the specified peripheral(s) of group A. Please refer to [6] Reference Manual Clock Control and Operation Mode (CG-M4K(2)-E), Revision 1.0, May 2018, Toshiba Electronic Devices & Storage Corporation for details on the M4KN clock or the corresponding manual for older families. Several clocks can be set with one call.
Prototype	int HAL_CG_EnableMiddleSpeedClockA(HAL_CG_CLK_A clock)
Parameters	clock - the peripheral middle speed clock to be enabled as specified in HAL_CG_CLK_A enumeration.
Return Value	0 – on success, otherwise failure
Limitations	The interface is split into two groups.
Usage example(s)	HAL_CG_EnableMiddleSpeedClockA(HAL_CG_CLK_T32A_CH1);

4.1.3.2. Middle Speed Clock Configuration B

Synopsis	Enable the middle speed clock(s) for the specified peripheral(s) of group B. Please refer to [6] Reference Manual Clock Control and Operation Mode (CG-M4K(2)-E), Revision 1.0, May 2018, Toshiba Electronic Devices & Storage Corporation for details on the M4KN clock or the corresponding manual for older families. Several clocks can be set with one call.
Prototype	int HAL_CG_EnableMiddleSpeedClockB(HAL_CG_CLK_B clock)
Parameters	clock - the peripheral middle speed clock to be enabled as specified in HAL_CG_CLK_B enumeration.
Return Value	0 – on success, otherwise failure
Limitations	The interface is split into two groups.
Usage example(s)	/* Enable the clock of the encoder instance */ HAL_CG_EnableMiddleSpeedClockB(HAL_CG_CLK_A_ENC_CH0);

4.1.3.3. Fc Clock Supply Control

Synopsis	Enable or disable supplying the clock fc to peripherals. Please refer to [6] Reference Manual Clock Control and Operation Mode (CG-M4K(2)-E), Revision 1.0, May 2018, Toshiba Electronic Devices & Storage Corporation for details on the M4KN clock or the corresponding manual for older families. Several clocks can be set with one call.
Prototype	void HAL_CG_SetFcSupplyPeriph(uint32_t Periph, FunctionalState NewState)
Parameters	Periph – the peripheral unit whose state will be altered.
	NewState – the desired new state, one of disable or enable
Return Value	None
Limitations	None
Usage example(s)	HAL_CG_SetFcSupplyPeriph((HAL_CG_FC_PERIPH_DNF_A | HAL_CG_FC_PERIPH_DNF_B | HAL_CG_FC_PERIPH_DNF_C), ENABLE);

4.1.3.4. ADC Clock Supply Control

Synopsis	Enable or disable supplying the clock (fc or fplladc) to the selected ADC unit. One clock at a time may only be set.
Prototype	void HAL_CG_SetAdcClkSupply(TSB_AD_TypeDef * ADx, FunctionalState NewState)
Parameters	ADx – the ADC unit whose state will be altered
	NewState – the desired new state, one of disable or enable
Return Value	None
Limitations	One channel may only be configured at a time
Usage example(s)	HAL_CG_SetAdcClkSupply(TSB_ADA, ENABLE);

4.1.4. Advanced Encoder

All declarations and definitions are provided in hal_encoder.h. The implementation is family specific and can be found in the corresponding family folder (M4KN) under hal_encoder.c.

4.1.4.1. Encoder Initialization

Synopsis	Initialize the Encoder channels, including clock gating, mode, usage and interrupt handling.
Prototype	void ENC_Init(unsigned char channel_number)
Parameters	channel_number – the encoder channel
Return Value	None
Limitations	A static relation between the encoder channel and the motor control channel.
Usage example(s)	ENC_Init(0);

4.1.4.2. Reset Counter

Synopsis	Reset the counter of the selected encoder channel.
Prototype	void ENC_ClearEncCounter(uint8_t channel_number)
Parameters	Channel_number – the encoder channel whose pulse counter it to be reset to 0
Return Value	None
Limitations	None
Usage example(s)	ENC_ClearEncCounter (0);

4.1.4.3. Angle Retrieval

Synopsis	Retrieve the current angular angle in electrical degrees and sector information.
Prototype	uint16_t ENC_GetTheta(unsigned char channel_number)
Parameters	Channel_number – the encoder channel
Return Value	16-bit value being the electrical angle in the sector we are currently in
Limitations	None
Usage example(s)	MotorControl[channel_number].Theta.part.reg = ((ENC_GetTheta(channel_number)) / 360; MotorControl[channel_number].Theta.part.reg *= FIXPOINT_16;

4.1.4.4. Angular Speed and Position Estimation Retrieval

Synopsis	Retrieve the sensor input based calculation of angular speed and position (angle).
Prototype	void ENC_Determine_Omega_Theta(unsigned char channel_number)
Parameters	channel_number – the encoder channel
Return Value	None
Limitations	The result is directly updated in the corresponding global state variables MotorControl[channel_number].Omega and MotorControl[channel_number].Theta, thus a the encoder channel is statically assigned to the motor control channel.
Usage example(s)	ENC_Determine_Omega_Theta(0);

4.1.4.5. Set Compare Value

Synopsis	Set a compare value for the completion interrupt generation in event counting mode.
Prototype	void ENC_SetEncoderIRQValue(uint8_t channel_number, uint8_t tweak_nr, uint32_t value)
Parameters	channel_number – the encoder channel
	tweak_nr – stage ID
	value – the counter value to be matched/waited upon
Return Value	None
Limitations	None
Usage example(s)	ENC_SetEncoderIRQValue(0, 1, WaitValueForShutdown[0]);

4.1.4.6. Enable Encoder Interrupt

Synopsis	Enable interrupt generation for the selected encoder channel.
Prototype	void ENC_EnableIRQ(unsigned char channel_number)
Parameters	channel_number – the encoder channel
Return Value	None
Limitations	None
Usage example(s)	ENC_EnableIRQ(0);

4.1.4.7. Disable Encoder Interrupt

Synopsis	Disable interrupt generation for the selected encoder channel.
Prototype	void ENC_DisableIRQ(unsigned char channel_number)
Parameters	channel_number – the encoder channel
Return Value	None
Limitations	None
Usage example(s)	ENC_DisableIRQ(0);

4.1.5. Flash Controller

Not fully implemented.

4.1.6. GPIO

All declarations and definitions are provided in hal_gpio.h. The implementation is family specific and can be found in the corresponding family folder (M4KN) under hal_gpio.c.

4.1.6.1. IO Functional Initialization

Synopsis	Configure the selected IO pin to the desired function.
Prototype	uint8_t HAL_GPIO_Init(HAL_GPIO_Port port, uint8_t bit, gpio_function function)
Parameters	port – the I/O PORT channel
	bit – the exact I/O
	function – the desired function
Return Value	0 – success 1 – failure
Limitations	None
Usage example(s)	/* Configure PG1 as SPI1 CS1 */ HAL_GPIO_Init(HAL_GPIO_PORT_G, HAL_GPIO_BIT_1, HAL_GPIO_SPI_OUTPUT);

4.1.6.2. Set Pin Output Level

Synopsis	Sets the pin level of the specified port pin
Prototype	uint8_t HAL_GPIO_SetBit (HAL_GPIO_Port port, uint8_t bit, uint8_t value)
Parameters	port – the I/O PORT channel
	bit – the exact I/O
	value – the output value
Return Value	0 – success 1 – failure
Limitations	Shall only be used with pins that are configured as general-purpose outputs.
Usage example(s)	HAL_GPIO_SetB (HAL_GPIO_PORT_C, HAL_GPIO_BIT_3, HAL_GPIO_HIGH_LEVEL);

4.1.6.3. Read Pin Input Level

Synopsis	Gets the pin level of the specified port pin.
Prototype	uint8_t HAL_GPIO_GetBit (HAL_GPIO_Port port, uint8_t bit)
Parameters	port – the I/O PORT channel
	bit – the exact I/O
	value – the output value
Return Value	0 – Low level 1 – High level
Limitations	Shall only be used with pins that are configured as general-purpose inputs.
Usage example(s)	Level = HAL_GPIO_GetB (HAL_GPIO_PORT_C, HAL_GPIO_BIT_3);

4.1.6.4. Set Port Output Levels

Synopsis	Sets the pin level of all pins in the specified port.
Prototype	uint8_t HAL_GPIO_SetValue (HAL_GPI_Port port, uint8_t value)
Parameters	port – the I/O PORT channel
	value – the output value
Return Value	0 – success 1 – failure
Limitations	Shall only be used with ports whose pins are configured as general-purpose outputs
Usage example(s)	HAL_GPIO_SetValue (HAL_GPIO_PORT_C, HAL_GPIO_LOW_LEVEL);

4.1.6.5. Read Port Input Levels

Synopsis	Gets the pin level of all pins in the specified port.
Prototype	uint8_t HAL_GPIO_GetValue (HAL_GPIO_Port port)
Parameters	port – the I/O PORT channel
Return Value	n – Pin level coded for each pin
Limitations	Shall only be used with pins that are configured as general-purpose inputs.
Usage example(s)	Level = HAL_GPIO_GetValue (HAL_GPIO_PORT_A);

4.1.7. I2C

All declarations and definitions are provided in hal_i2c.h. The implementation is family specific and can be found in the corresponding family folder (M4KN) under hal_i2c.c.

4.1.7.1. Initialize I2C Chanel

Synopsis	Initialize the I2C channels, setting the IOs and interrupt handling
Prototype	HAL_I2C_Result HAL_I2C_Init(uint8_t channel, uint8_t port)
Parameters	channel – the I2C channel to be configured
	port – the I/O PORT for the selected channel (limitations apply, please refer to the datasheet)
Return Value	HAL_I2C_OK – success HAL_I2C_ERROR – on failure HAL_I2C_INVALID_CH_PORT_COMB – invalid channel and port combination HAL_I2C_INVALID_CHANNEL – unsupported channel
Limitations	Shall not be invoked in interrupt/exception context.
Usage example(s)	HAL_I2C_Result HAL_I2C_Init(HAL_I2C_CH0, HAL_GPIO_PORT_C);

4.1.7.2. Configure Frequency & Clock

Synopsis	Initialize the I2C channel, including clock gating, frequency source and registers.
Prototype	HAL_I2C_Result HAL_I2C_Frequency(uint8_t channel, int32_t Hz);
Parameters	channel – the I2C channel to be configured
	hz – the used clock frequency in HZ
Return Value	HAL_I2C_OK – success HAL_I2C_ERROR – on failure
Limitations	Shall not be invoked in interrupt/exception context.
Usage example(s)	HAL_I2C_Frequency(HAL_I2C_CH0, HAL_I2C_FREQUENCY1);

4.1.7.3. Reset Channel

Synopsis	Perform software reset on the selected I2C channel.
Prototype	void HAL_I2C_Reset(uint8_t channel);
Parameters	channel – the I2C channel to be reset
Return Value	None
Limitations	Shall not be invoked in interrupt/exception context.
Usage example(s)	HAL_I2C_Reset (HAL_I2C_CH0);

4.1.8. Oscillation Frequency Detector

All declarations and definitions are provided in hal_ofd.h. The implementation is family specific and can be found in the corresponding family folder (M4KN) under hal_ofd.c.

4.1.8.1. Enable Oscillation Frequency Detection

Synopsis	Enable the monitoring and signaling of the OFD.
Prototype	void HAL_OFD_Enable(void)
Parameters	None
Return Value	None
Limitations	None
Usage example(s)	HAL_OFD_Enable ();

4.1.8.2. Disable Oscillation Frequency Detection

Synopsis	Disable the monitoring and signaling of the OFD.
Prototype	void HAL_OFD_Disable(void)
Parameters	None
Return Value	None
Limitations	None
Usage example(s)	HAL_OFD_Disable ();

4.1.8.3. Detection Parameters Configuration

Synopsis	Set the PLL and limiting count values for the detection
Prototype	void HAL_OFD_SetDetectionFrequency(halOfdPllState state, uint32_t higherDetectionCount, uint32_t lowerDetectionCount);
Parameters	state – usage status of the main PLL (on or off)
	higherDetectionCount – the count value of the higher detection frequency
	lowerDetectionCount – the count value of the lower detection frequency
Return Value	None
Limitations	None
Usage example(s)	HAL_OFD_SetDetectionFrequency (OFD_PLL_ON, OFD_MAX_VAL, OFD_MIN_VAL);

4.1.8.4. Configuration Code Write Control

Synopsis	Enable/disable the writing of a configuration code
Prototype	void HAL_OFD_SetRegWriteMode(halOfdFunctionalState newState)
Parameters	newState – Enable or disable
Return Value	None
Limitations	None
Usage example(s)	HAL_OFD_SetRegWriteMode (HAL_OFD_ENABLE);

4.1.9. PMD

All declarations and definitions are provided in hal_pmd.h. The implementation is family specific and can be found in the corresponding family folder (M4KN) under hal_pmd.c.

4.1.9.1. PMD Initialization

Synopsis	Configure and the selected programmable motor driver channel, including clock gating, parameters, IOs and interrupt handling.
Prototype	void HAL_PMD_HWInit (uint8_t channel_number)
Parameters	channel_number – the PMD channel to be initialized
Return Value	None
Limitations	None
Usage example(s)	HAL_PMD_HWInit(0);

4.1.9.2. Set Normal Output

Synopsis	Select normal output for motor control.
Prototype	void HAL_PMD_NormalOutput(unsigned char channel_number)
Parameters	channel_number – the PMD channel
Return Value	None
Limitations	None
Usage example(s)	HAL_PMD_NormalOutput(0);

4.1.9.3. Set Pattern Output

Synopsis	Select pattern output for motor control – open lower, upper or both sides.
Prototype	void HAL_PMD_OutputPattern(uint8_t channel_number, PMD_PATTERN pattern);
Parameters	channel_number – the PMD channel
	pattern – the PMD pattern output to be applied
Return Value	None
Limitations	None
Usage example(s)	HAL_PMD_OutputPattern(PMD_SHORTCUT_LOW_SIDE);

4.1.9.4. Disable Output

Synopsis	Disable the PMD output.
Prototype	void HAL_PMD_OutputOff(uint8_t channel_number);
Parameters	channel_number – the PMD channel
Return Value	None
Limitations	None
Usage example(s)	void HAL_PMD_OutputOff (0);

4.1.9.5. Disable Emergency Handling

Synopsis	Disable the monitoring and handling of the emergency signal.
Prototype	void HAL_PMD_EmergencyDisable(uint8_t channel_number)
Parameters	channel_number – the PMD channel
Return Value	None
Limitations	None
Usage example(s)	HAL_PMD_EmergencyDisable(0);

4.1.9.6. Handle Board Parameter Change

Synopsis	Update the board parameters affecting the PMD output at the next possible switching point.
Prototype	void HAL_PMD_HandleParameterChangeBoard(uint8_t channel_number);
Parameters	channel_number – the PMD channel
Return Value	None
Limitations	None
Usage example(s)	HAL_PMD_HandleParameterChangeBoard(0);

4.1.9.7. Handle System Parameter Change

Synopsis	Update the system parameters affecting the PMD output at the next possible switching point.
Prototype	void HAL_PMD_HandleParameterChangeSystem (uint8_t channel_number);
Parameters	channel_number – the PMD channel
Return Value	None
Limitations	None
Usage example(s)	HAL_PMD_HandleParameterChangeSystem(0);

4.1.9.8. Emergency Detection Reset

Synopsis	Reset the emergency detection and handling.
Prototype	void HAL_PMD_EmergencyReset (uint8_t channel_number);
Parameters	channel_number – the PMD channel
Return Value	None
Limitations	None
Usage example(s)	HAL_PMD_EmergencyReset(0);

4.1.9.9. Overvoltage Detection Reset

Synopsis	Reset the overvoltage detection and handling.
Prototype	void HAL_PMD_OvervoltageReset(uint8_t channel_number);
Parameters	channel_number – the PMD channel
Return Value	None
Limitations	None
Usage example(s)	HAL_PMD_OvervoltageReset(0);

4.1.10. SPI

All declarations and definitions are provided in hal_spi.h. The implementation is family specific and can be found in the corresponding family folder (M4KN) under hal_spi.c.

4.1.10.1. SPI Initialization

Synopsis	Configure the selected SPI channel, including clock gating, mode of operation and interrupt handling
Prototype	HAL_SPI_Err HAL_SPI_Init(const HAL_SPI_CfgData* initStruct)
Parameters	initStruct – configuration descriptor
Return Value	HAL_SPI_NO_ERR – success HAL_SPI_INIT_ERR – error upon initialization HAL_SPI_INVALID_PARAMETER – invalid parameter specified
Limitations	None
Usage example(s)	if (HAL_SPI_NO_ERR != HAL_SPI_Init(&drv8323SpiCfgData));

4.1.10.2. Reset SPI Channel

Synopsis	Perform software reset on the selected SPI channel.
Prototype	void HAL_SPI_SWReset(uint8_t channel)
Parameters	channel – the SPI channel to be reset
Return Value	None
Limitations	None
Usage example(s)	HAL_SPI_SWReset(HAL_SPI_CHANNEL_0);

4.1.10.3. Enable SPI Channel

Synopsis	Enable the selected SPI channel.
Prototype	void HAL_SPI_Enable(uint8_t channel)
Parameters	channel – the SPI channel to be enabled
Return Value	None
Limitations	None
Usage example(s)	HAL_SPI_Enable(HAL_SPI_CHANNEL_0);

4.1.10.4. Disable SPI Channel

Synopsis	Disable the selected SPI channel.
Prototype	void HAL_SPI_Disable(uint8_t channel)
Parameters	channel – the SPI channel to be disabled
Return Value	None
Limitations	None
Usage example(s)	HAL_SPI_Disable(HAL_SPI_CHANNEL_0);

4.1.10.5. Enable SPI Interrupt

Synopsis	Enable the interrupt handling for selected SPI channel in the desired direction.
Prototype	void HAL_SPI_EnableIRQ(uint8_t channel, HAL_SPI_Direction direction)
Parameters	channel – the SPI channel to be configured
	direction – one of HAL_SPI_RX or HAL_SPI_RX
Return Value	None
Limitations	None
Usage example(s)	HAL_SPI_EnableIRQ(HAL_SPI_CHANNEL_0, HAL_SPI_RX);

4.1.10.6. Disable SPI Interrupt

Synopsis	Disable the interrupt handling for selected SPI channel in the specified direction.
Prototype	void HAL_SPI_DisableIRQ(uint8_t channel, HAL_SPI_Direction direction)
Parameters	channel – the SPI channel to be configured
	direction – one of HAL_SPI_RX or HAL_SPI_RX
Return Value	None
Limitations	None
Usage example(s)	HAL_SPI_DisableIRQ(HAL_SPI_CHANNEL_0, HAL_SPI_RX);

4.1.10.7. Set SPI Interrupt Priority

Synopsis	Set the interrupt priority for the SPI channel in the desired direction.
Prototype	void HAL_SPI_SetIRQPriority(uint8_t channel, HAL_SPI_Direction direction, uint32_t prio)
Parameters	channel – the SPI channel to be configured
	direction – one of HAL_SPI_RX or HAL_SPI_RX
	prio – desired interrupt priority
Return Value	None
Limitations	None
Usage example(s)	HAL_SPI_EnableIRQ(HAL_SPI_CHANNEL_0, HAL_SPI_RX);

4.1.10.8. Read Single Data

Synopsis	Fetch one single data received via the SPI channel.
Prototype	uint32_t HAL_SPI_GetRxData(uint8_t channel)
Parameters	channel – the SPI channel to be read from
Return Value	n – the data being read
Limitations	None
Usage example(s)	u32Data = HAL_SPI_GetRxData(HAL_SPI_CHANNEL_0);

4.1.10.9. Write Single Data

Synopsis	Transmit one single data via the SPI channel.
Prototype	void HAL_SPI_SetTxData(uint8_t channel, uint32_t Data)
Parameters	channel – the SPI channel to be written to
	Data – the data to be written in the data register and sent out via the SPI channel
Return Value	None
Limitations	None
Usage example(s)	HAL_SPI_SetTxData(HAL_SPI_CHANNEL_0, u32Data);

4.1.10.10. Get Error State

Synopsis	Retrieve the current error status of the selected SPI channel.
Prototype	HAL_SPI_Err HAL_SPI_GetErrState(uint8_t channel)
Parameters	channel – the SPI channel to be checked
Return Value	HAL_SPI_NO_ERR – no error HAL_SPI_UNDERRUN – the error type HAL_SPI_OVERRUN – the error type HAL_SPI_PARITY_ERR – the error type
Limitations	One error only can be reported at a time in the following low to high priority – trigger, overrun, underrun, parity.
Usage example(s)	u32Err = HAL_SPI_GetErrState(HAL_SPI_CHANNEL_0);

4.1.10.11. Read Multiple Data

Synopsis	Read multiple data received via the SPI channel
Prototype	HAL_SPI_Err HAL_SPI_MasterReadBytes (uint8_t channel, const HAL_SPI_TransRec_t *read)
Parameters	channel – the SPI channel to be read from
	read – the data buffer with length field to store the data in
Return Value	HAL_SPI_NO_ERR – success HAL_SPI_RECEIVE_ERR– failure
Limitations	None
Usage example(s)	u32Err = HAL_SPI_MasterReadBytes(HAL_SPI_CHANNEL_0, &RXBuf);

4.1.10.12. Write Multiple Data

Synopsis	Write multiple data via the SPI channel.
Prototype	HAL_SPI_Err HAL_SPI_MasterWriteBytes(uint8_t channel, HAL_SPI_TransRec_t *write)
Parameters	channel – the SPI channel to be written to
	write – the data buffer with length field to take the data from
Return Value	HAL_SPI_NO_ERR – success HAL_SPI_TRANSMIT_ERR – failure
Limitations	None
Usage example(s)	u32Err = HAL_SPI_MasterWriteBytes(HAL_SPI_CHANNEL_0, &TiXBuf);

4.1.11. UART

All declarations and definitions are provided in hal_uart.h. The implementation is family specific and can be found in the corresponding family folder (M4KN) under hal_uart.c.

4.1.11.1. UART Initialization

Synopsis	Configure the selected UART channel, including clock gating, baud rate, frame and interrupt handling.
Prototype	void HAL_UART_Init(uint8_t channel, uart_direction direction , speed_mode baud, uint8_t bits, parity_mode parity, uint8_t stop_bits, flow_control flow)
Parameters	channel – the UART channel to be configured
	direction – transmitter or receiver
	baud – transfer rate
	bits – number of bits in a frame
	parity – even, odd or none
	stop_bits – number of stop bits
	flow – used control flow
Return Value	None
Limitations	None
Usage example(s)	HAL_UART_Init(UART_CHANNEL_1, HAL_UART_RXTX, SPEED_115200, 8, NO_PARITY, 1, NO_FLOW_CTRL);

4.1.11.2. Get UART Buffer State

Synopsis	Check the fill state of the UART TX or RX buffer.
Prototype	WorkState HAL_UART_GetBufState(uint8_t channel, uart_direction direction)
Parameters	channel – the UART channel to be configured
	direction – transmitter or receiver
Return Value	None
Limitations	None
Usage example(s)	WorkState State; State = HAL_UART_GetBufState(SERIAL_COMMUNICATION_UART_CHANNEL, HAL_UART_TX);

4.1.11.3. Send Single Data

Synopsis	Transmit one single data via the UART channel.
Prototype	void HAL_UART_SendData(uint8_t channel, uint32_t data)
Parameters	channel – the UART channel to be configured
	data – the data to transmitted
Return Value	None
Limitations	None
Usage example(s)	HAL_UART_SendData(SERIAL_COMMUNICATION_UART_CHANNEL, eot);

4.1.11.4. Receive Single Data

Synopsis	Fetch one single data received via the UART channel.
Prototype	uint32_t HAL_UART_GetReceivedData(uint8_t channel)
Parameters	channel – the UART channel to be configured
Return Value	n – the received data
Limitations	None
Usage example(s)	uint8_t HAL_UART_GetReceivedData (SERIAL_COMMUNICATION_UART_CHANNEL);

4.1.11.5. Enable UART Interrupt

Synopsis	Enable the interrupt handling for selected UART channel in the desired direction.
Prototype	void HAL_UART_EnableIRQ(uint8_t channel, uart_direction direction)
Parameters	channel – the UART channel to be configured
	direction – one of HAL_UART_TX, HAL_UART_RX or HAL_UART_RXTX
Return Value	None
Limitations	None
Usage example(s)	HAL_UART_EnableIRQ(SERIAL_COMMUNICATION_UART_CHANNEL, HAL_UART_RX);

4.1.11.6. Disable UART Interrupt

Synopsis	Disable the interrupt handling for selected UART channel in the desired direction.
Prototype	void HAL_UART_DisableIRQ(uint8_t channel, uart_direction direction)
Parameters	channel – the UART channel to be configured
	direction – one of HAL_UART_TX, HAL_UART_RX or HAL_UART_RXTX
Return Value	None
Limitations	None
Usage example(s)	HAL_UART_DisableIRQ(SERIAL_COMMUNICATION_UART_CHANNEL, HAL_UART_RX);

4.1.11.7. Set UART Interrupt Priority

Synopsis	Set the interrupt priority for the UART channel in the desired direction.
Prototype	void HAL_UART_SetIRQPriority(uint8_t channel, uart_direction direction, uint32_t prio)
Parameters	channel – the UART channel to be configured
	direction – one of HAL_UART_TX, HAL_UART_RX or HAL_UART_RXTX
	prio – desired interrupt priority
Return Value	None
Limitations	None
Usage example(s)	HAL_UART_SetIRQPriority(SERIAL_COMMUNICATION_UART_CHANNEL, HAL_UART_RX, 1);

4.1.11.8. Clear Pending UART Interrupt

Synopsis	Clear a pending interrupt of the UART channel in the set direction.
Prototype	void HAL_UART_ClearPendingIRQ(uint8_t channel, uart_direction direction)
Parameters	channel – the UART channel to be configured
	direction – one of HAL_UART_TX, HAL_UART_RX or HAL_UART_RXTX
Return Value	None
Limitations	None
Usage example(s)	HAL_UART_ClearPendingIRQ (SERIAL_COMMUNICATION_UART_CHANNEL, HAL_UART_RX);

4.1.11.9. Get UART Error State

Synopsis	Retrieve the current error status of the selected UART channel.
Prototype	uint32_t HAL_UART_GetError(uint8_t channel)
Parameters	channel – the UART channel to be checked
Return Value	n – the logical or of all error flags
Limitations	None
Usage example(s)	error = HAL_UART_GetError(SERIAL_COMMUNICATION_UART_CHANNEL);

4.1.11.10. Clear UART Error

Synopsis	Clear the selected errors for the UART channel specified.
Prototype	void HAL_UART_ClearError(uint8_t channel, uint32_t error)
Parameters	channel – the UART channel to be worked on
	error – the errors to be cleared
Return Value	None
Limitations	None
Usage example(s)	HAL_UART_ClearError (SERIAL_COMMUNICATION_UART_CHANNEL, OVRERR);

4.1.11.11. UART Transmission/Reception Control

Synopsis	Enable or disable the transmission and reception control for the selected UART channel.
Prototype	void HAL_UART_SetRxTx(uint8_t channel, uint8_t rxEnable, uint8_t txEnable)
Parameters	channel – the UART channel to be worked on
	rxEnable – desired state of the reception control
	txEnable – desired state of the transmission control
Return Value	None
Limitations	None
Usage example(s)	HAL_UART_SetRxTx(SERIAL_COMMUNICATION_UART_CHANNEL, 0, 1);

4.1.11.12. UART Reception Disable

Synopsis	Disable the reception control for the selected UART channel.
Prototype	void HAL_UART_DisableRX(uint8_t channel)
Parameters	channel – the UART channel to be worked on
Return Value	None
Limitations	None
Usage example(s)	HAL_UART_DisableRX(SERIAL_COMMUNICATION_UART_CHANNEL);

4.1.11.13. Clear UART Transmission FIFO

Synopsis	Clear the UART FIFO.
Prototype	void HAL_UART_ClearRXFIFO(uint8_t channel)
Parameters	channel – the UART channel to be cleared
Return Value	None
Limitations	None
Usage example(s)	HAL_UART_ClearRXFIFO (SERIAL_COMMUNICATION_UART_CHANNEL);

4.1.11.14. Retrieve FIFO Level

Synopsis	Retrieve the current fill level of the FIFO.
Prototype	uint32_t HAL_UART_GetRXFIFOLevel(uint8_t channel)
Parameters	channel – the UART channel to be checked
Return Value	n – the number of valid entries
Limitations	None
Usage example(s)	While(0 != HAL_UART_GetRXFIFOLevel (SERIAL_COMMUNICATION_UART_CHANNEL));

4.1.12. Advanced Vector Engine

All declarations and definitions are provided in hal_ve.h. The implementation is family specific and can be found in the corresponding family folder (M4KN) under hal_ve.c.

4.1.12.1. Vector Engine Initialization

Synopsis	Configure and start the vector engine, including clock gating, mode, trigger, measurement type and interrupt handling.
Prototype	void HAL_VE_Init(unsigned char channel_number)
Parameters	channel_number – the Vector Engine channel to be initialized
Return Value	None
Limitations	None
Usage example(s)	HAL_VE_Init(0);

4.1.12.2. PI Initialization

Synopsis	Initialize the PI-Control values
Prototype	void HAL_VE_InitPISettings(unsigned char channel_number
Parameters	channel_number – the vector engine channel to use the PI values
Return Value	None
Limitations	The PI settings are global and common for all channels. These need to be applied separately to every single hardware channel though. Software controlled channels do not need such initialization.
Usage example(s)	HAL_VE_InitPISettings(0);

4.1.12.3. Vector Engine Interrupt Configuration

Synopsis	Enable/disable the vector engine interrupt
Prototype	void HAL_VE_IRQ(unsigned char channel_number,VE_IRQ_State state)
Parameters	channel_number – the vector engine channel
	state – enable or disable
Return Value	None
Limitations	None
Usage example(s)	HAL_VE_IRQ(channel_number,VE_IRQ_OFF);

4.1.12.4. Vector Engine Output Control Configuration

Synopsis	Enable the Vector Engine energization output control to the PMD.
Prototype	void HAL_VE_PMD_Output(uint8_t channel_number, VE_PMD_State state)
Parameters	channel_number – the Vector Engine channel to be configured
	state – the output state being one of PMD_OFF or PMD_ON to disable or enabled the output respectively
Return Value	None
Limitations	None
Usage example(s)	HAL_VE_PMD_Output(0, PMD_OFF);

4.1.12.5. Vector Engine Emergency Protection Releases

Synopsis	Control the return from the Emergency protection enforced by the PMD.
Prototype	void HAL_VE_PMD_EMGReturn(uint8_t channel_number)
Parameters	channel_number – the Vector Engine channel to be configured
Return Value	None
Limitations	None
Usage example(s)	HAL_VE_PMD_EMGReturn(0);

4.1.13. Watchdog

All declarations and definitions are provided in hal_wdt.h. The implementation is family specific and can be found in the corresponding family folder (M4KN) under hal_wdt.c.

4.1.13.1. Watchdog Initialization

Synopsis	Initialize the Watchdog timer setting the maximum supervision time and desired reaction upon detection.
Prototype	void HAL_WDT_Init(const halWdtInitTypeDef * initStruct)
Parameters	initStruct – the configuration descriptor, containing the following fields:
	DetectTime – the detection time
	OverflowOutput –"Generate NMI interrupt" or "System Reset"
Return Value	None
Limitations	None
Usage example(s)	halWdtInitTypeDef WdtConf1 = {HAL_WDT_DETECT_TIME_EXP_21, HAL_WDT_WDOUT }; HAL_WDT_Init(WdtConf1);

4.1.13.2. Enable Watchdog Detection

Synopsis	Enable the watchdog.
Prototype	void HAL_WDT_Enable (void)
Parameters	None
Return Value	None
Limitations	None
Usage example(s)	HAL_WDT_Enable ();

4.1.13.3. Disable Watchdog Detection

Synopsis	Disable the watchdog.
Prototype	void HAL_WDT_Disable (void)
Parameters	None
Return Value	None
Limitations	None
Usage example(s)	HAL_WDT_Disable ();

4.1.13.4. Clear Watchdog Timer

Synopsis	Reset the watchdog counter value.
Prototype	void HAL_WDT_Clear (void)
Parameters	None
Return Value	None
Limitations	None
Usage example(s)	HAL_WDT_Clear ();

5. References

[1] TMPM4K Group (2) Datasheet, Revision 1.0, October 2018, Toshiba Electronic Devices & Storage Corporation
[2] Reference Manual Advanced Vector Engine Plus (A-VE+-B), Revision 3.0, May 2018, Toshiba Electronic Devices & Storage Corporation
[3] Reference Manual Advanced Programmable Motor Control Circuit (A-PMD-A), Revision 2.1, July 2020, Toshiba Electronic Devices & Storage Corporation
[4] Reference Manual Advanced Encoder Input Circuit(32-bit) (A-ENC32-A), Revision 1.1, October 2018, Toshiba Electronic Devices & Storage Corporation
[5] Reference Manual 12-bit Analog to Digital Converter (ADC-I), Revision 0.1, July 2020, Toshiba Electronic Devices & Storage Corporation
[6] Reference Manual Clock Control and Operation Mode (CG-M4K(2)-E), Revision 1.0, May 2018, Toshiba Electronic Devices & Storage Corporation

6. Revision History

Table 6.1 Revision History

Revision	Date	Changes
1.0.0	2022/04/29	Baselined Version
1.1.0	2022/10/21	Introduction section updated.

Trademarks

- FreeRTOS™ is a trademark of Amazon Web Services, inc in the US and/or elsewhere. All rights reserved.
- Microsoft® and Windows® are either registered trademarks Microsoft Corporation in the United States and/or elsewhere. All rights reserved.
- Arm® , Cortex® ,Cortex®-M3, Cortex®-M4,Keil® and µVision® are registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere.
- Click boards™ is a trademark of MIKROELEKTRONIKA. All rights reserved.
- FTDI may be registered trademarks of “Future Technology Devices International Limited”. All rights reserved.
- IAR Systems® and IAR Embedded Workbench® are registered trademarks are owned by IAR Systems. All rights reserved
- SEGGER and J-Link are trademarks or registered trademarks of SEGGER Microcontroller GmbH & Co. KG. All rights reserved.

Other Company names, product names and service names mentioned herein may be trademarks of their respective companies.

RESTRICTIONS ON PRODUCT USE

Toshiba Corporation and its subsidiaries and affiliates are collectively referred to as “TOSHIBA”.
Hardware, software and systems described in this document are collectively referred to as “Product”.

- TOSHIBA reserves the right to make changes to the information in this document and related Product without notice.
- This document and any information herein may not be reproduced without prior written permission from TOSHIBA. Even with TOSHIBA's written permission, reproduction is permissible only if reproduction is without alteration/omission.
- Though TOSHIBA works continually to improve Product's quality and reliability, Product can malfunction or fail. Customers are responsible for complying with safety standards and for providing adequate designs and safeguards for their hardware, software and systems which minimize risk and avoid situations in which a malfunction or failure of Product could cause loss of human life, bodily injury or damage to property, including data loss or corruption. Before customers use the Product, create designs including the Product, or incorporate the Product into their own applications, customers must also refer to and comply with (a) the latest versions of all relevant TOSHIBA information, including without limitation, this document, the specifications, the data sheets and application notes for Product and the precautions and conditions set forth in the "TOSHIBA Semiconductor Reliability Handbook" and (b) the instructions for the application with which the Product will be used with or for. Customers are solely responsible for all aspects of their own product design or applications, including but not limited to (a) determining the appropriateness of the use of this Product in such design or applications; (b) evaluating and determining the applicability of any information contained in this document, or in charts, diagrams, programs, algorithms, sample application circuits, or any other referenced documents; and (c) validating all operating parameters for such designs and applications. **TOSHIBA ASSUMES NO LIABILITY FOR CUSTOMERS' PRODUCT DESIGN OR APPLICATIONS.**
- **PRODUCT IS NEITHER INTENDED NOR WARRANTED FOR USE IN EQUIPMENTS OR SYSTEMS THAT REQUIRE EXTRAORDINARILY HIGH LEVELS OF QUALITY AND/OR RELIABILITY, AND/OR A MALFUNCTION OR FAILURE OF WHICH MAY CAUSE LOSS OF HUMAN LIFE, BODILY INJURY, SERIOUS PROPERTY DAMAGE AND/OR SERIOUS PUBLIC IMPACT ("UNINTENDED USE").** Except for specific applications as expressly stated in this document, Unintended Use includes, without limitation, equipment used in nuclear facilities, equipment used in the aerospace industry, lifesaving and/or life supporting medical equipment, equipment used for automobiles, trains, ships and other transportation, traffic signaling equipment, equipment used to control combustions or explosions, safety devices, elevators and escalators, and devices related to power plant. **IF YOU USE PRODUCT FOR UNINTENDED USE, TOSHIBA ASSUMES NO LIABILITY FOR PRODUCT.** For details, please contact your TOSHIBA sales representative or contact us via our website.
- Do not disassemble, analyze, reverse-engineer, alter, modify, translate or copy Product, whether in whole or in part.
- Product shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable laws or regulations.
- The information contained herein is presented only as guidance for Product use. No responsibility is assumed by TOSHIBA for any infringement of patents or any other intellectual property rights of third parties that may result from the use of Product. No license to any intellectual property right is granted by this document, whether express or implied, by estoppel or otherwise.
- **ABSENT A WRITTEN SIGNED AGREEMENT, EXCEPT AS PROVIDED IN THE RELEVANT TERMS AND CONDITIONS OF SALE FOR PRODUCT, AND TO THE MAXIMUM EXTENT ALLOWABLE BY LAW, TOSHIBA (1) ASSUMES NO LIABILITY WHATSOEVER, INCLUDING WITHOUT LIMITATION, INDIRECT, CONSEQUENTIAL, SPECIAL, OR INCIDENTAL DAMAGES OR LOSS, INCLUDING WITHOUT LIMITATION, LOSS OF PROFITS, LOSS OF OPPORTUNITIES, BUSINESS INTERRUPTION AND LOSS OF DATA, AND (2) DISCLAIMS ANY AND ALL EXPRESS OR IMPLIED WARRANTIES AND CONDITIONS RELATED TO SALE, USE OF PRODUCT, OR INFORMATION, INCLUDING WARRANTIES OR CONDITIONS OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, ACCURACY OF INFORMATION, OR NONINFRINGEMENT.**
- Do not use or otherwise make available Product or related software or technology for any military purposes, including without limitation, for the design, development, use, stockpiling or manufacturing of nuclear, chemical, or biological weapons or missile technology products (mass destruction weapons). Product and related software and technology may be controlled under the applicable export laws and regulations including, without limitation, the Japanese Foreign Exchange and Foreign Trade Law and the U.S. Export Administration Regulations. Export and re-export of Product or related software or technology are strictly prohibited except in compliance with all applicable export laws and regulations.
- Please contact your TOSHIBA sales representative for details as to environmental matters such as the RoHS compatibility of Product. Please use Product in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. **TOSHIBA ASSUMES NO LIABILITY FOR DAMAGES OR LOSSES OCCURRING AS A RESULT OF NONCOMPLIANCE WITH APPLICABLE LAWS AND REGULATIONS.**