TMC6460

Fully Integrated FOC BLDC Driver

Product Highlights

- Highly Miniaturized Servo Drive SoC
  - Drives 36V BLDC and DC Motors
  - Up to 3A$_{RMS}$ Continuous Current Capability
  - Up to 5A$_{FS}$ Current Sensing Capability
  - -40°C to 125°C Ambient Temperature
  - Easy to Use
- High Performance Motion Control
  - Hardware-Based Field-Oriented Control (FOC)
  - Up to 200kHz Control Loop and PWM Engine
  - Embedded 8-Point Ramp Controller
  - Flexible Feedback Engine (ABN, SinCos, Digital/Analog Hall, SPI Encoders)
- Integrated Power Stage and Signal Conditioning
  - Three 36V Half Bridges 55mΩ/FET
  - Integrated Scalable, Lossless Current Sensing
  - Accurate CSAs and ADCs
  - Short Circuit and Thermal Protection

See more in *Functional Description*.

Key Applications

- Robotics and Automation
- CCTV and PTZ Cameras
- SMT Pick and Place Machines
- Photography Gimbals
- Infusion and Syringe Pumps
- BLDC/PMSM and DC Servo Motors

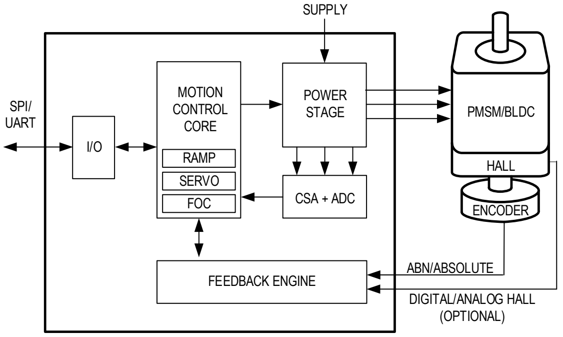
The TMC6460 is the first fully integrated Servo Drive IC including the FOC controller, power stage, current sensing and feedback engine, all in one chip. Its highly integrated design enables the control of torque, velocity, and position motion with unrivalled miniaturization.

Inside the TMC6460, the FOC Algorithm and Motion Control are implemented in the hardware, resulting in outstanding dynamic control of the motor compared to software-implemented solutions.

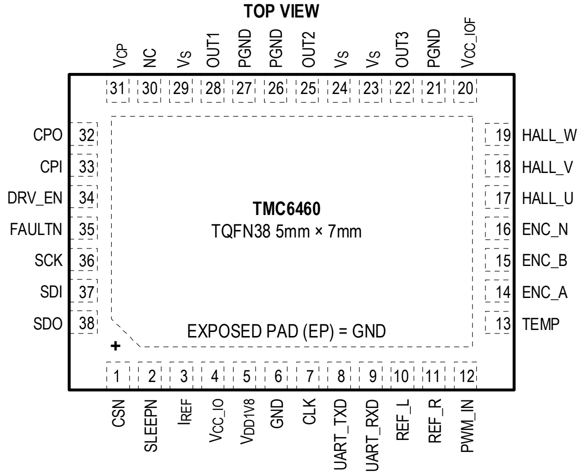
The high PWM frequency capabilities of the TMC6460 enable efficient and precise control of low-inductance and coreless motors.

The TMC6460 is a flexible and easy-to-use solution, making it the ideal choice in applications that require precise positioning and fast acceleration.

Simplified Application Diagram



Pin Configuration



Ordering Information ***is at the end of the data sheet.***

TABLE OF CONTENTS

Absolute Maximum Ratings

V_S to GND	-0.3V to 41V
V_{DD1V8} to GND	-0.3V to MIN(2.2, V_{CC_IO} + 0.3)V
V_{CC_IO} to GND	-0.3V to 6V
V_{CC_IOF} to GND	-0.3V to 6V
OUT1, OUT2, OUT3 to GND	-0.6V to V_S + 0.3V
V_{CP} to GND	V_S - 0.3V to Min. (44, V_S + 6)V
CPO to GND	V_S - 0.3V to Min. (44, V_S+6)V
CPI to GND	-0.3V to Min. (41, V_S + 0.3)V
I_{REF} to GND	-0.3V to Min. (2.2, V_{DD1V8} + 0.3)V
IO Pins to GND (On V_{CC_IO} ref.)	-0.3V to V_{CC_IO} + 0.3V
IO Pins to GND (On V_{CC_IOF} ref.)	-0.3V to V_{CC_IOF} + 0.3V
Operating Junction Temperature Range	-40°C to 125°C
Junction Temperature	160°C
Storage Temperature Range	-65°C to 150°C
Soldering Temperature (Reflow)	260°C

Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those indicated in the operational sections of the specifications is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

Package Information

38-Pin TQFN

Package Code	T3857+1C
Outline Number	21-0172
Land Pattern Number	90-0076
Thermal Resistance, Single Layer Board:	
Junction-to-Ambient (θ_{JA})	38°C/W
Junction-to-Case Thermal Resistance (θ_{JC})	1°C/W
Thermal Resistance, Four Layer Board:	
Junction-to-Ambient (θ_{JA})	28°C/W
Junction-to-Case Thermal Resistance (θ_{JC})	1°C/W

Electrical Characteristics

(V_S = 2.4V to 36V, R_{REF} = 60.4kΩ 0.1%, typical values assume T_A = 25°C and V_S = 24V, limits over the operating temperature range and relevant supply voltage range are guaranteed by design and characterization, specifications marked "GBD" are guaranteed by design and not production tested.)

PARAMETER	SYMBOL	CONDITIONS		MIN	TYP	MAX	UNITS
POWER SUPPLY							
Supply Voltage on V_S	V_S			2.4		36	V
Sleep Mode Current Consumption on V_S	I_{VS_SLEEP}	SLEEPN = 0V	Output leakage not included.		0	2	µA
Standby Mode Current Consumption on V_S	I_{VS_STOP}	SLEEPN = V_{CC_IO} LP_MODE = 1_b	Output leakage not included.		0	2	µA
Quiescent Mode Current Consumption on V_S	I_{VS_DRVOFF}	DRV_EN = 0V SLEEPN = V_{CC_IO} LP_MODE = 0_b			730	1100	µA
Supply Voltage on V_{CC_IO}	V_{CC_IO}			2.2		5.5	V
Sleep Mode Current Consumption on V_{CC_IO}	I_{VCCIO_SLEEP}	SLEEPN = 0V			0	2	µA
Standby Mode Current Consumption on V_{CC_IO}	I_{VCCIO_STOP}	SLEEPN = V_{CC_IO} LP_MODE = 1_b			3.1	5	mA
Quiescent Mode Current Consumption on V_{CC_IO}	I_{VCCIO_DRVOFF}	DRV_EN = 0V SLEEPN = V_{CC_IO} LP_MODE = 0_b			16	25	mA
Supply Voltage on V_{CC_IOF}	V_{CC_IOF}			2.2		5.5	V
Sleep Mode Current Consumption on V_{CC_IOF}	I_{VCCIOF_SLEEP}	SLEEPN = 0V			0	1	µA
Standby Mode Current Consumption on V_{CC_IOF}	I_{VCCIOF_STOP}	SLEEPN = V_{CC_IO} LP_MODE = 1_b			8	13	µA
Quiescent Mode Current Consumption on V_{CC_IOF}	I_{VCCIOF_DRVOFF}	DRV_EN = 0V SLEEPN = V_{CC_IO} LP_MODE = 0_b			70	100	µA

(V_S = 2.4V to 36V, R_{REF} = 60.4kΩ 0.1%, typical values assume T_A = 25°C and V_S = 24V, limits over the operating temperature range and relevant supply voltage range are guaranteed by design and characterization, specifications marked "GBD" are guaranteed by design and not production tested.)

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
1.8V Regulator Output Voltage	V_{DD1V8}	V_{CC_IO} = 2.2V I_{LOAD} = 40mA		1.8		V
V_{DD1V8} Short Circuit Current	I_{VDD1V8_SHT}	V_{DD1V8} = 0V			20	mA
V_{DD1V8} Current Limit	I_{VDD1V8_LIM}	Normal operation	40			mA
Charge Pump Voltage	V_{CP}	V_S > 3.3V I_{LOAD} = 1mA		V_S + 2.7		V
		V_S = 2.4V I_{LOAD} = 1mA	V_S + 1.8			
LOGIC INPUTS AND OUTPUTS						
V_{CC_IO} Input Voltage Level High	V_{VCCIO_IH}		0.7 × V_{CC_IO}			V
V_{CC_IO} Input Voltage Level Low	V_{VCCIO_IL}				0.3 × V_{CC_IO}	V
V_{CC_IO} Input Hysteresis	V_{VCCIO_HYS}			0.15 × V_{CC_IO}		V
V_{CC_IOF} Input Voltage Level High	V_{VCCIOF_IH}		0.7 × V_{CC_IOF}			V
V_{CC_IOF} Input Voltage Level Low	V_{VCCIOF_IL}				0.3 × V_{CC_IOF}	V
V_{CC_IOF} Input Hysteresis	V_{VCCIOF_HYS}			0.15 × V_{CC_IOF}		V
Pullup Resistance I/O Pins	R_{PU}	To V_{CC_IO}/V_{CC_IOF}	60	100	140	kΩ
Pulldown Resistance I/O Pins	R_{PD}	To GND	60	100	140	kΩ
Pulldown Resistance SLEEPN	R_{PD_SLEEPN}		0.6	1	1.4	MΩ
Open-Drain Output Logic Low Voltage	V_{OL}	I_{LOAD} = 5mA			0.4	V
Push-Pull Output Logic High Voltage	V_{OH}	I_{LOAD} = 5mA	V_{CC_IO} - 0.4			V
Open-Drain Output Logic High Leakage Current	I_{OH}	V_{PIN} = 5.5V	-1		+1	µA
POWER STAGE						
Max. Full-Scale Current	I_{FS}				5	A
Full-Scale Current		LS_RES_ON = 11_b			5	A
		LS_RES_ON = 10_b			3.75	
		LS_RES_ON = 01_b			2.5	
		LS_RES_ON = 00_b			1.25	

(V_S = 2.4V to 36V, R_{REF} = 60.4kΩ 0.1%, typical values assume T_A = 25°C and V_S = 24V, limits over the operating temperature range and relevant supply voltage range are guaranteed by design and characterization, specifications marked "GBD" are guaranteed by design and not production tested.)

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
Output ON-Resistance Low Side	$R_{ON,LS}$	LS_RES_ON = 11_b		55	110	mΩ
		LS_RES_ON = 10_b		78	150	
		LS_RES_ON = 01_b		112	220	
		LS_RES_ON = 00_b		200	400	
Output ON-Resistance High Side	$R_{ON,HS}$			56	110	mΩ
Output Leakage	I_{LEAK}	DRV_EN = 0V	-20	0	+20	µA
Output Slew Rate	SR	SLEW_RATE = 00_b		100		V/µs
		SLEW_RATE = 01_b		200		
		SLEW_RATE = 10_b		400		
		SLEW_RATE = 11_b		800		
Output Frequency Limit		See *Output Frequency Limit.*			599	Hz
PROTECTIONS						
Overcurrent Protection Threshold	OCP	LS_RES_ON = 11_b OCP_DETECION_MODE = 0_b	10			A
		LS_RES_ON = 10_p OCP_DETECION_MODE = 0_b	7.5			
		LS_RES_ON = 01_b OCP_DETECION_MODE = 0_b	5			
		LS_RES_ON = 00_b OCP_DETECION_MODE = 0_b	2.5			
Overcurrent Protection Threshold in VMODE	OCP_VMODE	OCP_DETECION_MODE = 1_b	1.6	2.2	2.8	V
Overcurrent Protection Blanking Time	t_{OCP}	OCP_DEGLITCH = 00_b	0.25	0.5	0.63	µs
		OCP_DEGLITCH = 01_b	0.3	0.8	1.06	
		OCP_DEGLITCH = 10_b	0.65	1.4	1.92	
		OCP_DEGLITCH = 11_b	1.3	2.6	3.65	
UVLO Threshold on VS	$UVLO_{VS}$	V_S rising	2.1	2.2	2.3	V
UVLO Threshold on VS Hysterisis	$UVLO_{VS\_HYS}$			0.1		V
UVLO Threshold on VCCIO	$UVLO_{VCCIO}$	V_{CC_IO} rising	1.6	1.9	2.1	V
UVLO Threshold on VCCIO Hysterisis	$UVLO_{VCCIO\_HYS}$			0.11		V
UVLO Threshold on VCCIOF	$UVLO_{VCCIOF}$	V_{CC_IOF} rising	1.6	1.9	2.1	V
UVLO Threshold on VCCIOF Hysteresis	$UVLO_{VCCIOF\_HYS}$			0.11		V
Thermal Protection Shutdown Threshold	TSD			165		°C

(V_S = 2.4V to 36V, R_{REF} = 60.4kΩ 0.1%, typical values assume T_A = 25°C and V_S = 24V, limits over the operating temperature range and relevant supply voltage range are guaranteed by design and characterization, specifications marked "GBD" are guaranteed by design and not production tested.)

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
Thermal Protection Threshold Hysteresis				20		°C
CURRENT SENSE AND SIGNAL CONDITIONING						
IREF Pin Resistor	R_{REF}			60.4		kΩ
IREF Output Voltage	V_{REF}		0.882	0.9	0.918	V
IREF Monitor Window	R_{REF_OK}		57		63	kΩ
Current Trip Regulation Accuracy	DITRIP1	ITRIG from 430mA to I_{FS}	-5		+5	%
	DITRIP1_INT	INT_IREF_SEL = 1_b ITRIG from 430mA to I_{FS}	-8		+8	
SYSTEM TIMINGS						
Sleep Time	t_{SLEEP}	SLEEPN falling edge to OUT1,2,3 in tristate.			50	µs
Wakeup Time from Sleep	t_{WAKE}	SLEEPN rising edge to normal operation.			4	ms
Wakeup Time from Standby Mode		LP_MODE changes to 0_b to normal operation.			3	ms
Enable Time	t_{EN}	DRV_EN_BIT changes to 1_b to driver on.			1	µs
Disable Time	t_{OFF}	DRV_EN falling edge to driver off.			650	ns
Internal Oscillator Frequency	f_{OSC}		14.25	15	15.75	MHz
PLL Frequency	f_{PLL}			120		MHz
System Clock Frequency	f_{SYS}			60		MHz
External Clock Frequency	f_{EXT_CLK}		1		32	MHz
PWM Frequency	f_{PWM}		1.831		200	kHz
SPI						
SCK Valid Time before or after Change of CSN	t_{CC}	See also *Figure 1* for all timings.	t_{SCK}			ns
CSN High Time	t_{CSH}		4 × t_{SCK}			ns
SCK Low Time	t_{CL}		20			ns
SCK High Time	t_{CH}		20			ns
SCK Frequency	$f_{SCK} = 1/t_{SCK}$	V_{CC_IO} = 3.0 V			8	MHz
SDI Setup Time before SCK Falling Edge	t_{DS}		10			ns
SDI Hold Time after SCK Falling Edge	t_{DH}		10			ns
Data Out Valid Time after SCK Rising Edge	t_{DO}	V_{CC_IO} = 2.2 V		55	75	ns
		V_{CC_IO} = 3.0 V		36	55	

(V_S = 2.4V to 36V, R_{REF} = 60.4kΩ 0.1%, typical values assume T_A = 25°C and V_S = 24V, limits over the operating temperature range and relevant supply voltage range are guaranteed by design and characterization, specifications marked "GBD" are guaranteed by design and not production tested.)

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
UART						
Baud Rate (No Autobaud, External Clock)		AUTOBAUD_EN = 0_b PLL_SRC = 1_b V_{CC_IO} = 3V	9600		12M	Bit/s
Baud Rate (No Autobaud, Internal Oscillator)		AUTOBAUD_EN = 0_b PLL_SRC = 0_b V_{CC_IO} = 3V	9600		12M	Bits/s
Baud Rate (Autobaud, External Clock)		AUTOBAUD_EN = 1_b PLL_SRC = 1_b V_{CC_IO} = 3V	9600		7.5M	Bit/s
Baud Rate (Autobaud, Internal Oscillator)		AUTOBAUD_EN = 1_b PLL_SRC = 0_b V_{CC_IO} = 3V	9600		6M	Bits/s
ENCODER INTERFACE AND ANALOG INPUTS						
A/B/N Input Minimum Low Time	$t_{ABNL,MIN}$			50		ns
A/B/N Input Minimum High Time	$t_{ABNH,MIN}$			50		ns
A/B/N Maximum Input Frequency	$f_{ABN,MAX}$	ENC_FLT = 00_b		10		MHz
A/B/N Input Filter Frequency[1]	$f_{FILTABN}$	ENC_FLT = 01_b		1.0		MHz
		ENC_FLT = 10_b		0.5		
		ENC_FLT = 11_b		0.1		
ADC Resolution				12		Bit
Analog Input Voltage Range	V_{AIN}	ANA_DIV_VCCIO = 11_b ANA_DIV_VCCIOF = 11_b	75m		1.1	V
		ANA_DIV_VCCIO = 10_b ANA_DIV_VCCIOF = 10_b	165m		2.4	
		ANA_DIV_VCCIO = 01_b ANA_DIV_VCCIOF = 01_b	225m		3.3	
		ANA_DIV_VCCIO = 00_b ANA_DIV_VCCIOF = 00_b	338m		5	
Analog Input Pulldown Resistance	R_{PD_AIN}	ANA_DIV_VCCIO = 11_b ANA_DIV_VCCIOF = 11_b	40	55	75	kΩ
		ANA_DIV_VCCIO = 10_b ANA_DIV_VCCIOF = 10_b	70	100	138	
		ANA_DIV_VCCIO = 01_b ANA_DIV_VCCIOF = 01_b	57	79	113	
		ANA_DIV_VCCIO = 00_b ANA_DIV_VCCIOF = 00_b	50	70	95	
AIN Analog Input Sampling Frequency	f_{AIN}			f_{PWM}		Hz

(V_S = 2.4V to 36V, R_{REF} = 60.4kΩ 0.1%, typical values assume T_A = 25°C and V_S = 24V, limits over the operating temperature range and relevant supply voltage range are guaranteed by design and characterization, specifications marked "GBD" are guaranteed by design and not production tested.)

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
Internal Temperature Measurement Accuracy				±10		°C
V_S Supply Voltage Measurement Accuracy			-5		+5	%

Note 1: The A/B/N input filter is disabled when ENC_FLT = 00_b.

Timing Diagrams

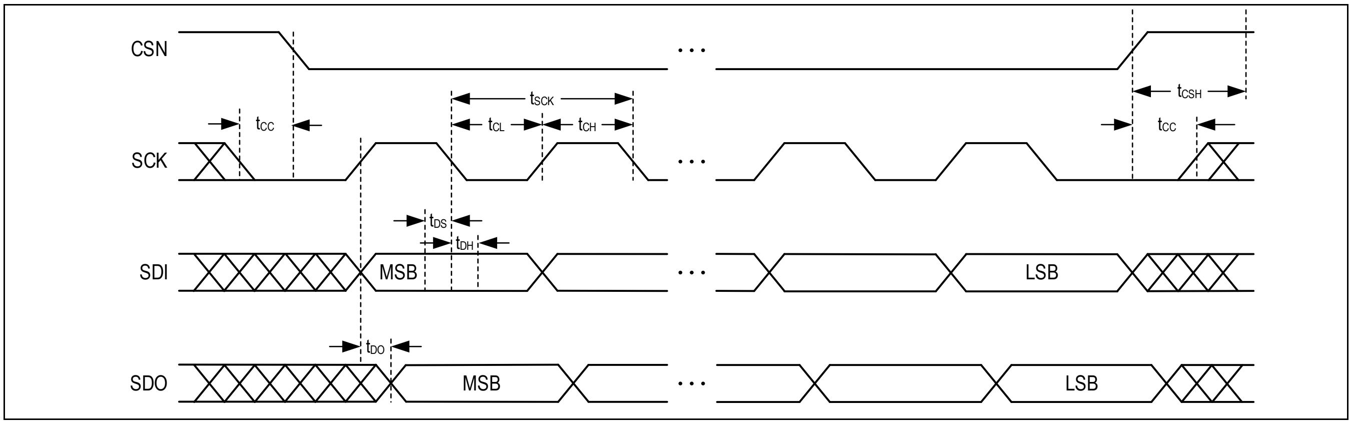
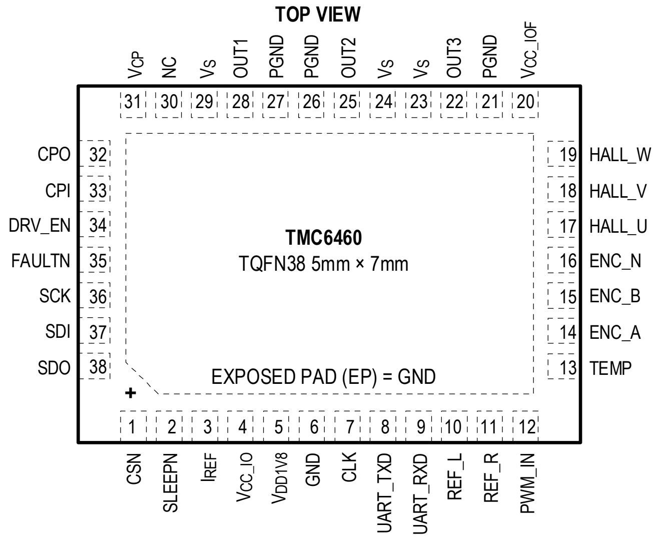


Figure 1. SPI Timing Diagram SPI MODE 1

Pin Configurations

TMC6460 TQFN Pin Configuration



Pin Descriptions

PIN	NAME	FUNCTION	REF SUPPLY	Type
1	CSN	SPI chip select input, active low.	VCC_IO	Digital Input (Pullup)
2	SLEEPN	Sleep/reset input, active low. The device enters sleep mode when the pin is asserted. Since register contents are not kept during sleep mode, this pin also acts as the reset input. This pin has an internal pulldown. Connect to VCC_IO if not used. Do not assert this pin while the motor is moving.	VCC_IO	Digital Input (Pulldown)
3	IREF	Current reference input. Connect a 60.4kΩ 0.1% resistor to GND. This pin can be directly connected to GND if the device is configured to use the internal reference resistor.		Analog Input
4	VCC_IO	Supply input for internal logic, I/O pins, and the internal 1.8V LDO. Connect a 1µF low ESR ceramic capacitor to GND close to the pin with short traces for best performance.	VCC_IO	Supply
5	VDD1V8	Internal 1.8V LDO output. Connect a 2.2µF low ESR ceramic capacitor to GND close to the pin with short traces for best perfomance.		Supply
6	GND	Common ground.		GND
7	CLK	External clock input. Provide a clock signal if the accuracy of the internal oscillator is not sufficient for the application. See more details in the *PLL and Clock Configuration* chapter.	VCC_IO	Digital Input (Pulldown)
8	UART_TXD	Default: UART_TXD – UART transmit output ALT1: BRAKE – Brake chopper PWM output ALT2: DIRECT_OUT[0] – I/O Controller direct output 0 ALT3: DIRECT_OUT[1] – I/O Controller direct output 1	VCC_IO	Digital Output
9	UART_RXD	Default: UART_RXD – UART receive input ALT1: DIR_IN – Direction control input (with the I/O Controller) ALT3: DIRECT_IN[0] – I/O Controller direct input 0	VCC_IO	Digital Input (Pullup/Pulldown)
10	REF_L	Default: REF_L – Left reference switch input ALT1: REF_H – Home reference switch input	VCC_IOF	Digital Input

		ALT3: DIRECT_IN[0] – I/O Controller direct input 0		(Pullup/Pulldown)
11	REF_R	Default: REF_R – Right reference switch input ALT1: REF_H – Home reference switch input ALT2: BRAKE – Brake chopper PWM output ALT3: DIRECT_OUT[1] – I/O Controller direct output 1	VCC_IOF	Digital Input (Pullup/Pulldown)/Digital Output
12	PWM_IN	Default: PWM_IN – PWM control input (with the I/O Controller) ALT1: AIN – Analog control input (with the I/O Controller) ALT3: DIRECT_IN[1] – I/O Controller direct input 1	VCC_IO	Digital Input (Pullup/Pulldown)/Analog Input
13	TEMP	Motor temperature analog input.	VCC_IO	Analog Input
14	ENC_A	Default: ENC_A – A input of the 1st ABN encoder (also used as the non-inverted signal for digital differential input). ALT1: AINN_U – Inverted analog Hall U or SinCos X signal for use with differential input. ALT2: HALL_U – U input of the digital hall sensor ALT3: DIRECT_OUT[0] – I/O Controller direct output 0	VCC_IOF	Digital Input (Pullup/Pulldown)/Digital Output/Analog Input
15	ENC_B	Default: ENC_B –B input of the 1st ABN encoder (also used as the non-inverted signal for digital differential input). ALT1: AINN_V – Inverted analog Hall V or SinCos N signal for use with differential input. ALT2: HALL_V – V input of the digital hall sensor ALT3: DIRECT_IN[0] – I/O Controller direct input 0	VCC_IOF	Digital Input (Pullup/Pulldown)/Analog Input
16	ENC_N	Default: ENC_N – N input of the 1st ABN encoder (also used as the non-inverted signal for digital differential input). ALT1: AINN_W – Inverted analog Hall W or SinCos Y signal for use with differential input. ALT2: HALL_W – W input of the digital hall sensor ALT3: DIRECT_IN[1]/DIRECT_OUT[1] – I/O Controller direct input 1 and direct output 1	VCC_IOF	Digital Input (Pullup/Pulldown)/Digital Output/Analog Input
17	HALL_U	Default: HALL_U – U input of the digital hall sensor ALT1: AINP_U –Analog Hall U/SinCos X input (also used as the non-inverted signal for analog differential input). ALT2: ENC2_A – A input of the 2nd ABN encoder ALT3: I2C_SDA – I/O Controller I^{2}C data in/out ALT6: DIG_DIFF_ENC_A_MONITOR – Monitoring of the inverted A signal of the 1st ABN encoder for digital differential input. ALT7: DIG_DIFF_ENC_A_N – Inverted A signal of the 1st ABN encoder for use with digital differential input.	VCC_IOF	Digital Input (Pullup/Pulldown)/Digital Output/Analog Input
18	HALL_V	Default: HALL_V – V input of the digital hall sensor ALT1: AINP_V – Analog Hall V/SinCos N input (also used as the non-inverted signal for analog differential input). ALT2: ENC2_B – B input of the 2nd ABN encoder ALT3: I2C_SCL – I/O Controller I2C clock as push-pull output ALT4: BRAKE – Brake chopper PWM output ALT5: I2C_SCL_OD – I/O Controller I2C clock as open-drain output ALT6: DIG_DIFF_ENC_B_MONITOR – Monitoring of the inverted B signal of the 1st ABN encoder for digital differential input. ALT7: DIG_DIFF_ENC_B_N – Inverted B signal of the 1st ABN encoder for use with digital differential input.	VCC_IOF	Digital Input (Pullup/Pulldown)/Digital Output/Analog Input
19	HALL_W	Default: HALL_W – W input of the digital hall sensor ALT1: AINP_W – Analog Hall W/SinCos Y input (also used as the non-inverted signal for analog differential input).	VCC_IOF	Digital Input (Pullup/Pul

		ALT2: ENC2_N – N input of the 2nd ABN encoder ALT3: DIRECT_OUT[2] – I/O Controller direct output 2 ALT4: REF_H – Home reference switch input ALT6: DIG_DIFF_ENC_N_MONITOR – Monitoring of the inverted N signal of the 1st ABN encoder for digital differential input. ALT7: DIG_DIFF_ENC_N_N – Inverted N signal of the 1st ABN encoder for use with digital differential input.		ldown)/Digital Output/Analog Input
20	V_{CC_IOF}	Supply input for motor position feedback and reference switch I/O pins. Connect a 100nF low ESR ceramic capacitor to GND close to the pin with short traces for best performance.		Supply
21, 26, 27	PGND	Power ground.		GND
22	OUT3	Half-bridge output 3 – Connect to BLDC motor phase W or unconnected for DC motor.	VS	Analog Output
23, 24, 29	V_S	Motor supply voltage input.		Supply
25	OUT2	Half-bridge output 2 – Connect to BLDC motor phase V or DC motor.	VS	Analog Output
28	OUT1	Half-bridge output 1 – Connect to BLDC motor phase U or DC motor.	VS	Analog Output
30	NC	Internally not connected.		
31	V_{CP}	Charge pump output. Connect a 1µF capacitor between V_{CP} and V_S.		Analog Output
32	CPO	Charge pump flying cap high terminal. Connect a 47nF capacitor between CPO and CPI.		Analog Output
33	CPI	Charge pump flying cap low terminal. Connect a 47nF capacitor between CPO and CPI.		Analog Output
34	DRV_EN	Driver enable input, active high.	VCC_IO	Digital Input (Pulldown)
35	FAULTN	Default: FAULTN_OD – Configurable fault output as open-drain, active low ALT1: INT_OD – Configurable interrupt/event output as open-drain ALT2: FAULTN_PP – Configurable fault output as push-pull, active low ALT3: INT_PP – Configurable interrupt/event output as push-pull	VCC_IO	Digital Output (Pullup)
36	SCK	Default: SCK – SPI clock input ALT3: DIRECT_IN[1] – I/O Controller direct input 1	VCC_IO	Digital Input (Pullup/Pulldown)
37	SDI	Default: SDI – SPI serial data input ALT1: DIR_IN – Direction control input (with the I/O Controller) ALT3: DIRECT_IN[0] – I/O Controller direct input 0	VCC_IO	Digital Input (Pullup/Pulldown)
38	SDO	Default: SDO – SPI serial data output ALT1: BRAKE – Brake chopper PWM output ALT3: DIRECT_OUT[0] – I/O Controller direct output 0	VCC_IO	Digital Output (Pullup)
EP	GND	Exposed Die Pad. Connect the exposed die pad to a GND plane. Provide as many as possible vias for heat transfer to the GND plane. Serves as the GND pin for power stage and internal circuitry.		GND

Functional Diagram

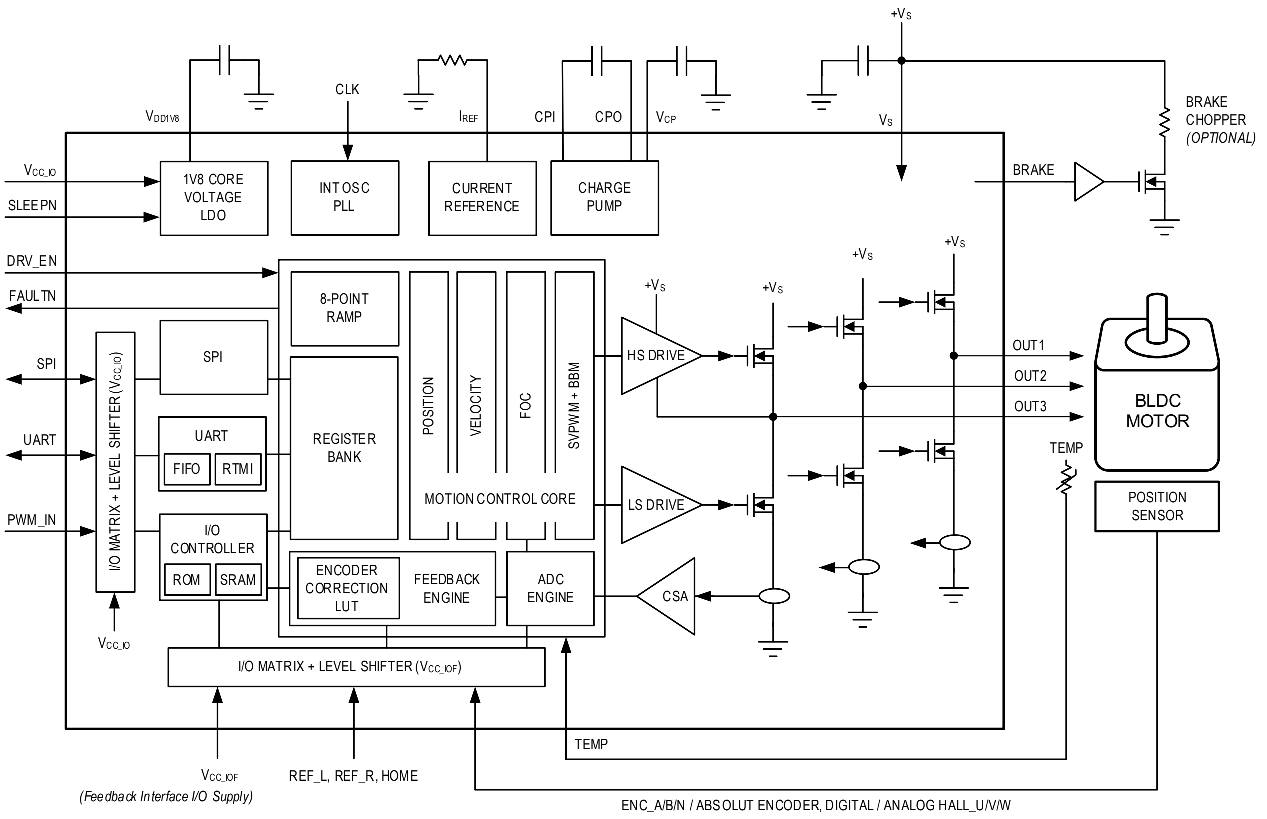


Figure 2. TMC6460 Functional Diagram

Functional Description

The TMC6460 is a 36V high-performance field-oriented control (FOC) servo drive for BLDC motors / 3-phase PMSM that can also drive DC motors. Highly integrated and easy to use, the TMC6460 simplifies the design of sensor-based FOC for motion control. The complete solution reduces the learning curve while giving best-in-class performance.

The TMC6460 includes the required peripheral interfaces for communication with a host controller, along with a flexible IO interface and a versatile motor angle feedback engine to adapt to most application needs.

The TMC6460 features a powerful motion control core (MCC) to carry out the FOC transformations. The MCC integrates the complete control loop cascade for torque, velocity and position control. All real-time critical tasks are implemented in hardware, resulting in high control loop speeds. The PWM engine of the MCC supports fast chopping frequencies for precise servo applications that use low inductance / coreless motors. An integrated eight-point ramp generator feeds the MCC to achieve highly dynamic position or velocity ramping motions with smooth transitions and minimum jerk.

The TMC6460 integrates three 36V-capable half bridges with low impedance FETs, resulting in high driving efficiency and low heat generation. With a total R_{ON} of 110mΩ (high side + low side), the TMC6460 can drive motors up to $3A_{RMS}$.

An accurate current acquisition based on low side FET measurement is integrated in the TMC6460. The internal current sense capabilities eliminate the need of bulky external power resistors, resulting in dramatic space and power savings compared with applications based on external sense resistors.

The TMC6460 provides comprehensive measurement and diagnostics capabilities, as well as robust failure detection and protection mechanisms. This allows continuous system monitoring for safe motor operation in the end application.

The TMC6460 is available as a 38-pin TQFN 5mm x 7mm package.

Communication and Control Interfaces

The TMC6460 integrates SPI and UART interfaces for communication with a host controller. Additionally, a single-wire control mode is supported.

SPI Interface

The TMC6460 integrates one SPI interface for communication with a host MCU. It uses SPI mode 1 (with CPOL = 0 and CPHA = 1), where the input (SDI) is sampled on the falling edge of the SPI clock (SCK) and the output (SDO) is generated on the rising edge of SCK. The SPI interface supports a maximum clock frequency of 8MHz.

The SPI interface offers support for daisy chain connection of multiple TMC6460 devices.

UART Interface

The UART interface in the TMC6460 supports full-duplex data exchange with external devices using industry standard NRZ asynchronous serial data format. This interface uses one start bit, eight data bits, one stop bit, and no parity bits.

The UART interface offers support for a wide range of baud rates as well as automatic baud rate detection (autobaud). A real time monitoring interface (RTMI) for fast-speed, periodic data streaming purposes is also incorporated.

The UART interface supports CRC error detection.

The UART interface may be used as the main communication channel with a host MCU if SPI is not required.

Alternatively, the UART interface can be used in parallel to the SPI interface. This can be useful, for instance, for real-time tuning / monitoring through the RTMI. In this case, only the TXD pin of the UART interface needs to be connected.

Single Wire Interface

The TMC6460 can be operated as a single-wire interface through the PWM_IN input pin. The single-wire input signal can be either a variable analog voltage or a variable duty cycle PWM signal. The input signal is then used as a control variable for any field in the register bank, for instance, to assign a velocity or position target for the controller.

Processing of the single-wire input values is done through the integrated I/O Controller. The scale of the input signal, the scale of the control target, as well as the register field to be written can all be independently configured.

The DIR_IN input can optionally be used as a second signal to define the sign of the control variable.

Note that either of the UART/SPI interfaces are still needed for initial configuration of this feature.

Feedback Engine

The TMC6460 integrates a versatile feedback engine with interfaces to support different types of sensors:

- Digital Hall sensors.
- ABN incremental encoders.
- Analog Hall sensors.
- Analog SinCos absolute encoders.
- SPI / SSI absolute encoders (through the integrated I/O Controller).

The feedback engine can also be configured for dual feedback setups, for example:

- Digital Hall for commutation + ABN for velocity and positioning.
- 1st ABN on the motor + 2nd ABN behind a gearbox.
- ABN on the motor + SSI encoder behind a gearbox.

The feedback engine also integrates:

- Velocity measurement modules.
- Multi-turn position counter (32 bits).

Additional encoder protocols such as BiSS-C, Tamagawa T-format, etc. can be integrated to the TMC6460 by uploading a custom program to the integrated I/O Controller.

Furthermore, arbitrary external angle feedback sources can be interfaced by writing the angle data through generic register access. For instance, if the host MCU itself is connected to the angle sensor.

Motor Control Core

The MCC of the TMC6460 contains the complete control loop architecture, that is, torque, velocity and position control loops. The MCC decouples all real-time critical tasks of the FOC from the user application layer. *Figure 3* depicts the simplified FOC architecture of the MCC.

The FOC torque PI controller forms the base component of the MCC, it includes the following transformations: Clark, Park, inverse Park, inverse Clark and space vector PWM (SVPWM). The MCC receives current measurements and motor angle information and outputs phase voltage signals to the PWM Engine. The FOC algorithm independently controls flux (I_D) and torque (I_Q) current components, which results in highly efficiency and quiet motor operation.

The PWM engine features a user programmable switching frequency, which can be configured from 2kHz up to 200kHz. The PWM modulation is based on a center aligned chopping scheme, a configurable SVPWM mode for maximum voltage utilization, and break before make (BBM) dead-time generator for the embedded power stage.

Outer velocity and position PI controllers are implemented on top of the inner FOC torque loop.

Finally, the TMC6460 integrates an eight-point ramp generator. Based on a single target position or target velocity, the ramp generator automatically calculates the motion profile based on the set velocity and acceleration parameters. The ramp generator supports calculation of velocity motion profiles with up to six independent acceleration and deceleration phases, plus a constant velocity segment. This way, highly dynamic position or velocity motions with smooth transitions and minimum jerk can be achieved. The output of the ramp generator is forwarded to the different loops in the control cascade, including direct control targets as well as feedforward input.

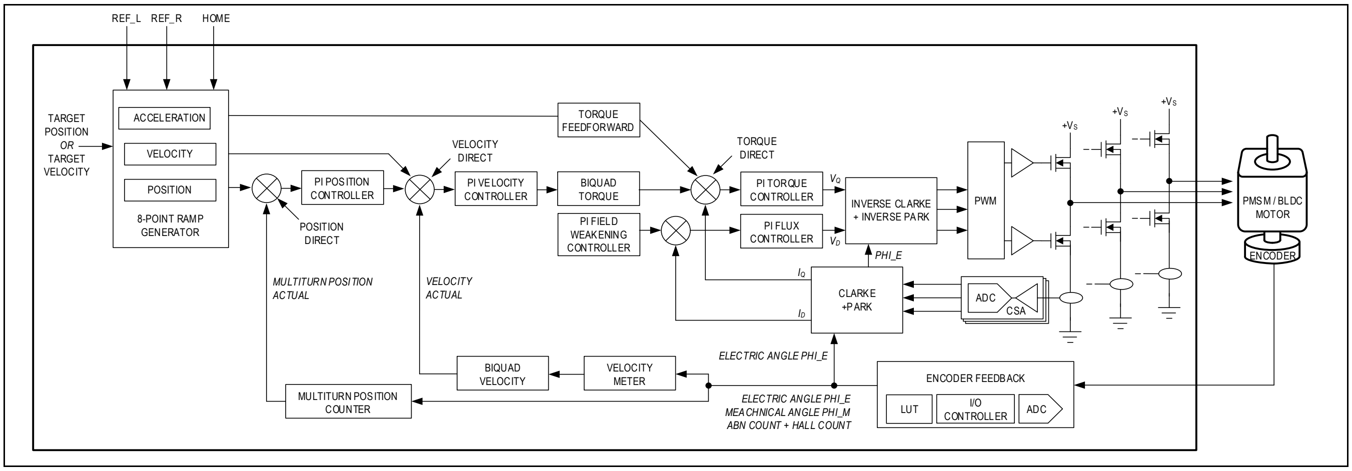


Figure 3. FOC and Motion Control Core

Power Output Stage

The TMC6460 integrates three 36V-capable half bridges with low impedance FETs. With a total R_{ON} of 110mΩ (high side + low side), the power stage allows high driving efficiency and minimal heat generation.

The TMC6460 can drive motors with up to $3A_{RMS}$ per phase. The maximum RMS current is primarily limited by thermal considerations, which are heavily dependent on the PCB layout and thermal characteristics of the application, e.g., PCB ground planes, heatsinks, ventilation, etc. The $3A_{RMS}$ rating refers to a JEDEC standard 4-layers 2s2p PCB operating at room temperature (T_A = 25°C).

Four different output slew rates for the half bridges can be configured, allowing to find the best compromise between electromagnetic emission and driving efficiency.

Current Acquisition

The internal current sensing capabilities of the TMC6460 eliminate the need for external power resistors, resulting in dramatic space and power savings. Instead, the power FETs are used as a current-sensing element, allowing this mechanism to not dissipate additional power.

The TMC6460 supports a variable current-sense scaling. The sensing element equivalent resistance ($R_{ON,LS}$) can be selected among four different values to adapt to different motor sizes and applications, optimizing accuracy and resolution.

In addition, it is possible to set a gain factor for the current-sense amplifier, to extend its equivalent range during analog-digital conversion. Four different gain values can be chosen. This setting can be modified during operation, for example: a low gain setting can be used in case of heavy acceleration when the motor current peaks, whereas a high gain setting can be used when the current demand decreases because the motor operates at constant velocity.

Diagnostics, Fault Handling and Protections

The TMC6460 provides comprehensive measurement and diagnostics capabilities. Parameters such as the actual position and velocity of the motor, as well as motor voltages, currents and power are all easily accessible for monitoring. Most variables pertinent to the control cascade and the FOC transforms are also available for querying.

An extensive selection of state flags and event bits provide an overview of the system state. This also allows efficient error detection and global diagnostics.

An overcurrent protection (OCP) mechanism is included in the TMC6460. It helps to protect the internal FETs in case of detected external short circuits or big current peaks. The threshold at which the OCP is triggered can be configured.

Other protection mechanisms triggered by events such as overvoltage of the power supply, overtemperature of the IC and overtemperature of an attached temperature sensor are also included in the TMC6460.

All these features provide the ability to do continuous system monitoring for safe motor operation in the end application.

Detailed Description

PLL and Clock Configuration

By default, the TMC6460 uses a f_{OSC} = 15MHz internal oscillator as its primary clock source. An integrated phase lock loop (PLL) uses the internal oscillator signal to generate a f_{PLL} = 120MHz clock, from which the rest of clock signals are generated. The PWM clock, used for generation of the PWM signals that drive the motor outputs, is generated directly from the PLL and thus runs at 120MHz. The system clock is generated at half of the PLL frequency and thus runs at f_{SYS} = 60MHz. The default clock configuration is carried out automatically during power-up and after reset, no further configuration is needed.

The system clock is used for some timing operation such as those of the ramp generator, thus, if the accuracy of the internal oscillator (see *Electrical Characteristics* table) is not sufficient for a particular application, a more accurate external clock signal can be provided in the CLK pin. To use the external clock signal as source for the PLL, set a value of 1_b to the PLL_SRC bitfield of the CLK_CTRL.CONFIG register. In addition, a clock divider must be set so that the PLL has a 1MHz clock input. To do this, set the CLOCK_DIVIDER field of the same register to a value of f_{EXT_CLK} / 1MHz - 1. For example, a value of 31 should be used if the external clock has a frequency of f_{EXT_CLK} = 32MHz. For normal operation, the CLK_FSM_EN, PWM_CLK_EN, ADC_CLK_EN and PLL_EN bits of the CLK_CTRL.CONFIG register must all be set. To apply the new clock configuration, write a 1_b to the COMMIT bit in the same register. The complete configuration can be carried out with a single write access to the CLK_CTRL.CONFIG register.

When a clock or PLL error occurs, for example due to a wrong configuration or a missing external clock signal, this is indicated by the CLK_PLL_FAIL_STATUS bit of the CHIP.STATUS_FLAGS register and the corresponding CLK_PLL_FAIL_EVENT of the CHIP.EVENTS register. After a fault is detected, the PLL is automatically disabled and the system clock source is changed back to the 15 MHz internal oscillator. After the original error cause is resolved, the CLK_PLL_FAIL_EVENT bit must be manually cleared. The configuration in CLK_CTRL.CONFIG register should be checked and the COMMIT bit set to 1_b once again. This restarts the clock and PLL with the given configuration.

The current status of the clock finite-state machine (FSM) can be queried from the CLK_CTRL.STATUS register.

Power Management

Setting LP_MODE_EN = 1_b will put the TMC6460 into low power/standby mode. The internal PLL is disabled in this mode, thus, all internal operations are executed with the internal oscillator clock f_{OSC} = 15MHz. Furthermore, PWM, ADC and I/O Controller operations are halted and thus consume no power.

Digital encoder and Hall sensor interfaces remain available in low power mode, allowing to track externally performed motor movements. Likewise, communication using UART and/or SPI interfaces remains available in this mode.

To exit low power mode, set LP_MODE_EN = 0_b, assert the DRV_EN pin, or reset the I/O Controller through register access.

To reduce current consumption even further, the SLEEPN pin can be asserted to set the TMC6460 into sleep mode. Note that communication through UART and/or SPI are not available during sleep mode and register contents will not be kept.

Refer to the *Electrical Characteristics* table to see the current consumptions during standby and sleep modes.

Fault Management

After the normal power-up sequence, the TMC6460 enters an off state where all functions operate normally except the output stage, which remains disabled. In this state, the SYS_READY_STATE bit of the CHIP.STATUS_FLAGS register is set. To enable the output stage and enter the driver on state, set the DRV_EN_BIT bitfield of MCC_CONFIG.GDRV register and assert the DRV_EN pin. If all conditions are met, the IC enters the driver on state, the bit GDRV_ON_STATE bit of the CHIP.STATUS_FLAGS register is set, and the SYS_READY_STATE bit is automatically cleared. In the driver on state, all functions of the TMC6460 can be used normally.

Various fault conditions may cause the TMC6460 to exit the driver on state and disable the output stage. The CHIP.STATUS_FLAGS and CHIP.EVENTS registers are a reference point for the cause of an error. When detecting any fault condition, the DRV_EN_BIT bitfield is automatically cleared. After resolving the error, set this bit again to enable the output stage. Note that resolving an error sometimes implies clearing several fault flags and reenabling other components.

ADC Re-initialization

If an ADC fault occurs, indicated by the ADC_FAIL_STATUS bit of the CHIP.STATUS_FLAGS register, it may be necessary to re-initialize the ADC engine. This can be done by issuing a reset of the I/O Controller through the RESET bit of the IO_CONTROLLER.CONTROL register.

Status Flags and Events

Status flags represent the current state of a certain error condition. They are cleared when the error condition disappears. Events latch the value of their corresponding status flag. They need to be cleared manually, which is possible only once the corresponding status flag is also inactive.

Events are not generated before successfully executing the first power-up, whereas status flags can be always read out.

Table 1 lists the status flags and events from the CHIP.STATUS_FLAGS and CHIP.EVENTS registers, respectively. Note that the PWRUP_FAIL_STATUS bit of CHIP.STATUS_FLAGS has no equivalent event bit. Instead, bit 0 of CHIP.EVENTS corresponds to the RST_EVENT bit. For all other cases, the Function/Comment column relates to the status flags.

Table 1. Overview: Status flags and events

Status Flag	Bit	Corresponding Event	Function/Comment
	0	RST_EVENT	A reset occurred. (EVENTS only)
PWRUP_FAIL_STATUS	0		Error occured during power-up. Power cycles is required. (STATUS_FLAGS only)
SPI_FAIL_STATUS	1	SPI_FAIL_EVENT	SPI timeout or interrupt error is active. Setting CSN high and SCK low resets the SPI interface to be ready for the next datagram.
UART_FAIL_STATUS	2	UART_FAIL_EVENT	Any UART error bit is active. Setting UART_RXD high and waiting for at least 70ms resets the UART interface to be ready for the next datagram.
VCC_IOF_FAIL_STATUS	3	VCCIOF_FAIL_EVENT	VCC_IOF supply voltage is below 1.85V.
CP_FAIL_STATUS	4	CP_FAIL_EVENT	Charge pump is not active.
UV_VM_STATUS	5	UV_VM_EVENT	VS supply voltage is below 2.2V.
POWER_FAIL_STATUS	6	POWER_FAIL_EVENT	Driver stage supply voltages are below the expected minimum values.
CLK_PLL_FAIL_STATUS	7	CLK_PLL_FAIL_EVENT	Any clock generation failure is active. The system clock uses the internal oscillator f_{SYS} = 15MHz. Reconfigure CLK_CTRL.CONTROL register manually, reset the I/O Controller or power cycle the IC to go back to the default clock configuration.
EXT_RES_FAIL_STATUS	8	EXT_RES_FAIL_EVENT	External reference resistor is beyond the expected range of 57kΩ - 63kΩ. Not active if the internal resistor is used.
ADC_FAIL_STATUS	9	ADC_FAIL_EVENT	Any ADC is not active. Reset the I/O Controller or power cycle the IC to reconfigure the ADCs.
TEMP_LIMIT_FAIL_STATUS	10	TEMP_LIMIT_FAIL_EVENT	Internal chip temperature is above 165°C.
VEL_FAIL_STATUS	11	VEL_FAIL_EVENT	Output frequency of the power stage is higher than the limit of 600Hz (electrical rev/s). The driver stage is disabled.
SHRT_FAIL_STATUS	12	SHRT_FAIL_EVENT	Short to ground or short to VS on any driver output detected for at least 3 consecutive PWM cycles. The driver stage is disabled.
OT_FAIL_STATUS	13	OT_FAIL_EVENT	The internal temperature of any driver stage channel exceeds 165°C.
DIFF_ENC_FAIL_STATUS	14	DIFF_ENC_FAIL_EVENT	Incorrect digital differential encoder input.
IO_CONTROLLER_STATUS	15	IO_CONTROLLER_EVENT	The I/O Controller is not active or the manually assigned I/O Controller flag is active.

RAMP_STALL_FAIL_STATUS	16	RAMP_STALL_FAIL_EVENT	Maximum position or velocity deviation of ramp generator is triggered. Refer to STALL_IN_POSITION_ERR and STALL_IN_V_ERR.
RAMP_TARGET_ REACHED_STATUS	17	RAMP_TARGET_ REACHED_EVENT	Ramp generator target position is reached. Refer to POSITION_REACHED_STATUS.
RAMP_V_REACHED_STATUS	18	RAMP_V_REACHED_EVENT	Ramp generator target/maximum velocity value is reached. Refer to V_REACHED_STATUS.
RAMP_V_ZERO_STATUS	19	RAMP_V_ZERO_EVENT	Ramp generator velocity value is equal to zero. Refer to V_ZERO.
RAMP_REF_LR_STATUS	20	RAMP_REF_LR_EVENT	Ramp generator left / right reference switch signal is active. Refer to RAMP_REF_L_STATUS and RAMP_REF_R_STATUS.
RAMP_REF_H_STATUS	21	RAMP_REF_H_EVENT	Ramp generator home reference switch signal is active. Refer to RAMP_REF_H_STATUS.
ABN_FAIL_STATUS	22	ABN_FAIL_EVENT	Combined ABN / ABN2 errors if these encoder interfaces are enabled.
HALL_FAIL_STATUS	23	HALL_FAIL_EVENT	Invalid combination on the Hall sensor decoder detected.
CURRENT_OVERLOAD_STATUS	24	CURRENT_OVERLOAD_EVENT	Any absolute phase current value exceeds manually defined limit.
I_CLIPPED_STATUS	25	I_CLIPPED_EVENT	Any of the measured phase current values is clipped.
AIN_CLIPPED_STATUS	26	AIN_CLIPPED_EVENT	Any of the measured analog values is clipped.
VM_TEMP_CLIPPED_STATUS	27	VM_TEMP_CLIPPED_EVENT	Any of the measured temperature or VS voltage is clipped.
OV_VM_LIMIT_FAIL_STATUS	28	OV_VM_LIMIT_FAIL_EVENT	Measured VS voltage value exceeds manually defined limits.
OVERTEMP_WARN_STATUS	29	OVERTEMP_WARN_EVENT	Measured external or internal temperature value exceeds manually defined limits.
SYS_READY_STATE	30	SYS_READY_EVENT	System ready state, the driver stage is disabled.
GDRV_ON_STATE	31	GDRV_ON _EVENT	Driver on state, the driver stage is enabled.

Some of the entries on this table have links with references to the relevant section in the manual.

FAULTN Pin

The TMC6460 provides a feature to drive the FAULTN pin according to the bits in CHIP.STATUS_FLAGS or CHIP.EVENTS registers. Whether the status flags or the events are used as a reference to drive the FAULTN pin is determined by the alternate function assignment for that pin. This is configured through the PIN_FAULTN field of the CHIP.IOMATRIX register as shown in Table 2.

Table 2. FAULTN pin alternate function assignment

PIN_FAULTN choice	FAULTN pin function
FAULTN_OD	Open drain output configuration with reference to CHIP.STATUS_FLAGS
INT_OD	Open drain output configuration with reference to CHIP.EVENTS
FAULTN_PP	Push-pull output configuration with reference to CHIP.STATUS_FLAGS
INT_PP	Push-pull output configuration with reference to CHIP.EVENTS

In addition to the pin alternate function assignment, an appropriate mask for selection of bits must be configured. This is done through FAULTN_INT_MASK field of the CHIP.FAULTN_INT_MASK register. Bits set to 1_b in the mask field select the corresponding bit from CHIP.STATUS_FLAGS or CHIP.EVENTS registers, according to the pin function assignment. If more than one bit is set in the mask, all the corresponding bits are OR wired into the FAULTN pin.

For example, if only the SHRT_FAIL_STATUS bit needs to be shown on the FAULTN output pin, then set the value of FAULTN_INT_MASK = 0x1000 and PIN_FAULTN = FAULT_OD or FAULT_PP. Similarly, if the FAULTN output pin shall show any of the CP_FAIL_EVENT or VCCIOF_FAIL_EVENT bits, then set the value of FAULTN_INT_MASK = 0x18 and PIN_FAULTN = INT_OD or INT_PP.

The FAULTN_INT_MASK field has a reset value of 0_b, which means that no status flag or event will be reflected on the FAULTN pin by default.

During power-up, the FAULTN pin is asserted to indicate that the power-up sequence is ongoing. If the power-up process is finished successfully, the FAULTN pin is deasserted and its configuration as described above takes place.

SPI Interface

The SPI interface in the TMC6460 has a simple command and control structure. It uses SPI mode 1 (with CPOL = 0 and CPHA = 1), where the input (SDI) is sampled on the falling edge of the SPI clock (SCK) and the output (SDO) is generated on the rising edge of SCK. A valid SPI command is at least 48 bits long and must be a multiple of 8 (like 48, 56, 64, etc.). If the number of bits of an SPI command is not a multiple of 8, or if it is less than 48 bits long, the command is ignored and the SPI_FAIL_STATUS bit of the CHIP.STATUS_FLAGS register is set.

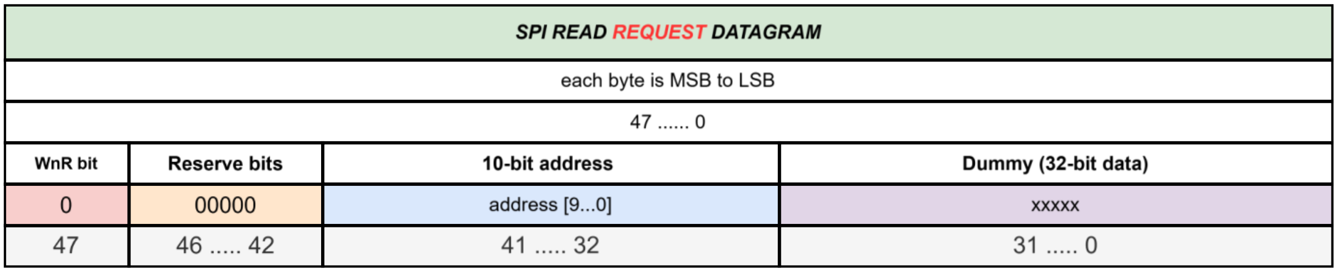
The SPI interface supports a timeout mechanism. A timeout occurs if no SCK rising edge occurs within t_{CCR_MAX} = 16µs after the CSN (Chip Select Not) falling edge, setting also the SPI_FAIL_STATUS bit. In low power mode, t_{CCR_MAX} increases to 67µs.

Regular communication with the TMC6460 through SPI is done with 48-bit datagrams. In each datagram, a request is sent through SDI, while the reply of the previous request is sent back simultaneously through SDO. SPI datagram requests are responded regularly with each incoming SCK cycle, even if only 8 bits are repeatedly requested. This allows the SPI main device to poll status bits, which are always sent first on the response datagrams.

SPI Read Request Datagram

Read request commands are used to read the contents of a register. The reply for the read request will be provided as the response datagram of the next issued command. The SPI read request datagram structure is shown in *Figure 4*.

- The MSB (bit 47) is sent first and the LSB (bit 0) is sent last.
- The MSB (bit 47) is the WRITE_notREAD (WnR) bit and it must be 0 for a read request.
- Bits 46 to 42 are reserved bits and must always be 0.
- Bits 41 to 32 are the address bits (10-bit address).
- Bits 31 to 0 are not relevant for read datagrams (32-bit dummy data).



SPI READ REQUEST DATAGRAM			
each byte is MSB to LSB			
47 0			
WnR bit	**Reserve bits**	**10-bit address**	**Dummy (32-bit data)**
0	00000	address [9...0]	xxxxx
47	46 42	41 32	31 0

Figure 4. SPI read request datagram structure

An example for an SPI read request datagram is 0x014000000000, which would be sent to read the address 0x140 (FOC.PID_CONFIG).

SPI Read Response Datagram

The read response contains the contents of a read back register. It is sent as the response datagram of the next issued command after a read request command. For example, to read the contents of a register without issuing a write command, send two consecutive read commands. The reply will then be contained in the response datagram of the second read command. The SPI read response datagram structure is shown in *Figure 5*.

- The MSB (bit 47) is sent first and the LSB (bit 0) is sent last.
- Bits 47 to 42 are 6 status bits, which can contain a selection of bits from the CHIP.EVENTS register by using the CHIP.SPI_STATUS_MASK register.
- Bits 41 to 32 is the 10-bit address from the read request datagram.
- Bits 31 to 0 is the 32-bit reply data from the specified address.

SPI READ RESPONSE DATAGRAM		
each byte is MSB to LSB		
47 0		
Status data[5:0]	**Address[9:0]**	**Reply data[31:0]**
xxxxxx	xxxxxx	xxxxx
47 42	41 32	31 0

Figure 5. SPI read response datagram structure

Figure 6 depicts an SPI datagram example. The main asserts CSN to select the subordinate and generates clock pulses on SCK to synchronize data transfer. Each clock edge shifts one bit, shown as numbered positions (7-0) for bytes, with logic levels depicted as high and low states. On SDI, the main sends a dummy read request to register 0x00, while on SDO, the subordinate returns the response for the previous request, including status bits, address (0x140), and the 32-bit response data.

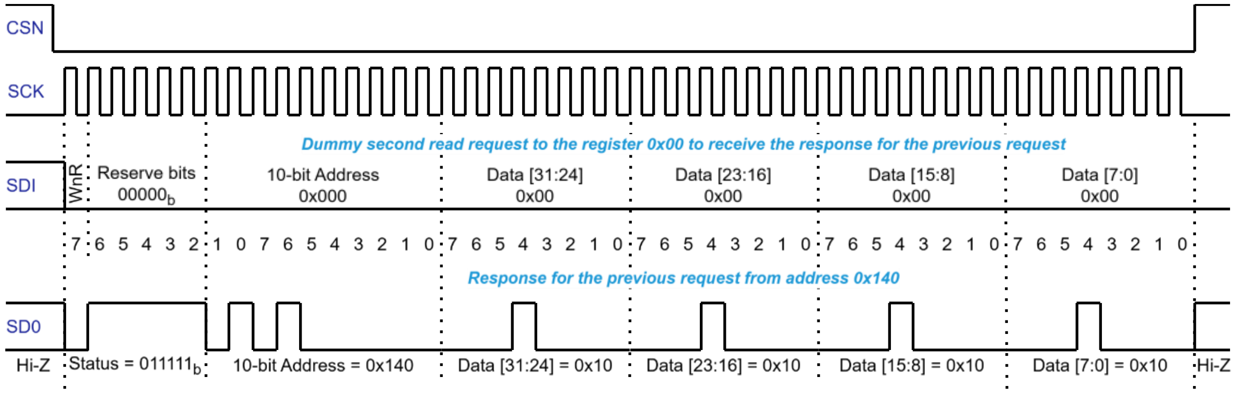


Figure 6. SPI dummy read request to receive data from previous read request datagram

SPI Write Request Datagram

Write request commands are used to modify the contents of a register. The reply for the write request will be provided as the response datagram of the next issued command. The SPI write request datagram structure is shown in Figure 7.

- The MSB (bit 47) is sent first and the LSB (bit 0) is sent last.
- The MSB (bit 47) is the WRITE_notREAD (WnR) bit and it must be 1.
- Bits 46 to 42 are reserved bits and it must be 0.
- Bits 41 to 32 are the address bits (10-bit address).
- Bits 31 to 0 are the data bits (32-bit data) that are written into the register at the defined address.

SPI WRITE REQUEST DATAGRAM			
each byte is MSB to LSB			
47 0			
WnR bit	**Reserve bits**	**10-bit address**	**Write data (32-bit data)**
1	00000	address [9...0]	xxxxx
47	46 42	41 32	31 0

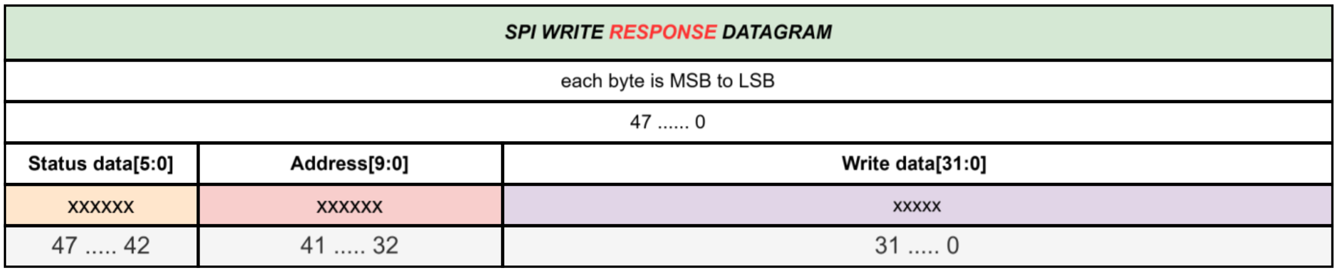
Figure 7. SPI write request datagram structure

An example of SPI write request datagram is 0x80C300000002 which writes data = 0x00000002 into the address 0x0C3 (MCC_ADC.CSA_GAIN).

SPI Write Response Datagram

The write response contains the read-after-write data of a written register. It is sent as the response datagram of the next issued command after a write request command. For example, to receive the reply of a write request without issuing a second write command, send a read request command. The reply will then be contained in the response datagram of the read command. The SPI read response datagram structure is shown in *Figure 8*.

- The MSB (bit 47) is sent first and the LSB (bit 0) is sent last.
- Bits 46 to 42 are status bits, which can contain a selection of bits from the CHIP.EVENTS register based on the CHIP.SPI_STATUS_MASK configuration.
- Bits 41 to 32 is the address from the write request datagram.
- Bits 31 to 0 is the read-after-write data from the specified address.



SPI WRITE RESPONSE DATAGRAM		
each byte is MSB to LSB		
47 0		
Status data[5:0]	**Address[9:0]**	**Write data[31:0]**
xxxxxx	xxxxxx	xxxxx
47 42	41 32	31 0

Figure 8. SPI write response datagram structure

SPI Daisy Chain Support

The SPI interface of the TMC6460 provides support for daisy chain connection of multiple peripherals. In this configuration, all daisy chained peripherals share the CSN and SCK signals. The host controller's SDO is connected to the SDI of the first TMC6460 device; the SDO of each device is then connected to the SDI of the next device in the chain. This sequence continues until the SDO of the last peripheral is connected back to the SDI of the host controller. *Figure 9* illustrates an example daisy chain setup with three SPI peripherals and one host microcontroller (MCU).

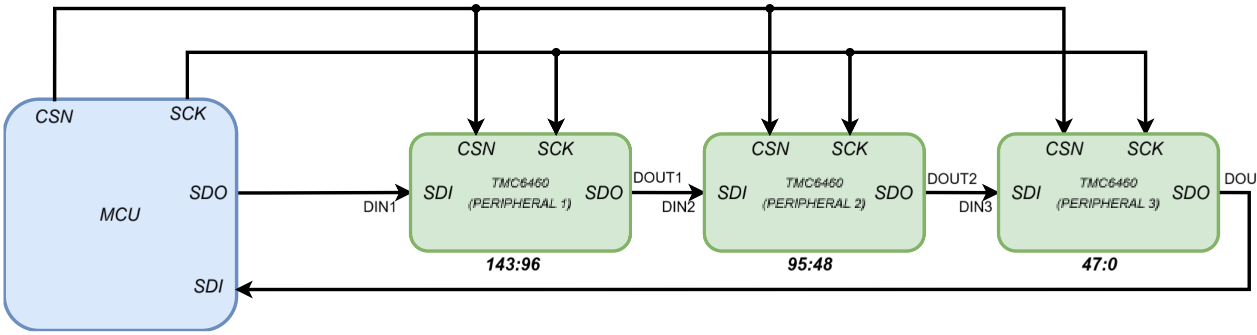


Figure 9. SPI daisy chain example with three SPI peripherals in a row

No additional configuration is required to enable daisy chain operation on the TMC6460. Daisy chaining is achieved by keeping the CSN signal asserted while transmitting multiple datagrams to the device chain. When the SPI interface on the TMC6460 receives more than 48 bits on SDI, the older bits are shifted out on SDO and forwarded to the next peripheral. When CSN is de-asserted, only the bits currently held by the peripheral are considered. The datagram held by any peripheral at the end of a transaction must still be 48 bits long for the SPI interface to correctly process it. *Figure 9* also shows which data bits are processed in each peripheral at the end of one daisy chain transaction.

A representation of the SPI transactions for reading a different register from each of the three peripherals is given in *Figure 10*. Note that in a real transaction, the SCK and data lines do not have an interruption in between datagrams. This is shown in this way for ease of visualization. Follow the explanation below:

- Request datagrams are sent from the MCU to all peripherals through peripheral 1.
- In the first 48 cycles of SCK, peripheral 1 receives (DIN1) the datagram from 47:0 (0x004000000000).
- In the second 48 cycles of SCK, peripheral 2 receives the output of the peripheral 1 (DOUT1 to DIN2) and peripheral 1 receives (DIN1) the datagram from 95:48 (0x8080000CC003).
- In the third 48 cycles of SCK, peripheral 3 receives the output of peripheral 2 (DOUT2 to DIN3), peripheral 2 receives the output of the peripheral 1 (DOUT1 to DIN2) and peripheral 1 receives (DIN1) the datagram from 143:96 (0x80C07FE20000).

- At the end of 144 SCK cycles, peripheral 1 has the read request datagram 143:96, peripheral 2 has the read request datagram 95:48 and peripheral 3 has the read request datagram 47:0.
- As the CSN is deasserted, each of the peripherals process the datagram they currently hold.
- Respond datagrams are performed next and a similar process with 144 cycles of SCK while asserting CSN. This time dummy datagrams (0x000000000000) are sent, as only the response datagram of each peripheral is of interest.
- In the first 48 cycles of SCK, peripheral 3 sends out (DOUT3) its response (0x004000000E3C) to the MCU, peripheral 2 sends its own response (0x0080000CC003) to peripheral 3 (DOUT2 to DIN3), while peripheral 1 also sends its own response (0x00C07FE20000) to peripheral 2 (DOUT1 to DIN2).
- In the second 48 cycles of SCK, peripheral 3 sends out (DOUT3) the original response from peripheral 2 (0x0080000CC003) to the MCU, while peripheral 2 sends the original response of peripheral 1 (0x00C07FE20000) to peripheral 3 (DOUT2 to DIN3).
- In the third 48 cycles of SCK, peripheral 3 sends out (DOUT3) the original response of peripheral 1 (0x00C07FE20000) to the MCU, then CSN is deasserted.

Alternatively to the dummy datagrams shown in this example, it is also possible to send the next series of read request datagrams or write request datagrams to make the communication process more efficient.

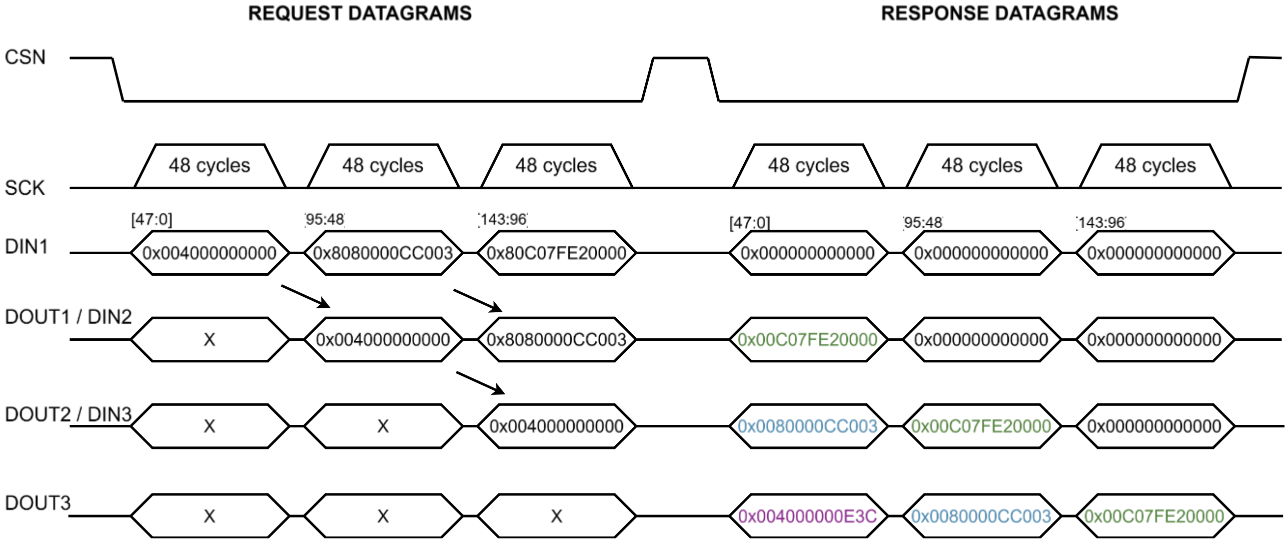


Figure 10. SPI Daisy Chain Datagram Example with three SPI peripherals in a row

UART Interface

The UART interface in the TMC6460 is a three-pin interface (GND, RXD, TXD) that supports up to 12Mbit/s transfer rates with one start bit, eight data bits, one stop bit, and no parity bits.

The TMC6460 UART supports standard baud rates, a selection of them is shown on *Table 3*. Additionally, an autobaud mode for automatic baud rate detection is included. In this mode, the baud rate is detected from the received datagrams, and the appropriate baud rate value is automatically set. The autobaud feature is enabled by default.

Additionally, the UART interface has an optional 8-bit cyclic redundancy check (CRC) feature to detect bit flips in the datagram. If there are one or more bit flips in the datagram, the datagram is ignored by the interface. The CRC feature is disabled by default.

Moreover, the UART interface incorporates a real-time monitoring interface (RTMI), which can be used for fast-speed, periodic data streaming purposes.

UART Baud Rate Settings

Autobaud is activated by default and the TMC6460 calculates the baud rate from the sync bits at the start of each UART transfer on UART_RXD. Disable the autobaud by clearing the AUTOBAUD_EN bit in the UART.CONTROL register. To set the baud rate to the desired value, write the MANTISSA_LIMIT bit field in the same register by using this formula:

$$MANTISSA\_LIMIT = \frac{f_{CLK}}{baud\_rate}$$

With clock frequency $f_{CLK} = f_{SYS}$ = 60MHz when the PLL is enabled or $f_{CLK} = f_{OSC}$ = 15MHz when the PLL is disabled. Since the MANTISSA_LIMIT is an integer value, and fractional baud generation is not supported, only specific baud rates can be achieved. Baud rates like 7Mbits/s, 8Mbits/s, 9Mbits/s and 11Mbits/s, for example, cannot be generated. Table 3 shows supported baud rates for a selection of mantissa limit values, assuming that the PLL is enabled.

Table 3. Baud Rate Settings

Baud rate [bits/s]	9600	19200	38400	57600	115200	230400	460800	921600	
MANTISSA_LIMIT	6250	3125	1563	1042	521	260	130	65	
Baud rate [Mbits/s]	1	2	3	4	5	6	7.5	10	12
MANTISSA_LIMIT	60	30	20	15	12	10	8	6	5

Note that baud rates greater than 12Mbits/s or smaller than 9600bits/s are not supported regardless of the clock settings.

Changing the baud rate dynamically is not supported. Any attempt to change the baud rate during an active UART_TXD transmission will be rejected, this rule applies in all cases. The baud rate will only change if the UART_TXD is inactive.

Changing the baud rate through a regular write request command in MANTISSA_LIMIT works, as the response to a write request is sent only after the datagram is fully processed by the UART interface. In this case, the write response would be sent back with the newly set baud rate.

Transfer rate limits

The UART transfer rate limits are determined by the clock configurations, the type of UART datagrams, and autobaud settings. The different limits are shown in Table 4.

Table 4. UART Speed Limitations

UART Configuration		**Clock Setup**		
Datagram Type	**Autobaud**	**PLL disabled**	**PLL enabled**	
			PLL source: Internal Oscillator	**PLL source: External Clock**
Regular read/write datagrams	Enabled	Up to 1.0Mbits/s	Up to 6.0Mbits/s	Up to 7.5Mbits/s
	Disabled	Up to 3.0Mbits/s	Up to 12.0Mbits/s	Up to 12.0Mbits/s
RTMI stream write datagram	Enabled	-	Up to 5.0Mbits/s	Up to 6.0Mbits/s
	Disabled	-	Up to 12.0Mbits/s	Up to 12.0Mbits/s

For very high baud rates, especially for RTMI streams, it is recommended to disable autobaud.

If the PLL is disabled, the system clock frequency is reduced to 15MHz. With this clock frequency, the UART interface can only support transfer rates up to 3Mbits/s. Additionally, since fractional baud rate generation is not supported, some standard transfer rates like 2Mbit/s are not supported anymore if the PLL is disabled.

UART Read Request Datagram

The UART read request datagram is 16 bits long (CRC disabled) or 24 bits long (CRC enabled). The structure of a UART read request datagram with CRC disabled is depicted in Figure 11. This applies irrespective of autobaud settings.

- The LSB (bit 0) is sent first and the MSB (bit 15 or 23) is sent last.
- Bits 0 to 2, 6, and 7 are sync bits used for autobaud calculations. The sync bits must be set to 010_b for bits 0 to 2 and 10_b for bits 6 and 7.
- Bit 3 is the Write_notRead bit and must be set to 0_b for a read request.
- Bits 8 to 15 represent the first part of the 10-bit address (address[7:0]), while bits 4 and 5 represent the second part (address[9:8]).
- (Only with CRC) Byte 2 is the 8-bit CRC.

UART READ REQUEST DATAGRAM (WITHOUT CRC)				
byte 0				byte 1
sync bits	WnR	address	sync bits	8-bit address
010	0	address [8...9]	10	address [0....7]
0...2	3	4 5	6, 7	8 15

Figure 11. UART read request datagram structure (NORMAL_CRC_EN = 0_b)

An example read request datagram is {0x52 0x40}, which would be used to read the FOC.PID_CONFIG register located at address 0x140.

When CRC is enabled (NORMAL_CRC_EN bit of UART.CONTROL register), the UART interface expects three bytes for the read request, as shown in *Figure 12*.

UART READ REQUEST DATAGRAM (WITH CRC)					
byte 0				byte 1	byte 2
sync bits	WnR	address	sync bits	8-bit address	8-bit CRC
010	0	address [8...9]	10	address [0....7]	checksum
0 ...2	3	4 5	6, 7	8 15	16 23

Figure 12. UART read request datagram structure (NORMAL_CRC_EN = 1_b)

An example read request datagram with CRC enabled is {0x52 0x40 0x12}, where 0x12 is the calculated CRC value. This command would also be used to read FOC.PID_CONFIG register at address 0x140.

UART Read Response Datagram

The UART read response datagram is 48 bits long (CRC disabled) or 56 bits long (CRC enabled). The UART read response datagram structure with CRC disabled is shown in *Figure 13*.

- The LSB (bit 0) is sent first and the MSB (bit 47 or 55) is sent last.
- The first two bytes (byte 0 and byte 1) are identical to those in the read request, containing sync bits, Write_notRead bit, and address bits.
- Bits 16 to 47 (bytes 2 to 5) contain the 32-bit read data from the requested address.
- (Only with CRC) Byte 6 is the 8-bit CRC

UART READ RESPONSE DATAGRAM (WITHOUT CRC)								
byte 0				byte 1	byte 2	byte 3	byte 4	byte 5
sync bits	WnR	address	sync bits	8-bit address	reply data (32-bit)			
010	0	address [8...9]	10	address [0....7]	data byte 3	data byte 2	data byte 1	data byte 0
0 ...2	3	4 5	6, 7	8 15	16 23	24 31	32 39	40 47

Figure 13. UART read response datagram structure (NORMAL_CRC_EN = 0_b)

An example read response datagram without CRC is {0x52 0x40 0x00 0x00 0x85 0x5C}, corresponding to a read data of 0x0000855C from the FOC.PID_CONFIG register at address 0x140.

When CRC is enabled (NORMAL_CRC_EN bit of UART.CONTROL register), the read response includes an extra, internally calculated CRC byte. This is shown in *Figure 14*.

UART READ RESPONSE DATAGRAM (WITH CRC)									
byte 0				byte 1	byte 2	byte 3	byte 4	byte 5	byte 6
sync bits	WnR	address	sync bits	8-bit address	reply data (32-bit)				8-bit CRC
010	0	address [8...9]	10	address [0....7]	data byte 3	data byte 2	data byte 1	data byte 0	checksum
0 ...2	3	4 5	6, 7	8 15	16 23	24 31	32 39	40 47	48 55

Figure 14. UART read response datagram structure (NORMAL_CRC_EN = 1_b)

An example read response datagram with CRC is {0x52 0x40 0x00 0x00 0x85 0x5C 0x6A} where byte 6 (0x6A) is the calculated CRC by the UART interface.

A detailed example of a read request and response datagram is shown in *Figure 15*. If CRC is enabled, and any of the bits in the first two bytes are corrupted or received incorrectly by the UART interface, then the request is ignored, and no response is sent.

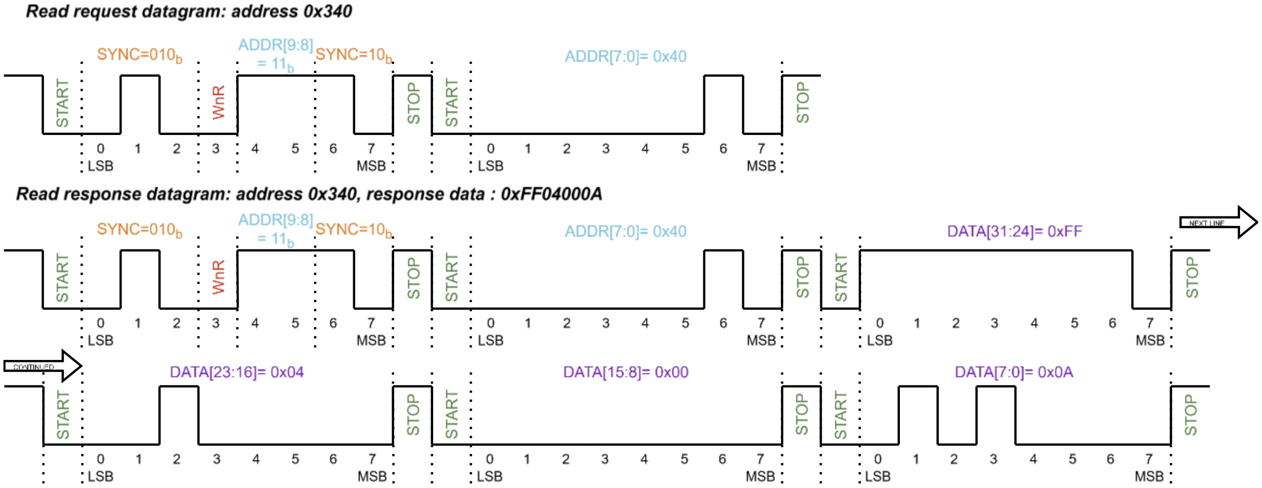


Figure 15. UART read access example

UART Write Request Datagram

The UART write request datagram is 48 bits long (CRC disabled) or 56 bits long (CRC enabled). The structure of a UART write request datagram with CRC disabled is depicted in *Figure 16*. This applies irrespective of autobaud settings.

- The LSB (bit 0) is sent first and the MSB (bit 47 or 55) is sent last.
- Bits 0 to 2, 6, and 7 are sync bits used for autobaud calculations. The sync bits must be set to 010_b for bits 0 to 2 and 10_b for bits 6 and 7.
- Bit 3 is the Write_notRead bit and must be set to 1_b for a write request.
- Bits 8 to 15 represent the first part of the 10-bit address (address[7:0]), while bits 4 and 5 represent the second part (address[9:8]).
- Bits 16 to 47 (bytes 2 to 5) are the 32-bit data to be written to the register at the specified address.
- (Only with CRC) Byte 6 is the 8-bit CRC.

UART WRITE REQUEST DATAGRAM (WITHOUT CRC)								
byte 0				byte 1	byte 2	byte 3	byte 4	byte 5
sync bits	WnR	address	sync bits	8-bit address	write data (32-bit)			
010	1	address [8...9]	10	address [0....7]	data byte 3	data byte 2	data byte 1	data byte 0
0 ...2	3	4 5	6, 7	8 15	16 23	24 31	32 39	40 47

Figure 16. UART write request datagram structure (NORMAL_CRC_EN = 0_b)

An example write request datagram is {0x5A 0x4E 0x01 0x00 0x01 0x00}, which would write the value 0x01000100 in the register FOC.PID_TORQUE_FLUX_TARGET at address 0x14E.

When CRC is enabled (NORMAL_CRC_EN bit of UART.CONTROL register), the UART interface expects an extra CRC byte for the write request, as shown in *Figure 17*.

UART WRITE REQUEST DATAGRAM (WITH CRC)									
byte 0				byte 1	byte 2	byte 3	byte 4	byte 5	byte 6
sync bits	WnR	address	sync bits	8-bit address	write data (32-bit)				8-bit CRC
010	1	address [8...9]	10	address [0....7]	data byte 3	data byte 2	data byte 1	data byte 0	checksum
0 ...2	3	4 5	6, 7	8 15	16 23	24 31	32 39	40 47	48 55

Figure 17. UART write request datagram structure (NORMAL_CRC_EN = 1_b)

An example write request datagram with CRC enabled is {0x5A 0x4E 0x01 0x00 0x01 0x00 0x5A} which writes into FOC.PID_TORQUE_FLUX_TARGET register at address 0x14E, where the last byte 0x5A is the calculated CRC value.

UART Write Response Datagram

The UART write response datagram is 40 bits long (CRC disabled) or 48 bits long (CRC enabled). The response to the write request is only sent if the bitfield CH_7_EN in the UART.RTMI_CH_7 register is set. If not set, then the write request will be processed, but no response will be sent. Refer to the section UART RTMI for more information. The write response datagram structure without CRC is shown in *Figure 18*.

- The LSB (bit 0) is sent first and the MSB (bit 39 or 47) is sent last.
- Bits 0 and 7 are sync bits that are 1_b and 0_b, respectively.
- Bits 1 to 3 represent the channel ID, which is always 111_b for normal write response datagrams.
- Bits 4 to 6 contain the write count, which indicates the number of executed write requests. The write count is shared for both stream and normal UART write requests, as well as SPI write requests. Executed write requests from any of these interfaces increase the same write count.
- Bits 8 to 39 (bytes 1 to 4) are the 32-bit response data.
- (Only with CRC) Byte 5 is the 8-bit CRC.

UART WRITE RESPONSE DATAGRAM (WITHOUT CRC)							
byte 0				byte 1	byte 2	byte 3	byte 4
sync bit	reg_id	wrt_cnt	sync bit	response data (32-bit)			
1	111	xxx	0	data byte 3	data byte 2	data byte 1	data byte 0
0	1...3	4...6	7	8 15	16 23	24 31	32 39

Figure 18. UART write response datagram structure (RTMI_CRC_EN = 0_b)

An example of a write response datagram is {0x1F 0x01 0x00 0x01 0x00} which would be the response of writing the value 0x01000100 in the FOC.PID_TORQUE_FLUX_TARGET register at address 0x14E. The first byte 0x1F indicates that one write access has been executed so far.

When CRC is enabled (RTMI_CRC_EN bit of UART.CONTROL register), the write response includes an extra, internally calculated CRC byte. This is shown in *Figure 19*. Note that the NORMAL_CRC_EN bit is **not** considered for this datagram.

UART WRITE RESPONSE DATAGRAM (WITH CRC)								
byte 0				byte 1	byte 2	byte 3	byte 4	byte 5
sync bit	reg_id	wrt_cnt	sync bit	response data (32-bit)				8-bit CRC
1	111	xxx	0	data byte 3	data byte 2	data byte 1	data byte 0	checksum
0	1...3	4...6	7	8 15	16 23	24 31	32 39	40 47

Figure 19. UART write response datagram structure (RTMI_CRC_EN = 1_b)

Figure 20 shows the structure of a UART write request and its corresponding response. The top waveform represents the host peripheral sending the write command with address 0x240 and data 0x80808091, while the bottom waveform shows the response confirming the write with the same address and data.

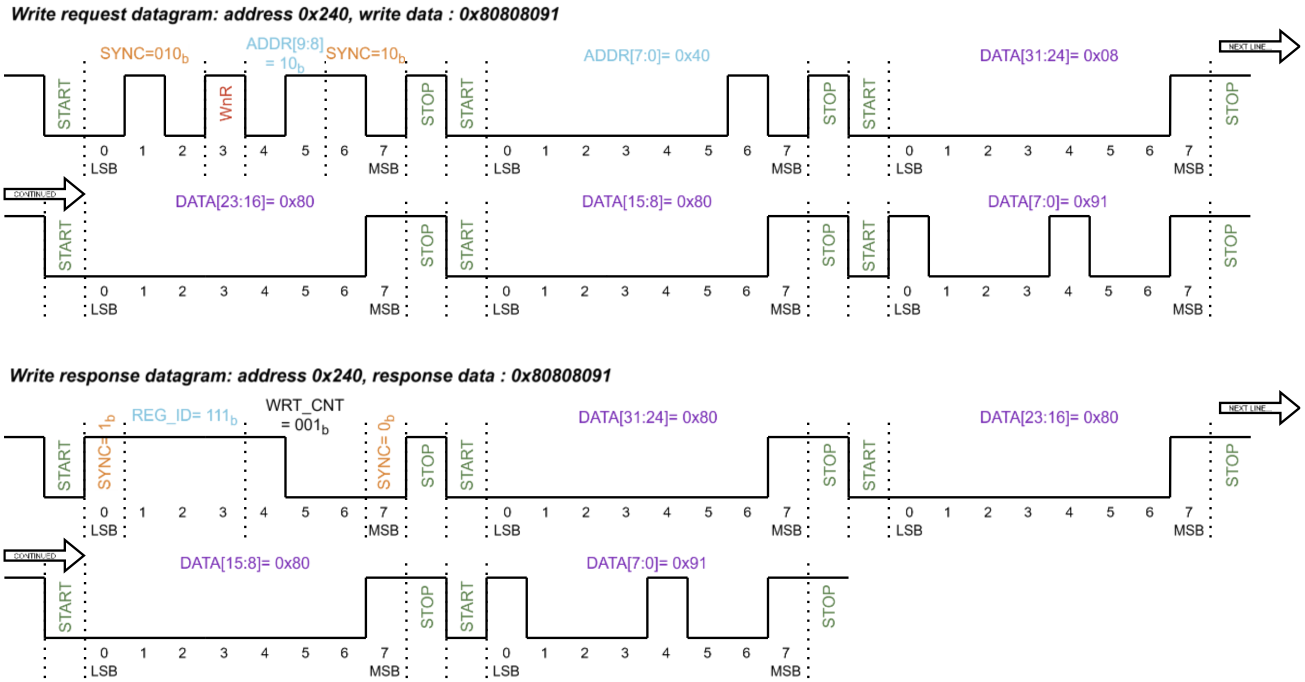


Figure 20. UART write access example

UART RTMI

The UART real-time monitoring interface (RTMI) is an additional interface that enables real-time data streaming from and to the TMC6460. Application-level software functions like identification of motor and system parameters are thereby facilitated. Up to eight channels can be mapped to any of the registers in the IC for monitoring purposes.

RTMI configuration

To use the UART RTMI feature, set the RTMI_EN bit in the UART.CONTROL register. The RTMI trigger frequency and each of the RTMI channels to be used must be configured beforehand.

Each of the available RTMI channels must be independently configured and enabled through registers UART.RTMI_CH_x (x = 0…7), each channel has two configuration options:

- Register field CH_x_ADDR (x = 0…7) defines the address of the register to be mapped to this RTMI channel, and
- Register field CH_x_EN (x = 0…7) enables the individual RTMI channel.

Data for all the enabled channels is sent out periodically on the UART_TXD pin based on a trigger with configurable frequency. The trigger is generated with the start of a PWM cycle, and according to the down sampling rate value in RTMI_SAMPLING field of the UART.CONTROL register. Use the following formula to calculate trigger frequency:

$$f_{TRIGGER} = \frac{f_{PWM}}{RTMI\_SAMPLING + 1}$$

For example, if the current PWM frequency is 50kHz, and RTMI_SAMPLING = 1, then new RTMI data streams are triggered with a frequency of 25kHz.

The length of the data stream according to the number of enabled channels and the set baud rate must be considered when choosing an RTMI trigger frequency. If a new RTMI trigger is issued during an ongoing data stream, then the new stream interrupts the previous one and the RTMI_INTERRUPT_EVENT field in the UART.EVENTS register is set.

The enabled channel responses can be received completely between RTMI triggers if the following relation is satisfied:

$$T_{TRIGGER} > T_{RTMI}$$

- $T_{TRIGGER} = \frac{1}{f_{TRIGGER}}$ and it is the time between two consecutive RTMI triggers.
- T_{RTMI} is the time taken for the RTMI stream data (enabled channels) to be sent, and it can be written as:

$$T_{RTMI} = n_{bytes} \times n_{channels} \times 10 \times T_{BIT}$$

- $n_{bytes} = 5$ when RTMI_CRC_EN = 0 or $n_{bytes} = 6$ when RTMI_CRC_EN = 1,
- $n_{channels}$ is the number of enabled channels,
- $T_{BIT} = \frac{1}{f_{UART}}$ is the time taken to send each UART bit.

For example, assuming an RTMI data stream with a PWM frequency of 50kHz, RTMI_SAMPLING of 4, number of enabled channels of 5, and the UART baud rate of 1MBaud with CRC disabled.

- $f_{TRIGGER} = \frac{f_{PWM}}{RTMI\_SAMPING+1} = \frac{50kHz}{4+1} = 10kHz$
- $T_{TRIGGER} = \frac{1}{f_{TRIGGER}} = \frac{1}{10KHz} = 100\mu s$
- $T_{RTMI} = n_{bytes} \times n_{channels} \times 10 \times T_{BIT} = 5 \times 5 \times 10 \times \frac{1}{1\,MBaud} = 250 \times 1\mu s = 250\mu s$

In this case, $T_{TRIGGER}$ is smaller than T_{RTMI}, so not all the channel responses will be sent between the RTMI triggers. This scenario would require a modification of the RTMI_SAMPLING, the PWM frequency or the UART baud rate to receive all the responses. If the UART baud rate is increased to 4MBaud, the T_{RTMI} becomes 62.5µs. Then, the $T_{TRIGGER}$ is greater than T_{RTMI} and the channels responses would be received without any interruption.

The following shows an example UART write request command to enable and configure the RTMI channel 0.

- Enable and map the RTMI channel 0 with ADC.I2_I1_RAW register: {0x7A 0x44 0x00 0x01 0x00 0x82}.
  - The first two bytes (0x7A 0x44) contain the sync and target address (RTMI_CH_0 register) data.
  - The last four bytes (0x00 0x01 0x00 0x82) contain the data to be written into the RTMI_CH_0 register.
    - Byte (0x01) corresponds to setting the CH_0_EN bitfield.
    - Bytes (0x0082) correspond to the address of the mapped ADC.I2_I1_RAW register.

A similar command can be issued for all other RTMI channels that need to be used.

UART RTMI Channel Response Datagram

The UART RTMI Channel Response Datagram is 40 bits long (CRC disabled) or 48 bits long (CRC enabled). As mentioned above, when the RTMI is enabled the UART interface periodically issues channel responses with the configured trigger frequency. The RTMI channel response datagram structure without CRC is shown in *Figure 21*.

- The LSB (bit 0) is sent first and the MSB (bit 39 or 47) is sent last.
- Bits 0 and 7 are sync bits that are 1_b and 0_b, respectively.
- Bits 1 to 3 are the channel ID bits, they are set to a value between 000_b (channel 0) and 111_b (channel 7), indicating the RTMI channel for which the data is sent.
- Bits 4 to 6 contain the write count, which indicates the number of executed write requests. The write count is shared for both stream and normal UART write requests, as well as SPI write requests. Executed write requests from any of these interfaces increase the same write count.
- Bits 8 to 39 (bytes 1 to 4) are the 32-bit response data for the RTMI channel corresponding to the channel ID.
- (Only with CRC) Byte 5 is the 8-bit CRC.

CHANNEL RESPONSE DATAGRAM (WITHOUT CRC)							
byte 0				byte 1	byte 2	byte 3	byte 4
sync bit	reg_id	wrt_count	sync bit	response data (32-bit)			
1	xxx	xxx	0	data byte 3	data byte 2	data byte 1	data byte 0
0	1...3	4...6	7	8 15	16 23	24 31	32 39

Figure 21. RTMI stream channel response datagram structure (RTMI_CRC_EN = 0_b)

Figure 22 shows a representation of an example RTMI channel response, in which RTMI channels 2, 4, 6, and 7 are not enabled. Data for those channels is therefore not included in the stream response. The channel response for all enabled channels is sent when the RTMI trigger is received, even if the RTMI_EN bit is cleared during an ongoing transmission. After a transmission is finished, the TXD pin remains idle until a new RTMI trigger is generated.

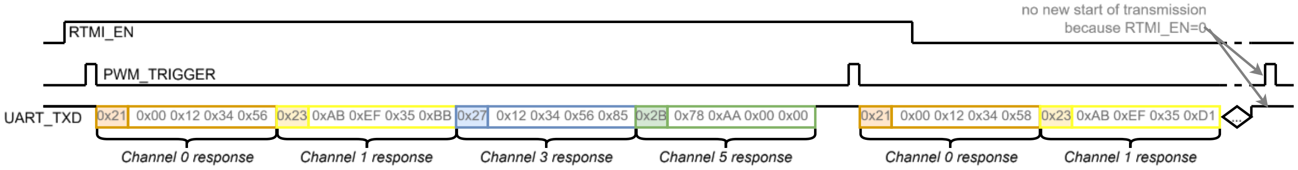


Figure 22. Example of a UART stream channel response datagram

These types of response datagram also include 8-bit CRC by enabling RTMI_CRC_EN bitfield from the UART.CONTROL register. The datagram structure is in *Figure 23*.

STREAM WRITE REQUEST DATAGRAM (WITH CRC)									
byte 0				byte 1	byte 2	byte 3	byte 4	byte 5
sync bit	reg_id	resv_bits	sync bit	write data (32-bit)				8-bit CRC
1	3-bit	001	0	data byte 3	data byte 2	data byte 1	data byte 0	checksum
0	1...3	4...6	7	8 15	16 23	24 31	32 39	40 47

Figure 23. RTMI stream channel response datagram structure (RTMI_CRC_EN = 1_b)

UART RTMI Stream Write Request Datagram

The stream write request is offered as an alternative to normal write requests. It has the advantage of being 1 byte shorter than the normal write request datagram, thereby allowing faster data transfer rates. However, only a selection of eight different registers can be written in this way. This operation relies on the UART RTMI channels, therefore, individual RTMI channels, as well as the RTMI feature must be enabled for stream write requests to function.

The registers that can be written through a stream write request command are determined by the configuration in registers UART.RTMI_CH_x (x = 0…7). For instance, if the RTMI channel 0 is mapped to register FOC.PID_POSITION_ACTUAL at address 0x155, then this register becomes available for stream write commands. Updating the value of this register can then be done by issuing a stream write request on the RTMI channel 0.

The stream write request datagram is 40 bits long (CRC disabled) or 48 bits long (CRC enabled). *Figure 24* shows the write request datagram structure. Please note that this datagram type is not used as a trigger for the RTMI.

- The LSB (bit 0) is sent first and the MSB (bit 39 or bit 47) is sent last.
- Bits 0 and 7 are sync bits, they must be set to 1_b for bit 0 and 0_b for bit 7. They are used for autobaud calculation.
- Bits 1 to 3 are the channel ID bits, they represent the selected RTMI channel used for the write operation. A value between 000_b (channel 0) and 111_b (channel 7) must be set.
- Bits 4 to 6 are the reserved bits and must be set to 001_b. These bits are also used for the autobaud calculation.
- Bits 8 to 39 (bytes 1 to 4) are the 32-bit data to be written at the register mapped on the specified RTMI channel.
- (Only with CRC) Byte 5 is the 8-bit CRC.

STREAM WRITE REQUEST DATAGRAM (WITHOUT CRC)							
byte 0				byte 1	byte 2	byte 3	byte 4
sync bit	reg_id	resv_bits	sync bit	write data (32-bit)			
1	3-bit	001	0	data byte 3	data byte 2	data byte 1	data byte 0
0	1...3	4...6	7	8 15	16 23	24 31	32 39

Figure 24. RTMI stream write request datagram structure (RTMI_CRC_EN = 0_b)

When the RTMI CRC is enabled (RTMI_CRC_EN bit of UART.CONTROL register), the UART interface expects an extra CRC byte for the stream write request, as shown in *Figure 25*.

STREAM WRITE REQUEST DATAGRAM (WITH CRC)								
byte 0				byte 1	byte 2	byte 3	byte 4	byte 5
sync bit	reg_id	resv_bits	sync bit	write data (32-bit)				8-bit CRC
1	3-bit	001	0	data byte 3	data byte 2	data byte 1	data byte 0	checksum
0	1...3	4...6	7	8 15	16 23	24 31	32 39	40 47

Figure 25. RTMI stream write request datagram structure (RTMI_CRC_EN = 1_b)

UART Timeout

The UART interface in the TMC6460 also provides a timeout mechanism that measures the UART_RXD signal idle time between consecutive bytes of a datagram. If the idle time is longer than the configured limit, then the RX_IDLE_TIMEOUT_EVENT bit of the UART.EVENTS register is set. The timeout mechanism is enabled by default.

The timeout is defined by a counter based on the system clock f_{SYS}. The limit for the counter can be defined through the LIMIT field of the UART.TIMEOUT register. The timeout can be calculated through the following formula:

$$timeout = \frac{LIMIT}{f_{SYS}} \qquad with\ f_{SYS} = 60\ MHz$$

Like this, the reset value of LIMIT = 65535 results in the default timeout of about 1.1ms.

An additional 4096 pre-divider of the system clock can be enabled by setting the TIMEOUT_PRE_DIVIDER_EN bit in the UART.CONTROL register. If the pre-divider is enabled, then the timeout can be calculated with:

$$timeout = \frac{LIMIT \times 4096}{f_{SYS}} \qquad with\ f_{SYS} = 60\ MHz$$

When the TIMEOUT_PRE_DIVIDER_EN bit is set, the lowest possible timeout of 68.26µs results when LIMIT = 1_b. The timeout feature is disabled if LIMIT is set to 0.

UART Failure Monitoring

This section explains possible failure events that can be found in the UART.EVENTS register.

- UART_SYNC_FAIL_EVENT
  - If autobaud mode is enabled:
    - This bit is set when any disruption of the datagram is detected, assuming the sync bits were correctly received.
    - This bit is set when the received datagram sync bits are not correct. Refer to datagram structures above.
  - If autobaud mode is disabled:
    - This bit is set when the received datagram sync bits are not correct. Refer to datagram structures above.
- RX_IDLE_TIMEOUT_EVENT
  - This bit is set if the UART_RXD pin is idle between consecutive bytes of a datagram for more than specified timeout period (about 1.1ms by default).
- INVALID_START_BIT_EVENT
  - This bit is set if the start bit of any byte is corrupted.
- INVALID_STOP_BIT_EVENT
  - This bit is set if the stop bit of any byte is corrupted.
- INVALID_CRC_BIT_EVENT
  - If CRC is enabled, an extra byte containing the externally calculated CRC value is expected.
  - This bit is set if the provided and the internally calculated CRC values are not the same.

All these events can be cleared by writing a 1_b into the corresponding event bits. Datagrams that result in any of these events being triggered are discarded, and therefore not processed.

The signal for all these events, including the NOISY_DATA_EVENT explained in the next section, are combined into the UART_FAIL_STATUS bit of CHIP.STATUS_FLAGS register and the corresponding UART_FAIL_EVENT bit of CHIP.EVENTS register. The UART_FAIL_STATUS bit remains active for as long any of the individual event bits are set. Similarly, all the individual event bits must be cleared before clearing the UART_FAIL_EVENT bit.

When the UART is not responsive, the UART_RXD pin must be set high for at least 70ms to reset the UART module.

UART Incoming Data Filtering

The UART interface of the TMC6460 has a filtering feature to mitigate the effects of noise or timing variations. To enable it, set the RX_FILTER_EN bitfield of the UART.CONTROL register. If this feature is enabled, the received data at UART_RXD is sampled three times per bit period. The final bit value is determined using a majority voting mechanism, where two out of the three read values define the final bit value.

If the sampled value changes within a bit period (e.g. 001_b, 101_b, etc.), then the NOISY_DATA_EVENT bit from the UART.EVENTS register is set. This event serves as a warning only, datagrams that result in this event are still processed.

This feature is disabled by default. When disabled, the RXD signal is sampled only once at the center of each bit period.

UART CRC Configuration and Calculation

CRC8 calculation is used to detect accidental changes in the UART datagram requests and responses. By default, CRC is disabled for the UART interface. There are two bitfields in the UART.CONTROL register to enable CRC evaluation and generation for UART datagrams:

- Setting the NORMAL_CRC_EN bit results in:
  - Enabling CRC evaluation for read request datagrams sent to the TMC6460.
  - CRC generation for read response datagrams sent from the TMC6460.
  - Enabling CRC evaluation of regular write request datagrams sent to TMC6460.
- Setting the RTMI_CRC_EN bit results in:
  - Enabling CRC evaluation of RTMI stream write request datagrams sent to the TMC6460.
  - CRC generation for write response datagrams sent from the TMC6460.
  - CRC generation for RTMI channel response datagrams sent from the TMC6460.

The last byte of the UART datagram holds the CRC value calculated from all preceding bytes. If the received CRC does not match the internally calculated CRC, the request is ignored, and no response is sent.

The checksum for CRC-8 calculations uses the polynomial $P(x) = x^8 + x^4 + x^3 + \ x^2 + 1$ which corresponds to the binary pattern 100011101_b. An 8-bit CRC polynomial is represented with nine bits because it has nine terms. To form the binary representation, each term in the polynomial maps to a bit position: if the term exists, that bit is set to 1. For this polynomial, the terms x^8, x^4, x^3, x^2, and x^0 are present, so bits 8, 4, 3, 2, and 0 are set to 1.

Since each byte in the UART datagram is transmitted with its least significant bit first, the CRC calculation must use the bytes in reversed bit order so that the most significant bit is placed first.

1. Each byte must be reversed. For example, 0x4A (01001010_b) becomes 01010010_b. After reversing all datagram bytes, concatenate them and append eight zero bits for the CRC calculation.

```
Sync/Address[9:8]   Address[7:0]   Data
01010010            00000010       00000000 00000000 00000000 10000000 00000000
```

2. Perform a bitwise XOR (^) operation between the polynomial and the input data. Begin by aligning the leftmost 1 bit of the polynomial with the leftmost 1 bit of the input. Apply XOR across all bits covered by the polynomial, leaving any bits to the right unchanged. Repeat this process, shifting the polynomial to the right as needed, until all original input bits (before the appended zeros) have been reduced to zero. The sequence below illustrates this step:

XOR Steps 1…10

```
 01010010100000000000000000000000000000001000000000000000
^ 100011101
=00010101110000000000000000000000000000001000000000000000
^   100011101
=00000100000100000000000000000000000000001000000000000000
^     100011101
=00000000011001000000000000000000000000001000000000000000
^         100011101
=00000000001000110100000000000000000000001000000000000000
^          100011101
=00000000000000001110000000000000000000001000000000000000
^                100011101
=00000000000000000110111010000000000000001000000000000000
^                 100011101
=00000000000000000010100111000000000000001000000000000000
^                  100011101
=00000000000000000000101001100000000000001000000000000000
^                    100011101
=00000000000000000000001010001000000000001000000000000000
^                      100011101
=00000000000000000000000010110010000000001000000000000000
```

(*resume on top of right side*)

XOR Steps 11…20

```
 00000000000000000000000010110010000000001000000000000000
^                        100011101
=00000000000000000000000000111100100000001000000000000000
^                          100011101
=00000000000000000000000000011111001000001000000000000000
^                           100011101
=00000000000000000000000000001110111100001000000000000000
^                            100011101
=00000000000000000000000000000110000110001000000000000000
^                             100011101
=00000000000000000000000000000010011011001000000000000000
^                              100011101
=00000000000000000000000000000000010101101000000000000000
^                                 100011101
=00000000000000000000000000000000000100011100000000000000
^                                   100011101
=00000000000000000000000000000000000000000001000000000000
^                                           100011101
=00000000000000000000000000000000000000000000000111010000
^                                               100011101
=00000000000000000000000000000000000000000000000011001101
```

3. Take the remaining 8 bits, with the most significant bit first, which represents the CRC value. In this example, the resulting CRC value is 11001101 (0xCD).

Pin Configurations

The TMC6460 offers a flexible pin configuration scheme to adapt to different systems and layout constraints. Refer to the *Pin Descriptions* table for an overview of the different pin configurations. The pin configuration scheme offers choice of input/output pull-up/pull-down resistors for several pins, a voltage divider for the analog inputs, as well as pre-processing options such as input level inversion and input noise filtering.

Digital Input Inversion

The pre-processing block provides the ability to invert some digital input signals. Input signal inversion can be useful, for example, to get the desired logic from reference switches, or to get inverted logic from a digital feedback system. The following input pins can have their signals inverted by using the corresponding INV_PIN_x bits (with e.g. x = ENC_A) of the CHIP.IO_CONFIG register:

- ABN encoder pins ENC_A, ENC_B and ENC_N,
- Hall sensor pins HALL_U, HALL_V, HALL_W,
- Reference switch pins REF_L and REF_R,
- SCK and SDI pins (only applicable when using an alternate function and not the default SPI function),
- UART pin RXD,
- Control input pin PWM_IN.

When used, the input signal inversion is applied to the corresponding pin signal directly, as long as the pin is used as a digital input. For example, if INV_PIN_ENC_A is set, then the input signal is inverted regardless of whether the ENC_A pin is used with its main function of incremental encoder input A or with an alternate function (e.g., digital Hall input U).

Care should be taken when reconfiguring this feature, as altering the logic of the UART RXD pin at runtime can cause unwanted interruptions of communication to the chip until reset.

Digital Input Filtering

The pre-processing block also provides the ability to use a filter for the digital input signals. The filter can be applied to the following inputs:

- ABN encoder pins ENC_A, ENC_B and ENC_N,

- Hall sensor pins HALL_U, HALL_V, HALL_W,
- Reference switch pins REF_L and REF_R,
- Control input pin PWM_IN.

Filters are configured block-wise, that is, one configuration for all the ENC_x pins (with x = A, B, N), one for all the HALL_x pins (with x = U, V, W), and so on. When used, input filters are applied on the corresponding IC pins directly, as long as the pin is used as a digital input. The only exception is on the HALL_W pin, which, if remapped as a Home reference switch, takes the filter configuration from the reference switches block.

Configuration is done by using the corresponding ENC_FLT, HALL_FLT, REF_FLT, and PWM_IN_FLT fields inside the CHIP.IO_CONFIG register. Each block can be configured to have no filtering or the following filter cutoff frequencies: 1MHz, 500kHz, 100kHz, with the set values of 00_b, 01_b, 10_b and 11_b, respectively.

The filter works like a majority voting filter with a variable window. The filter is used to remove spurious switching events from electrical noise. By default, the 1MHz filter is enabled for all interfaces mentioned above. The filters will remove some of the spurious events starting at their cutoff frequency and all events at about 15% over the stated frequency.

Analog Input Voltage Divider

An adjustable voltage divider can be set for the analog inputs of the TMC6460. Independent voltage divider values can be set for the pins on each of the VCC_IO and VCC_IOF reference voltages. To configure the voltage divider, use the corresponding ANA_DIV_VCCIO and ANA_DIV_VCCIOF fields of the CHIP.IO_CONFIG register. The possible divider options are: 4.5, 3.0, 2.2 and 1 (no division), with the set values of 00_b, 01_b, 10_b and 11_b, respectively. 4.5 is the default option for both ANA_DIV_VCCIO and ANA_DIV_VCCIOF fields.

See *ADC Configuration and Measurement* chapter for additional details on ADC configuration.

Pull-up / Pull-down Configuration

A pull-up or pull-down resistor can be configured individually for several pins of the TMC6460. Note that not all pins have both pull options. Refer to the *Electrical Characteristics* table to see the resistor's characteristics.

A pull-up resistor is available on the following pins:

- ABN encoder pins ENC_A, ENC_B and ENC_N,
- Hall sensor pins HALL_U, HALL_V, HALL_W,
- Reference switch pins REF_L and REF_R,
- SPI pins SDI, SCK, and SDO,
- UART pin UART_RXD,
- Control input pin PWM_IN,
- Control output pin FAULTN (only active if configured as open drain).

A pull-down resistor is available on the following pins:

- ABN encoder pins ENC_A, ENC_B and ENC_N,
- Hall sensor pins HALL_U, HALL_V, HALL_W,
- Reference switch pins REF_L and REF_R,
- SPI pins SDI and SCK,
- UART pin UART_RXD,
- Control input pin PWM_IN.

Pull-up / pull-down resistors are directly applied to the input pad. Hence, this manipulates pad levels before all digital adaptions presented in the previous sections. To enable this feature, set the corresponding PIN_x_PU or PIN_x_PD bits (with e.g. x = ENC_A) of the CHIP.IO_PU_PD register. If both pull-up and pull-down are enabled for a pin, neither of the internal pull resistors are connected.

Pin Remapping

Many of the pins on the TMC6460 can be remapped to fulfil different functions. The pin name simply describes the default pin mapping. For instance, the pin ENC_A (pin 14), has a default mapping as the A input of the 1st ABN encoder block. However, it can also be remapped, among other options, as the U input of the digital Hall sensor block.

All pin remapping is done through the PIN_x fields (with e.g. x = ENC_A) of the CHIP.IO_MATRIX register. To set an alternate function for a particular pin, simply set the value that corresponds to the desired function. For instance, PIN_PWM_IN = 01_b to set the AIN alternate function. Note, however, that not all alternate function values are valid. Values for unavailable / reserved options must not be set.

The following pins can be remapped with an alternate function:

- ABN encoder pins ENC_A, ENC_B and ENC_N,
- Hall sensor pins HALL_U, HALL_V and HALL_W,
- Reference switch pins REF_L and REF_R,
- SPI pins SDI, SDO and SCK (CSN pin cannot be remapped),
- UART pins UART_RXD and UART_TXD,
- Control input pin PWM_IN,
- Control output pin FAULTN.

Care should be taken when remapping the SPI or UART pins at runtime, as this could cause unwanted interruption of the communication with the IC.

The CHIP.INPUTS_RAW register contains the state of the raw pin inputs, as well as the potentially inverted and pre-processed signals for the different functional blocks. This allows verification of the mapping and preprocessing configurations. For example, the PIN_HALL_U_RAW bit reports the actual state at pin HALL_U (pin 17). Contrarily, the HALL_U bit represents the state of the U signal for the digital Hall block, signal which could come from either pin HALL_U or pin ENC_A (pin 14) when the appropriate alternate function is set.

Feedback Engine Pin Remapping

The feedback engine block offers the following interfaces:

- 1st ABN and 2nd ABN incremental encoders, allowing the usage of two incremental encoders concurrently,
- Digital Hall sensor,
- Analog Hall sensor / SinCos encoder using the integrated ADCs.

These interfaces can be used through pins ENC_x, and HALL_x (pins 14 - 19). Additionally, these pins give access to some of the I/O Controller inputs/outputs, the brake chopper output, as well as the home reference switch input. This is described in Table 5 and Table 6 (in these tables, functional groups are represented by the same color), from which the different combinations of used feedback interfaces can be derived. The feedback engine pins are all referenced to the VCC_IOF voltage supply.

Table 5. Feedback engine pin remapping

Pin no.	Name	Main Function	Alt. Func 1	Alt. Func 2	Alt. Func 3 (I/O Controller)	Type
14	ENC_A	Incremental encoder input A	Inverted analog sensor input U/X*	Digital Hall input U	DIRECT_OUT[0]	Input/Output + Analog input
15	ENC_B	Incremental encoder input B	Inverted analog sensor input V/N*	Digital Hall input V	DIRECT_IN[0]	Input + Analog input
16	ENC_N	Incremental encoder input N	Inverted analog sensor input W/Y*	Digital Hall input W	DIRECT_IN[1] / DIRECT_OUT[1]	Input/Output + Analog input
17	HALL_U	Digital Hall input U	Analog sensor input U/X	2nd incremental encoder input A	I2C SDA	Input/Output + Analog input
18	HALL_V	Digital Hall input V	Analog sensor input V/N	2nd incremental encoder input B	I2C SCL as push-pull	Input/Output + Analog input
19	HALL_W	Digital Hall input W	Analog sensor input W/Y	2nd incremental encoder input N	DIRECT_OUT[2]	Input + Analog input

(* for use with differential input)

Table 6. Additional Hall input pin remapping

Pin no.	Name	Main Function	Alt. Func 4	Alt. Func 5	Alt. Func 6	Alt. Func 7	Type
17	HALL_U	Digital Hall input U			Inverted incremental encoder input A, monitoring only*	Inverted incremental encoder input A*	Input/Output + Analog input
18	HALL_V	Digital Hall input V	Brake chopper output	I2C SCL as open drain	Inverted incremental encoder input B, monitoring only*	Inverted incremental encoder input B*	Input/Output + Analog input
19	HALL_W	Digital Hall input W	Home ref. switch		Inverted incremental encoder input N, monitoring only*	Inverted incremental encoder input N*	Input + Analog input

(* for use with differential input)

Refer to *Feedback Engine* chapter for more detailed information on the feedback engine usage.

SPI/SSI encoder input

SPI/SSI encoders are interfaced through the I/O Controller direct inputs/outputs. Feedback engine pins, as well as most of the remappable pins, can therefore be used for this type of encoder. Refer to the *Programmable I/O Controller* chapter and its *SPI_ENC* command for more details.

Digital differential ABN encoder input

The TMC6460 also provides support for digital differential ABN encoder input. In this case, the differential signal pairs must be provided as follows:

- Pin ENC_A and pin HALL_U are used as the non-inverted and inverted differential pair for input A.
- Pin ENC_B and pin HALL_V are used as the non-inverted and inverted differential pair for input B.
- Pin ENC_N and pin HALL_W are used as the non-inverted and inverted differential pair for input N.

Two different modes for the digital differential input are available, depending on the selected remapping option of the pins:

- Digital differential ABN encoder input, represented in *Figure 26*.
  - Active when the register CHIP.IO_MATRIX bitfields PIN_HALL_U, PIN_HALL_V, and PIN_HALL_W are set to 111_b and PIN_ENC_A, PIN_ENC_B and PIN_ENC_N are set to 00_b (their default function).
  - In this mode, the ABN encoder inputs are defined by their corresponding differential pair signals.
  - The value for an encoder input is not updated unless the differential input pair is valid.

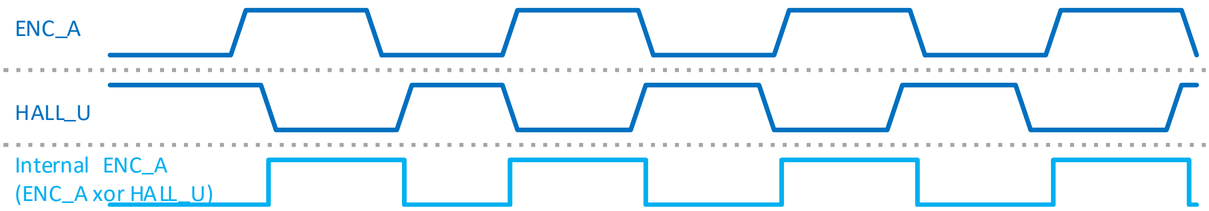


Figure 26. Digital differential ABN encoder input, example for ENC_A vs. HALL_U pins

- Digital differential ABN encoder monitoring only, represented in *Figure 27*.
  - Active when the register CHIP.IO_MATRIX bitfields PIN_HALL_U, PIN_HALL_V, and PIN_HALL_W of are set to 110_b and PIN_ENC_A, PIN_ENC_B and PIN_ENC_N are set to 00_b (their default function).
  - In this mode, the ABN encoder inputs are defined only by the non-inverted signals of the differential pair.
  - The inverted signals of the differential pair are used for monitoring purposes only.
  - When any of the differential signal pairs is invalid, the DIFF_ENC_FAIL_STATUS bit of the CHIP.STATUS_FLAGS register is set. Additionally, the INVALID_DIFF_ENC_x (with x = A, B, N) bits of the CHIP.OUTPUTS_RAW register show exactly which differential input is invalid.
  - An invalid differential signal results when the two signals in a pair have the same value. The value in DIFF_DELAY field of the CHIP.IO_CONFIG register defines the number of f_{SYS} clock cycles to wait after the non-inverted signal changes before invalid signals are registered. This grants some time for the signal values to settle. For example, the default value of 7 results in a waiting time of 7 / f_{SYS} = 117ns after the first signal changes.

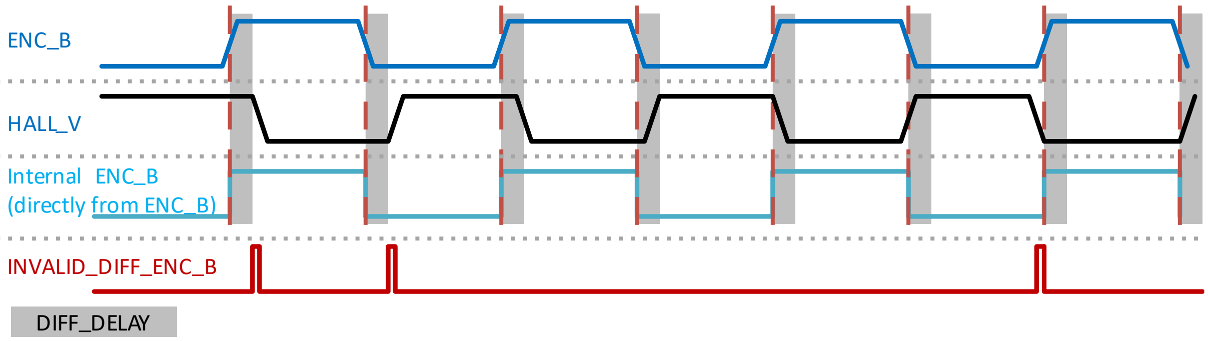


Figure 27. Digital differential ABN encoder monitoring, example for ENC_B vs. HALL_V pins, DIFF_DELAY is active after ENC_B switches

Reference Switches Pin Remapping

Left and right reference switch inputs are available by default on pins REF_L and REF_R. An additional home reference switch input is available, which can be mapped to REF_L, REF_R or HALL_W pins. Reference switch inputs are referenced to the VCC_IOF voltage supply.

These pins also give access through their alternate functions to the brake chopper output and to some of the I/O Controller inputs/outputs as described in *Table 7*.

Table 7. Reference switches pin remapping

Pin no.	Name	Main Function	Alt. Func 1	Alt. Func 2	Alt. Func 3 (I/O Controller)	Type	Voltage
10	REF_L	Left reference switch	Home reference switch		DIRECT_IN[0]	Input	VCC_IOF
11	REF_R	Right reference switch	Home reference switch	Brake chopper output	DIRECT_OUT[1]	Input/Output	VCC_IOF

The reference switch functionality is part of the ramp generator block. Refer to *Ramp Generator* chapter for more details.

Communication Interfaces Pin Remapping

The communication interface pins can also be remapped to other peripherals. This is especially useful if only one of UART or SPI are used to communicate with the TMC6460. Communication pins are referenced to the VCC_IO voltage supply.

SPI pins give alternative access to the brake chopper output and to some of the I/O Controller inputs/outputs as described in *Table 8*. Communication through SPI is not possible if any of their pins is used with an alternate function. CSN pin must not be asserted if SPI communication is not used.

Table 8. SPI pin remapping

Pin no.	Name	Main Function	Alt. Func 1	Alt. Func 2	Alt. Func 3 (I/O Controller)	Type	Voltage
1	CSN	Chip Select	(no alternate functions, keep at high level if SPI is not used)			Input	VCC_IO
36	SCK	Serial Clock			DIRECT_IN[1]	Input	VCC_IO
37	SDI	Serial Data In	DIR_IN input		DIRECT_IN[0]	Input	VCC_IO
38	SDO	Serial Data Out	Brake chopper output		DIRECT_OUT[0]	Output	VCC_IO

UART pins offer similar alternatives as described in *Table 9*.

Table 9. UART pin remapping

Pin no.	Name	Main Function	Alt. Func 1	Alt. Func 2 (I/O Controller)	Alt. Func 3 (I/O Controller)	Type	Voltage
8	UART_TXD	UART Data Output	Brake chopper output	DIRECT_OUT[0]	DIRECT_OUT[1]	Output	VCC_IO
9	UART_RXD	UART Data Input	DIR_IN input		DIRECT_IN[0]	Input	VCC_IO

Care must be taken when remapping the communication pins at runtime, as this could cause unwanted interruption of the communication with the IC.

Control Signals Pin Remapping

Control signals include a PWM input, driver enable input and fault output. Control signals are referenced to the VCC_IO voltage supply. It is also possible to reconfigure the function of some of the control signal pins, as described in *Table 10*.

Table 10. Control pin remapping

Pin no.	Name	Main Function	Alt. Func 1	Alt. Func 2	Alt. Func 3 (I/O Controller)	Type	Voltage
12	PWM_IN	PWM control input	AIN Analog control input		DIRECT_IN[1]	Digital input + Analog input	VCC_IO
34	DRV_EN	Driver enable	(no alternate functions available)			Input	VCC_IO
35	FAULTN	Open drain fault output	Open drain interrupt / event output	Push-pull fault output	Push-pull interrupt / event output	Output	VCC_IO

Two distinct cases can be distinguished for the FAULTN pin: fault output and interrupt output. The main difference being that fault output reflects the actual status of the selected flags (CHIP.STATUS_FLAGS register), while the interrupt output reflects the latched value of the selected flags (CHIP.EVENTS register), allowing to verify when a fault condition occurred, even if the condition is not currently active. In both cases, the output can be configured as open drain or push-pull. More details on the flag selection mask for the FAULTN pin can be found in *FAULTN Pin* section.

Input Configuration Diagrams

This section depicts the different input configurations that can be achieved for particular applications to summarize the prior *Pin Configurations* and *Pin Remapping* chapters.

In *Figure 28*, the different options to apply for the feedback signals of both ABN and ABN2 are displayed.

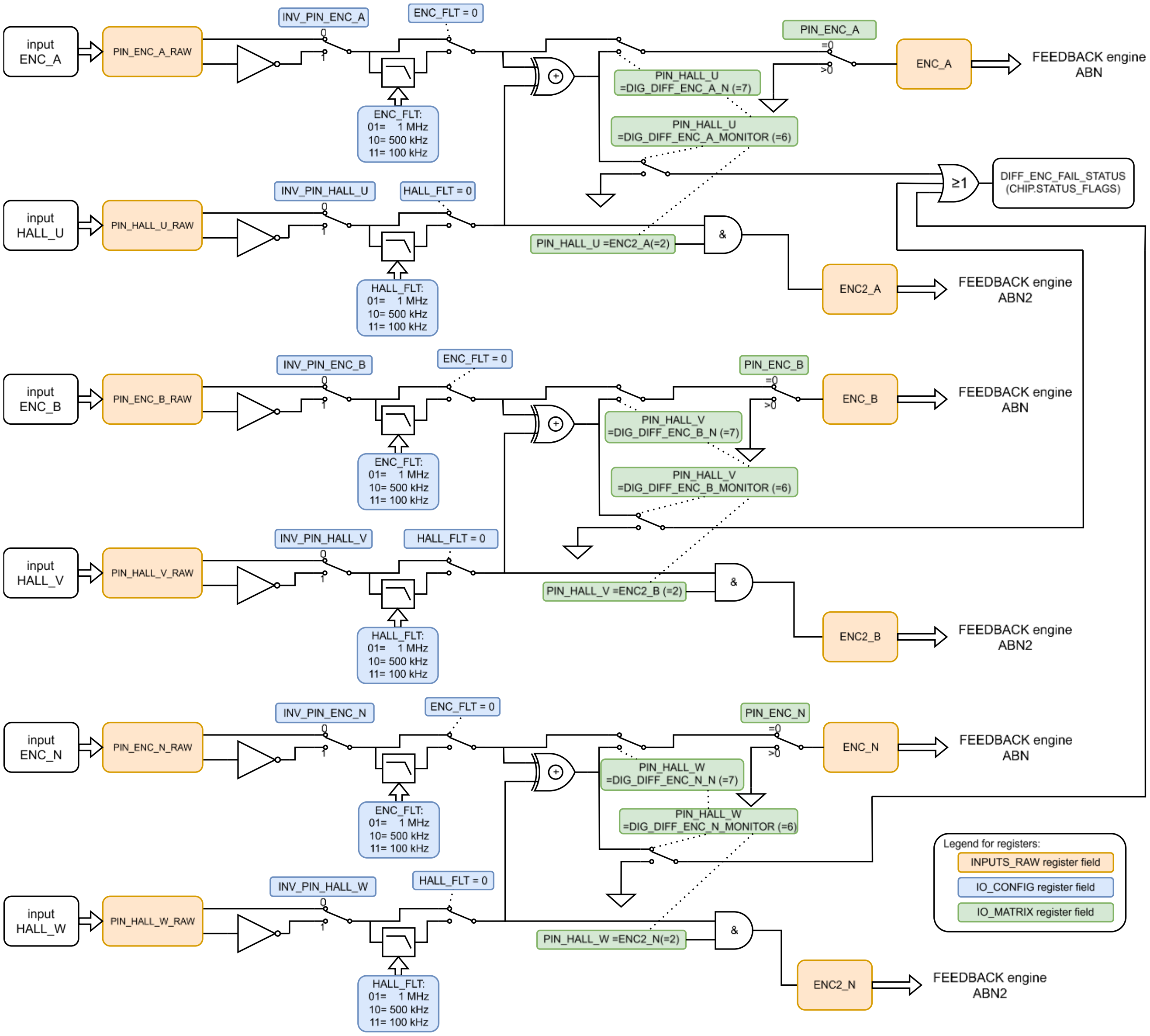


Figure 28. Encoder feedback configuration options

Figure 29 depicts the different options to apply for the feedback signals of the digital Hall. The additional HALL_MAP field can be found in the HALL.MAP_CONFIG register.

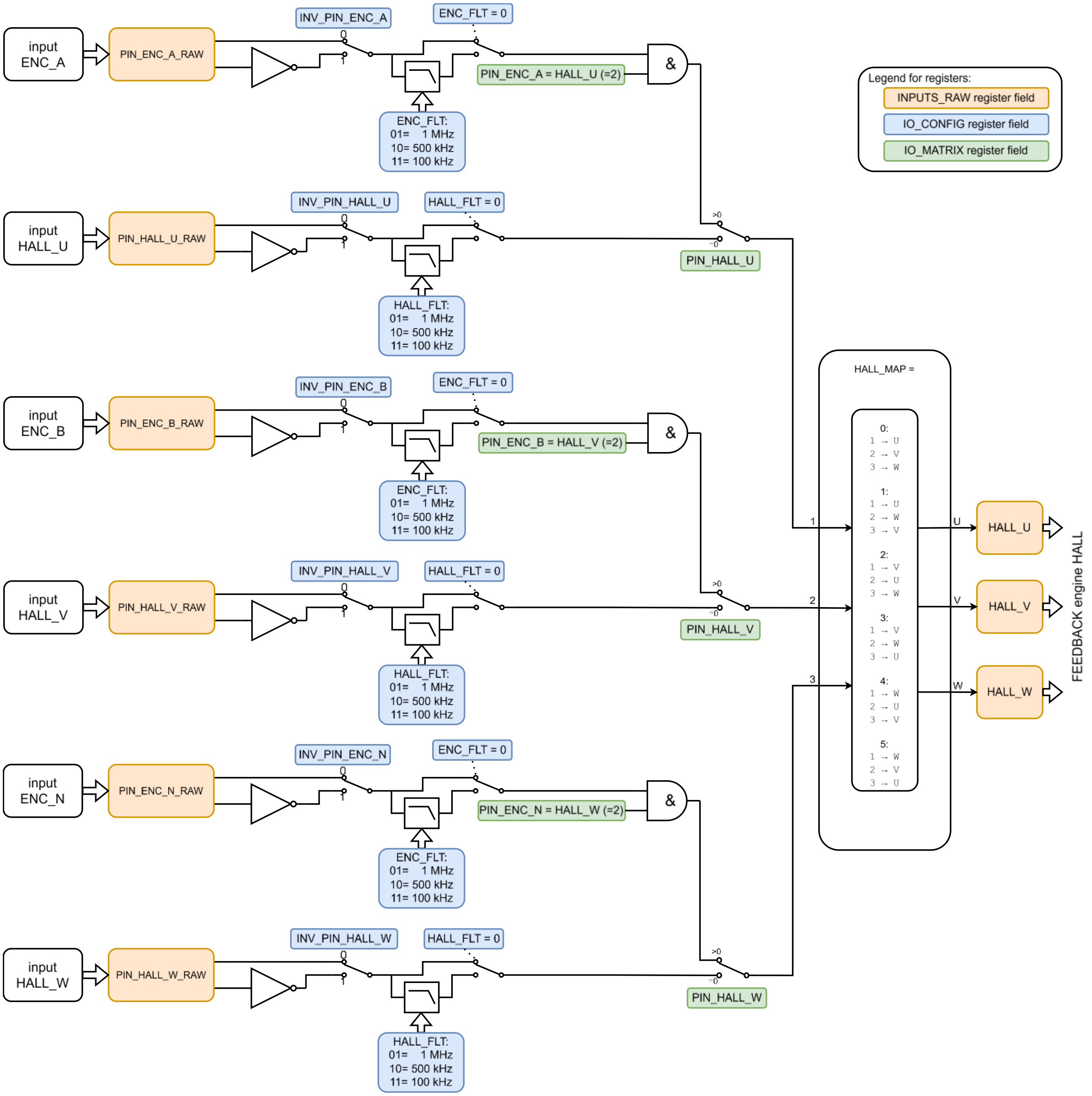


Figure 29. Digital Hall feedback configuration options

In *Figure 30* the different options for the analog sensor are depicted. Furthermore, the additional analog input option from PWM_IN input pin is shown. ANA_DIV_VCCIO and ANA_DIV_VCCIOF fields are part of the CHIP.IO_CONFIG register.

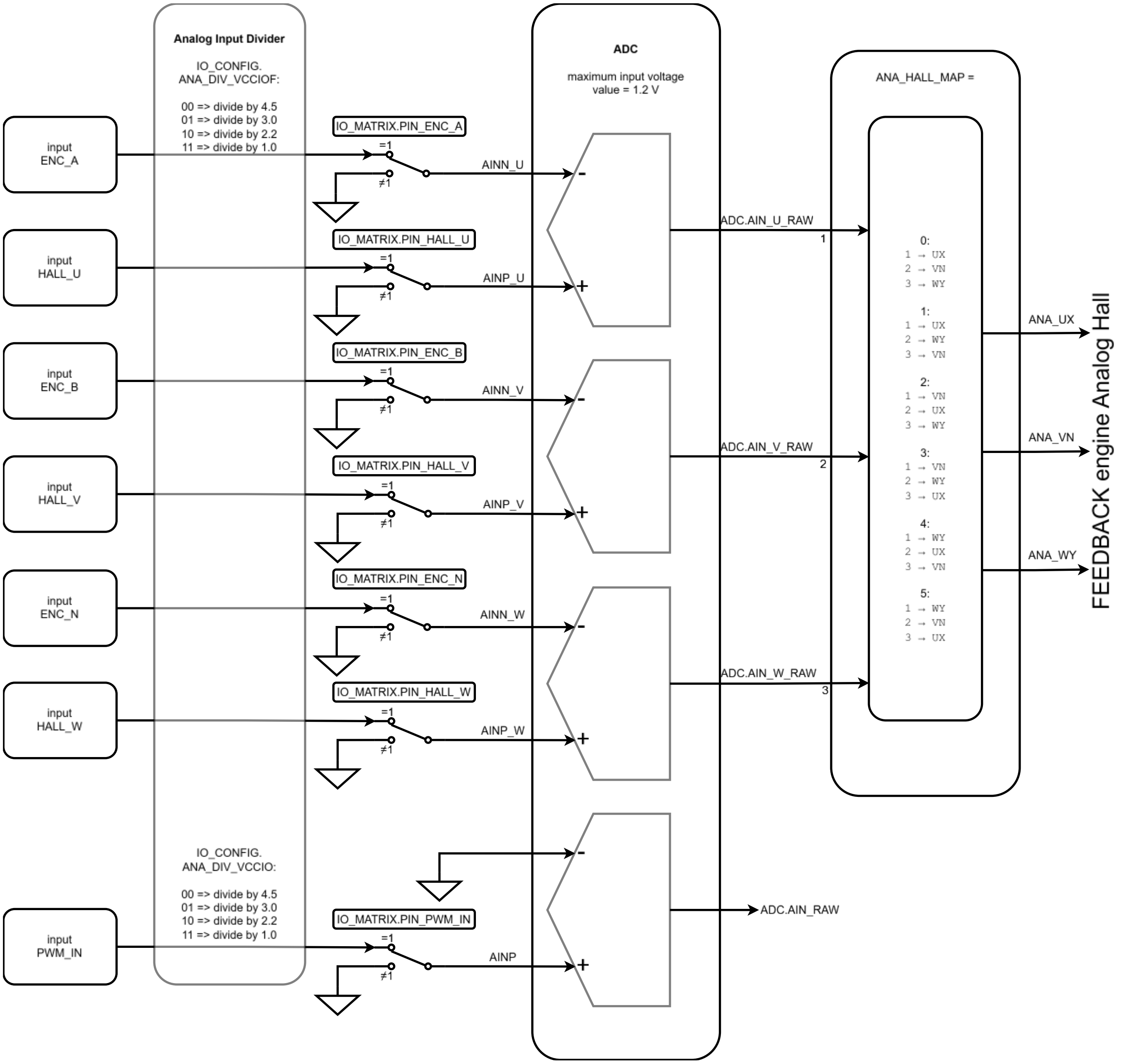


Figure 30. Analog Hall feedback and analog input configuration options

Figure 31 depicts the configuration options for the reference switch input pins. By setting either of the USE_PIN_REF_L_AS_REF_R or USE_PIN_REF_R_AS_REF_L bits of the CHIP.IO_CONFIG, a single input pin can be assigned to both internal reference signals. If both fields are activated, neither option is active.

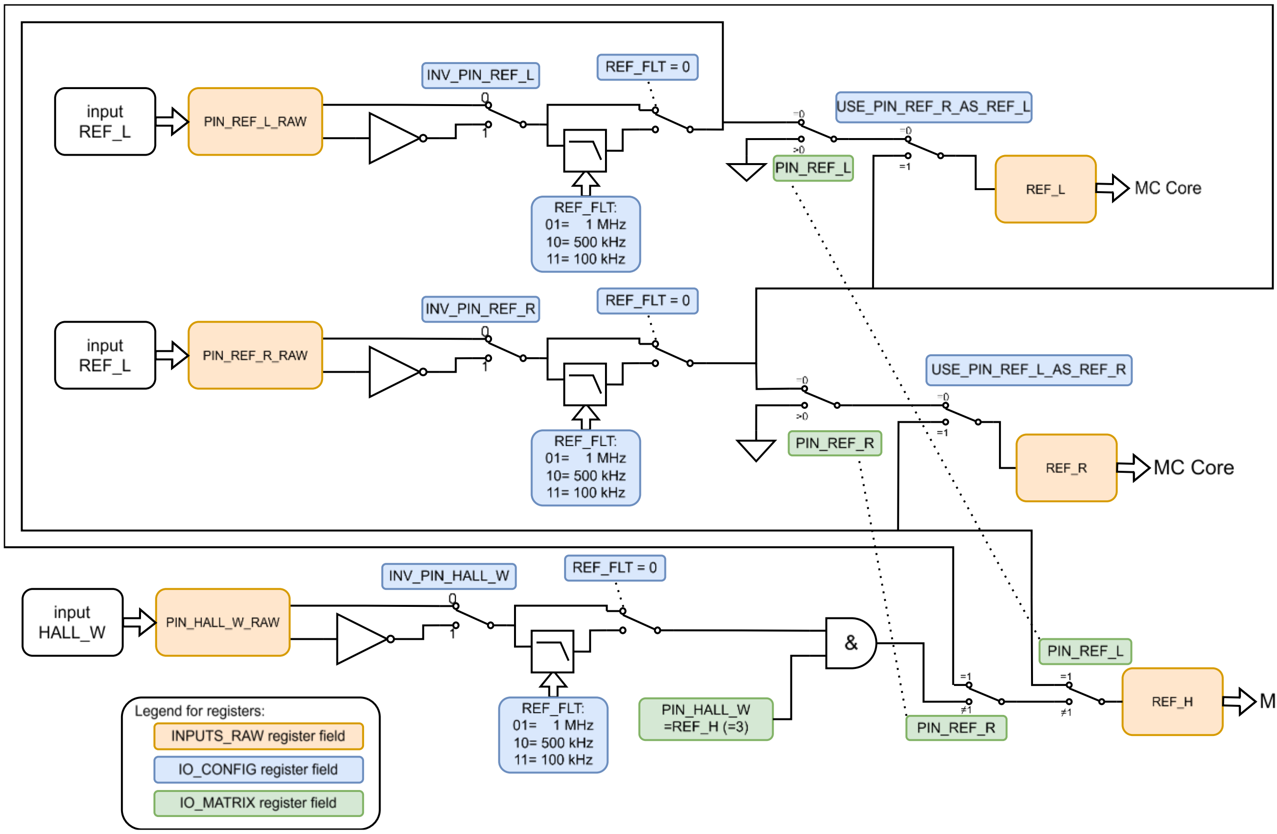


Figure 31. Reference Switch input configuration options

ADC Configuration and Measurement

The ADC engine operates with the system clock f_{SYS} = 60MHz. Its configuration is automatically executed during power-up, when the PLL is reconfigured or when the I/O Controller is reset. In these cases, no additional action is required. If the USE_INTERNAL_R_REF bit of register MCC_CONFIG.GDRV is changed after the ADC configuration was carried out, it is recommended to reissue the autoconfiguration of the ADC by manually resetting the I/O Controller (RESET bit of the IO_CONTROLLER.CONTROL register).

The ADC engine automatically measures the current for all three motor phases. Only the low side resistance and current sense amplifier gain settings need to be selected according to the application. Refer to the Current Acquisition section for guidance on how to find the correct settings.

As mentioned in the Analog Input Voltage Divider section, each of the IO voltage supply domains has an independent voltage divider for its analog inputs. Table 11 shows the different divisor values DIV_VCCIO and DIV_VCCIOF according to the configured value in ANA_DIV_VCCIO and ANA_DIV_VCCIOF fields of the CHIP.IO_CONFIG register.

A divisor value should be chosen in accordance with the estimated maximum voltage at the input pin so that the input voltage at the ADC (the divided voltage) does not exceed 1.25V. The maximum input voltage for the different divisor options is also shown in Table 11.

Table 11. Analog input divider options

ANA_DIV_VCCIO / ANA_DIV_VCCIOF	DIV_VCCIO / DIV_VCCIOF	Maximum Voltage Value for VCC_IO / VCC_IOF Analog Inputs
00 (default)	4.5	5.5
01	3.0	3.6
10	2.2	2.7
11	1.0	1.2

00_b is the default option for both ANA_DIV_VCCIO and ANA_DIV_VCCIOF fields.

Angle Feedback ADC Measurements

Further ADC measurements can be used to interface analog sensors that provide motor angle feedback. These sensors can be connected to the feedback engine pins HALL_U, HALL_V and HALL_W, and optionally to pins ENC_A, ENC_B and ENC_N if differential input is needed, all with some alternate function set. These ADC measurements are therefore referenced to the VCC_IOF supply. *Table 12* illustrates the source for the angle feedback ADC measurements according to the pin alternate functions set in CHIP.IO_MATRIX register.

Table 12. Calculation options for analog encoder input values

PIN_ENC_A PIN_HALL_U	= AINN_U	≠ AINN_U
= AINP_U	AINU = V(HALL_U vs. ENC_A)	AINU = V(HALL_U vs. GND)
≠ AINP_U	AINU = 0	AINU = 0
PIN_ENC_B PIN_HALL_V	**= AINN_V**	**≠ AINN_V**
= AINP_V	AINV = V(HALL_V vs. ENC_B)	AINV = V(HALL_V vs. GND)
≠ AINP_V	AINV = 0	AINV = 0
PIN_ENC_N PIN_HALL_W	**= AINN_W**	**≠ AINN_W**
= AINP_W	AINW = V(HALL_W vs. ENC_N)	AINW = V(HALL_W vs. GND)
≠ AINP_W	AINW = 0	AINW = 0

Three ADC measurements are provided from the feedback engine pins, their raw value can be read from AIN_U and AIN_V fields of AIN_V_AIN_U_RAW register, and AIN_W field of AIN_AIN_W_RAW register. All AIN_x fields (with x = U, V, W) hold 16-bit signed values that correlate to the measured input voltage at the ADC through the following equations:

$$AIN\_x = \frac{V_{AnalogSensor} \times 2^{15}}{DIV\_VCCIOF \times 1.25\ [V]} \qquad V_{AnalogSensor} = AIN\_x \times \frac{1.25\ [V]}{2^{15}} \times DIV\_VCCIOF$$

Divisor values DIV_VCCIOF can be derived from *Table 11*.

Additional ADC Measurements

Two further ADC measurements are referenced to the VCC_IO supply:

- AIN is an external analog measurement available at pin PWM_IN. The analog value at this pin is automatically calculated if PIN_PWM_IN = AIN on the CHIP.IO_MATRIX register is selected. The calculated value can be read from the AIN bitfield of the ADC.AIN_AIN_W_RAW register. This field is also a 16-bit signed value that uses this formula:

$$AIN = \frac{V_{AnalogPWM\_IN} \times 2^{15}}{DIV\_VCCIO \times 1.25\ [V]} \qquad V_{AnalogPWM\_IN} = AIN \times \frac{1.25\ [V]}{2^{15}} \times DIV\_VCCIO$$

- TEMP_EXT is an external analog measurement available at pin TEMP. An external temperature sensor or a thermistor-resistor arrangement can be interfaced to allow monitoring of, e.g. the motor temperature. The analog value at this pin is always calculated. This value can be read from TEMP_EXT bitfield of the ADC.TEMP_RAW register. This field is also a 16-bit signed value that uses the following formula:

$$TEMP\_EXT = \frac{V_{AnalogTEMP} \times 2^{15}}{DIV\_VCCIO \times 1.25\ [V]} \qquad V_{AnalogTEMP} = TEMP\_EXT \times \frac{1.25\ [V]}{2^{15}} \times DIV\_VCCIO$$

Divisor values DIV_VCCIO can be derived from *Table 11*.

Two additional analog values are also always calculated:

- TEMP_INT field of the ADC.TEMP_RAW register represents the internal temperature T_J. This is a 16-bit signed value that uses the following formula:

$$T_J = \left(TEMP\_INT \times \frac{1.25\ [V]}{2^{15}} - 0.622[V]\right) \times \frac{1250}{3}\left[\frac{°C}{V}\right]$$

- Finally, VM field of the ADC.VM_I3_RAW register represents the internally measured voltage value at pin VS. This is a 16-bit signed value that uses the following formula:

$$V_{VM} = VM \times \frac{1.25\ [V]}{2^{15}} \times \frac{287}{9}$$

Motor phase currents and the AIN analog input are measured every PWM cycle. All other analog values (TEMP_EXT, TEMP_INT and VM) are calculated every PWM cycle for f_{PWM} ≤ 50kHz, and every third PWM cycle for f_{PWM} > 50kHz.

If any of the raw values in AIN_U, AIN_V, AIN_W or AIN are clipped because the voltage at their inputs exceeds the ADC valid range, then the corresponding event bits AIN_U_CLIPPED, AIN_V_CLIPPED, AIN_W_CLIPPED, and AIN_CLIPPED of the ADC.STATUS register are set. These bits are merged into the AIN_CLIPPED_STATUS bit in the CHIP.STATUS_FLAGS register and the corresponding AIN_CLIPPED_EVENT bit in the CHIP.EVENTS register.

Similarly, if the raw values in VM, TEMP_EXT or TEMP_INT are clipped, then the corresponding event bits VM_CLIPPED, TEMP_EXT_CLIPPED, and TEMP_INT_CLIPPED of the ADC.STATUS register are set. These three events are merged into the VM_TEMP_CLIPPED_STATUS bit of the CHIP.STATUS_FLAGS register.

The relation of the merged status and event bits as described above is depicted in *Table 13*.

Table 13. ADC Clipping Event and Flags Overview

Raw ADC Input Register Field	Correpsonding Event Bit in ADC.STATUS Register	Merged Status Flag in CHIP.STATUS_FLAGS	Merged Event bit in CHIP.EVENTS
AIN_U	AIN_U_CLIPPED	AIN_CLIPPED_STATUS	AIN_CLIPPED_EVENT
AIN_V	AIN_V_CLIPPED		
AIN_W	AIN_W_CLIPPED		
AIN	AIN_CLIPPED		
VM	VM_CLIPPED	VM_TEMP_CLIPPED_STATUS	VM_TEMP_CLIPPED_EVENT
TEMP_INT	TEMP_INT_CLIPPED		
TEMP_EXT	TEMP_EXT_CLIPPED		

Note that the particular event bits in ADC.STATUS must be cleared before the corresponding merged event bit in CHIP.EVENTS can be cleared.

Motor Angle Feedback Configuration

Several different types of angle feedback are supported by the TMC6460. Namely, ABN encoders, digital and analog Hall sensors, SinCos absolute encoders and SPI/SSI absolute encoders. For each of these feedback types, the right configuration must be carried out before they can be used as an angle source for the feedback engine.

This section provides the configuration steps needed for each of these feedback types.

Incremental ABN Encoder

Incremental encoders provide two phase shifted pulse signals A and B, from which the amount and direction of movement can be determined. Some incremental encoders have an additional null position signal N (sometimes called zero pulse signal Z), from which an absolute positional reference can be derived. An incremental encoder (referred from mow on as ABN encoder) is characterized by its number of lines or pulses per revolution (PPR). There are two edges (rising and falling) on each pulse, and there are two signals that provide pulses (A and B). In this way, the TMC6460 can register four counts per line of the ABN encoder. The number of counts per revolution is abbreviated as CPR and equals PPR × 4.

The TMC6460 has two ABN decoder modules, which means that two ABN encoders can be used simultaneously. The first encoder module, corresponding to the ABN register block, takes its inputs from pins ENC_A, ENC_B and ENC_N. Conversely, the second encoder module, corresponding to the ABN2 register block, takes its inputs from pins HALL_U, HALL_V and HALL_W (with their alternate function set to 010_b). Note that the two decoder blocks are independent, so it is not possible to mix the decoder's inputs, for example, by connecting a single encoder to pins ENC_A and HALL_V.

If both A and B input signals change in the same internal clock cycle, the INVALID_SIGNAL_EVENT of the ABN.EVENTS register is set (or ABN2.EVENTS if the 2nd ABN encoder is used). This serves as detection mechanism for hardware or connection issues of the ABN encoder, or for input signals that exceed the update rate of the TMC6460 capabilities. If this failure occurs, the decoder count is neither increased nor decreased for that particular switching, and regular operation is maintained afterwards. The two signals for invalid input are OR wired into the ABN_FAIL_STATUS bit of the CHIP.STATUS_FLAGS and latched into the corresponding ABN_FAIL_EVENT bit in the CHIP.EVENTS register. To clear ABN_FAIL_EVENT, both individual event bits must be cleared first.

Configuring the ABN decoder

The ABN decoder increments or decrements the value in ABN.COUNT register for every detected edge of the A and B signals according to the detected direction of movement. The counting direction can be changed by setting the INV_DIR bit of the ABN.CONFIG register. Additionally, when an N pulse is detected, the N_EVENT bit of the ABN.EVENTS register is automatically set, and the current COUNT value is latched into the ABN.COUNT_N_CAPTURE register.

To allow for a more precise detection of the N pulse for some encoders, a combined signal mode is provided. In this mode, the N pulse is counted only when A, B, and N signals are all simultaneously high, as opposed to being based on the N signal alone. To achieve this, the A, B or N signals may be inverted as described in *Pin Configurations* chapter. To use this mode, set the COMBINED_N bit of the ABN.CONFIG register. The signals for a typical ABN encoder and the resulting combined N pulse signal when using this feature are represented in *Figure 32*. When using this feature, care must be taken that all three signals actually show a simultaneous high level in the used encoder, this must be true as the encoder rotates in both directions. If this condition is not met, the N pulse will not be detected.

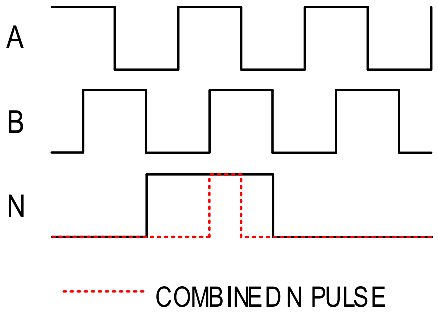


Figure 32. Typical ABN encoder signals and combined N pulse

For the decoder to work as intended, it needs a way to detect when a full revolution is completed. This can be done in two ways: by setting a reset value based on the encoder's CPR, or by using the encoder's N pulse as a reset signal.

To use the encoder's CPR as a count reset mechanism, set the CPR field in the ABN.CONFIG register to the maximum value the decoder will count before resetting back to zero. Therefore, write the CPR field with the value of cpr_encoder - 1. For example, if the connected ABN encoder has a resolution of 1024 lines (PPR = 1024), then write the value of CPR = (1024 × 4) - 1 = 4095. If this mechanism is not to be used, leave the CPR field with its reset value of 16777215.

Alternatively, the encoder's N signal can be used to reset the decoder's count value in ABN.COUNT. To use this mechanism, set the CLN bit of the ABN.CONFIG register. It is also possible to use the N pulse to write an arbitrary value to the COUNT register instead of resetting it. To do that, write the desired value in ABN.COUNT_N_WRITE (0 by default).

The configuration for the 2nd ABN encoder can be done in the same manner, by using registers in the ABN2 block instead, that is, registers ABN2.CONFIG, ABN2.COUNT, and so forth.

Digital Hall Sensor

Digital Hall sensors provide three digital signals corresponding to the three phases of the motor. From its signals, the current location of the motor among six distinct sectors can be determined. Each of which represents a 60° section of a full electrical revolution.

If all Hall input signals have the same value when measuring them, the INVALID_SIGNAL_EVENT bit in the HALL.DIG_EVENTS register is automatically set and the corresponding change of state is not processed. This serves as detection mechanism for hardware or connection issues of the Hall sensor. The signal for an invalid state is forwarded to the HALL_FAIL_STATUS bit of the CHIP.STATUS_FLAGS register and latched into the corresponding HALL_FAIL_EVENT bit in the CHIP.EVENTS register. To clear HALL_FAIL_EVENT, the INVALID_SIGNAL_EVENT bit must be cleared first.

Configuring the digital Hall sensor decoder

The digital Hall decoder takes three signals and compares their value against a lookup table to get the corresponding Hall sector. The current Hall sector represents the current position of the motor within six 60° sections of one electrical revolution. The Hall sector is represented by a value from 0 to 5, which can be read from the COUNT field in the HALL.DIG_COUNT register.

Assuming the pins are correctly configured for Hall sensor usage, the only configuration needed for the Hall decoder is the correct mapping of the Hall signals. This can be changed through the HALL_MAP field of the HALL.MAP_CONFIG register in combination with the pin input inversion as described in the *Pin Configurations* chapter. Changing the mapping of the signals can be used to compensate for a mismatch between the connection of the motor phases and the Hall sensor inputs. In other words, the remapping of the signals can be used to effectively apply an offset of the current Hall sector, or to effectively reverse the direction of the count. *Table 14* provides an overview of the possible Hall sector offsets and counting direction inversion according to the mapping and pin input inversion settings.

Table 14. Hall sensor sector offset and counting direction configuration

HALL_MAP[2:0]	Pin Input Inversion	HALL.DIGITAL_COUNT Resulting Sector Offset	HALL.DIGITAL_COUNT Resulting Counting Direction
000	Disabled	0°	Non-inverted
011	Enabled	60°	Non-inverted
100	Disabled	120°	Non-inverted
000	Enabled	180°	Non-inverted
011	Disabled	240°	Non-inverted
100	Enabled	300°	Non-inverted
001	Disabled	0°	Inverted
010	Enabled	60°	Inverted
101	Disabled	120°	Inverted
001	Enabled	180°	Inverted
010	Disabled	240°	Inverted
101	Enabled	300°	Inverted

In this table, the Pin Input Inversion column refers to the three pin inversion bits of the CHIP.IO_CONFIG register corresponding to the pins used for the Hall sensor. That is, the INV_PIN_HALL_x bits if using the HALL_x pins (with x = U, V, W) for the Hall sensor, or the INV_PIN_ENC_x bits (with x = A, B, N) if using the ENC_x pins.

Note that having the correct counting direction of the Hall decoder is crucial for correct motor commutation, as well as velocity and position tracking. However, the Hall sector offset as described above is only relevant for motor commutation, that is, it is only relevant for the alignment of the Hall sensor angle with the internal phi_e angle. This is described in more detail in the *Phi E Calculation* section.

Analog Hall Sensor

The analog sensor decoder accepts 3-phase (analog Hall) input, for which three sinusoid signals with a 120° offset between them are expected.

The analog sensor decoder provides the ability to compensate each of the analog inputs with an offset and scaling value. Each of the analog input values can be read both before and after compensation is applied. An arctangent calculation is performed with the compensated analog input values to derive the analog Hall angle.

Configuring the analog decoder for an analog Hall sensor

To configure the analog decoder for 3-phase (analog Hall) sensors, set ANA_MODE bitfield of the HALL.ANA_CONFIG register to 1_b, to the so-called UVW mode. Then, observe the following procedure using *Table 15* as a reference:

1. Offset values for each analog input must be set such that the compensated sinusoid is centered around zero when the motor is rotating. The offset is a signed value, and it is added to the raw input value before scaling.
2. Scale values for each analog input must be adjusted such that all three compensated sinusoids have the same amplitude. A scale value of 1024 represents a scaling of 1:1 from the raw ADC values. Negative scale values invert the raw analog input. When choosing scale values, the scaled ADC value must not exceed an absolute value of 21845 (two thirds of the signed 16-bit value).
3. It may be necessary to adjust the offsets again after the scaling has been changed.

Table 15. Analog Hall sensor configuration register fields

	U	V	W
Raw ADC value field	ADC.AIN_V_AIN_U_RAW:: AIN_U	ADC.AIN_V_AIN_U_RAW:: AIN_V	ADC.AIN_AIN_W_RAW:: AIN_W
Offset value field	HALL.ANA_UX_CONFIG:: UX_OFFSET	HALL.ANA_VN_CONFIG:: VN_OFFSET	HALL.ANA_WY_CONFIG:: WY_OFFSET
Scale value field	HALL.ANA_UX_CONFIG:: UX_SCALE	HALL.ANA_VN_CONFIG:: VN_SCALE	HALL.ANA_WY_CONFIG:: WY_SCALE
Compensated value field	HALL.ANA_UX_OUT:: UX_OUT	HALL.ANA_VN_OUT:: VN_OUT	HALL.ANA_WY_OUT:: WY_OUT

Register fields UX_OUT, VN_OUT, and WY_OUT represent the ADC values after applying offset and scaling.

An example plot comparing the raw ADC values with the compensated values can be found in *Figure 33*, where the following offset and scale values were used: UX_OFFSET = 15143, UX_SCALE = 2549, VN_OFFSET = 15108, VN_SCALE = 2501, WY_OFFSET = 15141, WY_SCALE = 2647.

The angle derived from the analog decoder can be read from the ANA_PHI field of the HALL.ANA_OUT register.

A sector offset and an inversion of the counting direction the analog Hall angle in ANA_PHI field can be achieved by setting an appropriate value in the ANA_HALL_MAP field of the HALL.MAP_CONFIG register in combination with analog input inversion as described in *Table 16*. Note that the mapping of the raw ADC values AIN_x (with x = U, V, W) to the analog sensor configuration and output values x_OFFSET, x_SCALE and x_OUT (with x = UX, VN, WY) as shown in *Table 15* is valid only when using the default value of ANA_HALL_MAP = 000_b. The different mapping options can be found on the field's description in the register map and on the right side in *Figure 30*.

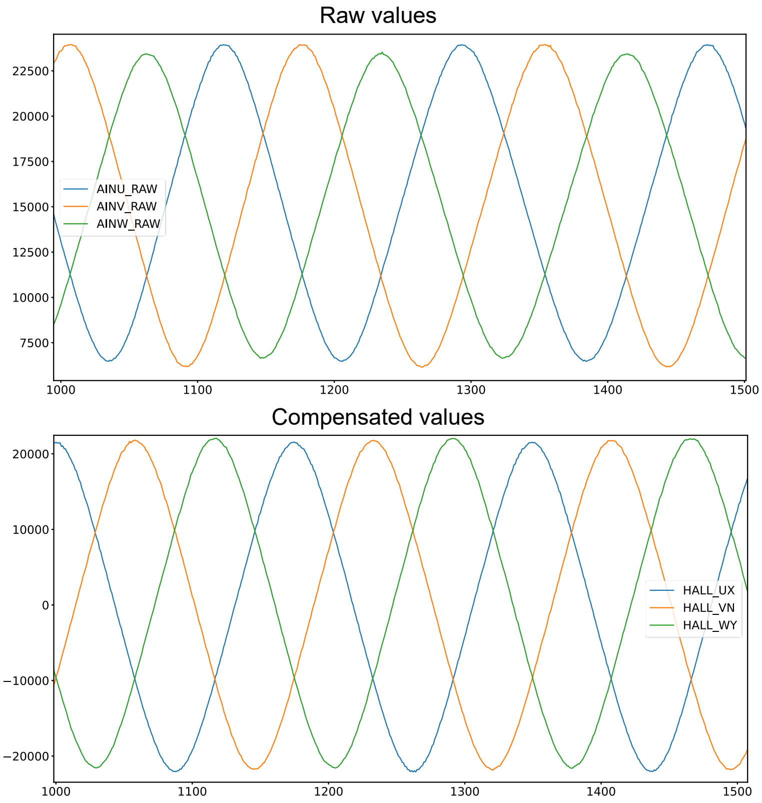


Figure 33. Scale and Offset Setup Example for an Analog 3-phase Encoder

Table 16. Analog Hall sensor sector offset and counting direction configuration

ANA_HALL_MAP[2:0]	Analog Input Inversion	HALL.ANA_OUT::ANA_PHI Resulting Angle Offset	HALL.ANA_OUT::ANA_PHI Resulting Counting Direction
000	Disabled	0°	Non-inverted
011	Enabled	60°	Non-inverted
100	Disabled	120°	Non-inverted
000	Enabled	180°	Non-inverted
011	Disabled	240°	Non-inverted
100	Enabled	300°	Non-inverted
001	Disabled	0°	Inverted
010	Enabled	60°	Inverted
101	Disabled	120°	Inverted
001	Enabled	180°	Inverted
010	Disabled	240°	Inverted
101	Enabled	300°	Inverted

In this table, the Analog Input Inversion column refers to using positive or negative values in the three UX_SCALE, VN_SCALE and WY_SCALE fields as described in *Table 15* and the corresponding procedure above it.

Note that having the correct counting direction of the analog sensor decoder is crucial for correct motor commutation, as well as velocity and position tracking. However, the angle offset as described above is only relevant for motor commutation, that is, it is only relevant for the alignment of the analog Hall sensor angle with the internal phi_e angle. This is described in more detail in the *Phi E Calculation* section.

Analog SinCos Encoder

The analog sensor decoder also accepts analog SinCos encoder input, for which two sinusoid signals with a 90° offset between them are expected.

The analog sensor decoder provides the ability to compensate each of the analog inputs with an offset and scaling value. Both analog input values, that is, before and after compensation, can be read from the register map for each of the inputs. An arctangent calculation is performed with the compensated analog input values to derive the encoder angle.

Configuring the analog decoder for a SinCos encoder

To configure the analog decoder for SinCos encoders, set ANA_MODE bitfield of the HALL.ANA_CONFIG register to 0_b, to the so-called XY mode.

Some SinCos encoders generate two sinusoid cycles per revolution of the motor. If using this type of encoder, set the TWO_CYCLE_MODE_EN bit of HALL.ANA_CONFIG register.

Similarly as for 3-phase sensors, offset and scale values must be defined so that the sinusoids are centered around zero and so that they have the same amplitude when the motor is rotating. Since this encoder type uses only two analog signals, only for the UX and WY inputs need to be configured in this manner. Use *Table 17* as a reference for configuration.

Table 17. Analog SinCos encoder configuration register fields

	X	Y
Raw ADC value field	ADC.AIN_V_AIN_U_RAW::AIN_U	ADC.AIN_AIN_W_RAW::AIN_W
Offset value field	HALL.ANA_UX_CONFIG::UX_OFFSET	HALL.ANA_WY_CONFIG::WY_OFFSET
Scale value field	HALL.ANA_UX_CONFIG::UX_SCALE	HALL.ANA_WY_CONFIG::WY_SCALE
Compensated value field	HALL.ANA_UX_OUT::UX_OUT	HALL.ANA_WY_OUT::WY_OUT

Register fields UX_OUT and WY_OUT represent the ADC values after applying offset and scaling.

The angle derived from the analog decoder can be read from the ANA_PHI field of the HALL.ANA_OUT register.

The ANA_HALL_MAP field of the HALL.MAP_CONFIG register or the analog input inversion can be used to invert the direction of counting of the angle in the ANA_PHI field. There are two ways to achieve this: either set a scale value with an opposite sign for either one of the UX_SCALE or WY_SCALE fields or set a value of ANA_HALL_MAP = 5. Note that the mapping of the raw ADC values AIN_x (with x = U, W) to the analog sensor configuration and output values x_OFFSET, x_SCALE and x_OUT (with x = UX, WY) as shown in *Table 17* is valid only when using the default value of ANA_HALL_MAP = 000_b. The different ANA_HALL_MAP mapping options can be found on the field's description in the register map and on the right side in *Figure 30*.

Since only two analog signals are used for standard SinCos encoders, it is possible to use the third analog input pin for a different purpose. For instance, by using pin HALL_V (which normally sources the AIN_V raw analog input) as a brake chopper output, or by using pin HALL_W (AIN_W raw analog input) as the home reference switch input or as the I/O Controller direct output 2. For those cases, it might be necessary to adjust the value of ANA_HALL_MAP so that the UX and WY analog sensor channels make use of the available analog input pins.

Configuring the optional N pulse for SinCos encoders

Some SinCos encoders provide an additional analog or digital signal that serves as a null position signal. If used, the additional signal must be connected to the VN analog input.

To use the analog sensor decoder N pulse feature, set the USE_N_PULSE bit of the HALL.ANA_CONFIG register. When enabled, the generated N pulse is used as a mechanism to detect the end of a cycle (as opposed to using the derived analog decoder angle value), this is true for encoders that generate two cycles per revolution. The generated N pulse is also used as trigger signal to latch the current analog decoder angle value (ANA_PHI) into ANA_PHI_N_CAPTURE field of the HALL.ANA_OUT register.

The value of the generated N pulse signal can be read in the N_PULSE_OUT bit of the HALL.ANA_CONFIG register.

To generate the N pulse, the appropriate threshold and polarity values must be set first. Do this through the N_PULSE_THRESHOLD and N_PULSE_POL fields of the HALL.ANA_CONFIG register. When N_PULSE_POL = 0_b (non-inverted), the generated N pulse signal is set high if the current VN input value is higher than the set threshold. Conversely, when N_PULSE_POL = 1_b (inverted), the generated N pulse signal is set high if the current VN input value is lower than the set threshold. Note that the comparison is done with the raw VN analog input value (ADC.AIN_V_AIN_U_RAW.AIN_V), not the offset and scale compensated one.

Finally, the N_PULSE_EDGE bit of the HALL.ANA_CONFIG register selects which edge of the generated N pulse (0_b for rising edge, 1_b for falling edge) is used to detect the end of a cycle and to trigger the capture of the derived angle.

SPI/SSI Encoder

SPI/SSI encoders are configured and interfaced through the I/O Controller. This feature makes use of the externally writable angle at FEEDBACK.PHI_EXT_x (with x = A, B) registers. Refer to the *Programmable I/O Controller* chapter and its *SPI_ENC* command for instruction on how to configure this type of encoders.

Feedback Engine

The feedback engine in the TMC6460 contains logic that is common to all angle feedback options. This includes 16-bit conversion, correction lookup table (LUT), extrapolation, phi_e calculation, velocity measurement and position tracking. A block diagram of the feedback engine is shown in *Figure 34*.

Two feedback channels are available, channel A and channel B. This provides the ability to use two angle feedback sources simultaneously, one for electrical commutation and the other for velocity and position measurement. Alternatively, a single angle feedback source can be used for both channels, where each channel is configured so that its output is more suitable for electrical commutation or velocity and position measurement. If the application allows it, it is also possible to use only one of the two available channels, for both commutation and velocity and position measurement.

Two 16-bit conversion modules are available, each of them can be configured to make use of any of the nine available feedback sources. The 16-bit conversion ensures that the correct scaling is used regardless of the selected feedback source. See the *16-bit Conversion* section below for more information on this module.

Note that only one instance of the correction LUT is available. The full table can be used for either channel A or channel B exclusively. Alternatively, each of the channels can use half of the table if the split mode is enabled. If the LUT is not enabled for a particular channel, the values from that channel will simply pass unchanged. See the *Correction Lookup Table* section below for more information on how to use and configure the correction LUT.

The extrapolation feature can be used for phi_e calculation. Similarly to the LUT, a single instance of the extrapolator is provided. However, only one of the channels can be used at a time as a source for the extrapolator. See *Phi Extrapolator* and *Phi E Calculation* sections for more information on this feature.

Note that there are two modules for velocity calculation, the period velocity meter and the frequency velocity meter. The two meters share their input, which can be configured to use either one of channel A or B. Even though both meters are active simultaneously, only the output of one of them can be selected as input to the velocity controller. See *Velocity Acquisition* sections for more information on this feature.

The position measurement can be selected to use either one of channel A or B. It is recommended that the same feedback channel is used for both velocity and position measurement, especially if the ramp generator is used for the application. See the *Position Acquisition* section for more detailed information.

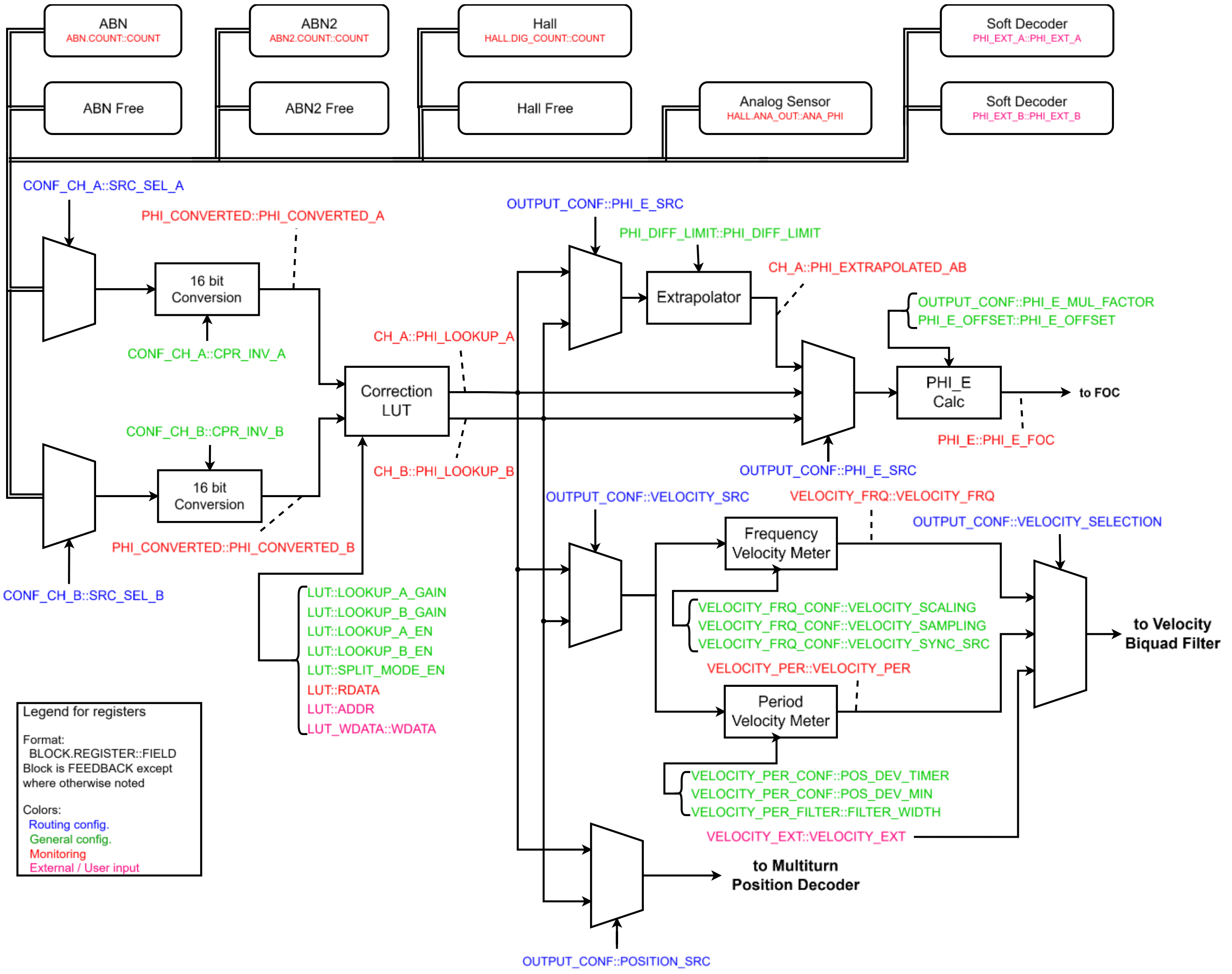


Figure 34. Feedback Engine overview, incl. register fields

Configuration Procedure

The following sections contain an overview of the configuration procedure for the feedback engine. Three scenarios according to the feedback source used in the application are distinguished. Follow the procedures referring to the detailed description for each of the feedback engine modules included in this chapter.

Usage with a single feedback source in one channel

Since both channels A and B offer the same capabilities, either of them can be selected when using a single feedback source. Channel A is used for the example procedure below.

Use the following procedure to configure a single feedback source for the motor:

1. Select the feedback source for channel A by setting the desired value in the SRC_SEL_A field of the FEEDBACK.CONF_CH_A register.
2. Configure the 16-bit conversion module of channel A. To do this, set the CPR_INV_A field in the FEEDBACK.CONF_CH_A register to 2^{24} / CPR, where the CPR represents the number of value increments per one revolution of the feedback. See more details in the respective section below.
3. Configure the correction LUT. This is an optional step and can be skipped if the LUT is not needed. Set the LOOKUP_EN_A bit in the FEEDBACK.LUT register and clear the SPLIT_MODE_EN bit in the same register. The LUT entries must be filled with the correction data beforehand. See more details in the respective section below.

4. Configure the phi extrapolator. This step is also optional and needed mostly for digital Hall sensor feedback. To do this, set an appropriate value in the FEEDBACK.PHI_DIFF_LIMIT register. See more in the respective section below.
5. Configure the feedback engine outputs. This is done through PHI_E_SRC, VELOCITY_SRC and POSITION_SRC fields of FEEDBACK.OUTPUT_CONF register.
    - For PHI_E_SRC: choose LOOKUP_A or EXTRAPOLATOR_A, depending on whether phi extrapolation is used.
    - For VELOCITY_SRC: select LOOKUP_A.
    - For POSITION_SRC: select LOOKUP_A.

Note that if the correction LUT is disabled, then the angle data will pass through it unmodified.

After this, continue with the phi_e calculation and alignment as well as the velocity and position acquisition steps, which are common to all the example procedures shown here.

Usage with a single feedback source in two channels

Both channels A and B can be used when an ABN encoder or a Hall sensor is used as a single feedback source. In this case, one channel is configured for motor commutation, and the other is configured for velocity and position measurement. Since both channels offer the same capabilities, it is not relevant which one is used for commutation and which one for velocity and position tracking. Channel A is used for commutation in the example procedure below.

Use the following procedure to configure an ABN encoder or a Hall sensor as a single feedback source:

1. Select the feedback source for channel A by setting the desired value in the SRC_SEL_A field of the FEEDBACK.CONF_CH_A register. Select any of the ABN_1, ABN_2 or HALL options.
2. Configure the 16-bit conversion module of channel A. To do this, set the CPR_INV_A field in the FEEDBACK.CONF_CH_A register to 224 / CPR, where the CPR represents the number of value increments per one revolution of the feedback. See more details in the respective section below.
3. Select the feedback source for channel B by setting the desired value in the SRC_SEL_B field of the FEEDBACK.CONF_CH_B register. Select any of the ABN_1_FREE, ABN_2_FREE or HALL_FREE options. These options are especially suitable for velocity and position tracking.
4. Configure the 16-bit conversion module of channel B. To do this, set the CPR_INV_B field in the FEEDBACK.CONF_CH_B register. It is recommended to use a value of 256 for ABN encoder feedback and a value of 4096 for Hall sensor feedback. With this, the 16-bit conversion can be “bypassed”, thereby allowing to have a scaling which is closer to the feedback’s true resolution. See more details in the respective section below.
5. Configure the correction LUT. This is an optional step and can be skipped if the LUT is not needed. The correction LUT is generally not needed for ABN encoder or Hall sensor feedback.
6. Configure the phi extrapolator. This step is also optional and needed mostly for digital Hall sensor feedback. To do this, set an appropriate value in the FEEDBACK.PHI_DIFF_LIMIT register. See more in the respective section below.
7. Configure the feedback engine outputs. This is done through PHI_E_SRC, VELOCITY_SRC and POSITION_SRC fields of FEEDBACK.OUTPUT_CONF register.
    - For PHI_E_SRC: choose LOOKUP_A or EXTRAPOLATOR_A, depending on whether phi extrapolation is used.
    - For VELOCITY_SRC: select LOOKUP_B, which contains the feedback angle suitable for velocity tracking.
    - For POSITION_SRC: select LOOKUP_B, which contains the feedback angle suitable for position tracking.

Note that if the correction LUT is disabled, then the angle data will pass through it unmodified.

After this, continue with the phi_e calculation and alignment as well as the velocity and position acquisition steps, which are common to all the example procedures shown here.

Usage with dual feedback source

Both channels A and B must be used when two feedback sources are available in the application. In this case, one channel is configured for motor commutation, and the other is configured for velocity and position measurement. Since both channels offer the same capabilities, it is not relevant which one is used for commutation and which one for velocity and position tracking. Channel A is used for commutation in the example procedure below.

Use the following procedure to configure dual feedback sources:

1. Select the feedback source for channel A by setting the desired value in the SRC_SEL_A field of the FEEDBACK.CONF_CH_A register.
   - If using an ABN encoder or a Hall sensor as a feedback source for this channel, select the corresponding ABN_1, ABN_2 or HALL options.
   - If using a different feedback type, select its corresponding option.
2. Configure the 16-bit conversion module of channel A. To do this, set the CPR_INV_A field in the FEEDBACK.CONF_CH_A register to 2^{24} / CPR, where the CPR represents the number of value increments per one revolution of the feedback. See more details in the respective section below.
3. Select the feedback source for channel B by setting the desired value in the SRC_SEL_B field of the FEEDBACK.CONF_CH_B register.
   - If using an ABN encoder or a Hall sensor as a feedback source for this channel, select the corresponding ABN_1_FREE, ABN_2_FREE or HALL_FREE options.
   - If using a different feedback type, select its corresponding option.
4. Configure the 16-bit conversion module of channel B. To do this, set the CPR_INV_B field in the FEEDBACK.CONF_CH_B register. See more details in the respective section below.
   - If using an ABN encoder as a feedback source for this channel, it is recommended to use a value of 256.
   - If using a Hall sensor as feedback source, it is recommended to use a value of 4096.
   - If using a different feedback type, set a value of 2^{24} / CPR, where the CPR represents the number of value increments per one revolution of the feedback.
5. Configure the correction LUT. This is an optional step and can be skipped if the LUT is not needed. The LUT entries must be filled with the correction data beforehand. See more details in the respective section below.
   - To enable the LUT for single channel use, clear the SPLIT_MODE_EN bit in the FEEDBACK.LUT register, set the LOOKUP_EN_x bit (with x = A or B) according to the channel that needs to be corrected and clear the LOOKUP_EN_x bit of the other channel.
   - To enable the LUT for both channels, set SPLIT_MODE_EN bit in the FEEDBACK.LUT register and set both LOOKUP_EN_A and LOOKUP_EN_A bits.
6. Configure the phi extrapolator. This step is also optional and needed mostly for digital Hall sensor feedback. To do this, set an appropriate value in the FEEDBACK.PHI_DIFF_LIMIT register. See more in the respective section below.
7. Configure the feedback engine outputs. This is done through PHI_E_SRC, VELOCITY_SRC and POSITION_SRC fields of FEEDBACK.OUTPUT_CONF register.
   - For PHI_E_SRC: choose LOOKUP_A or EXTRAPOLATOR_A, depending on whether phi extrapolation is used.
   - For VELOCITY_SRC: select LOOKUP_B, which contains the feedback angle suitable for velocity tracking.
   - For POSITION_SRC: select LOOKUP_B, which contains the feedback angle suitable for position tracking.

Note that if the correction LUT is disabled, then the angle data will pass through it unmodified.

After this, continue with the phi_e calculation and alignment as well as the velocity and position acquisition steps, which are common to all the example procedures shown here.

16-bit Conversion

Since different motor angle feedback sources are supported, the 16-bit conversion module serves as a way to ensure that the correct scaling is used regardless of the selected feedback source. To achieve this, all sources are scaled up to a resolution of 16 bits when processed by this module.

Two conversion modules are available, one for channel A and one for channel B. They are configured through FEEDBACK.CONF_CH_A and FEEDBACK.CONF_CH_B registers respectively. The SRC_SEL_x (with x = A, B) fields select the feedback source to be used for the corresponding channel. The CPR_INV_x (with x = A, B) fields determine the output of the conversion operation.

The 16-bit scaling is done such that one revolution of the feedback source, whether mechanical or electrical, maps to a value between 0 and 65535. The output of the conversion module is calculated as:

$$phi_{CONVERTED} = \frac{phi_{SOURCE} \times CPR\_INV\_x}{256}$$

Therefore, to achieve the 16-bit scaling, an appropriate cpr_inv value should be calculated as follows:

$$CPR\_INV\_x = \frac{2^{24}}{counts\_per\_rev}$$

For instance, use the following counts_per_rev values according to the selected choice in SRC_SEL_x field:

- ABN_1, ABN_2: use counts_per_rev equal to the CPR of the used encoder, for instance:
  - For an encoder with 1024 lines, its CPR value is 1024 × 4 = 4096. Therefore, CPR_INV_x = 2^{24} / 4096 = 4096.
  - For an encoder with 10000 lines, its CPR value is 40000. Therefore, CPR_INV_x = 2^{24} / 40000 = 419.
- ANALOG_SENSOR: use counts_per_rev equal to 2^{16} = 65536, as that is the full-scale range of the ANA_PHI field. Therefore, always use CPR_INV_x = 2^{24} / 2^{16} = 256.
- HALL: use counts_per_rev equal to 6, since the Hall decoder's COUNT field always cycles in the range from 0 to 5. Therefore, always use CPR_INV_x = 2^{24} / 6 = 2796202.
- PHI_EXT_A, PHI_EXT_B: in this case, counts_per_rev is determined by the the full-scale range of the externally written angle value, which would come from a microcontroller or from the integrated I/O Controller, for instance:
  - For a 14-bit resolution SPI encoder, use counts_per_rev equal to 2^{14}. Therefore CPR_INV_x = 2^{24} / 2^{14} = 1024.

Some special choices, the so-called FREE options, need special consideration. The goal with these options is to "bypass" the 16-bit conversion, thereby allowing converted values which are closer to the feedback's true resolution and scaling. The CPR_INV_x must be set to a power of 2 equal or bigger than 2^{8}, that is, 256, 512, 1024, 2048, etc. The greater the power of two, the bigger the resulting velocity and position values. A value of 256 results in the feedback's original scaling. This also affects the real-world unit conversion, as explained in *Real World Unit Conversion* section.

- ABN_1_FREE, ABN_2_FREE: it is recommended to use a CPR_INV_x of 256. In this way the original resolution and scaling of the encoder is retained.
- HALL_FREE: it is recommended to use a CPR_INV_x of 4096. Smaller powers of 2 are valid but not recommended, as they may result in too small converted values. This becomes problematic when using the ramp generator.

The output of the 16-bit conversion is the input for the correction LUT. The converted values can be read in PHI_CONVERTED_A and PHI_CONVERTED_B fields of the FEEDBACK.PHI_CONVERTED register.

Correction Lookup Table

The correction lookup table (LUT) of the TMC6460 is a 256-entry interpolated table that provides the ability to correct angle feedback error or distortion that is continuous and cyclic, for instance, the error resulting from the misalignment of an absolute magnetic encoder with respect to the rotating magnet attached to the motor shaft.

The LUT can be enabled for each feedback channel independently by setting either LOOKUP_A_EN or LOOKUP_B_EN bit of the FEEDBACK.LUT register. The LUT also features a split mode to operate on both feedback channels simultaneously. When the split mode is enabled, the 256-entry table is split into two 128-entry tables. The first 128 entries are used for feedback channel A, and the last 128 entries are used for channel B. The split mode of the LUT is enabled by setting the SPLIT_MODE bit of the FEEDBACK.LUT register. The split mode must be enabled if both LUT channels are enabled. However, it is possible to enable only one of the channels even if split mode is active, in which case only the 128 entries corresponding to the enabled channel are used.

Each entry of the LUT consists of an 8-bit value with the range of -128 to 127. However, the LUT has a configurable gain factor of 1, 2, 4, 8 and 16, which helps to extend the correction range. Like this, the LUT can correct angle error values with a maximum range of -2048 to 2032. An independent gain factor can be set for each LUT channel through the LOOKUP_A_GAIN and LOOKUP_B_GAIN fields of the FEEDBACK.LUT register.

The output of the LUT is the input for the extrapolator module, the phi_e calculation module, the velocity meters and the multiturn position decoder. The output of the LUT channels can be read from PHI_LOOKUP_A field of the FEEDBACK.CH_A register and PHI_LOOKUP_B field of the FEEDBACK.CH_B register.

Characterizing the encoder error

To successfully use the correction LUT the encoder error must be first characterized. For this, a second angle source must be used as a reference. The reference source should provide an angle that accurately follows the real angle of the motor shaft. A high-resolution ABN encoder is a good option for a reference angle source. Alternatively, the phi_e from the ramp generator can be used if no second external angle feedback is available.

The angle value from the two sources, the used feedback and the reference, must be tracked simultaneously while the motor spins. For this, the motor can be turned in open loop mode as described in Open Loop Motion Modes section. A minimum of 256, equally spaced samples must be taken as the motor completes one full revolution of the angle feedback source. The minimum number of samples is reduced to 128 if the split mode of the LUT is used. A higher number of samples can be taken to filter the noise in the captured data, ideally capturing a multiple of 256 samples (or 128 if using the split mode) per revolution of the feedback source. Two examples are given below:

- If an absolute encoder is used and no second feedback source is available, the internal phi_e from the ramp generator can be used as a reference. In this case, the data samples can be taken from the PHI_CONVERTED_A field of the FEEDBACK.PHI_CONVERTED register, and from the PHI_E field of the RAMP.PHI_E register. This assumes the used feedback source is correctly configured on the feedback channel A. However, to use the internal phi_e as a valid reference angle, it is crucial to consider the effects of the cogging torque of the motor. To compensate for this, a sufficiently high open loop current must be used while turning the motor during the data sampling, such that the motor shaft smoothly and closely follows the internal phi_e as it rotates. Additionally, the captured data for the internal phi_e must be adapted, such that the two angle values are comparable, that is, they must have the same scale. This is because the phi_e completes a cycle (from 0 to 65535) once per electrical revolution, while an absolute encoder completes the same cycle only after one mechanical revolution.
- If an absolute encoder is used and an ABN encoder is available, then the angle from the ABN encoder can be used as a reference. In this case, the data samples while the motor turns can be taken from the PHI_CONVERTED_A and PHI_CONVERTED_B fields of the FEEDBACK.PHI_CONVERTED register. This assumes such feedback sources are correctly configured on the feedback channels A and B. Since the angle from the ABN encoder already follows that of the motor shaft, the effects of the cogging torque do not need to be considered in this case. Therefore, the open loop current does not need to be as high as in the previous example. Additionally, the captured data for the ABN does not need to be adapted, since this angle and the absolute encoder angle are already comparable.

The encoder error function can then be obtained as the difference between the angle reported by the used feedback and the angle from the reference source. Any "DC offset" from the obtained error must be removed, for instance by computing the average of all the error data points and then subtracting the resulting value from each error point.

If a higher number of samples are available, a smoothing function such as a moving average filter can be applied on the error function. The resulting smoothed error function needs to be down sampled to 256 data points (or 128 if using the split mode) before it can be written into the LUT.

An additional adaptation is required before populating the LUT if any of the resulting error data points exceeds the valid range of the LUT entries of -128 to 127. Depending on the maximum absolute value of the error data points, the data needs to be divided by 2, 4, 8 or 16, corresponding to the available LUT gain factors. For instance, if the resulting error function has minimum and maximum values of -180 and 190, then the error function values need to be divided by 2, if the error function has a range from -510 to 500, then the error function values must be divided by 4, and so forth.

Theory of operation

The entries in the LUT are identified by their address, which is represented by a value between 0 and 255. Each entry of the table is an 8-bit value, interpreted as a signed integer in the range of -128 to 127. As described above, the encoder error function $f_{ERR}(phi)$ represents the difference between the angle reported by the feedback source $phi_{FB}(phi)$ and the actual mechanical angle (or the angle taken as a reference) phi, according to the following equation:

$$f_{ERR}(phi) = phi_{FB}(phi) - phi$$

For a given 16-bit input angle phi_{FB}, two LUT entries are read. The address of the first entry is defined by the upper 8 bits of phi_{FB} and the following address is taken as the second entry: $addr0 = phi_{FB} \gg 8$ and $addr1 = (phi_{FB} \gg 8) + 1$; If $addr0 = 256$, $addr1$ is set automatically to 0.

A weighted average of the values in the two LUT entries $lut(addr0)$ and $lut(addr1)$ is computed, utilizing the lower 8 bits of the angle phi_{FB} to determine the weights of each value. This quantity is called the interpolation result $res\_interpolation$:

$$res\_interpolation = lut(addr0) \times \frac{phi_{FB} \; \% \; 256}{256} + lut(addr1) \times \frac{256 - (phi_{FB} \; \% \; 256)}{256}; \qquad \% - modulo\ operator$$

For example, if $phi_{FB} = 20000 = 0\text{x}4\text{E}20$, the addresses are as follows: $addr0 = 0\text{x}4\text{E}$ and $addr0 = 0\text{x}4\text{F}$.

If the LUT entries have the values $lut(0\text{x}4\text{E}) = 10$ and $lut(0\text{x}4\text{F}) = -3$, the interpolation result is calculated as:

$$res\_interpolation = 10 \times \frac{20000\ \%\ 256}{256} - 3 \times \frac{256 - (20000\ \%\ 256)}{256} = 1.25 - 2.625 = -1.375$$

This interpolation result is then scaled by the set gain factor (LOOKUP_GAIN_A or LOOKUP_GAIN_B fields of FEEDBACK.LUT register), which results in the correction value $res\_correction$. The correction factor is added to the input angle to produce the corrected output angle. The whole procedure is depicted in the diagram of *Figure 35*.

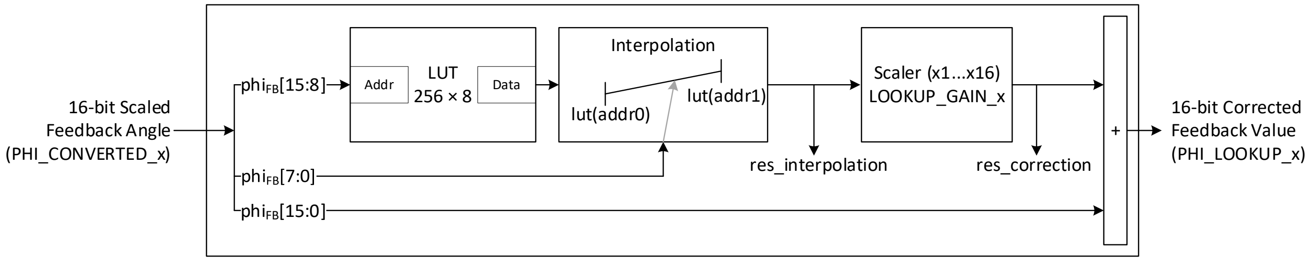


Figure 35. Correction LUT working scheme

Reading and writing the LUT entries

To read the value of an entry in the LUT, first write the address of the desired entry into the ADDR field of the FEEDBACK.LUT register. Then read out the RDATA field of the FEEDBACK.LUT register to retrieve the LUT value of the given address.

Writing the LUT can only be done in groups of four entries at a time. To write four entries of the LUT write the address of the first of the four entries into the ADDR field of the FEEDBACK.LUT register (the same address field as for reading). When populating the LUT, the passed address must be a multiple of 4, that is, 0, 4, 8, 12, and so on. Afterwards, write the 32-bit value containing the data for the four LUT entries into the FEEDBACK.LUT_WDATA register. The LUT automatically writes bits WDATA[7:0] to the table entry pointed by ADDR field, bits WDATA[15:8] into the table entry at ADDR+1, bits WDATA[23:16] into ADDR+2 and bits WDATA[31:24] into ADDR+3. Then, increase the value in the ADDR field by four and write the value for the next four LUT entries in the FEEDBACK.LUT_WDATA register. Repeat the process until the whole table is populated, after using the last address of 252.

Phi Extrapolator

The phi extrapolator module uses the time between angle feedback updates and the corresponding step size to predict the next update time and value. The extrapolator uses the predicted values to generate a large number of intermediate angle updates. This effectively reduces the step size of the feedback angle. The module is intended to be used with low resolution feedback such as digital Hall sensors; however, it can be used with any feedback type.

The phi extrapolator is enabled by selecting EXTRAPOLATOR_A or EXTRAPOLATOR_B options (10_b and 11_b, respectively) in the PHI_E_SRC field of the FEEDBACK.OUTPUT_CONF register. It is not possible to manually select the source channel (channel A or B) for the extrapolator module. Instead, the selection is automatically made based on the set option in the PHI_E_SRC field.

The extrapolator's predictions generate angles that are ahead of the actual feedback angle. The maximum distance the extrapolator will predict is configured by the PHI_DIFF_LIMIT field of the FEEDBACK.PHI_DIFF_LIMIT register. This field holds a 16-bit angle difference. Choose a value larger than your feedback system's angle resolution, but always smaller than a quarter revolution (16384). For example, for a hall sensor the minimum PHI_DIFF_LIMIT is 2^{16} / 6 = 10923. Larger PHI_DIFF_LIMIT values allow for smoother operation during steady-state velocity, but generate a more jitter when extrapolation automatically gets disabled at lower velocities.

Note that for applications where a Hall sensor is used, the PHI_DIFF_LIMIT value is typically set to 65° of an electrical revolution, that is, a value of 2^{16} × 65°/ 360° = 11833. This covers the 60° of a Hall sector plus some margin for the Hall sensor alignment with the magnetic field.

The phi extrapolator requires a minimum update rate to function correctly. When the minimum update rate is not met, the extrapolator is automatically disabled, and feedback values are passed through unmodified. The minimum update rate for extrapolation is defined by:

$$f_{EXTRAPOLATION,min} = \frac{f_{SYS}}{65536 \times 16} = 57.22\ Hz \qquad with\ f_{SYS} = 60MHz$$

As an extension, the minimum velocity for extrapolation to occur when using a Hall sensor can be calculated as:

$$v_{EXTRAPOLATION,min} = \frac{f_{SYS} \times 60}{65536 \times 16 \times 6 \times N_{polepairs}} \; [rpm] \qquad with \; f_{SYS} = 60MHz$$

The extrapolated angle can always be read from PHI_EXTRAPOLATED_AB field in the FEEDBACK.CH_A register. Despite its origin register FEEDBACK.CH_A, this bit field may show the extrapolated angle for channel B if channel B is selected for commutation.

Phi E Calculation

The phi_e calculation module is used to convert from mechanical angles to electrical angles, as well as for alignment of the feedback's angle with the internal phi_e used for FOC and therefore current control.

The phi_e calculation module takes its input from the LUT output or the extrapolator output. This is set through PHI_E_SRC field of FEEDBACK.OUTPUT_CONF register. The phi_e calculation module's output can be read from the PHI_E_FOC field of the FEEDBACK.PHI_E register.

Additionally, this module is configured through the PHI_E_MUL_FACTOR field in the FEEDBACK.OUTPUT_CONF register. Write a value equal to the amount of electrical rotations of the motor per rotation of the feedback angle. For feedback sources that produce the mechanical angle, for instance an ABN encoder or a SinCos encoder, write the PHI_E_MUL_FACTOR field with the number of pole pairs of the motor. For feedback sources that measure the electrical angle directly, for instance digital or analog Hall sensors, write the PHI_E_MUL_FACTOR field with a value of 1.

Phi E alignment

It is likely that the feedback's calculated electrical angle will not be perfectly aligned with the internal phi_e. Since this alignment is critical for correct FOC, an appropriate procedure to align the two angles must be carried out. The procedure must be chosen according to the used feedback source. A few different cases can be distinguished:

- Incremental ABN encoders:
  - The ABN encoder angle can be aligned with the internal phi_e angle by means of an "initialization".
  - Use the following procedure to initialize the ABN encoder:
    - Set the IC for operation in open loop position mode. Refer to *Open Loop Motion Modes* section for details.
    - Make sure that FOC.PID_POSITION_TARGET is set to zero.
    - Set a voltage value in the UD field of the EXT_CTRL.VOLTAGE register, or a target flux value in PID_FLUX_TARGET field of the PID_TORQUE_FLUX_TARGET register, depending on whether MOTION_MODE is set to VOLTAGE_EXT or TORQUE.
    - The rotor should snap into a specific position, aligning with the electrical angle of 0°. The higher the voltage/flux value, the more precise the alignment is, but the more current is forced through the motor's coils.
    - The voltage/flux value can be manually ramped up in small increments instead of setting the final value from the start. This can help reduce the chances of inducing an overcurrent state.
    - Wait for a short time to ensure that the motor is standstill. Depending on the motor characteristics and voltage/flux value used, a time between 0.5 and 1 seconds should suffice.
    - If the initialization must be done in the shortest time possible, the COUNT field of the ABN.COUNT register can be monitored after setting the voltage/flux value. The procedure can be continued as soon as the monitored value stops changing, meaning that the motor is standstill.
    - Set the value of COUNT field of the ABN.COUNT register to zero.
  - Fields PHI_E and PHI_E_FOC should now show very similar values as the motor is rotated, and the ABN encoder phi_e angle can now be used for closed-loop operation.
  - This procedure must be executed only once while the TMC6460 and the encoder stay active. It must be carried out after a power cycle or reset of the IC, or after disruption of the encoder signals or physical connections.
  - Note that this procedure is not infallible. For example, it is possible that a motor with a high load does not align with the target angle of 0° if the initial position is close to 180° of the electrical angle. Improved initialization methods are not discussed in detail here; however, a couple of examples include:
    - Actively shaking or swinging the target angle before ending in the target of 0°.
    - Setting an initial target angle of 90° and waiting for standstill before setting the final target of 0°.

- Digital Hall sensors – phi extrapolation is used:
  - The Hall sensor angle can be aligned with the internal phi_e angle by making use of an offset value.
  - An offset value can be configured through the PHI_E_OFFSET field of the FEEDBACK.PHI_E_OFFSET register.
  - Use the following procedure to find the offset value:
    - Rotate the motor in open loop mode (see *Open Loop Motion Modes* section) while capturing both the PHI_E field of the RAMPER.PHI_E register (open loop angle) and the PHI_E_FOC field of the FEEDBACK.PHI_E register (feedback's angle). During data capturing, the motor should rotate fast enough for phi extrapolation to occur (see *Phi Extrapolator* section for details on the minimum velocity).
    - Compute the average difference between the two angles from the captured data.
    - Use the computed average as the offset value in the PHI_E_OFFSET field.
    - For better results, it may be necessary to repeat the process while rotating the motor in the opposite direction, computing the average by using the data of the movement in both directions.
  - Fields PHI_E and PHI_E_FOC should now show very similar values as the motor rotates, and the resulting Hall sensor phi_e angle can now be used for closed-loop operation.
  - Since the Hall sensor provides an absolute (electrical) angle, the calculated offset will not change for this particular motor-sensor setup. This means that the procedure above must be carried out only once, and the resulting offset value can be used every time after resetting or power cycling the IC.
- Digital Hall sensors – no phi extrapolation used:
  - The Hall sensor angle can be aligned with the internal phi_e angle by making use of the digital Hall decoder sector offset as described in the *Digital Hall Sensor* section.
  - Use the following procedure to find the sector offset settings:
    - Slowly rotate the motor in open loop mode (see *Open Loop Motion Modes* section) while capturing both the PHI_E field of the RAMPER.PHI_E register (open loop angle) and the PHI_E_FOC field of the FEEDBACK.PHI_E register (Hall sensor angle).
    - Compute the average difference between the two angles from the captured data.
    - Calculate the equivalent in degrees of the computed average with $average\_diff/65535 \times 360°$.
    - Using *Table 14* as a reference, set the mapping and pin input inversion values according to the sector offset which is closest to the calculated average difference.
  - The angle sector from the PHI_E_FOC should now follow the open loop angle in PHI_E as the motor rotates, and the resulting Hall sensor phi_e angle can now be used for closed-loop operation.
  - Since the Hall sensor provides an absolute (electrical) angle, the obtained Hall sector offset will not change for this particular motor-sensor setup. This means that the procedure above must be carried out only once, and the resulting mapping and pin input inversion settings can be used every time after resetting or power cycling the IC.
- Analog Hall sensors, SinCos encoders and absolute SPI/SSI encoders, that is, feedback sources that produce an absolute angle (even if only the electrical angle):
  - The feedback's electrical angle can be aligned with the internal phi_e angle by making use of the PHI_E_OFFSET field of the FEEDBACK.PHI_E_OFFSET register.
  - Use the same procedure as with the digital Hall sensor when phi extrapolation is used to find the offset value.
  - Since phi extrapolation is generally not needed for these feedback sources, the motor does not need to rotate as fast as in the case above while capturing the phi_e data.
  - The procedure to find the offset value must be carried out only once, and the resulting offset can be used every time after resetting or power cycling the IC.

Phi E rounding error correction

When using feedback sources with a resolution that is not a power of 2 as sources for phi_e calculation, the formula for the CPR_INV_x of the 16-bit conversion module yields a non-integer value. In such cases the actual value in CPR_INV_x field must be set to the floor of the calculated value, which results in each feedback step being smaller than they should. The smaller steps lead to an accumulated error. To compensate for it, the final step of the revolution becomes larger than all the previous steps.

The correction LUT can be configured to completely eliminate this error. The key to successfully using the LUT to correct for a non-continuous error function is to avoid interpolation between the first and last entries on the table. To achieve this:

1. The CPR_INV should be reduced so that instead of scaling to a range of 0…0xFFFF the feedback angle is scaled to a range of 0…0xFEFF (use the formula CPR_INV = $(2^{24} - 2^{16})$ / CPR).

2. Using the adjusted CPR_INV value, calculate the difference (after 16-bit conversion) between the first step (0 to 1) and the last step (CPR − 1 to 0).
3. Subtract the first step size from the last step size, the resulting quantity is referred to as the "correction factor".
4. The correction factor may exceed 127 (maximum value for the LUT entries). In such cases a scaling factor must selected, such that it is the smallest number which can divide the correction factor into a number smaller or equal to 127.
5. Configure the LUT with the chosen scaling factor, and program all the entries (including the last one) with a linear ramp that starts at zero and ends at the correction factor divided by the scaling factor. Like this, the correction LUT can completely eliminate the error introduced by the 16-bit scaling.

Example rounding error correction

For the case of an encoder with CPR = 30000, the calculated CPR_INV setting is 2^{24} / 30000 = 559.24053. This value must be rounded down to 559. Note that CPR_INV must be always round down to the next integer value.

Calculating the effect of the uncompensated error

The 16-bit conversion module simply multiplies the encoder angle by the CPR_INV and then divides that by 256. Since the CPR_INV is rounded down, each encoder step results in a change in the converted angle that is slightly smaller than it should. This small error accumulates until the encoder angle reaches its highest value, at which point the error is at its highest.

At the end of the rotation when the encoder angle wraps around to zero, the resulting step in the converted angle is larger to compensate for all the steps that were slightly smaller. This is why the error never accumulates over multiple rotations.

To determine what the maximum error will be for a particular encoder, the converted angle is compared against the angle with and without rounding when the encoder is at its maximum value:

- For this example, the maximum encoder value is 29999 (the CPR − 1).
- At this maximum value the converted angle without rounding is: (2^{24} / 30000) × 29999 / 256 = 65533.81547
- The converted angle using the rounded CPR_INV is: 559 × 29999 / 256 = 65505.~~62891~~
- The 16-bit conversion module always rounds down.
  - Consequently, the example result must be rounded down to 65505.
- The maximum error results in 65533.81547 − 65505 = 28.81547

Eliminating the rounding error

If the calculated maximum error is unacceptable for the application, the correction LUT feature can be used to correct it. A modified CPR_INV calculation is needed for the procedure to work: CPR_INV = (2^{24} − 2^{16}) / CPR.

Using the correction LUT to compensate for encoder errors is still possible in this configuration, but the procedure to configure the LUT becomes more complex.

To configure the LUT to eliminate the conversion error, a modified CPR_INV formula is needed. The modified CPR_INV equation is required because the correction LUT interpolates between adjacent entries, and the first and last entries are counted as being adjacent. Since the error function has a discontinuity at the end of the rotation it is not desireable to interpolate between the first and last entries.

The modified CPR_INV value scales the angle to a slightly reduced range which prevents interpolation between these entries. Scaling to a reduced range actually makes the error much larger, but also makes it possible for the LUT to fully correct it.

1. To configure the LUT first calculate the modified CPR_INV value. For the example encoder (CPR = 30000):
2. $CPR\_INV = \frac{2^{24}-2^{16}}{CPR} = \frac{2^{24}-2^{16}}{30000} \approx 557 \ (rounded\ down)$
3. Using this value calculate the size of the first converted step. This is the converted angle when the encoder angle is 1. The first step size is consequently 1 × 557 / 256 ≈ 2 (rounded down).
4. Calculate the converted angle with the maximum encoder value: 29999 × 557 / 256 ≈ 65271 (rounded down).
5. With this value calculate the size of the last converted step (step back to 0). The last step size results in 2^{16} − 65271 = 265. (Note that the converted value wraps around 2^{16}, which is the top of the 16-bit range +1).

6. Now that the size of the first and last steps have been calculated the next step is to calculate the difference between these values, which represent the correction_factor = 265 − 2 = 263.
7. In this case the calculated correction factor is larger than 127, the maximum value of the 8-bit LUT entries, so a gain factor must be calculated. Gain factor options are 1, 2, 4, 8, and 16.
8. Set the gain factor of the LUT to the minimum value such that the correction factor divided by the gain factor is less than or equal to 127. For our example, the minimum lookup_gain_factor = 4. This gain factor must be defined also in the FEEDBACK.LUT register.
9. Next program the LUT for each index ADDR ϵ {0…255} with a linear ramp that starts at 0 and ends at the correction factor divided by the gain factor (rounded to the nearest integer). Note that the 255 value is the index of the last entry (this assumes that split mode is disabled). The value of any particular entry is consequently:
10. $LUT[ADDR] = \frac{correction\_factor}{lookup\_gain\_factor} \times \frac{ADDR}{255} \qquad with\ ADDR \in \{0 \ldots 255\}$
11. Now that the gain factor and LUT entries are set, the LUT can be enabled to fully eliminate the 16-bit conversion error.

If the LUT is used in split mode, then the value of any particular entry is (rounded to the nearest integer):

$$LUT[ADDR] = \frac{correction\_factor}{lookup\_gain\_factor} \times \frac{ADDR}{127} \qquad with\ ADDR \in \{0 \ldots 127\}$$

Additionally, the CPR_INV must be calculated slightly differently (rounding down):

$$CPR\_INV = \frac{2^{24} - 2^{17}}{CPR}$$

Before trying to configure the LUT for correction of the rounding error, evaluate if the current controller response is satisfactory without the rounding error correction. In many cases the rounding error results in small or even insignificant changes of the current controller behavior.

Velocity Acquisition

To enable velocity control, it is required to configure the feedback path for the velocity acquisition. The VELOCITY_SRC field of the FEEDBACK.OUTPUT_CONF register is used to select the source from one of the two feedback channels.

If using an ABN encoder or a digital Hall sensor for velocity measurement, the input source for the used feedback channel should be one of the ABN_1_FREE, ABN_2_FREE or HALL_FREE options respectively (SRC_SEL_x fields of the FEEDBACK.CONF_CH_x registers, with x = A, B). This is especially true for Hall sensors and for ABN encoders whose CPR is not a power of 2. Note that the feedback channel used for phi_e calculation must still use the normal ABN_1, ABN_2, or HALL options. This results in the necessity of both feedback channels A and B even if only a single feedback source (ABN encoder or Hall sensor) is used.

As noted above, there are two modules for velocity calculation: the period velocity meter and the frequency velocity meter. They operate simultaneously, but only the output of one of them can be selected as input for the velocity controller. The input for the velocity controller is defined by the VELOCITY_SELECTION field of the FEEDBACK.OUTPUT_CONF register. The three options are: VELOCITY_FRQ, VELOCITY_PER, and VELOCITY_EXT, corresponding to the frequency velocity meter, period velocity meter, and an externally writable velocity value, respectively.

Period velocity meter

The period velocity meter works by measuring the elapsed time between consecutive angle updates at its input. This meter is suitable for measuring even very low motor velocities.

The period velocity meter is configured through the FEEDBACK.VELOCITY_PER_CONF register. The period velocity measurement is updated only when the angle of the selected feedback changes more than the value defined in the POS_DEV_MIN field. The defined minimum change must also happen within the time determined by POS_DEV_TIMER field. The timeout is calculated as POS_DEV_TIMER / f_{SYS}, with f_{SYS} = 60MHz.

Additionally, a moving average filter is available for the measurement of this velocity meter. The width of the moving average window is configured through the FILTER_WIDTH field of the FEEDBACK.VELOCITY_PER_FILTER register. To disable the filter, set a value of FILTER_WIDTH = 0_b.

The calculated value of the period velocity meter can always be read from the FEEDBACK.VELOCITY_PER register.

Frequency velocity meter

The frequency velocity meter works by measuring the change of the angle value at a constant, configurable rate. This meter is more suitable for very high motor velocities, and not recommended for very low velocities. The frequency velocity meter is configured through the FEEDBACK.VELOCITY_FRQ_CONF register.

The update rate of the frequency velocity meter f_{vel} is defined by the active PWM frequency and the down sampling factor from VELOCITY_SAMPLING field of the FEEDBACK.VELOCITY_FRQ_CONF register and the following formula:

$$f_{vel} = \frac{f_{PWM}}{VELOCITY\_SAMPLING + 1}$$

Note that the VELOCITY_SAMPLING field also defines the update rate of the whole velocity controller. For this reason, its value must be considered even if only the period velocity meter is used.

The frequency velocity meter has an additional scaling factor configured through the VELOCITY_SCALING field of the FEEDBACK.VELOCITY_FRQ_CONF register. It is recommended to use the scaling factor for the frequency velocity meter so that its output and the one from the period velocity meter match each other. Only then is it possible to switch between both meters dynamically during operation. Use the following formula to choose the right scaling factor:

$$VELOCITY\_SCALING = \frac{2^{24} \times f_{vel}}{f_{SYS}} \quad with\ f_{SYS} = 60MHz$$

The frequency velocity meter's calculated value can always be read from the FEEDBACK.VELOCITY_FRQ register.

Velocity meters comparison

The period velocity meter is more accurate than the frequency velocity meter in most scenarios. There are some limitations for the period velocity meter when the update frequency of the feedback is very high. This usually only occurs when using high resolution ABN encoders in combination with high motor velocities.

In the following equations, $\mathrm{counts\_per\_mrev}$ refers to the equivalent counts per mechanical revolution as if calculated on the 16-bit conversion module (see formula above). Note that this value is not used for configuration of the IC, but only to aid the velocity meter comparison. This value depends on the configuration in SRC_SEL_x and CPR_INV_x fields of the FEEDBACK.CONF_CH_x registers (with x = A, B) for the corresponding channel, as well as whether the feedback source produces a mechanical angle or an electrical angle. A few example values are given below:

- If SRC_SEL_x = ABN_1_FREE and CPR_INV_x = 256 (for a 4096-line ABN encoder):
  $$\mathrm{counts\_per\_mrev} = 16384$$
- If SRC_SEL_x = PHI_EXT_A and CPR_INV_x = 1024 (for a 14-bit SPI encoder):
  $$\mathrm{counts\_per\_mrev} = 65536$$
- If SRC_SEL_x = HALL_FREE and CPR_INV_x = 256:
  $$\mathrm{counts\_per\_mrev} = 6 \times N_{polepairs}$$
- If SRC_SEL_x = HALL_FREE and CPR_INV_x = 4096:
  $$\mathrm{counts\_per\_mrev} = 96 \times N_{polepairs}$$
- If SRC_SEL_x = ANALOG_SENSOR and CPR_INV_x = 256:
  $$\mathrm{counts\_per\_mrev} = 2^{16} \times N_{polepairs}$$

The following formula can be used to calculate the maximum motor velocity in RPM that the period velocity meter can measure without distortion when the meter's moving average filter is disabled, that is, when the value in the FILTER_WIDTH field of the FEEDBACK.VELOCITY_PER_FILTER register is set to 0_b.

$$v_{PER,max} = \frac{f_{SYS} \times POS\_DEV\_MIN \times 60}{53 \times \mathrm{counts\_per\_mrev}} [rpm] \quad with\ f_{SYS} = 60MHz$$

The following formula can be used to calculate the maximum motor velocity in RPM that the period velocity meter can measure without distortion when the meter's moving average filter is enabled, that is, when the value in the FILTER_WIDTH field of the FEEDBACK.VELOCITY_PER_FILTER register is set to anything other than 0_b.

$$v_{PER,max} = \frac{f_{SYS} \times POS\_DEV\_MIN \times 60}{105 \times \mathrm{counts\_per\_mrev}} [rpm] \quad with\ f_{SYS} = 60MHz$$

If the velocity for the given application is expected to reach anywhere close to the calculated $v_{PER,max}$, then it is recommended to either increase the value in POS_DEV_MIN field or switch the velocity controller's input to the frequency velocity meter. The real value of $v_{PER,max}$ may vary from the calculations depending on the actual setup and configuration.

The performance of the period velocity meter decreases the higher the velocity gets, while the performance of the frequency velocity meter is constant. The equation below can be used to calculate the crossover point (v_{COP}) at which the performance of the period velocity meter falls behind that of the frequency velocity meter.

$$v_{COP} = 60 \times \frac{f_{vel} + \sqrt{f_{vel}^2 + f_{vel} \times f_{SYS} \times POS\_DEV\_MIN \times 4}}{2 \times \text{counts\_per\_mrev}} [rpm] \qquad with\ f_{SYS} = 60MHz$$

Note that the equation of v_{COP} assumes ideal position feedback from the feedback, its resulting value is therefore only meant as a guide. The period velocity meter is sensitive to noisy or inaccurate angle feedback. This means that the actual crossover point may be lower in the real application. If the intended velocity range of the application includes velocities which are close to the calculated v_{COP}, then it is recommended that the actual performance of both meters is measured experimentally, and the crossover point decided based on the real measurements.

A curve representing measurement noise for increasing motor velocities for both velocity meters is displayed in *Figure 36*. In this case, a low measurement noise is desirable.

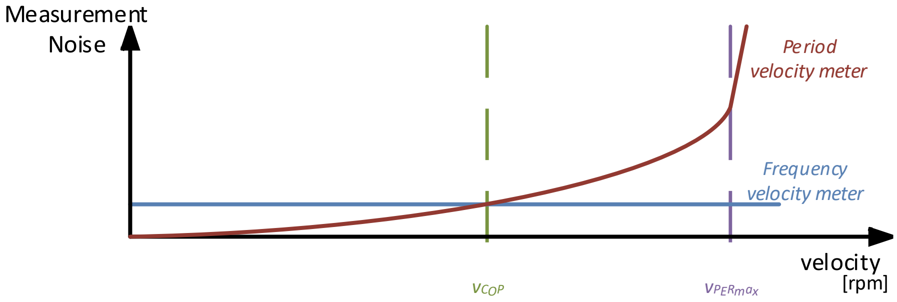


Figure 36. Velocity meters measurement noise for increasing motor velocities

Depending on the configuration, v_{COP} can be higher or lower compared to $v_{PER,max}$. This means both thresholds need to be considered when evaluating which velocity meter to use.

Position Acquisition

To enable position control, it is required to configure the feedback path for the position acquisition. The POSITION_SRC field of the FEEDBACK.OUTPUT_CONF register is used to select the source from one of the two feedback channels.

If using an ABN encoder or a digital Hall sensor for position tracking, the input source for the used feedback channel should be one of the ABN_1_FREE, ABN_2_FREE or HALL_FREE options respectively (SRC_SEL_x fields of the FEEDBACK.CONF_CH_x registers, with x = A, B). This is especially true for Hall sensors and for ABN encoders whose CPR is not a power of 2. Note that the feedback channel used for phi_e calculation must still use the normal ABN_1, ABN_2, or HALL options. This results in the necessity of both feedback channels A and B even if only a single feedback source (ABN encoder or Hall sensor) is used.

Real World Unit Conversion

The TMC6460 can use an internal oscillator or an external clock signal as reference for the system clock f_{SYS}. Velocity measurement, as well the ramp generator motion profiles are referenced to the system clock. For this reason, it is recommended to use an external quartz oscillator, or to provide an external clock signal from a microcontroller for best stability and reproducibility of the measurements and the ramp motion profiles.

The TMC6460 uses an internal scaling for the different PI controllers, as well as for the ramp generator. This section provides formulas to convert internal values into real-world units during closed-loop operation (when RAMP_USE_PHI_E bit of the MCC_CONFIG.MOTOR_MOTION register is not set).

For real-world unit conversion in open loop mode, see the *Open Loop Real Word Unit Conversion* section.

Position conversion

The position scaling depends on the configuration in SRC_SEL_x and CPR_INV_x fields of the FEEDBACK.CONF_CH_x registers (with x = A, B) for the corresponding channel. Table 18 provides the equations to convert from the internal rotor position P_{int} to the mechanical rotor revolutions P_{mech} for different SRC_SEL_x options.

Table 18. Position real-world unit conversion equations

SRC_SEL_x	P_{mech}[rev]
ABN_1_FREE / ABN_2_FREE	$P_{int} \times \frac{256}{CPR\_INV\_x \times CPR_{ABN}}$
ANALOG_SENSOR, ANA_MODE = UVW	$P_{int} \times \frac{256}{CPR\_INV\_x \times 2^{16} \times N_{polepairs}}$
ANALOG_SENSOR, ANA_MODE = XY	$P_{int} \times \frac{256}{CPR\_INV\_x \times 2^{16}}$
HALL_FREE	$P_{int} \times \frac{256}{CPR\_INV\_x \times 6 \times N_{polepairs}}$
PHI_EXT_A / PHI_EXT_B	$P_{int} \times \frac{256}{CPR\_INV\_x \times CPR_{EXT}}$

In this table, CPR_{ABN} refers the counts per revolution of the ABN encoder, CPR_{EXT} refers to the counts per revolution of the externally written angle value and $N_{polepairs}$ is the number of pole pairs of the motor. P_{int} refers to the value of any position register that uses the internal position scaling. This includes but is not limited to: FOC.PID_POSITION_ACTUAL, FOC. PID_POSITION_TARGET and RAMPER.POSITION.

Velocity conversion

Use the following formula and the scaling factor k to convert internal velocity values into rotor velocities in RPM.

$$v\ [rpm] = \frac{v_{int}}{k}$$

In this case, v_{int} refers to the value of any velocity register that uses the internal velocity scaling. This includes but is not limited to: FEEDBACK.VELOCITY_FRQ, FEEDBACK.VELOCITY_PER, FOC.PID_VELOCITY_ACTUAL and FOC.PID_VELOCITY_TARGET as well as any velocity parameter from the ramp generator registers.

The scaling factor k depends on the configuration in SRC_SEL_x and CPR_INV_x fields of the FEEDBACK.CONF_CH_x registers (with x = A, B) for the corresponding channel. Additionally, when using SRC_SEL_x = ANALOG_SENSOR the configuration in ANA_MODE field of the HALL.ANA_CONFIG is also considered.

The equations to calculate the scaling factor for each variation are given in Table 19. Here, k_{PER} and k_{FRQ} refer to the scaling factors for the frequency and period velocity meters, respectively. Note that when using the recommended value for VELOCITY_SCALING field as described above, both velocity scaling factors k_{PER} and k_{FRQ} are the same.

Table 19. Velocity real-world unit scaling factors

SRC_SEL_x	k_{PER}	k_{FRQ}
ABN_1_FREE / ABN_2_FREE	$\frac{CPR_{ABN} \times CPR\_INV\_x \times 2^{24}}{256 \times f_{SYS} \times 60}$	$\frac{CPR_{ABN} \times CPR\_INV\_x \times VELOCITY\_SCALING}{256 \times f_{vel} \times 60}$
ANALOG_SENSOR, ANA_MODE = UVW	$\frac{2^{16} \times N_{polepairs} \times CPR\_INV\_x \times 2^{24}}{256 \times f_{SYS} \times 60}$	$\frac{2^{16} \times N_{polepairs} \times CPR\_INV\_x \times VELOCITY\_SCALING}{256 \times f_{vel} \times 60}$
ANALOG_SENSOR, ANA_MODE = XY	$\frac{2^{16} \times CPR\_INV \times 2^{24}}{256 \times f_{SYS} \times 60}$	$\frac{2^{16} \times N_{polepairs} \times CPR\_INV\_x \times VELOCITY\_SCALING}{256 \times f_{vel} \times 60}$
HALL_FREE	$\frac{6 \times N_{polepairs} \times CPR\_INV\_x \times 2^{24}}{256 \times f_{SYS} \times 60}$	$\frac{6 \times N_{polepairs} \times CPR\_INV\_x \times VELOCITY\_SCALING}{256 \times f_{vel} \times 60}$
PHI_EXT_A / PHI_EXT_B	$\frac{CPR_{EXT} \times CPR\_INV\_x \times 2^{24}}{256 \times f_{SYS} \times 60}$	$\frac{CPR_{EXT} \times CPR\_INV\_x \times VELOCITY\_SCALING}{256 \times f_{vel} \times 60}$

Here f_{vel} is the update rate of the frequency velocity meter as described above. CPR_{ABN} refers to the counts per revolution of the ABN encoder, CPR_{EXT} refers to the counts per revolution of the externally written angle value, $N_{polepairs}$ is the number of pole pairs of the motor, and $f_{SYS} = 60MHz$.

Acceleration conversion

In contrast to velocity and position, acceleration values are used exclusively through the ramp generator block.

The acceleration conversion depends only on the system clock frequency f_{SYS} and the velocity scaling factor k. The velocity scaling factor must be chosen according to the used velocity meter and the feedback channel configurations as described above. Use the following to convert internal acceleration values a_{int} into rotor accelerations in rev/s^2.

$$a\left[\frac{rev}{s^2}\right] = a_{int} \times \frac{f_{SYS}}{k \times 60 \times 2^{17}} \qquad with\ f_{SYS} = 60MHz$$

In this case, a_{int} refers to the value of any of the acceleration parameters from the ramp generator registers. Some examples include: RAMPER.A1, RAMPER.D_MAX, and RAMPER.ACCELERATION.

Motor Control Core

The TMC6460 integrates an advanced motor control core (MCC) that implements the FOC method including torque and flux control loops, as well as velocity and position control loops, and a PWM generator.

Current Acquisition

The current is measured inside the TMC6460 using lossless current sensing on all three low side MOSFETs. Measurements for the ADCs are triggered at the start of a new PWM period, since typically all low side switches are turned on at this point.

Raw current values I1, I2, and I3 bitfields of the ADC.I2_I1_RAW and ADC.VM_I3_RAW registers are calculated by automatically taking the current sense amplifier (CSA) gain into account. This means that, for a given measured current, setting different CSA gain values does not change the resulting raw current value in I1, I2, and I3.

The real-world units current sense values I_{CSx} (with x = 1, 2, 3) depend only on the low side resistor setting in LS_RES_ON field of the MCC_CONFIG.GDRV register and can be calculated from the raw current values I1, I2 and I3 with the formulas shown on Table 20.

Table 20. Current sense calculation

LS_RES_ON[1:0]	Current value calculation (with x = 1, 2, 3)
00	I_{CSx} [A] = Ix × 0.00164 [A] / 8 × 1/4 × 1.000
01	I_{CSx} [A] = Ix × 0.00164 [A] / 8 × 2/4 × 0.992
10	I_{CSx} [A] = Ix × 0.00164 [A] / 8 × 3/4 × 0.988
11	I_{CSx} [A] = Ix × 0.00164 [A] / 8 × 4/4 × 0.982

Note that these formulas are chosen depending on the LS_RES_ON setting. However, they are not directly derived from the output low side ON-resistance $R_{ON,LS}$ value, which also changes according to the value in the LS_RES_ON field.

All three current values, I1, I2, and I3, are summed up in the I123 field of the ADC.I123 register. Too much deviation from the ideal sum value of 0 indicates an inaccurate current measurement. This might be caused when measuring the current on phases with duty cycles close to 100%. It is possible to counteract this inaccurate measurement by using the auto Kirchhoff feature, which is explained below.

A new value can be defined for the I123_LIMIT field of the ADC.I123 register. If the absolute value of I123 exceeds the value in I123_LIMIT field, then the I123_FAIL bit of ADC.STATUS register is set. Additionally, if any of the raw current values I1, I2, or I3 are clipped because their current exceeds the ADC range, the corresponding I1_CLIPPED, I2_CLIPPED, or I3_CLIPPED bits of ADC.STATUS register are set. These bits are also merged into the I_CLIPPED_STATUS bit of the CHIP.STATUS_FLAGS register and latched into the corresponding I_CLIPPED_EVENT bit of CHIP.EVENTS register. The individual event bits in the ADC.STATUS register must be cleared before I_CLIPPED_EVENT can be cleared.

Current sense amplifier gain

As stated above, CSA gain settings are transparent for the raw current values I1, I2, and I3. Higher gain values result in higher resolution of the raw values measured by the internal ADC, whereas lower gain values allow for a higher current to be measured. The CSA gain value in case of static gain setup is defined through the CSA_GAIN field of the MCC_ADC.CSA_GAIN register. *Table 21* shows the absolute value of the equivalent maximum current that can be measured by the ADC for different LS_RES_ON and CSA_GAIN values.

Table 21. Maximum measurable current for different gain settings

CSA_GAIN[1:0]	Sense current multiplier (gain_factor)	Absolute maximum sense current [A] dependent on LS_RES_ON[1:0]:			
		00	01	10	11
00 (default)	1x	1.68	3.33	4.98	6.60
01	2x	0.84	1.67	2.49	3.30
10	3x	0.56	1.11	1.67	2.20
11	4x	0.42	0.83	1.24	1.65

These values correspond only to the ADC measuring limits. Please be aware that the output stage of the TMC6460 is qualified for $3A_{RMS}$ and $5A_{FS}$ at most. Refer to *Table 28* for the full scale current for each LS_RES_ON value.

The raw current values I1, I2, or I3 get clipped if the limits for a given LS_RES_ON and CSA_GAIN combination are exceeded. The absolute raw value at which the current is clipped is defined by 2^{15} / gain_factor.

Current sense amplifier dynamic gain

It might be useful for some applications to adapt the CSA gain during operation. The TMC6460 supports a dynamic gain mode. For this to work, the needed gain values must be individually enabled through the DYN_GAIN_2X_EN, DYN_GAIN_3X_EN, and DYN_GAIN_4X_EN fields of the MCC_ADC.CSA_GAIN register. There is no individual bit for the gain_factor of 1x as this one is always available when using dynamic gain mode.

In addition to enabling the individual dynamic gain factors, it is also needed to set a hysteresis value bigger than 0 in the DYN_GAIN_HYST field of the MCC_ADC.CSA_GAIN register. The hysteresis value relates to the allowable current change during one PWM cycle that does not trigger a gain change during dynamic gain mode. As an example, if PWM frequency f_{PWM} = 25kHz, then the theoretical maximum current change during one PWM cycle of a motor with L = 0.5mH at a supply of 24V would be defined by: di = V / L × dt = 24V / 0.0005H × 40µs = 1.92A.

The current value to monitor during dynamic gain mode is the bigger of the actual and target current values of the current controller. During dynamic gain mode, the maximum possible enabled gain is automatically assigned according to the monitored current value. The currently assigned gain during dynamic gain mode can be read from the CSA_GAIN field of the MCC_ADC.CSA_GAIN register.

To decide which CSA_GAIN value is selected during dynamic gain mode, the current limit for the used gain values must be defined: DYN_GAIN_LIMIT_2X field of the MCC_ADC.DYN_GAIN_LIMIT_2X register and DYN_GAIN_LIMIT_3X and DYN_GAIN_LIMIT_4X fields of the MCC_ADC.DYN_GAIN_LIMITS_4X_3X register. If any of the target or actual current values exceed a certain limit, then the next lower CSA_GAIN is selected, assuming that particular gain is enabled for dynamic gain change. If the limits of all enabled gains are exceeded, then CSA_GAIN = 00_b will be assigned.

Auto Kirchhoff

As the current can only be measured when the low side MOSFET is on, it can occur that at very high high-side duty cycles the current cannot be acquired by the ADC with sufficient accuracy. In this case the measurement that has the highest duty cycle can be replaced via Kirchhoff's law by using the other two measurements.

This feature can be switched on by setting the MEAS_MODE filed of the MCC_ADC.I_GEN_CONFIG register. The corresponding STATUS field of the same register indicates that a measurement has been replaced. The AUTO_KIRCHHOFF_LIM field of the MCC_CONFIG.PWM_PERIOD register specifies the duty cycle value compared to the UU, UV, and UW fields of the FOC.FOC_UW_UU and FOC.FOC_UV registers. In case any of the three duty cycle values exceeds AUTO_KIRCHHOFF_LIM, the auto Kirchhoff feature is used for the phase with the highest duty cycle.

Measured currents after the auto Kirchhoff calculation can be monitored in IU, IV, and IW fields of MCC_ADC.IW_IU and MCC_ADC.IV registers. *Table 22* shows the relation between the MCC current values and the ADC raw current values if none of the values are taken from the auto Kirchhoff calculation.

Table 22. ADC current values vs. MCC_ADC raw values

MCC_ADC register field	Corresponding ADC register field
MCC_ADC.IW_IU::IU	ADC.I2_I1_RAW::I1
MCC_ADC.IV::IV	ADC.I2_I1_RAW::I2
MCC_ADC.IW_IU::IW	ADC.VM_I3_RAW::I3

The equations to calculate the AUTO_KIRCHOFF_LIM are provided in the PWM Engine section below.

PWM Engine

The switching frequency for the PWM generated output can be configured through MAX_COUNT field of the MCC_CONFIG.PWM_PERIOD register. It is not recommended to change the PWM frequency during operation, as it is the base frequency of all control loops and might result in loop instability. The reset value of MAX_COUNT = 4800 results in a PWM frequency of 25kHz. The PWM frequency can be calculated using the equation below. The maximum supported PWM frequency is 200kHz. Above that frequency, the current sensing and the control loop may work incorrectly.

$$f_{PWM} = \frac{f_{PLL}}{MAX\_COUNT} \qquad with\ f_{PLL} = 120MHz$$

For test purposes, the PWM chopper scheme can be manipulated through the CHOP field of the MCC_CONFIG.PWM register. Note that proper operation of the motor is only possible when the CHOP field is set to CENTERED.

In normal operation when driving a BLDC motor, the duty cycle for each output is calculated based on the target output voltage, which in turn is calculated as an output of the current controller after the FOC output transformation. See the *Current Control Loop* section for more detailed information. The calculated output voltage can be constrained by setting a limit value in the PID_UQ_UD_LIMITS field of the FOC.PID_UQ_UD_LIMITS register.

To ensure that the calculated output duty cycle never exceeds 100%, the output voltage has an internal maximum value in addition to the user-defined limit value. The internal maximum value depends on the modulation scheme currently in use. One among four different modulation schemes can be selected by setting a value in the SV_MODE field of the MCC_CONFIG.PWM register. *Table 23* provides a brief description of the different modulation schemes along with the internal maximum output voltage value.

Table 23. Space Vector PWM modes

SV_MODE[1:0]	SVPWM Mode	Output voltage internal maximum value	Description
00	DISABLED	16383	Sine PWM mode. Space vector modulation is disabled.
01	HARMONIC	18900	Space vector modulation is enabled, and third harmonic injection is added to the duty cycle. This enables an effective peak voltage higher than the supply voltage VS inside the motor.
01	BOTTOM	18900	Space vector modulation is enabled as in HARMONIC mode. The lowest phase voltage is subtracted from all phases so that one phase is always at 0% duty cycle.
11	BOTTOM_OFFSET	18900	Same as BOTTOM with an optional offset that can be added to all phases using the FLAT_BOTTOM_OFFSET field of the MCC_CONFIG.PWM register.

The default modulation scheme is the HARMONIC mode.

A representation of the generated PWM duty cycle value for the different modulation schemes described above is shown in *Figure 37*. When using SV_MODE = 00_b (DISABLED), a PWM duty cycle of 100% (65535) is generated when the calculated output voltage value peaks at the limit value of 16383. Conversely, when using any other SV_MODE, a PWM duty cycle of 100% is generated when the calculated output voltage peaks at 18900. A PWM duty cycle of 100% represents a voltage equal to the supply voltage VS.

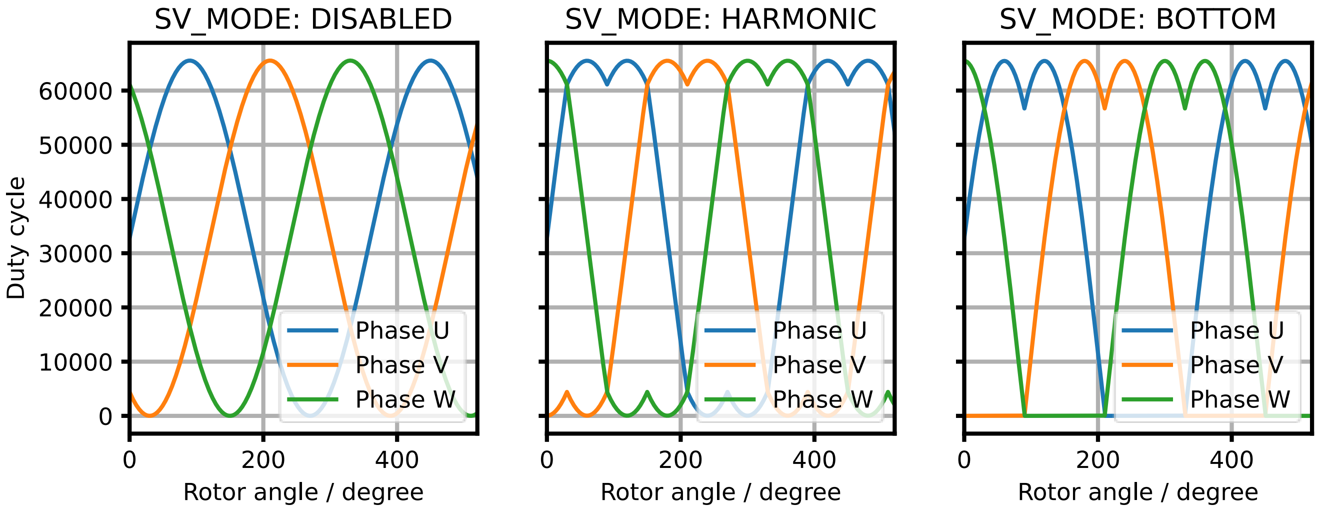


Figure 37. PWM modulation mode

If the automatic switch mode (auto Kirchhoff calculation) of the MEAS_MODE field of the MCC_ADC.I_GEN_CONFIG register is used, the AUTO_KIRCHHOFF_LIM field of the MCC_CONFIG.PWM_PERIOD register must also be set for the PWM engine. To avoid corrupted current measurements due to too high duty cycles, the AUTO_KIRCHHOFF_LIM sets an upper threshold for the duty cycle. If the duty cycle of one phase exceeds the set switch limit, the current for that particular phase will not be measured. Instead, the other two phases will be used to calculate the third one. The maximum switch limit can be calculated with the following formula:

$$AUTO\_KIRCHHOFF\_LIM_{max} = (1 - T_{set} \times f_{PWM}) \times 2^{16}$$

The AUTO_KIRCHHOFF_LIM register should be set below the calculated AUTO_KIRCHHOFF_LIM$_{MAX}$ to avoid corrupted current measurements. The calculated value depends on the configured PWM frequency f$_{PWM}$ and the settling time T$_{set}$ of the CSAs, which in turn is dependent on the value in SLEW_RATE field of the MCC_CONFIG.GDRV register. *Table 24* shows the possible T$_{set}$ values for the different slew rate choices.

Table 24. CSA Settling time T$_{SET}$

SLEW_RATE[1:0]	00	01	10	11
T$_{SET}$ [µs]	2.4	1.8	1.3	1.0

The minimum limit on the other hand can be calculated with the equations below, depending on the selected SV_MODE.

- SV_MODE= DISABLED

$$AUTO\_KIRCHHOFF\_LIM_{min} = 24576 + 1.5 \times PID\_UQ\_UD\_LIMITS$$

- SV_MODE= HARMONIC

$$AUTO\_KIRCHHOFF\_LIM_{min} = 32768 + 1.5 \times PID\_UQ\_UD\_LIMITS$$

- SV_MODE= BOTTOM

$$AUTO\_KIRCHHOFF\_LIM_{min} = 3 \times PID\_UQ\_UD\_LIMITS$$

The minimum switch limit depends on the active voltage limit configured in the PID_UQ_UD_LIMITS field of the FOC.PID_UQ_UD_LIMITS register. The minimum limit is needed to ensure that there is no point in the waveform when two phases are above the switch limit at the same time. Therefore, it is recommended to set the AUTO_KIRCHHOFF_LIM value above the calculated minimum and below the calculated maximum. If this is not possible due to a minimum value being higher than the maximum, the output voltage limit set in the PID_UQ_UD_LIMITS field must be reduced. For this, it is important to know that the voltage limit has an internal maximum value that also depends on the selected SV_MODE as described above. Adapting PID_UQ_UD_LIMITS above this value has no effect.

Current Control Loop

The current control loop in the TMC6460 comprises PI controller for Torque and Flux (i_Q and i_D, respectively), input Clarke and Park transformations, output voltage limiter and output voltage inverse Park and inverse Clarke transformations with the space vector PWM block. The overall structure of the current control loop is shown in *Figure 38*.

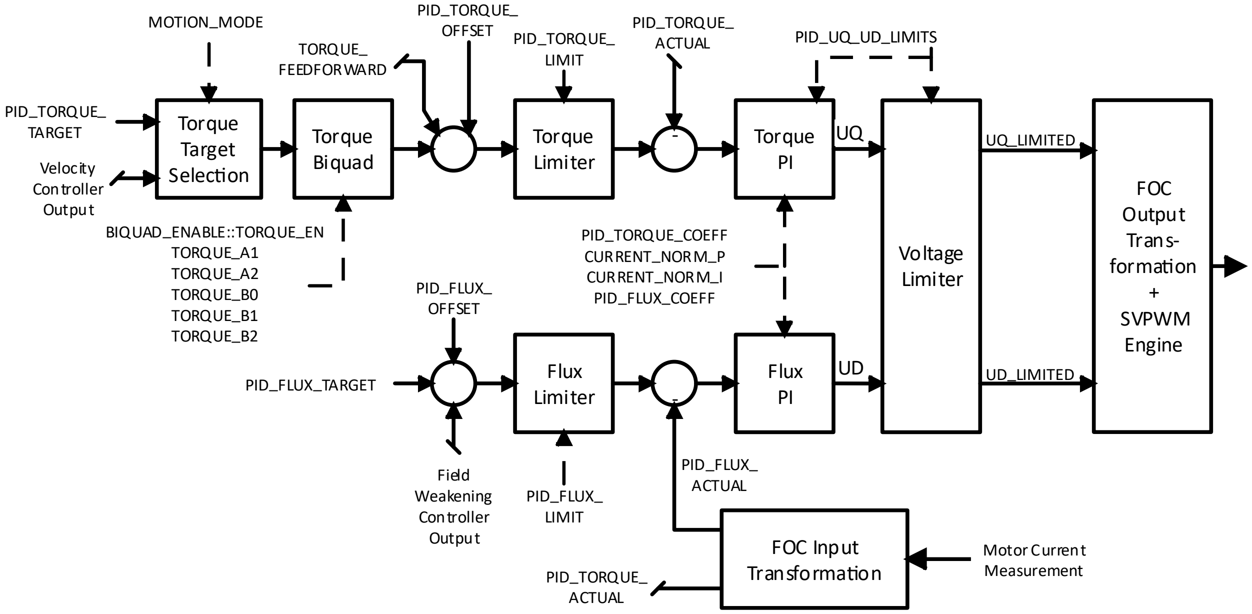


Figure 38. Torque and Flux control loop.

The input transformation blocks perform the classic transformations. Interim results can be monitored in IALPHA and IBETA fields of the FOC.FOC_IBETA_IALPHA register. The phi_e angle is used for the Park transformation and it is computed in the feedback engine during normal operation.

The main input target values for the current controllers can be switched according to MOTION_MODE field of the MCC_CONFIG.MOTOR_MOTION register. A target is set either through the direct targets in the PID_TORQUE_TARGET and PID_FLUX_TARGET fields of FOC.PID_TORQUE_FLUX_TARGET register or through the output of the velocity controller and the output of the field-weakening controller. It is possible to filter the main input for the torque controller through the implemented biquad filter.

Configurable offsets can be applied for the current controllers through the PID_TORQUE_OFFSET and PID_FLUX_OFFSET fields of the FOC.PID_TORQUE_FLUX_OFFSET register.

Additionally for the torque controller, a feedforward component from the MCC_CONFIG.TORQUE_FEEDFORWARD register is added. This summed up feedforward component integrates three torque-related feedforward components: acceleration, viscous friction and Coulomb friction. The independent torque feedforward components are in turn generated from the ramp generator velocity and acceleration outputs.

Each torque feedforward must be independently enabled and configured. They can be enabled through the TORQUE_FF_ACC_EN, TORQUE_FF_VISC_FRIC_EN and TORQUE_FF_COULOMB_FRICTION_EN bits of the MCC_CONFIG.MOTOR_MOTION register, and configured through the MCC_CONFIG.TORQUE_FF_ACC_CONFIG, MCC_CONFIG.TORQUE_FF_VISC_FRIC_CONFIG and MCC_CONFIG.TORQUE_FF_COLOUMB_FRIC registers.

The current controllers' summed up target values are constrained by the limit values in PID_TORQUE_LIMIT and PID_FLUX_LIMIT fields of the FOC.PID_TORQUE_FLUX_LIMITS register. The limited target values are available in PIDIN_TORQUE_TARGET_LIMITED and PIDIN_FLUX_TARGET_LIMITED fields of the FOC.PIDIN_TORQUE_FLUX_TARGET_LIMITED register.

The resulting target for the current controllers is compared against their actual values to calculate the control current errors, which can be read in FOC.PID_TORQUE_ERROR and FOC.PID_FLUX_ERROR registers.

The torque and flux PI controllers have their own proportional and integral coefficients to be set through the TORQUE_P, TORQUE_I fields of the PID_TORQUE_COEFF register and through FLUX_P, FLUX_I fields of the PID_FLUX_COEFF register. The proportional and integral coefficients for the current controller can be normalized according to the fixed-point formats of Q16.0 or Q8.8 (default). This is configured through the CURRENT_NORM_I and CURRENT_NORM_P fields of the FOC.PID_CONFIG register for the integral and proportional coefficients, respectively. The normalization factors are shared for the torque and flux controllers.

When the current controller is active, its integrator values for torque and flux can be read and overwritten through the FOC.PID_TORQUE_INTEGRATOR and FOC.PID_FLUX_INTEGRATOR registers.

The outputs of the current controllers, UQ and UD fields of the FOC.FOC_UQ_UD register, are forwarded to the voltage limiter. Although these controller's outputs are already limited by the FOC.PID_UQ_UD_LIMITS register field, an additional limit stage is processed afterwards. The voltage limiter constrains the voltage values in a way that UD always gets the priority over UQ in demand, while also making sure the voltage vector length does not exceed the defined duty cycle limit in register FOC.PID_UQ_UD_LIMITS. The limited output voltages in UQ_LIMITED and UD_LIMITED fields of the FOC.FOC_UQ_UD_LIMITED register are then forwarded to the output transformations.

The inverse Park transformation uses the phi_e angle to calculate the value in UBETA and UALPHA fields of the FOC.FOC_UBETA_UALPHA register. These values are then forwarded to the inverse Clarke transformation which in turn computes the values of UU, UV, and UW fields of the FOC.FOC_UW_UU and FOC.FOC_UV registers.

The actual motor voltage and total motor current can be monitored through I_S_ACTUAL and U_S_ACTUAL fields in the FOC.U_S_ACTUAL_I_S_ACTUAL register. The actual target current value can be found in the I_T_ACTUAL field of the FOC.I_T_ACTUAL register. These values are calculated as follows:

$$I\_S\_ACTUAL = \sqrt{PID\_TORQUE\_ACTUAL^2 + PID\_FLUX\_ACTUAL^2}$$

$$U\_S\_ACTUAL = \sqrt{U\_Q\_LIMITED^2 + U\_D\_LIMITED^2}$$

$$I\_T\_ACTUAL = \sqrt{PIDIN\_TORQUE\_TARGET\_LIMITED^2 + PIDIN\_FLUX\_TARGET\_LIMITED^2}$$

The output power can be read from the FOC.P_MOTOR register, which is calculated as the product of the values in I_S_ACTUAL and U_S_ACTUAL fields.

Field Weakening Controller

The TMC6460 is equipped with a field-weakening controller that is used to inject negative flux current (i_D) above a configurable voltage limit. This can effectively counteract the motor's back-EMF voltage, allowing the motor to run at higher velocities, at the expense of increasing the current consumption and/or reducing torque (i_Q).

When using this feature, it is important to monitor the total current in the motor, as this might be thermally relevant. The total motor current, calculated from both i_D and i_Q components, can be read from the I_S_ACTUAL field of the FOC.U_S_ACTUAL_I_S_ACTUAL register.

In *Figure 39*, the behavior of the field-weakening controller over velocity is displayed. The voltage in the motor increases as the motor velocity increases. Up until the configured voltage limit, denoted here as nominal voltage, it is possible to use the current exclusively for torque generation (i_Q). As the voltage limit is reached, the controller starts using a proportion of the current to inject the negative flux (i_D). This results in a reduction in torque but also allows for higher velocities of the motor.

The structure of the field-weakening controller is illustrated in *Figure 40*.

The input of the field-weakening controller is calculated as the difference between the set voltage limit in U_S_MAX field of the FOC.PID_U_S_MAX register, and the actual motor voltage from U_S_ACTUAL field of the FOC.U_S_ACTUAL_I_S_ACTUAL register. When U_S_ACTUAL exceeds U_S_MAX, the controller builds a negative output value via the proportional and integral coefficients in FIELDWEAK_P and FIELDWEAK_I fields of the FOC.PID_FIELDWEAK_COEFF register. Both coefficients are interpreted as a Q8.8 fixed-point format.

The output of the field-weakening controller is combined with other user-defined flux values and shown in register FOC.PIDIN_FLUX_TARGET. This value is constrained according to the set limit value in PID_FLUX_LIMIT field of the PID_TORQUE_FLUX_LIMITS register. The limited value is then used as an input for the flux controller and available for reading in the PIDIN_FLUX_TARGET_LIMITED field of the FOC.PIDIN_TORQUE_FLUX_TARGET_LIMITED register.

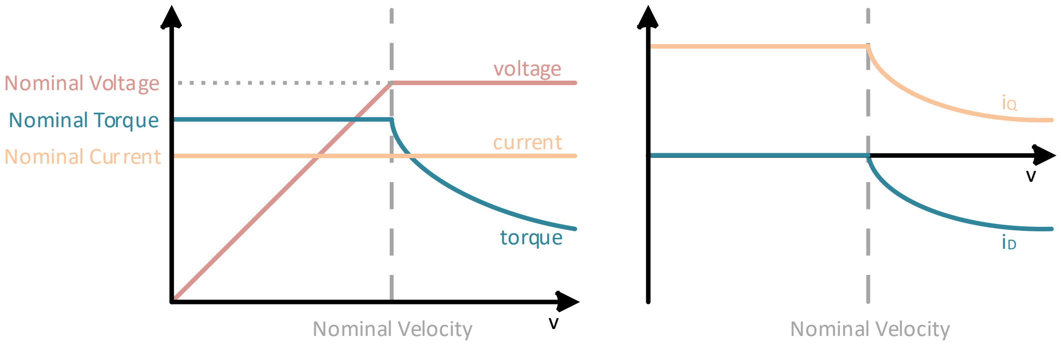


Figure 39. Field-weakening behavior

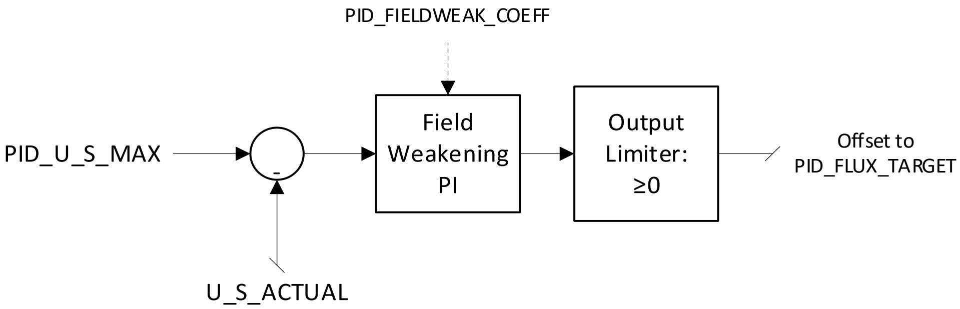


Figure 40. Field-weakening control loop

The use of the field-weakening controller output as input of the flux controller is displayed in *Figure 38*.

Velocity Control Loop

The velocity controller in TMC6460 is implemented as a PI controller with limitation for input and output values. Its structure is shown in *Figure 41*.

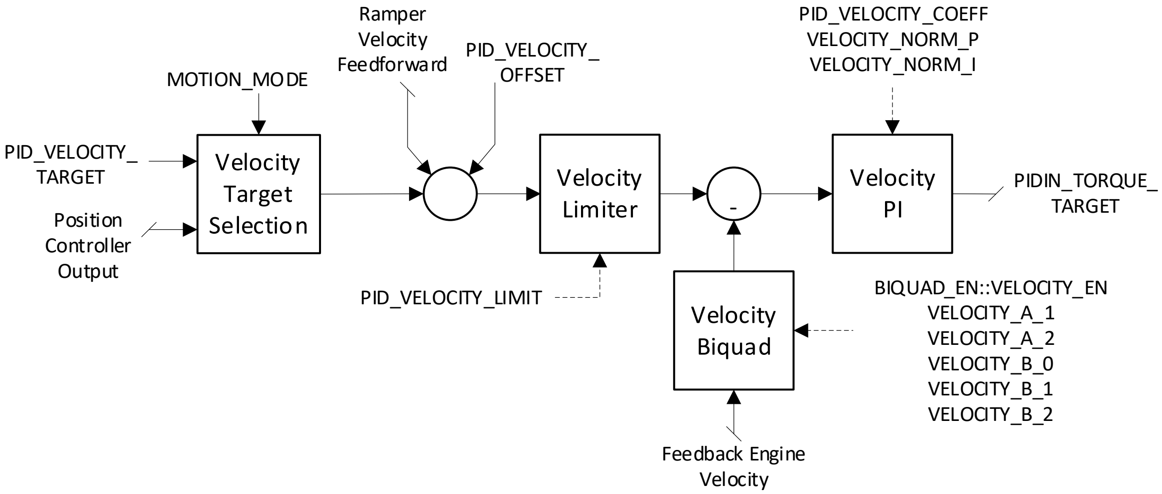


Figure 41. Velocity control loop.

The actual velocity FOC.PID_VELOCITY_ACTUAL register is the output of the velocity biquad filter, which in turn comes from the feedback engine velocity acquisition block.

The main input target value for the velocity controller can be switched according to MOTION_MODE field of the MCC_CONFIG.MOTOR_MOTION register. If the motion mode is set to position mode, then the input target comes from the output of the position controller. If the motion mode is set to velocity mode, then the input target comes from the RAMPER.VELOCITY register, if the ramp generator is enabled; or directly from FOC.PID_VELOCITY_TARGET register, if the ramp generator is disabled. Note however, that the user-defined velocity target for velocity mode is always specified through the FOC.PID_VELOCITY_TARGET register regardless of whether the ramp generator is enabled or not. The resulting input target can be read out from the FOC.PIDIN_VELOCITY_TARGET register.

An offset can be applied for the velocity controller through the FOC.PID_VELOCITY_OFFSET register. This is meant to be used as a user-defined feedforward value.

Additionally, a feedforward component from the RAMPER.V_ACTUAL register of the ramp generator block can be added. This is done when the VELOCITY_FF_EN bit of the MCC_CONFIG.MOTOR_MOTION register is set.

The velocity controller's summed up target value is constrained by the limit values in FOC.PID_VELOCITY_LIMIT register. The limited target value is available in FOC.PIDIN_TARGET_VELOCITY_LIMITED register.

The resulting limited input target for the velocity controller is compared against the actual velocity to calculate the control velocity error, which can be read in FOC.PID_VELOCITY_ERROR register.

The proportional and integral coefficients for the velocity controller can be specified through the VELOCITY_P and VELOCITY_I fields of the FOC.PID_VELOCITY_COEFF register. The proportional coefficient can be normalized according to the fixed-point format of Q16.0, Q8.8 (default), Q0.16 and Q0.24, which is set in the VELOCITY_NORM_P field of the FOC.PID_CONFIG register. Similarly, the integral coefficient can be normalized according to the fixed-point format of Q8.8, Q0.16 (default), Q0.24 and Q0.32, which is set in the VELOCITY_NORM_I field of the same register.

In addition to the individual normalization factors, the total output of the velocity controller can be shifted setting a value in the VELOCITY_SHIFT field of the FOC.PID_CONFIG register. The limit value in the FOC.PID_VELOCITY_LIMIT register is shifted accordingly.

When the velocity controller is active, its integrator value can be read and overwritten through the register FOC.PID_VELOCITY_INTEGRATOR.

The following equations represent the output of the velocity controller neglecting constraints from the limit value:

$$PIDIN\_TORQUE\_TARGET = \frac{\frac{PID\_VELOCITY\_ERROR \times VELOCITY\_P}{2^{VELOCITY\_NORM\_P}} + PID\_VELOCITY\_INTEGRATOR}{2^{VELOCITY\_SHIFT}}$$

$$with\ PID\_VELOCITY\_INTEGRATOR = \frac{PID\_VELOCITY\_ERROR \times VELOCITY\_I}{2^{VELOCITY\_NORM\_I}}$$

$$with\ VELOCITY\_NORM\_P \in \{0,8,16,24\}, \qquad VELOCITY\_NORM\_I \in \{8,16,24,32\}, \qquad VELOCITY\_SHIFT \in \{0 \ldots 15\}$$

The velocity controller update rate f_{vel} can be configured through the VELOCITY_SAMPLING field of the FEEDBACK.VELOCITY_FRQ_CONF register, as discussed in the *Frequency velocity meter* section.

Depending on the value in the MOTION_MODE field of the MCC_CONFIG.MOTOR_MOTION register, the output of the velocity controller is shown in the PIDIN_TORQUE_TARGET field of the FOC.PIDIN_TORQUE_FLUX_TARGET register.

Position Control Loop

The position controller in TMC6460 is implemented as a PI controller with limitation for input and output values. A simplified structure of the position controller is shown in *Figure 42*.

The actual position value is shown in register FOC.PID_POSITION_ACTUAL, which comes from the output of the position decoder of the feedback engine.

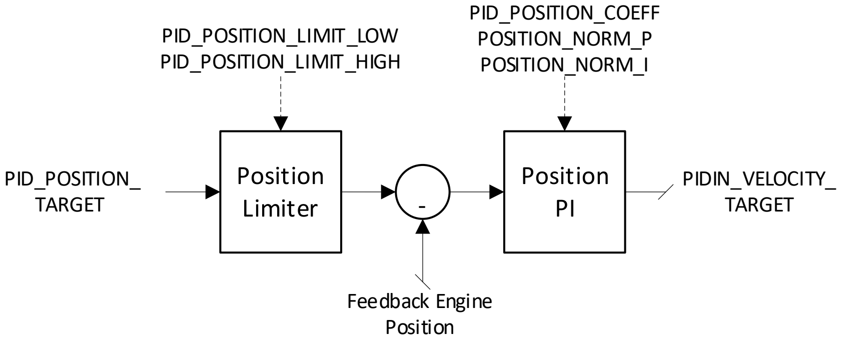


Figure 42. Position control loop.

The position controller's input target value can come from the RAMPER.POSITION register, if the ramp generator is enabled; or directly from FOC.PID_POSITION_TARGET register, if the ramp generator is disabled. Note however, that the user-defined position target is always specified through the FOC.PID_POSITION_TARGET register regardless of whether the ramp generator is enabled or not. The resulting input target for the position controller can be read out from the FOC.PIDIN_POSITION_TARGET register.

The position controller's input target value is constrained by the limit values in FOC.PID_POSITION_LIMIT_LOW and FOC.PID_POSITION_LIMIT_HIGH registers. No limitation is applied if both limit values are identical. The limited target value is available in FOC.PIDIN_POSITION_TARGET_LIMITED register.

The resulting limited target for the position controller is compared against the actual value to calculate the control position error, which can be read in FOC.PID_POSITION_ERROR register.

The proportional and integral coefficients for the position controller can be specified through the POSITION_P and POSITION_I fields of the FOC.PID_POSITION_COEFF register. The proportional coefficient can be normalized according to the fixed-point format of Q16.0, Q8.8 (default), Q0.16 and Q0.24, which is set in the POSITION_NORM_P field of the FOC.PID_CONFIG register. Similarly, the integral coefficient can be normalized according to the fixed-point format of Q8.8, Q0.16 (default), Q0.24 and Q0.32, which is set in the POSITION_NORM_I field of the same register.

When the position controller is active, its integrator value can be read and overwritten through the register FOC.PID_POSITION_INTEGRATOR.

The following equations represent the output of the position controller neglecting constraints from the limit value:

$$PIDIN\_VELOCITY\_TARGET = \frac{PID\_POSITION\_ERROR \times POSITION\_P}{2^{POSITION\_NORM\_P}} + PID\_POSITION\_INTEGRATOR$$

$$PID\_POSITION\_INTEGRATOR = \frac{PID\_POSITION\_ERROR \times POSITION\_I}{2^{POSITION\_NORM\_I}}$$

$$with\ POSITION\_NORM\_P \in \{0, 8, 16, 24\}, \qquad POSITION\_NORM\_I \in \{8, 16, 24, 32\}$$

The position controller's update rate can be configured through POSITION_SAMPLING field of the FOC.PID_CONFIG register, as a down sampled frequency of the velocity controller's update rate f_{vel}:

$$f_{pos} = \frac{f_{vel}}{POSITION\_SAMPLING + 1} = \frac{f_{PWM}}{(VELOCITY\_SAMPLING + 1) \times (POSITION\_SAMPLING + 1)}$$

The VELOCITY_SAMPLING field is part of the FEEDBACK.VELOCITY_FRQ_CONF register. Its configuration is discussed in the *Frequency velocity meter* section.

Depending on the value in the MOTION_MODE field of the MCC_CONFIG.MOTOR_MOTION register, the output of the position controller is shown in the FOC.PIDIN_VELOCITY_TARGET register.

It is possible to neglect small position errors once the motor reaches the target position. In this case, the control cascade can enter an idle state where the position controller is disabled, and output of the velocity controller is held static. To enable this feature, registers FOC.PID_POSITION_TOLERANCE and FOC.PID_POSITION_TOLERANCE_DELAY must be configured accordingly. The ramp generator must also be enabled for this feature to work.

The control cascade enters the idle state when the following conditions are met: 1) the ramp generator's RAMP.POSITION must have reached the target position, and 2) the absolute value of the controller's error FOC.PID_POSITION_ERROR must be less than the set FOC.PID_POSITION_TOLERANCE value for at least the number of PWM cycles defined by the value in FOC.PID_POSITION_TOLERANCE_DELAY. As soon as the position error value leaves the tolerance band, or the target position changes, the position control is activated again to ensure normal controller behavior.

Biquad Filters

The TMC6460 provides three biquad filters, with two of them distributed in the control cascade and one that can be used from the integrated I/O Controller. For the control cascade, the filters are provided for the velocity measurement, and for the velocity controller output, that is, the input for the torque controller.

The velocity biquad filter processes the actual velocity as calculated in the feedback engine. Its input comes from either of the velocity meters, as configured through the VELOCITY_SELECTION field of the FEEDBACK.OUTPUT_CONF register. The velocity biquad filter is triggered at the velocity controller update rate f_{vel}. It is enabled by default, as seen in the VELOCITY_EN bit of the BIQUAD.BIQUAD_EN register. The filter's coefficients can be set through the BIQUAD.VELOCITY_A1, BIQUAD.VELOCITY_A2, BIQUAD.VELOCITY_B0, BIQUAD.VELOCITY_B1 and BIQUAD.VELOCITY_B2 registers.

The torque biquad filter processes the output of the velocity controller. The torque biquad filter is also triggered at the velocity controller update rate f_{vel}. It is enabled through the TORQUE_EN bit of the BIQUAD.BIQUAD_EN register. The filter's coefficients can be configured through the BIQUAD.TORQUE_A1, BIQUAD.TORQUE_A2, BIQUAD.TORQUE_B0, BIQUAD.TORQUE_B1 and BIQUAD.TORQUE_B2 registers.

All the coefficients are normalized according to the fixed-point format of Q4.20. *Figure 43* shows a block diagram of the included biquad filters. Their transfer function can be described as follows:

$$G(z) = \frac{y(n)}{x(n)} = \frac{b_0 + b_1 \times z^{-1} + b_2 \times z^{-2}}{1 - a_1 \times z^{-1} - a_2 \times z^{-2}}$$

For this, the sign of the A1 and A2 coefficients must be considered.

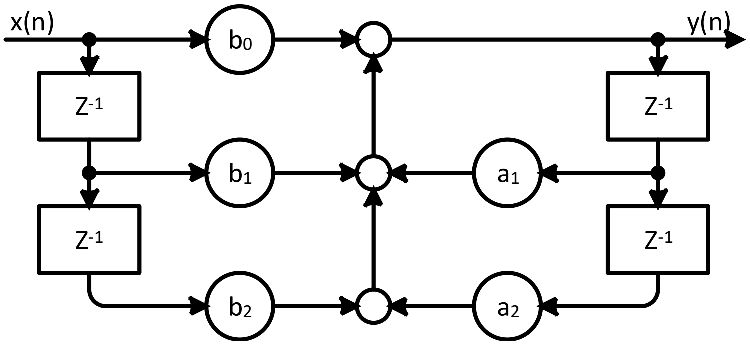


Figure 43. Biquad Filter Structure

Tuning tools for the biquad filters are provided within the *TMCL-IDE package*.

Direct Motion Modes

In addition to the standard closed-loop motion modes, the TMC6460 provides some direct control modes. These are offered mainly for system setup and for testing purposes, as they bypass some of the control loops within the IC.

PWM off and PWM on modes

PWM off is the default mode after the TMC6460 is reset. This mode can be manually selected by setting MOTION_MODE = 0 (PWM_OFF) in the MCC_CONFIG.MOTOR_MOTION register. In this mode, the PWM engine and the whole control cascade are disabled. The output stage is at tristate.

By setting MOTION_MODE = 1 (PWM_ON) in the MCC_CONFIG.MOTOR_MOTION register, the PWM engine and power stage are activated, and all half bridges start to toggle to produce the PWM output with frequency f_{PWM}. In this mode, changes in the MAX_COUNT field of the MCC_CONFIG.PWM_PERIOD register, or in the CHOP field of the MCC_CONFIG.PWM register result in a change in the generated PWM outputs. Internal processes that rely on the PWM triggers according to f_{PWM} are also made available. In this mode, the PWM duty cycle is fixed to 50%, and the different controllers in the cascade are not used. This means that no current should flow through the motor in this mode, and no control of the motor is possible.

External PWM duty cycle mode

During normal operation, the duty cycle of the three PWM outputs is calculated automatically based on the output voltage of the current controller. By using the external PWM duty cycle mode, it is also possible to bypass all the control loops and directly set a specific duty cycle for each output channel.

The external PWM duty cycle mode is enabled by setting the field MOTION_MODE = 9 (PWM_EXT) in the MCC_CONFIG.MOTOR_MOTION register. In this mode, the duty cycle of the three half-bridges can be directly set through the PWM_U, PWM_V and PWM_W fields of the EXT_CTRL.PWM_V_U and EXT_CTRL.PWM_W registers. Current measurements are still carried out, but no control inside the TMC6460 is performed.

By setting a value between 0 and 65535 in the PWM_U, PWM_V and PWM_W fields, a duty cycle between 0% and 100% is generated for each channel accordingly.

Open Loop Motion Modes

If no feedback for the motor position is available, the TMC6460 can still drive a motor in open loop configuration. In this mode, the ramp generator is used as a source for the phi_e angle. This is generally useful only during setting up of a real angle feedback source.

If the RAMP_EN and RAMP_USE_PHI_E bits of the MCC_CONFIG.MOTOR_MOTION register are set, the generated angle in the PHI_E field of the RAMPER.PHI_E register is used for motor commutation in the FOC engine. The ramp generator can be configured for velocity or position mode through the RAMP_MODE field of the MCC_CONFIG.MOTOR_MOTION register. Depending on the chosen ramp mode, a corresponding target value can be set in the FOC.PID_VELOCITY_TARGET or FOC.PID_POSITION_TARGET registers.

More information on the ramp generator, including real-world unit conversions for open loop operation can be found in the *Ramp Generator* chapter.

External voltage open loop mode

In normal FOC operation, the motor angle feedback and the current measured for each phase are used to calculate the torque and flux components (i_Q and i_D) of the current running through the motor. From the current controllers, corresponding u_Q and u_D voltage components are calculated, which in turn are used to drive the PWM duty cycle of the output stages. In contrast to normal operation, the external voltage mode provides a way to directly set the u_Q and u_D voltage components.

The external voltage mode is enabled by setting the field MOTION_MODE = 11 (VOLTAGE_EXT) in the MCC_CONFIG.MOTOR_MOTION register. In this mode, the voltage can be manually set through the UQ and UD fields of the EXT_CTRL.VOLTAGE register. Current measurements are still carried out, but no current control is performed. The voltage limiter and output transformation parts of the PWM engine are active in this mode. Which means a duty cycle according to the set voltage values is generated.

While in external voltage mode, the open loop mode using the ramp generator can be enabled as described above. Then, a flux voltage value in the UD field of the EXT_CTRL.VOLTAGE register can be set. If the set voltage value is high enough, the rotor will follow ramp generator open loop angle and the corresponding generated magnetic field. The higher the voltage value, the closer the rotor will follow the generated angle, but the more current that is forced through the motor’s coils. If the rotor cannot follow the generated magnetic field, the voltage value must be increased.

When using this configuration for setting up of an angle feedback source, it is recommended to set UQ to zero and UD to a positive value that is high enough to allow the motor to turn smoothly.

Torque open loop mode

The torque open loop mode is different from the external voltage mode, in that the current controller is active. However, the phi_e used for FOC is not coming from a real angle feedback source, but rather from the ramp generator as described above. This mode has the advantage over the external voltage mode that a known, measurable and controlled current can be applied to the motor instead of just a fixed voltage value, which might, or might not, result in the desired current.

The torque open loop mode is enabled when MOTION_MODE = 2 (TORQUE) in the MCC_CONFIG.MOTOR_MOTION register and the RAMP_EN and RAMP_USE_PHI_E bits of the MCC_CONFIG.MOTOR_MOTION register are set.

When using this configuration for setting up of an angle feedback source, it is recommended to set PID_TORQUE_TARGET to zero and PID_FLUX_TARGET to a positive value that is high enough to allow the motor to turn smoothly.

It is important to note that, since the current controller is used for this mode, it is necessary to configure the torque and/or flux controllers' coefficients as described in the *Current Control Loop* section before using this mode successfully.

Standard Motion Modes

Besides the direct and open loop motion modes, the MCC can be operated in various standard motion modes. This is configured through the MOTION_MODE field of the MCC_CONFIG.MOTOR_MOTION register.

In torque mode (MOTION_MODE = 2), the current controllers (torque and flux), as well as the FOC transformations are enabled. The register fileds PID_TORQUE_TARGET and PID_FLUX_TARGET are typically used to provide target values to the controllers.

In velocity mode (MOTION_MODE = 3), the velocity controller is also activated. Its output (PIDIN_TORQUE_TARGET) is forwarded to the torque controller. Register field PID_TORQUE_TARGET is no longer used as a target for the torque controller. The target for the velocity controller is provided through the PID_VELOCITY_TARGET register field.

In position mode (MOTION_MODE = 4), the position controller is also activated. Its output (PIDIN_VELOCITY_TARGET) is forwarded to the velocity controller. Register PID_VELOCITY_TARGET is no longer used as target for the velocity controller. The target for the position controller is provided through the PID_POSITION_TARGET register field.

Note that register field PID_FLUX_TARGET remains to be used as a target for the flux controller even for velocity and position modes. This field should therefore be set to 0 for normal operation in velocity and position modes.

Switching between motion modes

The safest way to switch between two different motion modes during operation is to ensure that the motor is in standstill and that the target values of the mode being switched to are deliberately chosen. Additionally, if the ramp generator is used when switching between position and velocity modes, it is crucial to also change the ramp mode accordingly.

In general, it is possible to switch between the three standard motion modes during operation and while the motor is moving. However, some considerations must be taken to prevent abrupt movements or big current spikes. The exact behavior when switching depends on the origin and target motion modes and on the value of the OVERWRITE_TARGET bit of the FOC.PID_CONFIG register at the moment of the change.

The general idea when OVERWRITE_TARGET is set, is to allow a jerk free transition between the two motion modes. This is achieved by an automatic overwriting of the appropriate integrator and/or target values of the different controllers. The change of the ramp generator mode must be observed even if OVERWRITE_TARGET is set.

When switching from one motion mode to another on a “lower” level in the control cascade (velocity→torque, position→torque or position→velocity) while OVERWRITE_TARGET is set, the particular target register of the motion mode being changed to is automatically overwritten. Concretely, when switching from position or velocity modes to torque mode, the PID_TORQUE_TARGET register field is automatically overwritten with the internal torque target value at the moment of the change. When changing from position to velocity mode, the PID_VELOCITY_TARGET register field is automatically overwritten with internal velocity target value.

In contrast, when switching from one motion mode to another on a “higher” level in the control cascade (torque→velocity, torque→position or velocity→position), the particular target register of the motion mode being switched to must be manually assigned to ensure a smooth transition, even if OVERWRITE_TARGET is set. For the particular cases of changing from torque mode to position or velocity modes, the velocity integrator register is automatically overwritten with the current value of the torque target register to aid in minimizing the jerk.

PRBS Motion Modes

Several modes are available for system identification, the so-called pseudo random binary sequence (PRBS) modes. These modes can also be selected through the MOTION_MODE field of the MCC_CONFIG.MOTOR_MOTION register. In these modes, a PRBS generator is used to create an input signal for the corresponding control mode. The PRBS generator includes a feedback shift register of 20 bits (bits 0 to 19) where the result of the XOR operation of bit 16 and 19 is shifted back into the register, as represented in *Figure 44* and the expression below.

$$PRBS\_REG[0] = PRBS\_REG[16] \quad xor \quad PRBS\_REG[19]$$

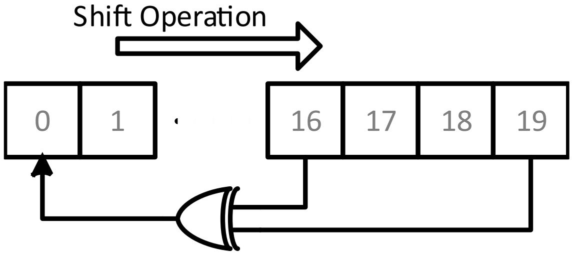


Figure 44. PRBS shift register

PRBS_REG is initialized with all bits set to 1 when FOC.PRBS_AMPLITUDE register equals zero. The shift operation is executed in sync with f_{PWM}, with a configurable down sampling rate through FOC.PRBS_DOWN_SAMPLING_RATIO register. The PRBS shift frequency is thereby defined as:

$$f_{PRBS} = \frac{f_{PWM}}{PRBS\_DOWN\_SAMPLING\_RATIO + 1}$$

After each shift, the output of the PRBS generator is formed. If PRBS_REG[0] is 1, then PRBS_AMPLITUDE is used as the output. If PRBS_REG[0] is 0, then -PRBS_AMPLITUDE is used. In *Table 25* the various modes are further described.

Table 25. PRBS motion modes

MOTION_MODE	PRBS output on	Use case
PRBS_UD	UD	Identification of flux controller plant
PRBS_FLUX	TARGET_FLUX	Identification of flux controller bandwidth
PRBS_TORQUE	TARGET_TORQUE	Identification of torque controller bandwidth
PRBS_VELOCITY	TARGET_VELOCITY	Identification of velocity controller bandwidth
PRBS_POSITION	TARGET_POSITION	Identification of position controller bandwidth

As stated before, these modes are used for identification of the controller's characteristics and not used for motion control.

DC Motor Mode

The TMC6460 can also be utilized to drive a DC motor instead of a 3-phase BLDC motor. This mode is enabled by setting the field MOTOR_TYPE = 1 (DC) in the MCC_CONFIG.MOTOR_MOTION register.

The DC motor only uses the first two output channels OUT1 and OUT2. The third motor output is disabled.

For the current measurement, both channels are measured redundantly. The results are shown in the ADC.I2_I1_RAW register. Only one of the measured currents will be used for current control depending on the set auto Kirchhoff limit. For this, the AUTO_KIRCHHOFF_LIM field of the MCC_CONFIG.PWM_PERIOD register must be set to around 50% duty cycle (32768) if the MEASUREMENT_MODE is set to AUTOMATIC_SWITCH (MCC_ADC.I_GEN_CONFIG register). The selected current for current control can be read in the IU register field (MCC_ADC.IW_IU register). None of the other phases are used while in DC motor mode.

The PWM modulation can also be adjusted using the SV_MODE field of the MCC_CONFIG.PWM register. Since it is not possible to use the third harmonic injection with a DC motor, the options DISABLED, HARMONIC and BOTTOM_OFFSET always generate the default modulation around 50% duty cycle. The BOTTOM mode is also available and generates a modulation where one of the two outputs is always zero and connected to ground. Refer to *PWM Engine* section for more details about the PWM modulation.

Since the concept of an electrical angle (phi_e) does not exist for DC motors, the input and output transforms of the FOC are simply bypassed for the DC motor mode. Instead, the torque current i_Q is simply assumed to be equal to the direct phase current measurement as shown in the IU field, and no flux current i_D is calculated.

For this reason, only the torque controller is used to regulate the DC motor's current. If operating in torque mode, the target must therefore be set through the corresponding PID_TORQUE_TARGET register field.

Similarly, only the output voltage u_Q of the torque controller is used for the PWM generation. Therefore, when operating in external voltage mode, the target output voltage must be set in the corresponding UQ register field.

PRELIMINARY

Other processes that normally rely on the phi_e angle are also different in DC motor mode. For instance, it is no longer possible to use the open loop modes to turn the motor in sync with the ramp's generated phi_e angle. Also, the process required for alignment of the feedback's angle with the internal phi_e is no longer needed.

Besides that, the velocity and position motion modes can be used as usual. This is because the velocity and position controllers do not rely on a phi_e angle, but rather on the mechanical angle of the motor as provided by an external angle feedback source.

Power Stage Configuration

Only a few settings are required to be set for the proper operation of the internal power driver stage. Motor current considerations, including the current sense amplifier gain and the low side ON-resistance value, are discussed in the *Current Acquisition* section.

The slew rate of the output stage can be changed by setting a value in the SLEW_RATE field of the MCC_CONFIG.GDRV register. *Table 26* shows the output slew rate values for the different SLEW_RATE options.

Table 26. SLEW_RATE register field

SLEW_RATE[1:0]	00	01 (default)	10	11
Output Slope [V/µs]	100	200	400	800

To enable the driver stage and enter the driver on state, set the DRV_EN_BIT bitfield. The DRV_EN pin must also be asserted for this to happen. The GDRV_ON_STATE bit of the CHIP.STATUS_FLAGS register can be used as an indication on whether the system is in driver on state. The SYS_READY_STATE bit of the same register indicates if the system is in driver off state.

Some driver stage fail events lead to an automatic deactivation of the driver stage. In most cases, the DRV_EN_BIT bitfield is also automatically cleared. This ensures a safe and user-controlled reactivation of the driver stage.

Specific scenarios on which the DRV_EN_BIT bitfield is not automatically cleared are described in the sections below.

Overcurrent Protection

The TMC6460 features an overcurrent protection (OCP) mechanism to protect the internal FETs in case of detected external short circuits or big current peaks. Two overcurrent protection modes are provided. The currently active overcurrent protection mode is set through the OCP_DETECTION_MODE bit of the MCC_CONFIG.GDRV register. A description of each mode can be found on *Table 27*.

Table 27. OCP modes

OCP Control Mode	OCP_DETECTION_MODE	Description
Current	0	Overcurrent detection based on measured current through the power FETs.
Voltage	1	Overcurrent detection based on measured V_{DS} of the power FETs.

It is possible to configure the OCP threshold, which is done through the LS_RES_ON field of the MCC_CONFIG.GDRV register. *Table 28* shows the current values at which the OCP is triggered when in current detection mode.

Table 28. Overcurrent Protection Threshold

LS_RES_ON[1:0]	Full Scale Current [A]	OCP Threshold Minimum [A]
00	I_{FS} × (1/4)	I_{FS} × (1/2)
01	I_{FS} × (1/2)	I_{FS}
10	I_{FS} × (3/4)	I_{FS} × (3/2)
11	I_{FS}	I_{FS} × 2

Note that the maximum full scale current I_{FS} = 5A.

In case an overcurrent event occurs, the driver is switched off for the particular PWM cycle. Additionally, fault flags in the MCC_CONFIG.MCC_STATUS register are activated depending on the location of the short as described below:

- Short to ground (with x = 1, 2, 3):
  - SHRT2G_CHx_STATUS is set for as long as the short is active.
  - SHRT2G_CHx_EVENT is set to indicate a short to ground occurred. Write '1' to clear.
  - Only high side driver is disabled.

- Short to VS (with x = 1, 2, 3):
  - SHRT2V_CHx_STATUS is set for as long as the short is active.
  - SHRT2V_CHx_LATCH is set to indicate a short to VS occurred. Write '1' to clear.
  - Low and high side driver are disabled.

All six status bits are merged into the SHRT_FAIL_STATUS bit of the CHIP.STATUS_FLAGS register.

If a short is detected on three consecutive PWM cycles, the whole driver stage is disabled, and the IC will be set to driver off (SYS_READY) state to protect the IC.

By selecting a retry time in the OCP_AUTORETRY field of the MCC_CONFIG.GDRV register, an automatic restart of the driver stage can be attempted after 1ms, 5ms, or 50ms. If this field is set to 11_b (default), no automatic restart is attempted.

A deglitch time can be defined to mask overcurrent states that are detected only for a short time. A deglitch time between 0.3µs and 2.4µs can be chosen through the OCP_DEGLITCH field of the MCC_CONFIG.GDRV register. *Table 29* presents the OCP_DEGLITCH and OCP_AUTORETRY options.

Table 29. Overcurrent Protection Deglitch and Automatic Retry Time

OCP_DEGLITCH[1:0]	Deglitch Time [µs]
00	0.3
01	0.6
10	1.2
11	2.4

OCP_AUTORETRY[1:0]	Retry Time [ms]
00	1
01	5
10	50
11	No retry

Care must be taken when using the auto retry function, as the driver could remain in a cycle of OCP triggering and restarting if the underlying cause of the overcurrent is not resolved.

Soft overcurrent limits

The TMC6460 provides an additional soft overcurrent limit. If the absolute value of any of the raw current measurements in I1, I2, and I3 bitfields of the ADC.I2_I1_RAW and ADC.VM_I3_RAW registers exceed the value in the CURRENT_OVERLOAD field of MCC_ADC.CURRENT_OVERLOAD register, the CURRENT_OVERLOAD_STATUS bit of the MCC_ADC.EVENTS register is set. This bit is also forwarded to the CHIP.STATUS_FLAGS register.

Thermal Protection

The TMC6460 also provides thermal protection. If the driver stage temperature on any phase exceeds 165°C, the corresponding OVERTEMP_CHx_STATUS bit of the MCC_CONFIG.MCC_STATUS register (with x = 1, 2, 3) is set and the driver stage is switched off.

All OVERTEMP_CHx_STATUS bits are merged into the OT_FAIL_STATUS bit of the CHIP.STATUS_FLAGS register.

If the OVERTEMP_LATCH bit of the MCC_CONFIG.GDRV register is not set (default) and the temperature falls below 145°C, then the driver stage is automatically activated, and thus the system is set back to the driver on (GDRV_ON) state. If the OVERTEMP_LATCH bit is set, the OVERTEMP_CHx_EVENT bits of the MCC_CONFIG.MCC_STATUS register must be manually cleared (by writing a '1' to them) before the driver stage can be manually reenabled.

Soft thermal limits

The TMC6460 provides additional soft overtemperature limits for the internal temperature measurement and the external temperature analog input in the TEMP_INT and TEMP_EXT fields of the ADC.TEMP_RAW register.

To configure this feature, set the values of TEMP_INT_LIMIT and/or TEMP_EXT_LIMIT fields of the MCC_ADC.TEMP_LIMITS register. If the internal temperature measurement or the external temperature analog input exceed their set limit, the corresponding TEMP_INT_LIMIT_EXCEEDED and/or TEMP_EXT_LIMIT_EXCEEDED bits of the MCC_ADC.EVENTS register are set.

Both bits are merged into the OVERTEMP_WARN_STATUS bit of CHIP.STATUS_FLAGS register.

Overvoltage Failure and Brake Chopper Output

If the internally measured supply voltage in the VM field of the ADC.VM_I3_RAW register exceeds the value in UPPER_SWITCH_LIM of the MCC_CONFIG.BRAKE_CHOPPPER_LIMITS register, the OV_VM_LIMIT_FAIL_STATUS bit of the CHIP.STATUS_FLAGS register is set. The bit is kept active until the value in VM falls below that in the LOWER_SWITCH_LIM field of the same register. This allows for hysteresis on the VM supply voltage monitoring.

Additionally, the OV_VM_LIMIT_FAIL_STATUS bit is directly connected to the BRAKE output. This way, an attached brake chopper circuit may be driven. Note that for this to work, the BRAKE output must be assigned as an alternate function of one of the pins of the TMC6460. Refer to *Pin Remapping* chapter to see which pins can be used.

Additionally, note that when driving a brake chopper circuit using the BRAKE output, the selected pin must not be directly connected to the driving transistor. Instead, the pin must be connected to an external component such as a gate driver, a Schmitt trigger or a buffer circuit. This is represented in the example application diagrams of *Figure 60* and *Figure 61*.

Output Frequency Limit

The TMC6460 has a fixed output frequency limit of 599Hz. If an output frequency higher than 599Hz is detected, then the driver stage is automatically deactivated, and the VEL_FAIL_STATUS bit of the CHIP.STATUS register is set. The corresponding VEL_FAIL_EVENT bit of the CHIP.EVENTS is also set.

In this case, the output frequency refers to that of the generated sinusoid current as seen on the motor's coils. This means that the velocity of the motor is limited to 599 electrical rev/s. The corresponding velocity limit of the motor in rpm is then calculated as a factor of the number of pole pairs of the motor according to the following formula:

$$v_{599Hz,lim}[rpm] = \frac{599 \times 60}{N_{polepairs}}$$

For example, if a motor has one pole pair, then its velocity limit due to the output frequency limit is 35940rpm. For a four-pole-pair motor, the velocity limit is 8985rpm, and so forth.

Note that the output frequency limit is always calculated with the internal oscillator that runs at f_{OSC} = 15MHz ± 5%. To guarantee that the real output frequency does not exceed 599Hz, the limit is calculated assuming the worst-case scenario of 15MHz + 5%. For this reason, the frequency limit might be triggered before reaching a real output frequency of 599Hz.

For instance, if the internal oscillator for a particular sample runs at 15MHz, then the frequency limit would be triggered at a real output frequency of approximately 570Hz. Similarly, if the internal oscillator for another sample runs at 15MHz - 5%, then the frequency limit would be triggered at a real output frequency of approximately 542Hz.

Zero Current Limitations

Due to the non-ideal switching nature of the power stage in the TMC6460, it is not possible to achieve a state where exactly zero current flows through the motor once the power stage is actively switching. The effects of this limitation can become apparent when at least two power stage channels operate with the same PWM duty cycle. Such a condition can occur, for instance, when the current controller attempts to regulate the current to zero, which may also happen in certain scenarios during velocity or position motion modes.

This limitation might result in small oscillations of the motor currents. The amplitude and frequency of the oscillations depend on the motor impedance, the selected PWM frequency, and the set PI tuning parameters of the control loop cascade. This means that the effects can be more or less apparent for a given application.

The most straightforward method to mitigate the impact of this limitation is to apply a more aggressive PI tuning to the current controller.

Ramp Generator

The ramp generator in the TMC6460 allows for smooth motion based on a single target value. It automatically calculates the motion profile based on the configured ramp parameters to allow for jerk minimized motion. Depending on the type of profile to be generated, the ramp generator can operate in either velocity mode or position mode.

The ramp generator allows for real-time changes in target position or target velocity, also during an ongoing motion profile. Additionally, the ramp generator can be configured to automatically stop the motion based on certain events, such as the detection of a reference switch or when a too big deviation from target values is detected.

The ramp generator is enabled by setting the RAMP_EN bit of the MCC_CONFIG.MOTOR_MOTION register. The ramp mode is configured through the RAMP_MODE bitfield of the same register.

Note that a separate MOTION_MODE field in the MCC_CONFIG.MOTOR_MOTION register is available. As described *above* this field sets the motion mode of the MCC, which is independent from the ramp mode. *Table 30* shows the recommended ramp mode usage for different motion modes. Care should be taken not to use the wrong ramp mode during velocity or position motion modes, as this could result in unwanted movement of the motor.

Table 30. Ramp generator usage for different MOTION_MODE choices

MOTION_MODE choice	Ramp generator usage (RAMP_MODE)	Comment
VOLTAGE_EXT	RAMP_VELOCITY or RAMP_POSITION	Ramp generator needed for motor rotation during open loop operation. See *Open Loop Motion Modes* for details.
TORQUE	RAMP_VELOCITY or RAMP_POSITION	Ramp generator needed for motor rotation during open loop operation. Not used for closed-loop operation.
VELOCITY	RAMP_VELOCITY only	Velocity feedforward must be disabled.
POSITION	RAMP_POSITION only	Velocity feedforward can be used.

A block diagram representation of the controller's input target according to the ramp mode and motion mode is shown in *Figure 45*. When the ramper generator is enabled, ramper values in RAMPER.V_ACTUAL and RAMPER.POSITION registers are used as input target values for the velocity and position controllers, respectively.

The user-defined target values are always set through FOC.PID_VELOCITY_TARGET register (for motion mode velocity) and FOC.PID_POSITION_TARGET register (for motion mode position) regardless of whether the ramp generator is enabled or not. For example, when operating in motion mode velocity, the velocity target is defined through PID_VELOCITY_TARGET. If the ramp generator is disabled, then the value in PID_VELOCITY_TARGET is taken directly as an input to the velocity controller. However, if the ramp generator is enabled, the value in PID_VELOCITY_TARGET is taken as input for the ramp generator, and the value in RAMPER.V_ACTUAL (the output of the ramp generator) is then used as input for the velocity controller.

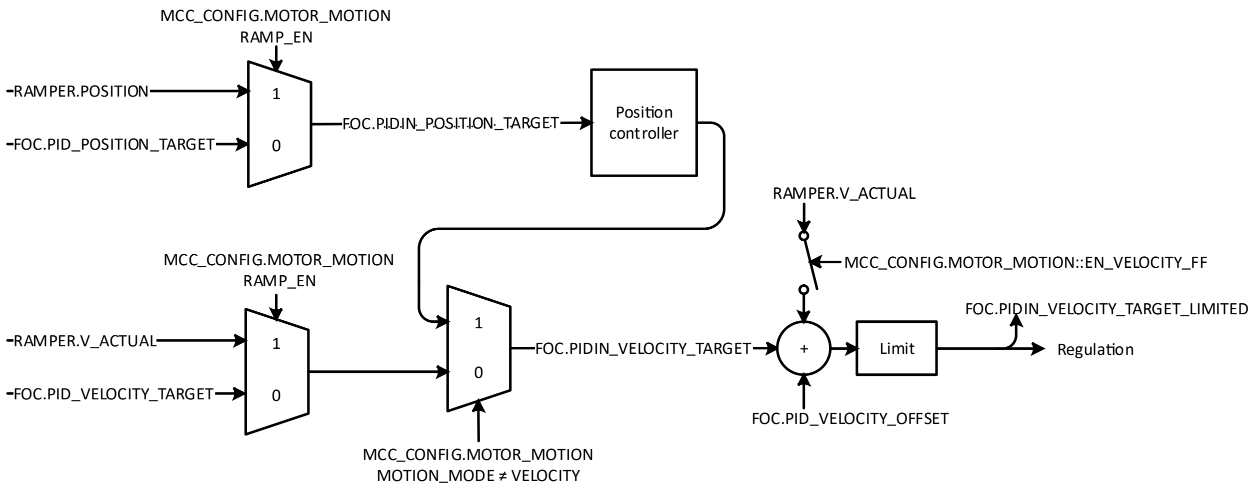


Figure 45. Ramper Controller Target Value Generation

As described in *Current Control Loop* and *Velocity Control Loop* sections, the ramp generator can be used to generate feedforward input for the velocity or the torque controllers. The value in RAMPER.V_ACTUAL register can be used as input for the velocity controller. While values in RAMPER.V_ACTUAL and RAMPER.ACCELERATION registers can be used as input for the torque controller.

Ramp Positioning Mode

The ramp generator is configured for positioning profiles when setting the field RAMP_MODE = 0_b in the MCC_CONFIG.MOTOR_MOTION register. When a new user-defined target position is commanded through the FOC.PID_POSITION_TARGET register, the ramp generator calculates an eight-point velocity ramp that generates a positioning motion profile to reach the target position.

The current velocity and position values from the generated profile can be read from RAMPER.V_ACTUAL and RAMPER.POSITION registers. If the ramp generator is enabled, then the value in RAMPER.POSITION register is used as input for the position controller.

The eight-point velocity ramp is generated according to the defined ramp parameters as described in *Table 31*. Up to six distinct acceleration and deceleration segments plus a constant velocity segment can be differentiated for the eight-point ramp profiles. This is shown in *Figure 46*.

Table 31. Position mode ramp parameters

Register Field	Short Explanation
POSITION	Current position value of the ramp generator. It is used as a target for the position controller when the ramp generator is enabled. After a write access to FOC.PID_POSITION_ACTUAL register, POSITION is automatically overwritten with the new value in PID_POSITION_ACTUAL.
V_ACTUAL	Read-only current velocity value of the ramp generator. It is used as a target for the velocity controller, or as feedforward input to the position controller when the ramp generator is enabled.
ACCELERATION	Read-only current acceleration value of the ramp generator. It is optionally used as feedforward input to torque controller.
PID_POSITION_TARGET	Part of FOC.PID_POSITION_TARGET register. Input target position for the ramp generator. After a write access to FOC.PID_POSITION_ACTUAL register, PID_POSITION_TARGET is overwritten with the new value in PID_POSITION_ACTUAL, unless the KEEP_POS_TARGET bit of the FOC.PID_CONFIG register is set.
V_START	Initial velocity of the eight-point velocity ramp. Used on motion start or when switching driving direction (crossing velocity of 0). It is an unsigned value; the sign of the current velocity is determined by the relative direction of the position target.
V1	Velocity value at which ramp generator switches between A1 and A2 during acceleration phase or between D2 and D1 during deceleration phase. It must be bigger than V_START. Alternatively, it can be set to zero if the A1/D1 sections are not needed. It is an unsigned value; the sign of the current velocity is determined by the relative direction of the position target.
V2	Velocity value at which ramp generator switches between A2 and A_MAX during acceleration phase or between D_MAX and D2 during deceleration phase. It must be bigger than V1. Alternatively, it can be set equal to V1 if the A2/D2 sections are not needed. It is an unsigned value; the sign of the current velocity is determined by the relative direction of the position target.
V_MAX	Maximum velocity of the eight-point velocity ramp. It must be bigger than V2. It is an unsigned value; the sign of the current velocity is determined by the relative direction of the position target.
V_STOP	Velocity value right before stopping. Used when reaching the position target or when switching driving direction (crossing velocity of 0). The ramp velocity is not set to V_STOP if |V_ACTUAL| is smaller than V_STOP when the target position is reached.
A1	Acceleration value between V_START and V1.
A2	Acceleration value between V1 and V2.
A_MAX	Acceleration value between V2 and V_MAX.
D1	Deceleration value between V_START and V1.
D2	Deceleration value between V1 and V2.
D_MAX	Deceleration value between V2 and V_MAX.
T_ZEROWAIT	Part of the RAMPER.TIME_CONFIG register. Hold the velocity at 0 for T_ZEROWAIT × 8.5µs before starting a new movement. The timer starts after reaching the target position, when V_ACTUAL crosses zero, or after a hard/soft stop event.
T_VMAX	Part of the RAMPER.TIME_CONFIG register. Hold a constant velocity for at least T_VMAX × 8.5µs after reaching V_MAX or before starting any acceleration or deceleration due of a change in position target, even if the acceleration sign changes.

Unless otherwise noted, register fields shown on this table belong to the RAMPER register block.

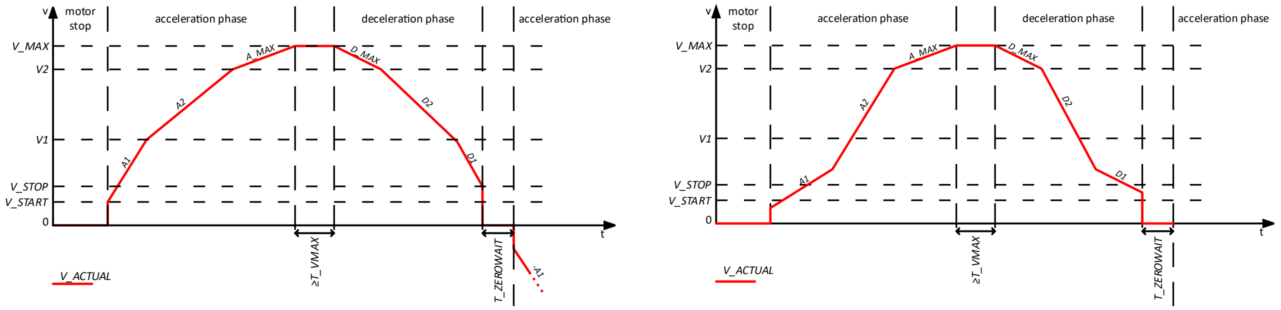


Figure 46. Typical eight-point velocity ramp for positioning motion profiles

Early ramp termination occurs when the distance needed for the constant velocity segment (set in T_VMAX field of RAMPER.TIME_CONFIG register) and the deceleration phase of the ramp profile exceeds the remaining distance given by the difference of FOC.PID_POSITION_TARGET − RAMPER.POSITION. In this case, an adapted ramp profile is calculated, on which not all sections of the eight-point ramp are executed. This is shown in the examples of *Figure 47*.

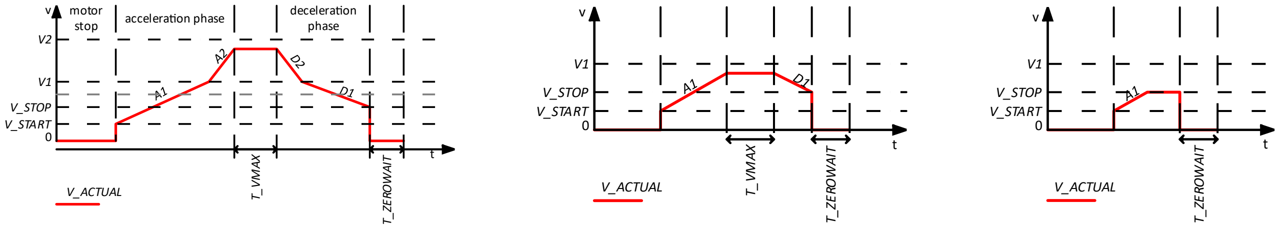


Figure 47. Adapted ramp positioning profile examples

Changes of the target position in FOC.PID_POSITION_TARGET during an ongoing ramp positioning profile may lead to early ramp termination. Different scenarios are illustrated in *Figure 48*.

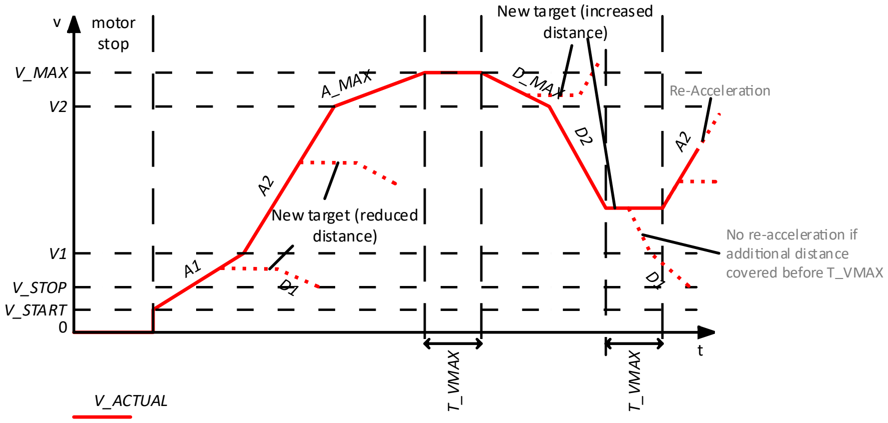


Figure 48. Possible adaptations of the ramp positioning profile due to changes in the target position

Ramp Velocity Mode

The ramp generator is configured for velocity profiles when setting the field RAMP_MODE = 1_b in the MCC_CONFIG.MOTOR_MOTION register. When a new user-defined target velocity is commanded through the FOC.PID_VELOCITY_TARGET register, the ramp generator calculates a velocity motion profile to reach from the current velocity to the set target velocity.

The current velocity value from the generated profile can be read from RAMPER.V_ACTUAL register. If the ramp generator is enabled, then the value in RAMPER.VELOCITY register is used as input for the velocity controller.

The velocity profile is generated according to the defined ramp parameters as described in *Table 32*. Note that, in contrast to positioning profiles, there is no distinct deceleration phase for the ramp velocity profiles. Instead, whether the profile is accelerating or decelerating depends only on the magnitude of the target velocity compared to that of the current velocity.

Up to three distinct acceleration/deceleration segments plus a constant velocity segment are generated for the ramp velocity profiles. The constant velocity segment represents only the one defined by the T_VMAX register field.

T_ZEROWAIT, V_START, V_STOP, V_MAX, D1, D2 and D_MAX are not used for the ramp velocity profiles.

Note that the defined ramp velocity parameters V1 and V2 and the corresponding acceleration/deceleration phases are honored for both velocity signs, even during a single velocity motion profile. For instance, if the current velocity has a value of $+v_1 \geq V2$, and a new velocity target of $-v_1$ is set, then the following velocity profile sequence is generated: $+v_1 \rightarrow +V2 \rightarrow +V1 \rightarrow -V1 \rightarrow -V2 \rightarrow -v_1$. This behavior is shown in *Figure 49*, where two velocity ramp profiles are displayed: one from a velocity of 0 to PID_VELOCITY_TARGET, and a second one from +PID_VELOCITY_TARGET to -PID_VELOCITY_TARGET. The constant velocity segment defined by the T_VMAX ramp parameter is also shown.

Table 32. Velocity mode ramp parameters

Register Field	Short Explanation
POSITION	Current position value of the ramp generator. It is used as a target for the position controller when the ramp generator is enabled. After a write access to FOC.PID_POSITION_ACTUAL register, POSITION is automatically overwritten with the new value in PID_POSITION_ACTUAL.
V_ACTUAL	Read-only current velocity value of the ramp generator. It is used as a target for the velocity controller, or as feedforward input to the position controller when the ramp generator is enabled.
ACCELERATION	Read-only current acceleration value of the ramp generator. It is optionally used as feedforward input to torque controller.
PID_VELOCITY_TARGET	Part of the FOC.PID_VELOCITY_TARGET register. Input target velocity for the ramp generator.
V1	Velocity value at which ramp generator switches between A1 and A2 during acceleration/deceleration phase. It must be bigger than V_START. Alternatively, it can be set to zero if the A1 section is not needed. It is an unsigned value; the sign of the current velocity is determined by the relative magnitude of the velocity target.
V2	Velocity value at which ramp generator switches between A2 and A_MAX during acceleration/deceleration phase. It must be bigger than V1. Alternatively, it can be set equal to V1 if the A2 section is not needed. It is an unsigned value; the sign of the current velocity is determined by the relative magnitude of the velocity target.
A1	Acceleration value between velocity before starting the ramp profile and V1.
A2	Acceleration value between V1 and V2.
A_MAX	Acceleration value between V2 and PID_VELOCITY_TARGET.
T_VMAX	Part of the RAMPER.TIME_CONFIG register. Hold a constant velocity for at least T_VMAX × 8.5µs after reaching PID_VELOCITY_TARGET or before starting any acceleration/deceleration due to a change in velocity target, even if the acceleration sign changes.

Unless otherwise noted, register fields shown on this table belong to the RAMPER register block.

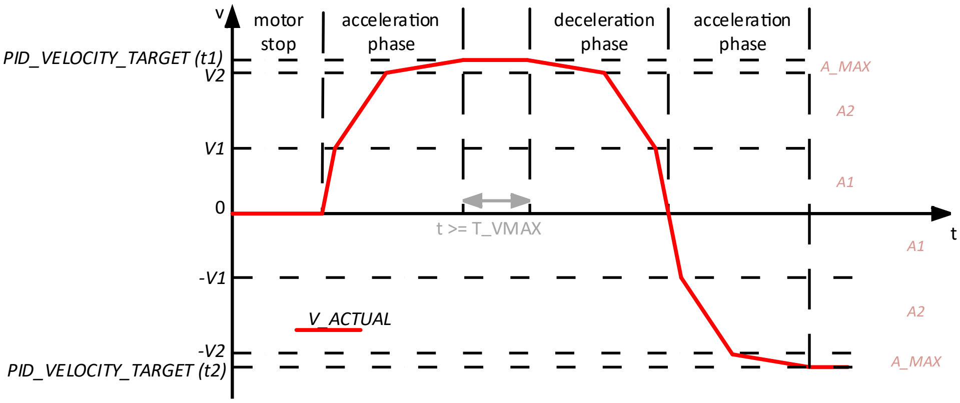


Figure 49. Typical ramp velocity profiles

Changes of the target velocity in FOC.PID_VELOCITY_TARGET during an ongoing ramp velocity profile may lead to early ramp termination, as are illustrated in *Figure 50*. Here, the constant velocity sections defined by T_VMAX are used.

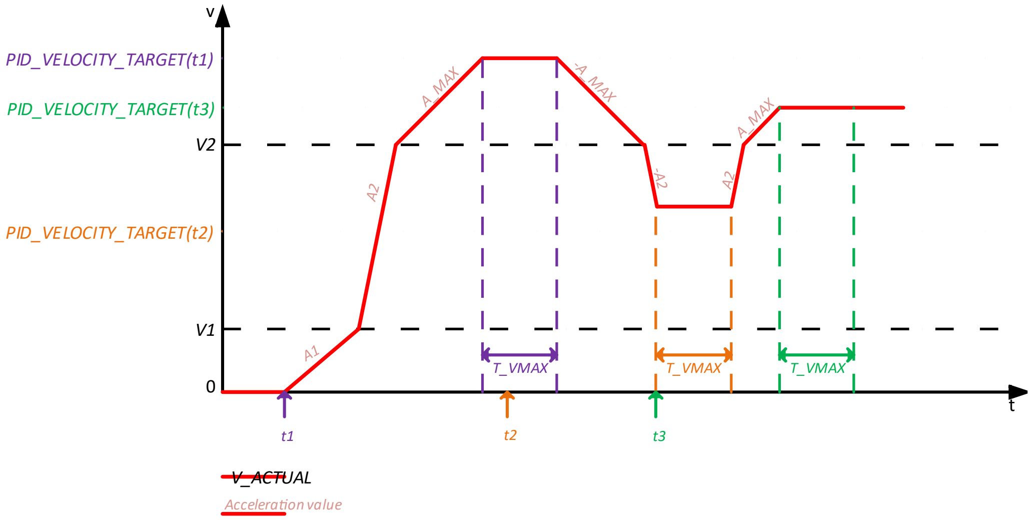


Figure 50. Possible adaptations of the ramp velocity profile due to changes in the target velocity

Open Loop Real Word Unit Conversion

The TMC6460 can use an internal oscillator or an external clock signal as reference for the system clock f_{SYS}. Velocity and acceleration parameters for the ramp generator are referenced to the system clock. For this reason, it is recommended to use an external quartz oscillator, or to provide an external clock signal from a microcontroller for best stability and reproducibility of the ramp motion profiles.

The TMC6460 uses an internal scaling for the ramp generator values. This section provides formulas to convert internal values into real-world units. The conversions provided here are only valid when operating in open loop mode. That is, when the ramp generator's phi_e is used instead of the angle of an external feedback source (when RAMP_USE_PHI_E bit of the MCC_CONFIG.MOTOR_MOTION register is set).

For real-world unit conversion for closed-loop operation (when an external angle feedback source is used), see the *Real World Unit Conversion* section.

Position conversion

The position scaling for open loop operation depends only on the number of pole pairs of the motor. Below are the equations to convert from the internal position P_{int} to the electrical and mechanical rotor revolutions P_e and P_m.

$$P_e[erev] = \frac{P_{int}}{2^{16}}$$

$$P_m[rev] = \frac{P_{int}}{2^{16} \times N_{polepairs}}$$

P_{int} refers in this case to the value of any position register used when in open loop mode. This includes but is not limited to: FOC.PID_POSITION_ACTUAL, FOC.PID_POSITION_TARGET and RAMPER.POSITION.

Velocity conversion

Use the following formula and the scaling factor k_{OL} to convert internal velocity values into rotor velocities in RPM when operating in open loop mode.

$$v\ [rpm] = \frac{v_{int}}{k_{OL}}$$

The velocity scaling factor for open loop operation k_{OL} depends only on the system clock frequency f_{SYS}.

$$k_{OL} = \frac{2^{16} \times 2^{24}}{f_{SYS} \times 60} \quad with\ f_{SYS} = 60MHz$$

In this case, v_{int} refers to the value of any velocity register used when in open loop mode. This includes but is not limited to: FOC.PID_VELOCITY_ACTUAL and FOC.PID_VELOCITY_TARGET, as well as any velocity parameter from the ramp generator registers.

Acceleration conversion

In contrast to velocity and position, acceleration values are used exclusively through the ramp generator block.

The acceleration conversion when operating in open loop mode depends only on the velocity scaling factor k_{OL} and the system clock frequency f_{SYS}. Use the following to convert internal acceleration values a_{int} into rotor accelerations in rev/s^2.

$$a\left[\frac{rev}{s^2}\right] = a_{int} \times \frac{f_{SYS}}{k_{OL} \times 60 \times 2^{17}} \qquad with\ f_{SYS} = 60MHz$$

In this case, a_{int} refers to the value of any of the acceleration parameters from the ramp generator registers.

Reference Switches

The TMC6460 supports the detection of reference switches, which are commonly used with linear motors or linear stages. Three reference switches are supported, they are referred to as left, right and home switches. Devices that can be used as reference switch input include mechanical switches, photo interrupters and digital Hall effect sensors. The reference switch inputs can be used to latch the current position or to stop the ramp motion. Settings for this feature are assigned in the RAMPER.SWITCH_MODE register.

Both normally open and normally closed switches can be used by setting the appropriate switch polarity and selecting the pull-up or pull-down resistor configuration accordingly. Normally closed switches are failsafe with respect to an interruption of the switch connection.

In case of long cables or noisy signals, additional RC filtering might be required near the TMC6460 reference switch inputs. Adding an RC filter also reduces the danger of destroying the logic level inputs by wiring faults, but it adds a small delay which should be considered with respect to the application. An example setup with left and right normally closed reference switches and an RC filter is shown in *Figure 51*.

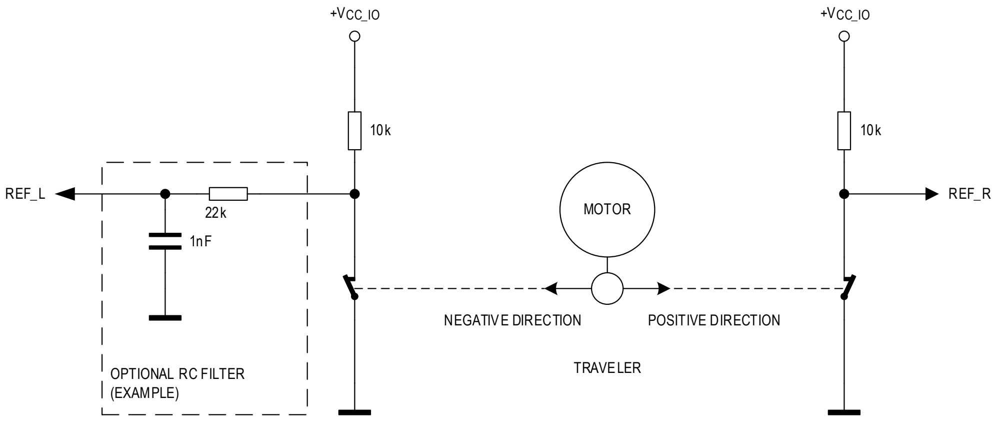


Figure 51. Example of reference switch usage

As described in the *Pin Configurations* chapter, most of the input pin signals can be digitally inverted and filtered. In addition to these settings, the TMC6460 allows to configure the active polarity for all functions of the ramp generator block that use reference switch inputs, for instance the latching of the current position or the automatic stopping of motion. This is set through the RAMP_REF_x_POL bits of the RAMPER.SWITCH_MODE register (with x = H, L, R).

Additionally, the left and right reference switch inputs can be swapped by setting the SWAP_RAMP_REF_LR bit on the RAMPER.SWITCH_MODE register.

Position Latching

The ramp generator block provides the ability to latch the current position upon detection of any of the reference switch inputs. It is also possible to configure whether the latching happens upon the active going edge or the inactive going edge of the reference switch input.

To use and configure this feature, set either of the LATCH_REF_x_INACTIVE or LATCH_REF_x_ACTIVE bits of RAMPER.SWITCH_MODE register (with x = H, L, R), according to the reference switch input that should trigger the position latching. The active/inactive edge is taken according to the configured switch polarity in the RAMP_REF_x_POL bits of the same register.

Two position values are latched when a position latching trigger occurs: the current value of RAMPER.POSITION register is written into RAMPER.POSITION_LATCH register, and the current value of FOC.PID_POSITION_ACTUAL register is written into RAMPER.POSITION_ACTUAL_LATCH register.

The LATCH_REF_x_READY bits of the RAMPER.EVENTS register (with x = H, L, R) can be used to monitor whether a position was latched due to a reference switch event.

Stop Modes

Three different event types can be configured to bring the ramp motion to a stop. These are reference switch inputs, position or velocity deviations, and a manually settable trigger. The stop behavior works for both velocity and position ramp modes. Note that the motion stop is performed at the output of the ramp generator. The ramp generator must therefore be enabled for this behavior to work.

To enable the deviation stall stop signals to stop the ramp motion, set the STALL_STOP_EN bit of the RAMPER.SWITCH_MODE register and additionally the STOP_ON_POS_DEVIATION or STOP_ON_V_DEVIATION bits according to the desired behavior. To configure the deviation value at which the stall stop signals are generated, set the appropriate values in the FOC.PID_POSITION_ERROR_MAX and/or FOC.PID_VELOCITY_ERROR_MAX registers. The status of the deviation stall stop signals can be queried from STALL_IN_POSITION_ERR and STALL_IN_V_ERR bits of the RAMPER.STATUS register.

To trigger a stall stop event manually, set the FORCE_HARD_STOP bit of the RAMPER.SWITCH_MODE register. The STALL_STOP_EN bit of the same register must also be set for this to work.

If any of the stall stop events occurred, the STALL_STOP_EVENT bit of the RAMPER.EVENTS register is set.

To enable a particular reference switch input to stop the ramp motion, set the corresponding RAMP_REF_x_STOP_EN bits of the RAMPER.SWITCH_MODE register (with x = H, L, R). The signal is triggered according to the active switch polarity as configured in the RAMP_REF_x_POL bits of the same register. The status of the stop signal can be queried from the RAMP_REF_x_STOP_STATUS bits of the RAMPER.STATUS register.

Hard vs soft stop

By default, all stop signals mentioned above trigger a hard stop of the motion. This means that the ramp velocity is set immediately to zero. Hard stops honor the wait time set in T_ZEROWAIT field of the RAMPER.TIME_CONFIG register to release the ramp motion. Hard stops might be suitable for reference switch finding, or whenever the velocity of the motion is kept relatively slow. This might however not be suitable for normal operation of the system.

For scenarios where a hard stop might not be desirable, an additional soft stop mode is available. To enable the soft stop behavior, set the SOFT_STOP_EN bit of the RAMPER.STATUS register. When enabled, soft stop sequences are used for the reference switch stop signals, the stall on position/velocity deviation signals, as well as the manual stop trigger.

Soft stop sequences make use of the deceleration ramp settings DMAX, D2, D1, V2, V1, VSTOP and T_ZEROWAIT to stop the ramp motion.

Care must be taken when using soft stop mode in combination with reference switches that are too close to the mechanical limit of a particular system. In those cases, it is possible that the rotor or linear stage hits the mechanical limit while the soft stop sequence is still ongoing. In some scenarios, especially when the ramp position and the motor's actual position are not totally in sync, this might lead to the position controller to drive the motor into the mechanical limit with maximum torque. To avoid this situation, it is recommended to place the reference switch at a safe distance to the mechanical limit.

Implementing a Homing Routine

Use the sequence below to implement a homing routine using the reference switch inputs and the ramp position mode.

1. Make sure that the used switch is not pressed, e.g. by moving away from the switch for a short distance.
2. Activate position latching upon the used switch event and the desired polarity.
3. Activate the motor (soft) stop upon activation of the used switch.

4. Start a ramp motion into the position of the switch (move to a more negative position for a left switch, or to a more positive position for a right switch). This motion may be timed out by using a ramp positioning motion.
5. As soon as the switch is hit, the position becomes latched and the motor begins to stop.
6. Wait until the motor is in standstill by polling the V_ZERO status flag.
7. Calculate the difference between the latched position and the actual position.
8. Make sure the KEEP_POS_TARGET bit is not set.
9. Write the calculated difference into the actual position register.

Homing is now finished. Moving to a position of 0 brings the motor back to the exact point where the switch was hit.

Switch Mode Register Overview

Table 33. RAMPER.SWITCH_MODE bit fields

Register Field(s)	Remarks
RAMP_REF_R_POL RAMP_REF_L_POL RAMP_REF_H_POL	These bit fields define the active polarity of the right, left or home reference switches: ▪ If set to 0, the active polarity is 1. ▪ If set to 1, the active polarity is 0 (inverted).
RAMP_REF_R_STOP_EN RAMP_REF_L_STOP_EN RAMP_REF_H_STOP_EN	These enable bit fields trigger a stop event for the ramp motion in case: ▪ Right reference switch is active and actual velocity value is positive (V_ACTUAL > 0) ▪ Left reference switch is active and actual velocity value is negative (V_ACTUAL < 0) ▪ Home reference switch is active.
STALL_STOP_EN	Set this bit to enable any of FORCE_HARD_STOP, STOP_ON_POS_DEVIATION and STOP_ON_V_DEVIATION to trigger a stall stop event.
FORCE_HARD_STOP	Activating this bit triggers a stall stop event (STALL_STOP_EN must be set)
STOP_ON_POS_DEVIATION	Trigger a stall stop event in case the absolute position error of the position controller exceeds FOC.PID_POSITION_ERROR_MAX. (STALL_STOP_EN must be set)
STOP_ON_V_DEVIATION	Trigger a stall stop event in case the absolute velocity error of the velocity controller exceeds FOC.PID_VELOCITY_ERROR_MAX. (STALL_STOP_EN must be set)
SOFTSTOP_EN	If this bit is cleared, any triggered stop event performs a hard stop of the ramp motion, setting RAMPER.V_ACTUAL immediately to 0. Set this bit to enable the soft stop behavior for any triggered stop event, for which a deceleration ramp (incl. all defined phases RAMPER.D1, RAMPER.D2, and RAMPER.D_MAX) is executed.
LATCH_REF_R_ACTIVE LATCH_REF_L_ACTIVE LATCH_REF_H_ACTIVE	Set any of these bits to latch the value of RAMPER.POSITION into RAMPER.POSITION_LATCH register as well as the value of FOC.PID_POSITION_ACTUAL into RAMPER.POSITION_ACTUAL_LATCH register upon an active going edge of the particular reference switch input.
LATCH_REF_R_INACTIVE LATCH_REF_L_INACTIVE LATCH_REF_H_INACTIVE	Set any of these bits to latch the value of RAMPER.POSITION into RAMPER.POSITION_LATCH register as well as the value of FOC.PID_POSITION_ACTUAL into RAMPER.POSITION_ACTUAL_LATCH register upon an inactive going edge of the particular reference switch input.
SWAP_RAMP_REF_LR	If this bit is set, the internal processing of the left and right reference switches is interchanged. It might be useful to avoid external circuit changes.

Unless otherwise noted, register fields shown on this table belong to the RAMPER.SWITCH_MODE register.

Ramp Generator Status Flags and Events

The ramp generator block provides some status flags to indicate specific ramp conditions, including those related to the reference switches. The status flags represent the current state (read-only) of a particular condition, while the events signal that a certain condition has occurred. An event bit stays active for as long as the particular state is active and the corresponding bit is not manually cleared. Table 34 shows all status flags, along with their bit position. These are available in the RAMPER.STATUS register. Table 35 shows the event bits, available in the RAMPER.EVENTS register. Certain status bits are forwarded to the CHIP.STATUS_FLAGS register. Refer to Ramp Generator Status Flags and Events section for more information.

Table 34. RAMPER.STATUS bit fields

Flag Name	Bit No.	Access	Description
RAMP_REF_L_STATUS	0	R	Status of left reference switch: 0= inactive, 1= active
RAMP_REF_R_STATUS	1	R	Status of right reference switch: 0= inactive, 1= active
RAMP_REF_H_STATUS	2	R	Status of home reference switch: 0= inactive, 1= active
RAMP_REF_L_STOP_STATUS	6	R	Signals a stop condition due to an active left stop switch.*
RAMP_REF_R_STOP_STATUS	7	R	Signals a stop condition due to an active right stop switch.*
RAMP_REF_H_STOP_STATUS	8	R	Signals a stop condition due to an active home ref switch.*
V_REACHED_STATUS	11	R	Maximum/target velocity is equal to current ramp velocity value. Ramp position mode: V_ACTUAL = RAMPER.V_MAX. Ramp velocity mode: V_ACTUAL = FOC.PID_VELOCITY_TARGET.
POSITION_REACHED_STATUS	12	R	Ramp current velocity is equal to 0 and target position is equal to ramp current position: RAMPER.POSITION = FOC.PID_POSTION_TARGET. This bit is forwarded to the CHIP.STATUS_FLAGS register.
V_ZERO	13	R	Current ramp velocity value RAMPER.V_ACTUAL is equal to 0.
T_ZEROWAIT_ACTIVE	14	R	T_ZEROWAIT phase is active after ramp motion has been stopped. During this time, the ramp motion is standstill.
STALL_IN_V_ERR	16	R	Ramp has been stopped because maximum velocity deviation exceeds FOC.PID_VELOCITY_ERROR_MAX value.
STALL_IN_POSITION_ERR	17	R	Ramp has been stopped because maximum position deviation exceeds FOC.PID_POSITION_ERROR_MAX value.

*The stop condition and its interrupt can be cleared by commanding a move in the opposite direction. If soft stop mode is enabled, the condition remains active until the motor has stopped motion into the direction of the reference switch. Disabling the reference switch or its stop function also clears the interrupt, but the motor might continue its motion.

Table 35. RAMPER.EVENTS bit fields

Flag Name	Bit No.	Access	Description
POSITION_REACHED_EVENT	0	R/W1C	The ramp motion has been stopped and target position reached: RAMPER.POSITION = FOC.PID_POSTION_TARGET.
STALL_STOP_EVENT	1	R/W1C	Signals an active stall stop event due to velocity or position tracking errors. Writing '1' into this bit field clears the stop condition and the motor may re-start motion. (The bit and its interrupt condition are cleared upon writing '1').
SECOND_MOVE_EVENT	2	R/W1C	Ramp generator calculated a new ramp motion that is moving in the opposite direction, e.g. due to an update of the target values.
LATCH_REF_H_READY	3	R/W1C	Latching position was executed by a triggered home reference switch.
LATCH_REF_L_READY	4	R/W1C	Latching position was executed by a triggered left reference switch.
LATCH_REF_R_READY	5	R/W1C	Latching position was executed by a triggered right reference switch.

The bits shown on this table have W1C access, meaning that a '1' must be written to clear them.

Programmable I/O Controller

The TMC6460 contains a versatile programmable I/O Controller. Its architecture and command set are optimized for the specific purpose of converting serial data into parallel data and vice versa. This way, synchronous and asynchronous bit-streams are supported.

As discussed in *Pin Remapping* chapter, most of I/O pins can be assigned an alternate function related to the I/O Controller. Up to three output pins and two input pins of the TMC6460 can be directly controlled or interfaced with the I/O Controller. The 4th output of the I/O Controller is forwarded to the IO_CONTROLLER_STATUS bit of the CHIP.STATUS_FLAGS register. Since this bit is combined with the I/O Controller reset flag, the 4th output is only reflected on the IO_CONTROLLER_STATUS bit when the I/O Controller is not in the reset state. Some internal synchronization signals (e.g. PWM_ZERO) can also be assigned as inputs to the I/O Controller.

Further on, the complete register map can be accessed by the I/O Controller in parallel to any register access from an external host through SPI or UART. The controller is versatile enough to support tasks that involve capturing or manipulating register values with low latency and with periodically triggered signals.

The I/O Controller in the TMC6460 includes a ROM code that provides a basic feature set to extend the capabilities of the TMC6460. This includes interfacing with SPI/SSI encoders, loading programs from an external EEPROM, data filtering with a biquad filter, and more.

The I/O Controller contains different accelerator modules that can be controlled by utilizing specific addresses within the program running on it. The following accelerator modules are available:

- I2C master,
- Unsigned 32-bit by 16-bit divider,
- Signed 16-bit multiplier, and
- Signed biquad filter.

Accelerator modules are only explained in the context of the included ROM code features in this data sheet.

A 1024x16-bit RAM is also available for user programming. Programming and reading the RAM is possible through the included ROM code features. Programming of the I/O Controller is optional and only required if features other than those offered by the ROM code are needed.

Further details and explanations of the specific RAM access, as well as accelerator module requests and all the available opcodes are part of an additional application note.

ROM Code Features

The TMC6460 provides several features using the I/O Controller without the need of a custom program. These are referred to as ROM code features. The IO_CONTROLLER.COMMAND register is the main interface through which the different features are executed. Some ROM code features are executed as a single command, while others execute repeatedly. Only one feature can be executed at a time. *Table 36* shows an overview of the available ROM code features.

Table 36. ROM code feature overview

Command	Feature	(S)ingular / (R)epeated Execution
GET_VER	Returns the ROM code version.	S
I2C_INIT_PP	Initializes the I2C protocol in push-pull outputs.	S
I2C_INIT_OD	Initializes the I2C protocol, open drain outputs.	S
I2C_READ	Read from an I2C client.	S
I2C_WRITE	Write to an I2C client.	S
I2C_EEPROM	Readout the contents of an interfaced I2C EEPROM into the RAM.	S
RAM_READ	Reads one address from the internal RAM.	S
RAM_WRITE	Writes one address of the internal RAM.	S
RAM_EXE	Switch to RAM code execution.	S
FILTER_SETUP	Setup of the internal biquad filter.	S
CALC_FILT	Executes one or all of the following: signed multiplication, unsigned division, signed biquad filtering.	S
DEBUG	Debugs a particular register, by getting minimum, maximum, and filtered values.	R
PWM_INPUT	Executes PWM input pin processing and stores the calculated value in a defined register.	R
AIN_INPUT	Executes AIN input pin processing and stores the calculated value in a defined register.	R
SPI_ENC	Sets up a SPI master interface to communicate to an interfaced SPI encoder with either single or continuous readouts and optionally stores the received value for use with the feedback engine. SSI encoders with compatible datagram structure can also be interfaced through this feature.	S / R

Single execution commands do not require additional finishing commands, whereas repeatedly executed ones must be finished by setting an “idle” command as IO_CONTROLLER.COMMAND = 0x00000000.

Generally, the IO_CONTROLLER.RESPONSE_0 register is set equal to the requested IO_CONTROLLER.COMMAND as soon as the command is successfully executed. Some slight variations occur for the repeatedly executed commands. This is discussed in the particular command’s section.

Important to note is the requirement to start a new command execution. The command defined by the value in the IO_CONTROLLER.COMMAND register will be executed only when the command value differs from that of the IO_CONTROLLER.RESPONSE_0 register. Therefore, if two identical commands must be executed one after the other, an “idle” command (0x00000000) must be set in between executions. This works since setting IO_CONTROLLER.COMMAND = 0x00000000 results in IO_CONTROLLER.RESPONSE_0 = 0x00000000.

The 4th I/O Controller output that is connected to the IO_CONTROLLER_STATUS bit can be assigned by the I/O Controller to the internal IO_CONTROLLER.RESPONSE_0 status bit. As soon as IO_CONTROLLER.RESPONSE_0 equals the IO_CONTROLLER.COMMAND register – that indicates the command has been finished – this status bit is set to 1. This way, permanent polling of the IO_CONTROLLER.RESPONSE_0 register can be avoided by using the FAULTN pin to be informed about the completion of the requested command.

Depending on the ROM code feature used, additional parameters other than the command itself might be required. For that purpose, some or all the IO_CONTROLLER.RESPONSE_x registers (with x = 1, 2, 3) must be used. Details are also discussed in the particular command’s section.

The following notation is used to describe the ROM code feature commands: letters in lowercase represent one byte of the particular register whose value must be chosen according to the desired configuration; letters in uppercase and integers represent a predefined hexadecimal value that is required for the command to work.

GET_VER

Table 37. GET_VER Commands and replies

	User action: Write to register		I/O Controller reply: Read from		Comments
	IO_CONTROLLER…	Value	IO_CONTROLLER…	Value	
1	COMMAND	0xAA000000			
2			RESPONSE_0	0xAA000000	As soon as command is executed.
3			RESPONSE_1	0x646000B3	ROM version no.

Description:

Returns the current ROM version.

I2C_INIT_PP

Table 38. I2C_INIT_PP Commands and replies

	User action: Write to register		I/O Controller reply: Read from		Comments
	IO_CONTROLLER…	Value	IO_CONTROLLER…	Value	
1	COMMAND	0xAA01yyyy			0xyyyy: Baud rate setting.
2			RESPONSE_0	0xAA01yyyy	As soon as command is executed.

Description:

Initializes the I2C interface with a certain I2C clock frequency defined by the 16 bits “yyyy”. The clock frequency is calculated as: f_{SYS} / 4 / (yyyy + 1), e.g. yyyy = 0x95 results in a clock frequency of f_{I2C} = 100kHz at f_{SYS} = 60MHz.

I2C SDA and SCK pins are only available as alternate pin functions for feedback engine pins HALL_U and HALL_V.

SCL pin is configured as push-pull, whereas SDA is configured as open drain during ACK and read phases, as expected for I2C protocol.

Either this or the *I2C_INIT_OD* commands must be executed before using the EEPROM read/write commands.

I2C_INIT_OD

Table 39. I2C_INIT_OD Commands and replies

	User action: Write to register		I/O Controller reply: Read from		Comments
	IO_CONTROLLER….	Value	IO_CONTROLLER….	Value	
1	COMMAND	0xAA02yyyy			0xyyyy: Baud rate setting.
2			RESPONSE_0	0xAA02yyyy	As soon as command is executed.

Description:

Initializes the I2C interface with a certain I2C clock frequency defined by the 16 bits “yyyy”. The clock frequency is calculated as: f_{SYS} / 4 / (yyyy + 1), e.g. yyyy = 0x95 results in a clock frequency of f_{I2C} = 100kHz at f_{SYS} = 60MHz.

I2C SDA and SCK pins are only available as alternate pin functions for feedback engine pins HALL_U and HALL_V.

SCL is not configured as push-pull. Instead, the SCL is driven with strong 0s and weak 1s, which enables potential clock stretching behavior of the connected I2C client. SDA is configured as open drain during ACK and read phases, as expected for I2C protocol.

Either this or the *I2C_INIT_PP* commands must be executed before using the EEPROM read/write commands.

I2C_READ

Table 40. I2C_READ Commands and replies

	User action: Write to register		I/O Controller reply: Read from		Comments
	IO_CONTROLLER….	Value	IO_CONTROLLER….	Value	
1	COMMAND	0xDyyy0000			0xyyy: Readout address for I2C client.
2			RESPONSE_0	0xDyyy0000	As soon as command is executed.
3			RESPONSE_1	0x0000zzzz	Readout response from I2C client.

Description:

Executes a single read access from the connected I2C client. “yyy” represents the appointed 12-bit address.

The 16 bits “zzzz” represent the reply of the client and are stored in IO_CONTROLLER.RESPONSE_1 register and to be read out after the command is executed.

The I2C interface must be initialized with one of the I2C_INIT_OD or I2C_INIT_PP features before issuing this command.

I2C_WRITE

Table 41. I2C_WRITE Commands and replies

	User action: Write to register		I/O Controller reply: Read from		Comments
	IO_CONTROLLER....	**Value**	**IO_CONTROLLER....**	**Value**	
1	COMMAND	0xEyyyzzzz			0xyyy: Write address, 0xzzzz: Write data.
2			RESPONSE_0	0xEyyyzzzz	As soon as command is executed.

Description:

Executes a single write access to the connected I2C client. The 16 bits “zzzz” represent the value that would be written into the appointed 12-bit address of the “yyy” bits.

The I2C interface must be initialized with one of the I2C_INIT_OD or I2C_INIT_PP features before issuing this command.

I2C_EEPROM

Table 42. I2C_EEPROM Commands and replies

	User action: Write to register		I/O Controller reply: Read from		Comments
	IO_CONTROLLER....	**Value**	**IO_CONTROLLER....**	**Value**	
1	COMMAND	0xAA030000			
2			RESPONSE_0	0xAA030000	As soon as command is executed.

Description:

Executes a complete read out of a connected EEPROM via I2C into the RAM.

The I2C interface must be initialized with one of the I2C_INIT_PP or I2C_INIT_OD features before issuing this command.

RAM_READ

Table 43. RAM_READ Commands and replies

	User action: Write to register		I/O Controller reply: Read from		Comments
	IO_CONTROLLER....	**Value**	**IO_CONTROLLER....**	**Value**	
1	COMMAND	0xCyyy0000			0xyyy: RAM addr
2			RESPONSE_0	0xCyyy0000	As soon as command is executed.
3			RESPONSE_1	0x0000zzzz	Readout value from RAM at addr=0xyyy.

Description:

Executes a single read access from the internal RAM. “yyy” contains the appointed 10-bit address. The two upper bits of “yyy” must be set to 0.

The 16 bits “zzzz” represent the contents of the RAM and are stored in IO_CONTROLLER.RESPONSE_1 register and to be read out after the command is executed.

RAM_WRITE

Table 44. RAM_WRITE Commands and replies

	User action: Write to register		I/O Controller reply: Read from		Comments
	IO_CONTROLLER....	**Value**	**IO_CONTROLLER....**	**Value**	
1	COMMAND	0xByyyzzzz			0xyyy: RAM write addr, 0xzzzz: Write data.
2			RESPONSE_0	0xByyyzzzz	As soon as command is executed.

Description:

Executes a single write access to the internal RAM. The 16 bits “zzzz” represent the value that would be written into the appointed 10-bit address of the “yyy” bits. The two upper bits of “yyy” must be set to 0.

RAM_EXE

Table 45. RAM_EXE Commands and replies

	User action: Write to register		I/O Controller reply: Read from		Comments
	IO_CONTROLLER….	Value	IO_CONTROLLER….	Value	
1	COMMAND	0xFFFFFFFF			
2			RESPONSE_0	0xFFFFFFFF	As soon as command is executed, just before RAM code execution.

Description:

Executes the transfer from ROM code execution to RAM code execution. Any further I/O Controller actions are defined by the RAM opcodes.

FILTER_SETUP

Table 46. FILTER_SETUP Commands and replies

	User action: Write to register		I/O Controller reply: Read from		Comments
	IO_CONTROLLER….	Value	IO_CONTROLLER….	Value	
1	RESPONSE_1	0xzzpppppp			0xpppppp: Coefficient b0, 0xzz: Part of a2.
2	RESPONSE_2	0xyyssssss			0xssssss: Coefficient b1, 0xyy: Part of a2.
3	RESPONSE_3	0xwwtttttt			0xtttttt: Coefficient b2, 0xww:Part of a2.
4	COMMAND	0x70uuuuuu			0xuuuuuu: Coefficient a1.
5			RESPONSE_0	0x70uuuuuu	As soon as command is executed.

Description:

Configures the biquad filter connected to the I/O Controller with all its 24 bits parameters (refer to *Biquad Filters* section for more details about biquad parameterization):

- Coefficient b0 [23:0] = pppppp
- Coefficient b1 [23:0] = ssssss
- Coefficient b2 [23:0] = tttttt
- Coefficient a1 [23:0] = uuuuuu
- Coefficient a2 [23:0] = wwyyzz

Coefficients b0, b1, b2 and a2 must be set before executing the command. Therefore, coefficient a1 must be set last.

CALC_FILT

Table 47. CALC_FILT Commands and replies

	User action: Write to register		I/O Controller reply: Read from		Comments
	IO_CONTROLLER….	Value	IO_CONTROLLER….	Value	
1	RESPONSE_1	0xzzzzzzzz			Numerator (u32).
2	RESPONSE_2	0xsssswwww			0xssss and 0xwwww: Multipliers (s16).
3	RESPONSE_3	0xyyyyyyyy			Biquad input (s32).
4	COMMAND	0x7100uuuu			0xuuuu: Divisor (u16).
5			RESPONSE_0	0x7100uuuu	As soon as command is executed.
6			RESPONSE_1 RESPONSE_2 RESPONSE_3	0xqqqqqqqq 0xpppppppp 0xtttttttt	Unsigned division result (u32). Signed multiplication result (s32). Biquad filter output (s32).

Description:

Executes three different calculations using the accelerator modules of the I/O Controller.

- Unsigned Division (Numerator[31:0] / Divisor[15:0] = Quotient [31:0]).
  - Calculation: 0xzzzzzzzz / 0x0000uuuu = 0xqqqqqqqq.
  - RESPONSE_1 contains the division result.
  - All numbers are unsigned.
- Signed Multiplication (Multiplier[15:0] × Multiplier[15:0] = Product [31:0]).
  - Calculation: 0xssss × 0xwwww = 0xpppppppp.
  - RESPONSE_2 contains multiplication result.

  - All numbers are signed.
- Biquad filter.
  - 0xyyyyyyyy is the signed input to the biquad filter.
  - 0xtttttttt (RESPONSE_3) contains the biquad filter output.
  - All numbers are signed.
  - Biquad filter must be setup before, using the *FILTER_SETUP* command.
  - Refer to *Biquad Filters* for more details.

All calculations are executed in parallel. Only the parameters which are needed for the desired calculation are required.

DEBUG

Table 48. DEBUG Commands and replies

	User action: Write to register		I/O Controller reply: Read from		Comments
	IO_CONTROLLER....	Value	IO_CONTROLLER....	Value	
1	COMMAND	0x6yyyzwuu			0xyyy: Address, 0xz: signed/unsigned. 0xw: Data mask, 0xuu: trigger setup.
2			RESPONSE_0	0x55yyzwuu	During calculations, reply data NOT valid!
3			RESPONSE_0	0x6yyyzwuu	After caclulation phase, reply data is valid.
			RESPONSE_1	0xqqqqqqqq	Maximum value since debug started (u32).
			RESPONSE_2	0xpppppppp	Mi(s32).
			RESPONSE_3	0xtttttttt	Minimum value since debug started (s32).
4	COMMAND	0x00000000			Finish debug feature.

Description:

Repeatedly debugs a certain register. With every new trigger, a new calculation phase is executed.

On each calculation phase, the value of the defined debug register will be loaded and checked against:

- Maximum stored positive value (IO_CONTROLLER.RESPONSE_1 = 0xpppppppp), and
- Minimum stored negative value (IO_CONTROLLER.RESPONSE_3 = 0xnnnnnnnn).

Note that the values in IO_CONTROLLER.RESPONSE_1 and IO_CONTROLLER.REPONSE_3 are each related only to positive or negative values. For instance, if only negative values are acquired for a debugged register, then IO_CONTROLLER.RESPONSE_1 will contain a value of zero and not the negative value with the smallest magnitude. Conversely, if only positive values are acquired, then the IO_CONTROLLER.RESPONSE_3 will contain a value of -1 and not the smallest positive value.

IO_CONTROLLER.RESPONSE_2 contains a filtered value. Each new value will be filtered by the connected biquad filter. For this reason, the filter must be set up before starting DEBUG by using the *FILTER_SETUP* command.

The debugged register is defined by the 10-bit address in the "yyy" bits. The two upper bits of "yyy" must be set to 0.

The "z" bits define if the values are signed (=1) or unsigned (=0).

The "w" bits define the mask value for the debugged register:

- Each bit of this byte represents a byte of the debugged register to keep.
- For example:
  - w = 0011_b will process the two lower bytes (register bits [15:0]) of the debugged register.
  - w = 0110_b will process the register bits [23:8] of the debugged register.
  - w = 1111_b will process all register bits [31:0] of the debugged register.

The "uu" bits define the trigger of each acquisition/calculation phase:

- Every new acquisition/calculation phase is triggered with a defined signal.
- When a 0 to 1 change of the signal is detected, a new acquisition and calculation cycle is processed.
- During processing, the highest byte of IO_CONTROLLER.RESPONSE_0 changes to 0x55 to indicate that the response data is not valid.
- After the processing is concluded, the highest byte of IO_CONTROLLER.RESPONSE_0 changes back to 0x6y to indicate that the response data is valid.

Table 49 shows the signals that can be used for the debug trigger. Note that more than one trigger signal can be selected. However, it is recommended to select at most one signal so that no trigger is missed.

Table 49. ROM code DEBUG: Trigger options

“uu”	Trigger	“uu”	Trigger
0x01	PWM_IN input signal	0x10	ADC_I_VALID: Internal CSA calculations done
0x02	DIR_IN input signal	0x20	ADC_EXT_VALID: External ADC calculations done
0x04	Internal PWM_ZERO signal	0x40	FOC_VALID: FOC calculations done
0x08	Internal PWM_CENTER signal	0x80	Internal BRAKE signal

If “uu” = 0x00, then PWM_CENTER signal is defined as the trigger signal.

To finish debugging, send the “idle” command of IO_CONTROLLER.COMMAND = 0x00000000. The response data will be retained for as long as no new command is issued.

PWM_INPUT

Table 50. PWM_INPUT Commands and replies

	User action: Write to register		I/O Controller reply: Read from		Comments
	IO_CONTROLLER.…	Value	IO_CONTROLLER.…	Value	
1	RESPONSE_1	0xttttzzzz			Normalized duty cycle limits (u16).
2	RESPONSE_2	0xuuuuuuuu			Minimum final output value limit (s32).
3	RESPONSE_3	0xvvvvvvvv			Maximum final output value limit (s32).
4	COMMAND	0x9yyymsww			0xyyy: Target address, 0xm: Byte mask, 0xs: Byte shift, 0xww: Sample rate delay.
5			RESPONSE_0	0x55yymsww	During calculations, reply data NOT valid!
6			RESPONSE_0	0x9yyymsww	After caclulation phase, reply data is valid.
			RESPONSE_1	0xnnnnnnnn	Last target value (s32).
			RESPONSE_2	0xqqqqpppp	0xqqqq: Last filtered duty cycle value, 0xpppp: Last relative duty cycle value.
			RESPONSE_3	0xkkkkhhhh	0xkkkk: PWM=1 count, 0xhhhh: PWM=0 count.
7	COMMAND	0x00000000			Finish PWM_INPUT feature.

Description:

Repeatedly processes the PWM_IN input and optionally the DIR_IN input to use them as control signals for any target register. During this process, the target register is repeatedly written with new values, regardless of the validity of the response registers. The response registers are only for debugging purposes.

Note that the optional digital input filter and/or signal inversion of the PWM_IN and DIR_IN input signals must be configured before issuing this command. Refer to *Pin Configurations* and *Pin Remapping* chapters for more details.

Different types of duty cycle are calculated for this command. Duty cycles are represented with 16-bits values which correspond to a percentage between 0 - 100%. Keep in mind that the input PWM signal cannot have a duty cycle of 0% or 100%, as the 1→0 and 0→1 transition edges are essential to start the different calculation steps. Since the input duty cycles can be normalized, processed duty cycle values can in practice reach either of the limits.

Since duty cycles are represented with 16-bit values, the real duty cycle percentage can be obtained by dividing the internal value by 65536, e.g. an internal duty cycle value of 8192 corresponds to 8192 / 65536 × 100% = 12.5%.

The base sampling rate of the input PWM signal is f_{SYS} / 30 = 2MHz. This frequency can also be reduced if needed. The “ww” bits of the command datagram define the divider for the base sampling rate with the following formula: sampling_rate = 2MHz / (ww + 1). For instance, with ww = 4 the sampling rate reduces to 400kHz, whereas setting ww = 0 results in the maximum sampling rate of 2MHz.

A slower sampling rate might be needed if the input PWM signals has a period larger than 1 / 2MHz × 2^{16} = 32.768ms, (equivalent to 30.52Hz), which is the case as the input counter is restricted to a 16-bit value.

Since most of the calculations are executed after the 1→0 transition edge of the input PWM signal, the low phase of this input must be at least 6µs long to ensure the correct processing of every PWM_IN duty cycle.

During the calculation phase, the highest byte of IO_CONTROLLER.RESPONSE_0 is set to 0x55 to indicate that the reply data is not valid. After all calculation steps are executed and the target assignment is finished, that same byte is set again to 0x9y to indicate valid response registers. After this, the response registers contain the following data:

- IO_CONTROLLER.RESPONSE_1 = 0xnnnnnnnn: Last target assignment value.
- IO_CONTROLLER.RESPONSE_2[15:0] = 0xpppp: Last monitored relative duty cycle value $DC_{REL} \times 2^{16}$.
- IO_CONTROLLER.RESPONSE_2[31:16] = 0xqqqq: Last filtered duty cycle value $DC_{FILT} \times 2^{16}$.
- IO_CONTROLLER.RESPONSE_3[15:0] = 0xhhhh: Last monitored PWM_IN = 1 count value.
- IO_CONTROLLER.RESPONSE_3[31:16] = 0xkkkk: Last monitored PWM_IN = 0 count value.

To finish the PWM input processing, send the “idle” command of IO_CONTROLLER.COMMAND = 0x00000000. The response data will be retained for as long as no new command is issued.

The basic acquisition and calculation sequence is explained in the following. After the PWM_IN = 0→1 edge, the relative duty cycle is calculated with:

$$DC_{REL} = \frac{count\{PWM\_IN=1\}}{count\{PWM\_IN=1\}+count\{PWM\_IN=0\}}$$

After the PWM_IN = 1→0 edge, the DIR_IN polarity is stored, all remaining calculations are performed, and the target value assignment for PWM_IN processing is started. This is described in the following process:

1. Normalize relative duty cycle DC_{REL} according to duty cycle limits.
   - IO_CONTROLLER.RESPONSE_1 configures the 16-bit limits when the COMMAND is issued. It defines minimum and maximum relative duty cycle values for the normalization calculation.
   - Minimum relative duty cycle DC_{MIN} = 0xzzzz / 65536 (defined in RESPONSE_1[15:0]):
     - If the relative duty cycle is below this limit, the normalized duty cycle DC_{NORM} is set to 0.
   - Maximum relative duty cycle DC_{MAX} = 0xtttt / 65536 (defined in RESPONSE_1[31:16]):
     - If the relative duty cycle exceeds this limit, the normalized duty cycle DC_{NORM} is set to 1.
   - If DC_{REL} is within DC_{MIN} and DC_{MAX}, the normalized duty cycle DC_{NORM} is calculated proportionally to these limits.
   - If duty cycle inversion is enabled (bit 27 of COMMAND is set) and DIR_IN = 1 at the PWM_IN 1→0 edge, then the normalized duty cycle DC_{NORM} is inverted.
   - Summarized calculations and assignments during this phase:
     - If $DC_{REL} < DC_{MIN}$: $DC_{NORM} = 0$
     - Else if $DC_{REL} > DC_{MAX}$: $DC_{NORM} = 65535 / 65536 \approx 1.0$
     - Else: $DC_{NORM} = (DC_{REL} - DC_{MIN}) / (DC_{MAX} - DC_{MIN})$
     - If duty cycle inversion is active and DIR_IN = 1: $DC_{NORM} = 1.0 - DC_{NORM}$
2. Filter the normalized duty cycle DC_{NORM} value with the connected biquad filter. Refer to *Biquad* Filters section for details.
   - Output of the biquad filter is named DC_{FILT} in the following.
   - Important to note that the biquad filter must be set up before by using the *FILTER* SETUP command.
3. Calculate the final output value with the filtered duty cycle DC_{FILT} and the set limits. The 32-bit signed limit values are defined by IO_CONTROLLER.RESPONSE_2 and IO_CONTROLLER.RESPONSE_3 when the COMMAND is issued:
   - Minimum output value OUT_{MIN} = 0xuuuuuuuu (defined in IO_CONTROLLER.RESPONSE_2, signed)
   - Maximum output value OUT_{MAX} = 0xvvvvvvvv (defined in IO_CONTROLLER.RESPONSE_3, signed)
   - Final output value OUT_{FINAL} is calculated proportionally to these limits.
   - If output value inversion is enabled (bit 27 of COMMAND is set) and DIR_IN = 1 at the PWM_IN 1→0 edge, then the final output value OUT_{FINAL} is inverted.
   - Summarized calculations and assignments during this phase:
     - Final output calculation: $OUT_{FINAL} = (OUT_{MAX} - OUT_{MIN}) \times DC_{FILT} + OUT_{MIN})$
     - If output value inversion is active and DIR_IN = 1: $OUT_{FINAL} = -1.0 \times OUT_{FINAL}$
4. Shift output value and mask target register value:
   - If the target register contains more than one register field, the output value OUT_{FINAL} can be shifted, and the original register value can be masked so that only the particular register field is overwritten.
   - The “m” byte defines the mask of the target register value. Each bit of “m” represents a byte of the target register. If any bit equals 0, the particular byte is kept during the write access to the target register. Refer to following examples:
     - If m = 0xC, only the upper two bytes of the target register are overwritten by the final output value.
     - If m = 0xF, all bytes are overwritten by the final output value OUT_{FINAL}.

- “s” defines a potential shift operation of the final output value OUT_{FINAL}.
  - For instance, if “s” = 0x3 the final output value is left shifted by 2 bytes to fit into the target register field.
  - The following shift operation are available for the final output value (if “s” = 0x0 no shift is carried out):

“s” =	0x1	0x3	0x7	0x9	0xB	0xF
Shift	Left by 1 byte	Left by 2 bytes	Left by 3 bytes	Right by 1 byte	Right by 2 bytes	Right by 3 bytes

5. Transfer the output value into the target register.
   - Target register address is defined by the “yyy” part of the COMMAND.
   - The lower 10 bits represent the address of the target register.
   - After all the prior calculations, the target register value is overwritten by the shifted OUT_{FINAL} value.
   - Masked bytes (if particular bit of “m” equals 0) are transferred from the prior value of this target register.

The DIR_IN input signal can be utilized to invert the duty cycle or the final output value as described above. It is important to note that DIR_IN evaluation is always done at the PWM_IN 1→0 edge. If DIR_IN equals 0 in both cases, no inversion is executed.

Figure 52 depicts the usage of the PWM_INPUT in an example situation. where PID_TORQUE_TARGET field of the FOC.PID_TORQUE_FLUX_TARGET register is adapted by the PWM input without changing the value of the PID_FLUX_TARGET field of the same register, by making use of the byte shift and mask settings.

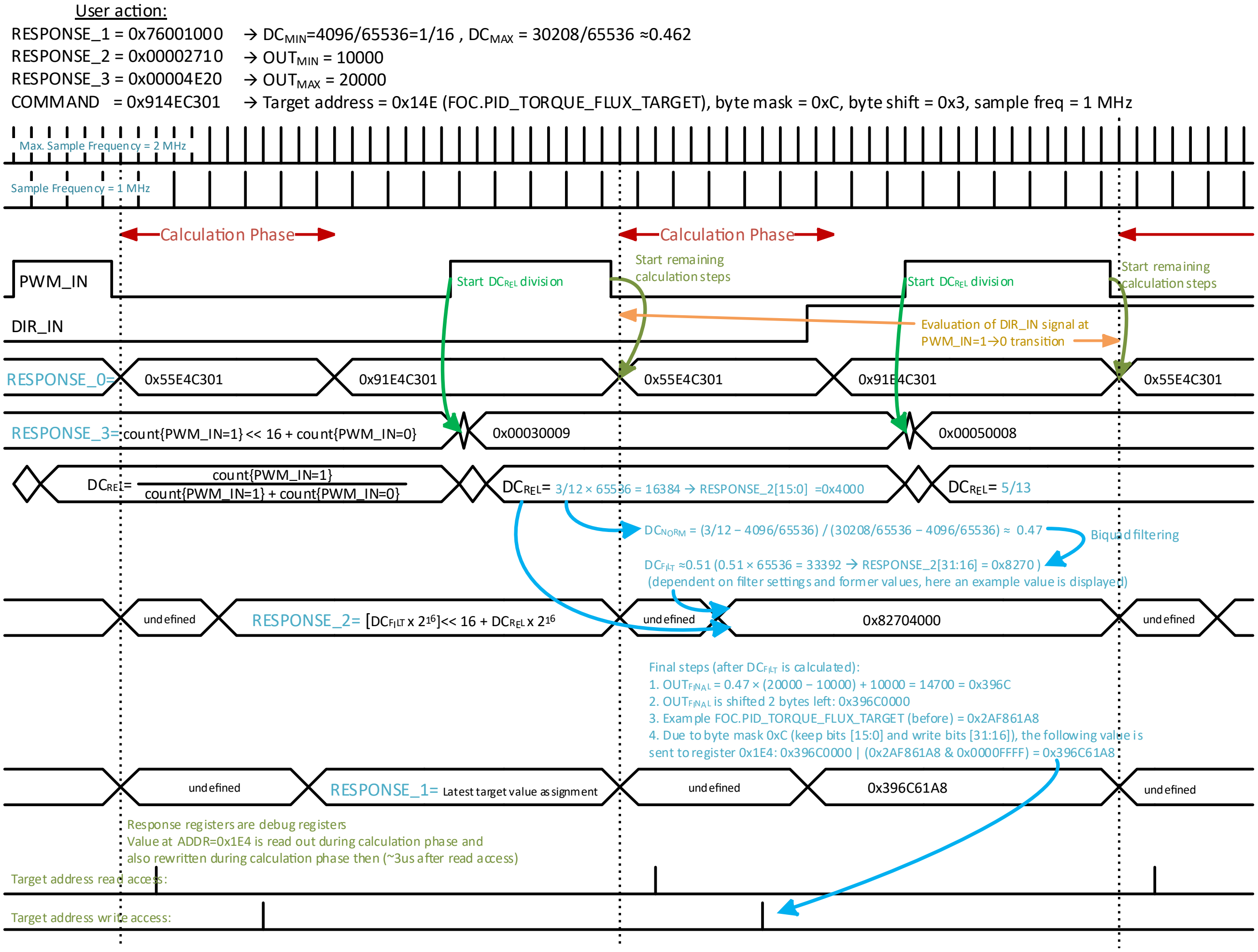

User action:
RESPONSE_1 = 0x76001000 → DCMIN=4096/65536=1/16 , DCMAX = 30208/65536 ≈0.462
RESPONSE_2 = 0x00002710 → OUTMIN = 10000
RESPONSE_3 = 0x00004E20 → OUTMAX = 20000
COMMAND = 0x914EC301 → Target address = 0x14E (FOC.PID_TORQUE_FLUX_TARGET), byte mask = 0xC, byte shift = 0x3, sample freq = 1 MHz
Max. Sample Frequency = 2 MHz
Sample Frequency = 1 MHz
Calculation Phase
Calculation Phase
PWM_IN
Start DCREL division
Start remaining calculation steps
Start DCREL division
Start remaining calculation steps
DIR_IN
Evaluation of DIR_IN signal at PWM_IN=1→0 transition
RESPONSE_0=
0x55E4C301
0x91E4C301
0x55E4C301
0x91E4C301
0x55E4C301
RESPONSE_3= count{PWM_IN=1} << 16 + count{PWM_IN=0}
0x00030009
0x00050008
DCREL= count{PWM_IN=1} / count{PWM_IN=1} + count{PWM_IN=0}
DCREL= 3/12 × 65536 = 16384 → RESPONSE_2[15:0] =0x4000
DCREL= 5/13
DCNORM = (3/12 − 4096/65536) / (30208/65536 − 4096/65536) ≈ 0.47
Biquad filtering
DCFILT ≈0.51 (0.51 × 65536 = 33392 → RESPONSE_2[31:16] = 0x8270)
(dependent on filter settings and former values, here an example value is displayed)
undefined
RESPONSE_2= [DCFILT x 2^16]<< 16 + DCREL x 2^16
undefined
0x82704000
undefined
Final steps (after DCFILT is calculated):
1. OUTFINAL = 0.47 × (20000 − 10000) + 10000 = 14700 = 0x396C
2. OUTFINAL is shifted 2 bytes left: 0x396C0000
3. Example FOC.PID_TORQUE_FLUX_TARGET (before) = 0x2AF861A8
4. Due to byte mask 0xC (keep bits [15:0] and write bits [31:16]), the following value is sent to register 0x1E4: 0x396C0000 | (0x2AF861A8 & 0x0000FFFF) = 0x396C61A8
undefined
RESPONSE_1= Latest target value assignment
undefined
0x396C61A8
undefined
Response registers are debug registers
Value at ADDR=0x1E4 is read out during calculation phase and
also rewritten during calculation phase then (~3us after read access)
Target address read access:
Target address write access:

Figure 52. PWM_INPUT ROM code feature example

PRELIMINARY

AIN_INPUT

Table 51. AIN_INPUT Commands and replies

	User action: Write to register		I/O Controller reply: Read from		Comments
	IO_CONTROLLER....	Value	IO_CONTROLLER....	Value	
1	RESPONSE_1	0xttttzzzz			Normalized duty cycle limits (u16).
2	RESPONSE_2	0xuuuuuuuu			Minimum final output value limit (s32).
3	RESPONSE_3	0xvvvvvvvv			Maximum final output value limit (s32).
4	COMMAND	0x8yyyms00			0xyyy: Target address, 0xm: Byte mask, 0xs: Byte shift
5			RESPONSE_0	0x55yyms00	During calculations, replay data NOT valid!
6			RESPONSE_0	0x8yyyms00	After caclulation phase, reply data valid.
			RESPONSE_1	0xnnnnnnnn	Last target value (s32).
			RESPONSE_2	0xqqqqpppp	0xqqqq: Last filtered duty cycle value. 0xpppp: Last relative duty cycle value.
7	COMMAND	0x00000000			Finish AIN_INPUT feature.

Description:

Repeatedly processes the AIN analog input and optionally the DIR_IN input to use them as control signals for any target register. During this process, the target register is repeatedly written with new values, regardless of the validity of the response registers. The response registers are only for debugging purposes.

Most of the calculation steps are the same as with the PWM_INPUT command. The only difference lies in the acquisition of the relative duty cycle DC_{REL}. For AIN_INPUT, DC_{REL} is directly taken from the analog input measurement, once per PWM cycle. For this reason, RESPONSE_3 register is not used as a reply register, only as definition for the maximum final output value when issuing the COMMAND. Please note that the maximum analog value that can be read is $(2^{15} - 1)$. The definition of the minimum and maximum duty cycle values ("zzzz" and "tttt" values) must be set accordingly. Negative analog input values for AIN are clipped to 0.

Figure 53 depicts the usage of the AIN_INPUT in an example situation where the whole PID_POSITION_TARGET register is adapted by the AIN input, by making use of the byte shift settings.

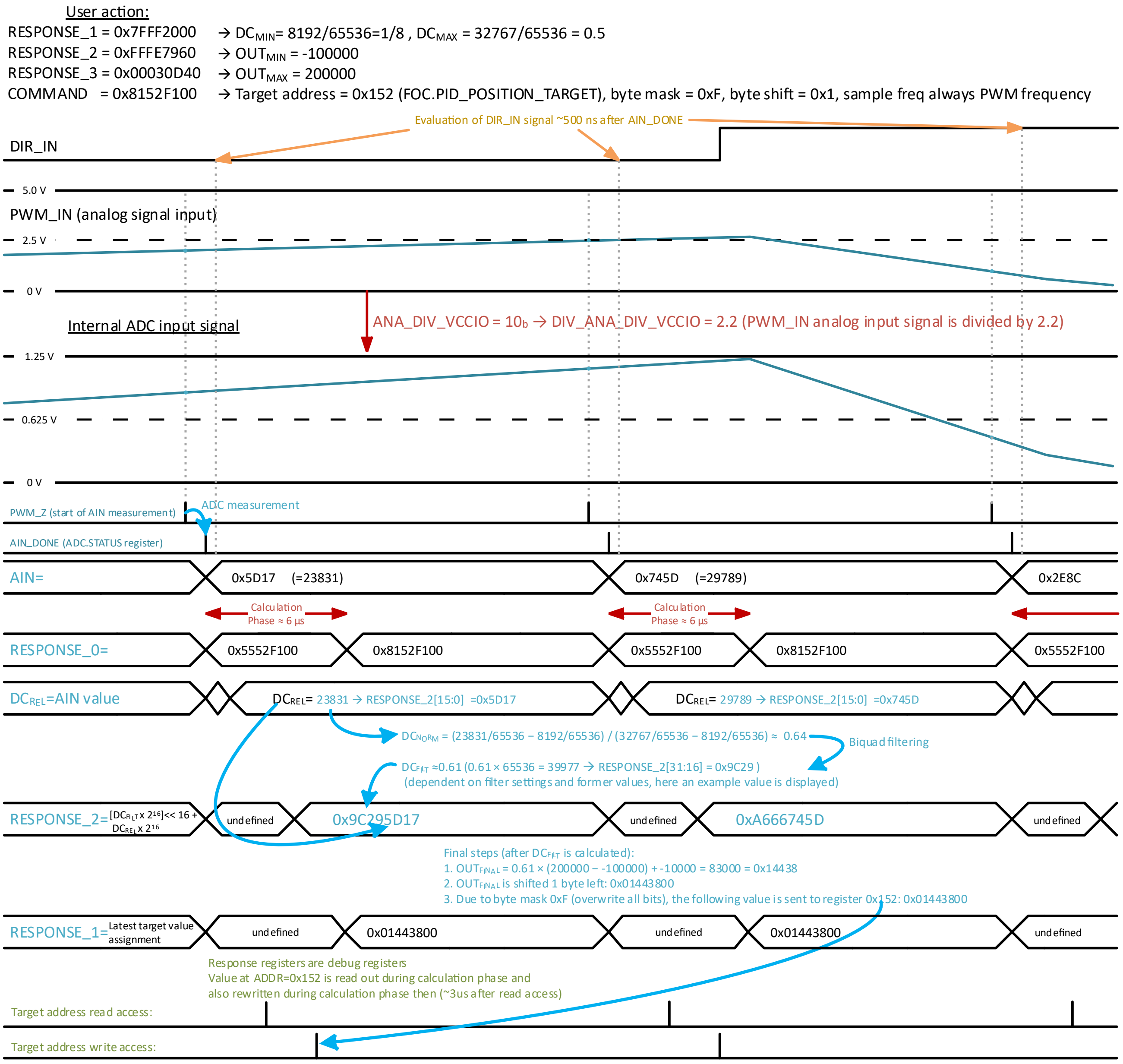

User action:
RESPONSE_1 = 0x7FFF2000 → DC_MIN= 8192/65536=1/8 , DC_MAX = 32767/65536 = 0.5
RESPONSE_2 = 0xFFFE7960 → OUT_MIN = -100000
RESPONSE_3 = 0x00030D40 → OUT_MAX = 200000
COMMAND = 0x8152F100 → Target address = 0x152 (FOC.PID_POSITION_TARGET), byte mask = 0xF, byte shift = 0x1, sample freq always PWM frequency
Evaluation of DIR_IN signal ~500 ns after AIN_DONE
DIR_IN
5.0 V
PWM_IN (analog signal input)
2.5 V
0 V
Internal ADC input signal
ANA_DIV_VCCIO = 10b → DIV_ANA_DIV_VCCIO = 2.2 (PWM_IN analog input signal is divided by 2.2)
1.25 V
0.625 V
0 V
PWM_Z (start of AIN measurement)
ADC measurement
AIN_DONE (ADC.STATUS register)
AIN=
0x5D17 (=23831)
0x745D (=29789)
0x2E8C
Calculation Phase ≈ 6 µs
Calculation Phase ≈ 6 µs
RESPONSE_0=
0x5552F100
0x8152F100
0x5552F100
0x8152F100
0x5552F100
DCREL=AIN value
DCREL= 23831 → RESPONSE_2[15:0] =0x5D17
DCREL= 29789 → RESPONSE_2[15:0] =0x745D
DCNORM = (23831/65536 – 8192/65536) / (32767/65536 – 8192/65536) ≈ 0.64
Biquad filtering
DCFILT ≈0.61 (0.61 × 65536 = 39977 → RESPONSE_2[31:16] = 0x9C29)
(dependent on filter settings and former values, here an example value is displayed)
RESPONSE_2=[DCFILT x 2^16]<< 16 + DCREL x 2^16
undefined
0x9C295D17
undefined
0xA666745D
undefined
Final steps (after DCFILT is calculated):
1. OUTFINAL = 0.61 × (200000 – -100000) + -10000 = 83000 = 0x14438
2. OUTFINAL is shifted 1 byte left: 0x01443800
3. Due to byte mask 0xF (overwrite all bits), the following value is sent to register 0x152: 0x01443800
RESPONSE_1=Latest target value assignment
undefined
0x01443800
undefined
0x01443800
undefined
Response registers are debug registers
Value at ADDR=0x152 is read out during calculation phase and
also rewritten during calculation phase then (~3us after read access)
Target address read access:
Target address write access:

Figure 53. AIN_INPUT ROM code feature example

SPI_ENC

Table 52. SPI_ENC Commands and replies

	User action: Write to register		I/O Controller reply: Read from		Comments
	IO_CONTROLLER....	Value	IO_CONTROLLER....	Value	
1	RESPONSE_1	0xzzzzzzzz			Addr/Data bits to be sent to SPI client.
2	RESPONSE_2	0xssssssss			Response bit mask.
3	COMMAND	0xFywwuuvv			0xy: Mode, 0xww: CSN duration, 0xuu: pin setup + bit count, 0xvv: Period.
4			RESPONSE_0	0x55wwuuvv	During calculations, replay data NOT valid!
5			RESPONSE_0	0xFywwuuvv	After caclulation phase, reply data valid.
			RESPONSE_3	0xrrrrrrrr	Complete client response.
6	COMMAND	0x00000000			Finish SPI_ENC feature.

Description:

Requests and processes angle data from serial clients like SPI or SSI encoders.

The SPI_ENC command allows to make singular client requests, that is, reading and writing the encoder's registers. In this way, an interfaced encoder can be configured during setup.

Additionally, the SPI_ENC command allows to repeatedly read raw data from the SPI encoder. The raw data can be processed and then written into either of the FEEDBACK.PHI_EXT_A or FEEDBACK.PHI_EXT_B registers. This way, the SPI encoder angle can be further utilized by the feedback engine.

In the following, the SPI encoder requests are explained in detail up until the point the processed angle data is written into the feedback engine registers. Refer to *Feedback Engine* chapter for the rest of steps.

Table 53 shows the I/O Controller signals used as the four SPI communication lines.

Table 53. SPI/SSI encoder feature pin assignment

u[7]	CSN	SCK	SDO	SDI
0	DIRECT_OUT[2]	DIRECT_OUT[0]	DIRECT_OUT[1]	DIRECT_IN[0]
1	DIRECT_OUT[1]	DIRECT_OUT[0]	DIRECT_OUT[2]	DIRECT_IN[0]

In the table above, u[7] represents bit 7 of the 0xuu byte from the IO_CONTROLLER.COMMAND.

Note that the connections between the I/O Controller inputs and outputs to the physical I/O pins of TMC6460 are adaptable. Use *Table 54* as a reference for the pins that can be remapped as the relevant I/O Controller inputs and outputs. Refer to the configurations of the CHIP.IO_MATRIX register in the *Pin Remapping* chapter for further details.

Table 54. Pin availability for I/O Controller inputs and outputs

Pin Name	DIRECT_OUT[2]	DIRECT_OUT[0]	DIRECT_OUT[1]	DIRECT_IN[0]
ENC_A		X		
ENC_B				X
ENC_N			X	
HALL_W	X			
REF_L				X
REF_R			X	
SDI				X
SDO		X		
UART_TXD		X	X	
UART_RXD				X

Also note that the communication pins (SDI, SDO, UART_TXD and UART_RXD) are referenced to the V_{CC_IO} supply, while the rest of pins on the table above are referenced to the V_{CC_IOF} supply. This must be considered when choosing the pins to be used with the encoder.

In contrast to the other inputs and outputs of the I/O Controller, the DIRECT_OUT[2] output signal is available as the alternate function of only one pin (HALL_W) of the TMC6460. This results in the fact that if an SPI encoder is to be used, digital and analog Hall sensors, as well as SinCos encoders become unavailable as a simultaneous second feedback source, as they all require the HALL_W pin to function. The 2nd ABN encoder becomes partially unavailable as well, as its N signal also makes use of the HALL_W pin.

In contrast to SPI encoders, SSI encoders only make use of two (SCK and SDO) or sometimes three (SCK, SDO and CSN) signals, as they do not have data input. By allowing to swap the assignment of the CSN and SDO signals among the DIRECT_OUT[1] and DIRECT_OUT[2] outputs, it is made possible to simultaneously use an SSI encoder and a second feedback source as available through the HALL_U, HALL_V and HALL_W pins and their alternate functions.

The COMMAND itself contains several configurations for the SPI communication:

- The "y" byte defines SPI mode and request type:
  - COMMAND[27:26] configures the commonly known SPI mode (SPI mode 0, 1, 2, or 3) that is used.
  - COMMAND[25:24] defines the request type:

00_b: One datagram will be sent, and the response – which is the reply of the previous datagram for typical SPI communication – is stored in IO_CONTROLLER.RESPONSE_3. Useful for SPI write requests.

01_b: Two identical datagrams will be sent. The response to the first sent datagram which is received with the second datagram is stored in IO_CONTROLLER.RESPONSE_3. Useful for SPI read requests.

10_b: Datagrams will be sent repeatedly to obtain SPI raw data. The acquired data will be processed and eventually written into the FEEDBACK.PHI_EXT_A register as regular feedback data.

11_b: Datagrams will be sent repeatedly to obtain SPI raw data. The acquired data will be processed and eventually written into the FEEDBACK.PHI_EXT_B register as regular feedback data.

- The two “vv” bytes (COMMAND[7:0]) define SCK_HALF_PERIOD that represents half of the encoder clock period.
  - SCK switching frequency is defined by this value.
  - If this value is lower than 10, the fastest possible SCK frequency (SCK_FREQ) of f_{SYS} / 20 = 3MHz is used.
  - Else, the set value defines the SCK frequency with:
    - $SCK\_FREQ\ = \frac{f_{SYS}}{(SCK\_HALF\_PERIOD+1)\times 2} \qquad with\ f_{SYS} = 60\ MHz$
  - This results in SCK frequency range of 117.2kHz…3MHz
- The two “uu” bytes define the pin assignment (COMMAND[15]) as explained above, and the number of data bits (COMMAND[12:8]) that are sent and received for each SPI datagram (SPI_BIT_CNT).
  - SPI_BIT_CNT = COMMAND[12:8] + 1.
  - The maximum bit count is SPI_BIT_CNT = 32. Longer datagrams are not supported by this ROM feature.
- The two “ww” bytes (COMMAND[23:16]) define the number of system clock cycles SPI_CSH_CNT to wait between two consecutive SPI datagrams (when CSN = 1), resulting in the idle time of t_{SPI_CSH}. If SPI request type equals 0, this configuration is not considered by the feature, as only one datagram is sent.
  - t_{SPI_CSH} is proportional to the square of SPI_CSH_CNT, but shortest possible time is 6µs.
    - $t_{SPI\_CSH} > \frac{SPI\_CSH\_CNT\times(SPI\_CSH\_CNT+1)}{f_{SYS}} + t_{SPI\_CSH\_OFST} \qquad with\ f_{SYS} = 60\ MHz$
  - $t_{SPI\_CSH\_OFST}$ ≈ 2µs if SPI request type = 1, else $t_{SPI\_CSH\_OFST}$ ≈ 3µs.
    - This means that t_{SPI_CSH} is slightly different for repeated requests than for a two-datagram request.

The address and optional data that will be sent to the SPI encoder client is defined in 0xzzzzzzzz of the IO_CONTROLLER.RESPONSE_1 register. Note that the data is sent MSB first, which means that bit[31] is always sent first. For instance, in case the 12-bit 0xABC data needs to be sent to the client, IO_CONTROLLER.RESPONSE_1 must be set as 0xABC00000.

The reply of the encoder client is stored in IO_CONTROLLER.RESPONSE_3. There, the reply data is assigned to the lower bytes. In case of our prior example with 12-bit data, IO_CONTROLLER.RESPONSE_3 will be 0x00ABCDEF in case the reply contains 0xDEF as response to the request of 0xABC. The response contains parts of the sent datagram because, internally, the output register is the same as the input register. In this case only the lower 12 bits are essential, as these are the reply from the encoder client.

For the repeated request types, the response value is automatically processed and stored in the defined register of feedback engine according to the chosen request type.

The restrictions for the automatic angle data acquisition and processing to work are as follows: 1) the response data containing the angle value must not exceed 32 bits, and 2) it must be possible to obtain the angle value from a single response datagram. For instance, it is not possible to combine the data from two consecutive datagrams. Most SPI encoder clients provide the option to stream out the raw angle value in a single datagram when the data is separated into several consecutive registers.

For some SPI encoder clients, angle data cannot be directly interpreted from the raw response datagram, for instance because certain status bits are part of the datagram. For such cases, the SPI_ENC feature provides a mask functionality to automatically process the raw response data.

The value set in IO_CONTROLLER.RESPONSE_2 (= 0xssssssss) serves as the mask datagram. All data bits of the client response datagram that are relevant for the encoder’s angle value must be set to 1 in the mask datagram, whereas data bits that should be omitted must be set to 0. If the status bits are placed in between the angle data bits, they can also be masked out, with the only restriction that these bits must be located at the end or the start of a byte. The final angle value is arranged by right shifting the bits that are not masked. *Table 55* shows examples for different mask values.

Table 55. SPI/SSI encoder mask/shift examples

SPI_BIT_CNT	Bit mask (RESPONSE_2)	Encoder raw data	Final feed-back data	Masked raw data bits	Comment
12	0x000003FC	0x00000789	0x000000E2	[11:10] [1:0]	Final value 8 bits wide: [7:0] = Raw[9:2]
12	0x00000E7F	0x00000789	0x00000189	[8:7]	Final value 10 bits wide: [9:0] = Raw[11:9][6:0]
14	0x00001CFC	0x00003E9F	0x000001E7	[13] [9:8] [1:0]	Final value 9 bits wide: [8:0] = Raw[12:10][7:2]
14	0x000006FF	0x00003E9F	0x0000039F	[13:11] [8]	Final value 8 bits wide: [7:0] = Raw[10:9][7:0]
16	0x0000FFF0	0x00008049	0x00000804	[3:0]	Final value 12 bits wide: [11:0] = Raw[15:4]
16	0x00007E1F	0x00008049	0x00000009	[15] [8:5]	Final value 11 bits wide: [10:0] = Raw[14:9][4:0]
16	0x00006C1F				**Not valid**: Masked bits are in the middle of byte2
24	0x00F83F78	0x007C7864	0x00003F8C	[18:14] [7] [2:0]	Final value 15 bits wide: [14:0] = Raw[23:19][13:8][6:3]

After the raw response data is arranged (masked and shifted), the resulting angle value is automatically written into the target register FEEDBACK.PHI_EXT_A or FEEDBACK.PHI_EXT_B to be further processed by the feedback engine.

The update rate of the angle data is defined by the SCK frequency and the idle time as described above.

To finish repeated SPI encoder requests, send an “idle” command IO_CONTROLLER.COMMAND = 0x00000000. The response data will be retained for as long as no new command is issued.

In *Figure 54*, a continuous SPI encoder (SPI mode 3) data read out is depicted. Only four out of eight transferred bits are used as the angle value (the remaining consists of for instance status flags), which is defined by the byte mask in IO_CONTROLLER.RESPONSE_2 register. Several timings that have been explained above are also displayed here. Further on, IO_CONTROLLER.RESPONSE_1 register is filled with the value 0x56023456. This contains the data sent to the SPI encoder (0x56), as well as some other data that is received from the SPI encoder (0x023456) after issuing the command. These last bits are neither sent to the SPI encoder nor transferred to the internal FEEDBACK.PHI_EXT_A register, and they are only shown here to depict the combined usage of the input and output stream data.

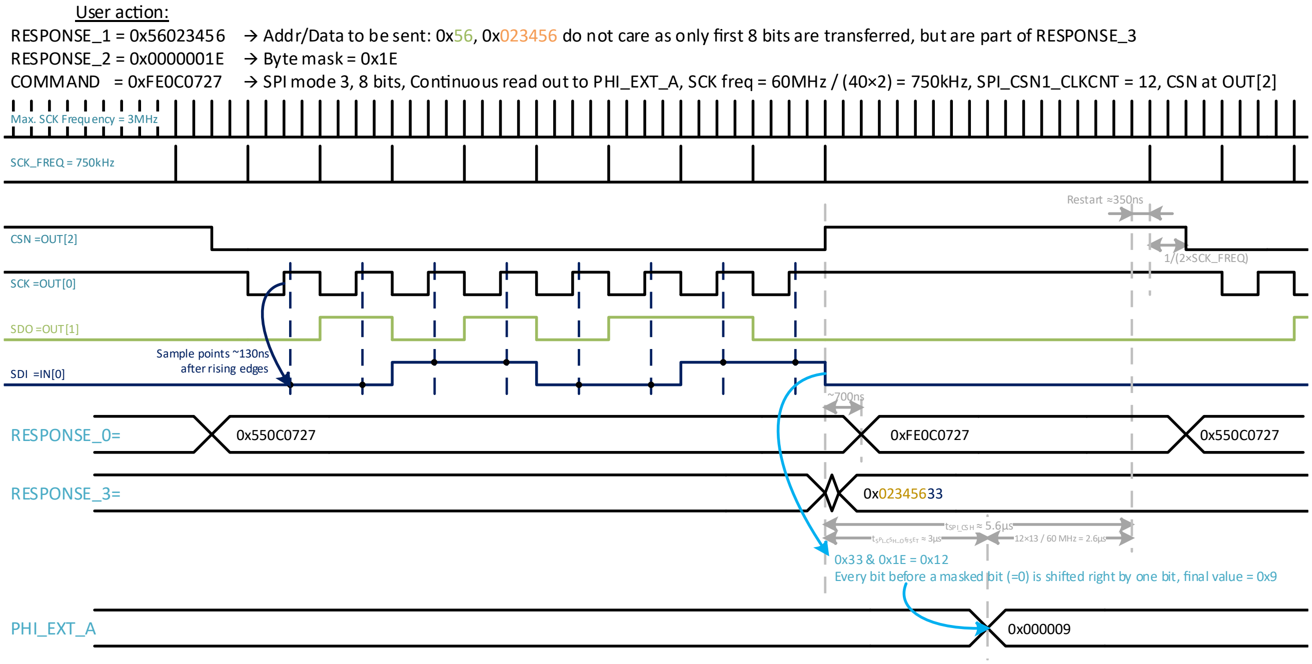

User action:
RESPONSE_1 = 0x56023456 → Addr/Data to be sent: 0x56, 0x023456 do not care as only first 8 bits are transferred, but are part of RESPONSE_3
RESPONSE_2 = 0x0000001E → Byte mask = 0x1E
COMMAND = 0xFE0C0727 → SPI mode 3, 8 bits, Continuous read out to PHI_EXT_A, SCK freq = 60MHz / (40×2) = 750kHz, SPI_CSN1_CLKCNT = 12, CSN at OUT[2]

Figure 54. Example of a continuous SPI encoder read out. Target register is PHI_EXT_A.

A one-time SPI encoder (SPI mode 0) data read out is depicted *Figure 55*. This uses the COMMAND[25:24] = 01_b, which results in only two sent datagrams. The client’s response, contained in the second datagram, can then be read out from IO_CONTROLLER.RESPONSE_3 register. Filling IO_CONTROLLER.RESPONSE_2 register is not required as there is no transfer of data to either of FEEDBACK.PHI_EXT_A or FEEDBACK.PHI_EXT_B registers.

User action:

RESPONSE_1 = 0xA5000000 → Addr/Data to be sent: 0xA500, e.g. read out 8 bit SPI encoder data from address 0xA5
RESPONSE_2 = 0x00000000 → Byte mask = 0 (do not care for non-continuous requests)
COMMAND = 0xF10A8F0E → SPI mode 0, 16 bits, 2 datagrams to read out, SCK freq = 60MHz / (15×2) = 2MHz, SPI_CSN1_CLKCNT = 10, CSN at OUT[1]

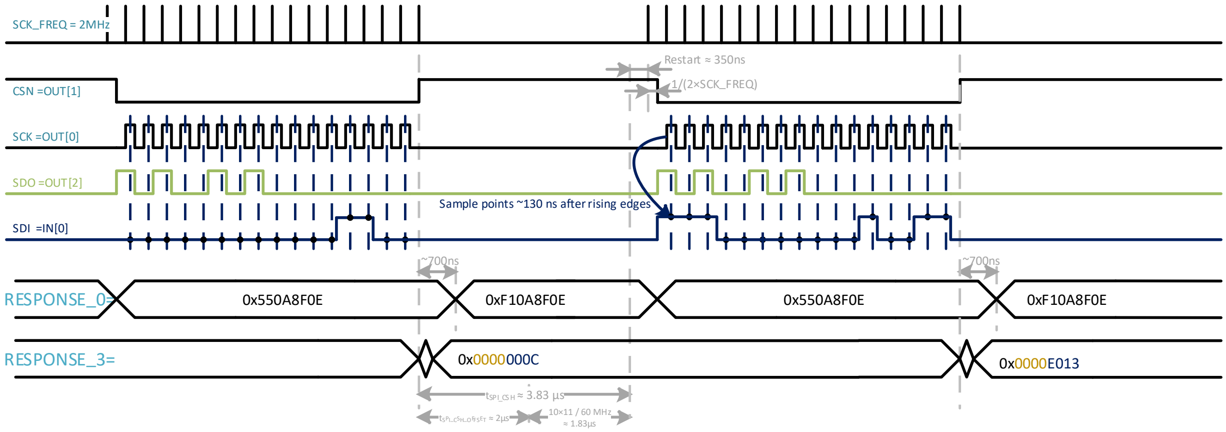


Figure 55. Example of a one-time SPI encoder read out by sending the same datagram twice to get reply in RESPONSE_3 register.

Finally, SSI encoders could be also utilized with the same command sequence. Figure 56 shows this application setup. The data path from SDO to the SSI encoder is not required. CSN is optional for some SSI encoders when they only use one clock input and one data output. In this example, the usage of the byte mask is also depicted. Most encoders do not provide a data stream that is full of gaps, but as long as the bits that are not part of the angle value are located at the start or the end of a byte, this ROM code feature is able to handle the masking and shifting to provide a valid angle value for the internal FEEDBACK.PHI_EXT_A or FEEDBACK.PHI_EXT_B registers.

User action:

RESPONSE_1 = 0x00000000 → Addr/Data not required: SSI only utilizes SCK, SDI, and optionally CSN
RESPONSE_2 = 0x01FF783C → Byte mask = 0x1FF783C, bits[27:25][15][10:6][1:0] are omitted
COMMAND = 0xFB009B1D → SPI mode 2, 28 bits, Continuous read out to PHI_EXT_B,
SCK freq = 60MHz / (30×2) = 1MHz, SPI_CSN1_CLKCNT = 0, CSN at OUT[1]

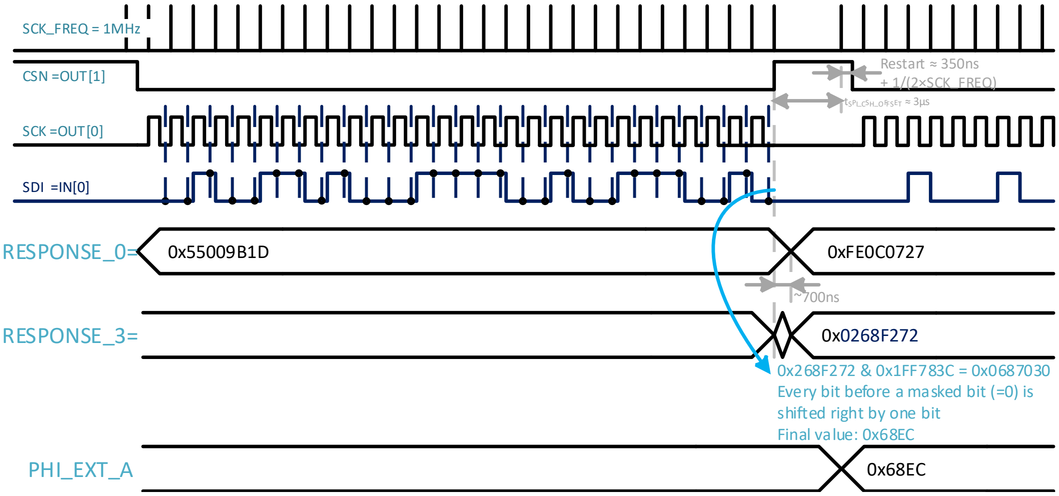


Figure 56. Example of a continuous SSI encoder read out. Target register is PHI_EXT_B.

Application Information

The TMC6460 can drive BLDC motors / 3-phase PMSM with different feedback systems. Some examples are shown in the application block diagrams below.

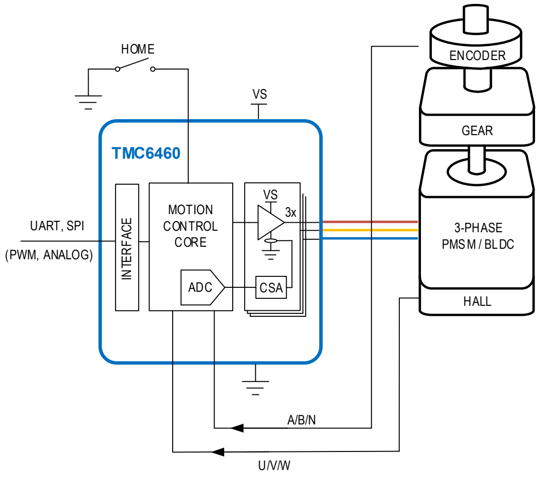


Figure 57. BLDC with incremental optical encoder behind the gearbox and digital Hall sensor on the motor

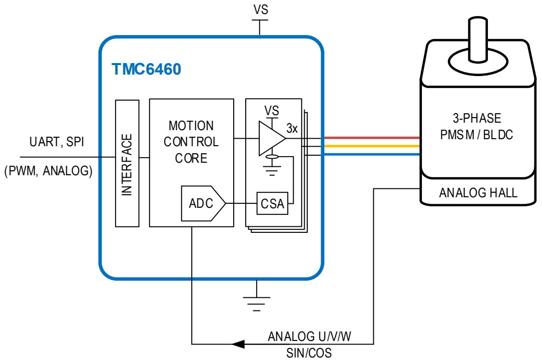


Figure 58. BLDC with Analog Hall sensor or SIN/COS encoder

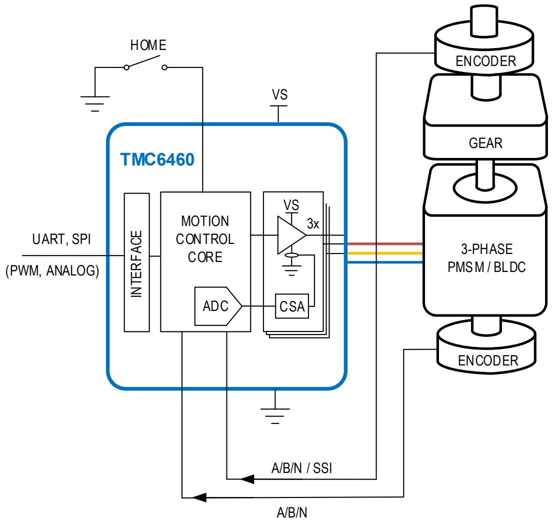


Figure 59. BLDC with two encoders, one incremental A/B/N encoder and a second incremental or absolute (e.g. SSI) encoder. One encoder mounted on the motor and a second one behind the gearbox.

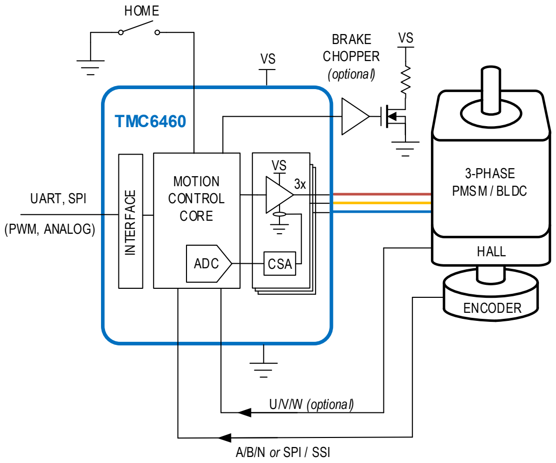


Figure 60. BLDC With Optical Encoder and Digital Hall on Motor + Brake Chopper

Typical Application Circuits

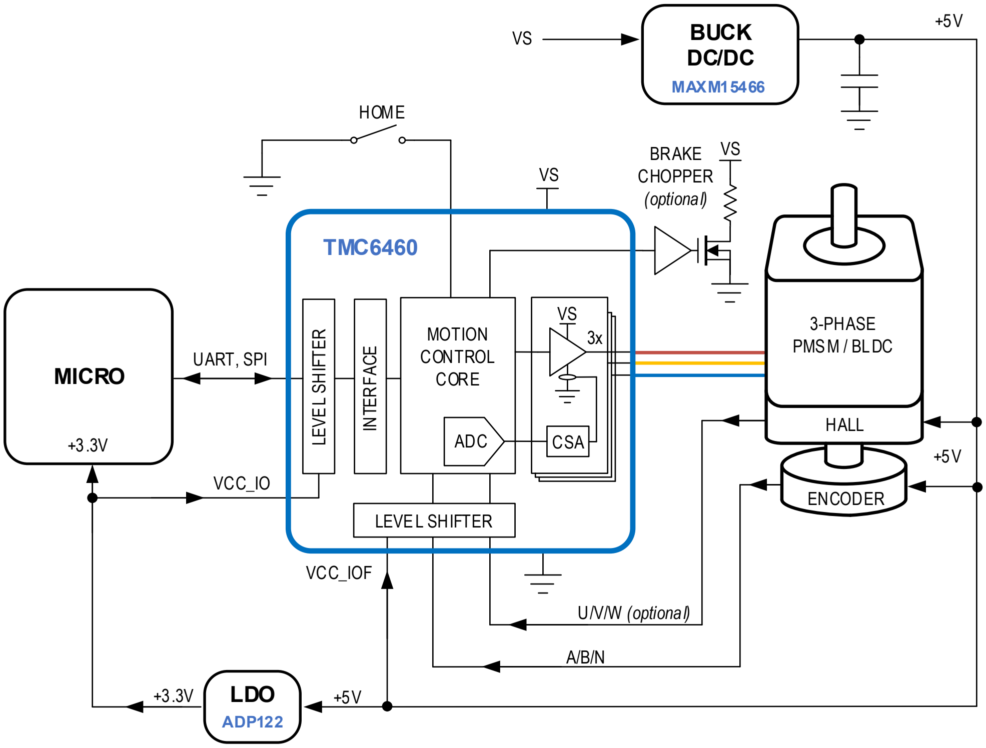


Figure 61. Typical Application Circuit Block Diagram

Register Map

Register Overview

An overview of all registers of the TMC6460 is provided below.

ADDRESS & NAME	FIELDS							MSB (TOP LEFT) TO LSB (BOTTOM RIGHT)
0x000 **CHIP.ID**	ID[31:24]							
	ID[23:16]							
	ID[15:8]							
	ID[7:0]							
0x001 **CHIP.VARIANT**	VARIANT[31:24]							
	VARIANT[23:16]							
	VARIANT[15:8]							
	VARIANT[7:0]							
0x002 **CHIP.REVISION**		REVISION[30:24]						
	REVISION[23:16]							
	REVISION[15:8]							
	REVISION[7:0]							
0x004 **CHIP.INPUTS_RAW**	PWM_IN	DIR_IN	DIRECT_IN1	DIRECT_IN0	REF_H	REF_R	REF_L	HALL_W
	HALL_V	HALL_U	ENC2_N	ENC2_B	ENC2_A	ENC_N	ENC_B	ENC_A
	PIN_DRV_EN		PIN_UART_RXD_RAW	PIN_PWM_IN_RAW	PIN_CSN_RAW	PIN_SCK_RAW	PIN_SDO_RAW	PIN_SDI_RAW
	PIN_REF_R_RAW	PIN_REF_L_RAW	PIN_HALL_W_RAW	PIN_HALL_V_RAW	PIN_HALL_U_RAW	PIN_ENC_N_RAW	PIN_ENC_B_RAW	PIN_ENC_A_RAW
0x005 **CHIP.OUTPUTS_RAW**						INVALID_DIFF_ENC_N	INVALID_DIFF_ENC_B	INVALID_DIFF_ENC_A
	FAULTN_OUT	INT_OUT	PWM_W_H	PWM_W_L	PWM_V_H	PWM_V_L	PWM_U_H	PWM_U_L
0x006 **CHIP.IO_MATRIX**	PIN_FAULTN		PIN_UART_TXD		PIN_UART_RXD		PIN_PWM_IN	
	PIN_SCK	PIN_SDO		PIN_SDI		PIN_REF_R		PIN_REF_L[1]
	PIN_REF_L[0]	PIN_HALL_W			PIN_HALL_V			PIN_HALL_U[2]
	PIN_HALL_U[1:0]		PIN_ENC_N		PIN_ENC_B		PIN_ENC_A	
0x007 **CHIP.IO_PU_PD**					PIN_UART_RXD_PD	PIN_PWM_IN_PD	PIN_SCK_PD	PIN_SDI_PD
		PIN_FAULTN_PU		PIN_SDO_PU	PIN_UART_RXD_PU	PIN_PWM_IN_PU	PIN_SCK_PU	PIN_SDI_PU
	PIN_REF_R_PD	PIN_REF_L_PD	PIN_HALL_W_PD	PIN_HALL_V_PD	PIN_HALL_U_PD	PIN_ENC_N_PD	PIN_ENC_B_PD	PIN_ENC_A_PD
	PIN_REF_R_PU	PIN_REF_L_PU	PIN_HALL_W_PU	PIN_HALL_V_PU	PIN_HALL_U_PU	PIN_ENC_N_PU	PIN_ENC_B_PU	PIN_ENC_A_PU
0x008 **CHIP.IO_CONFIG**	DIFF_DELAY						USE_PIN_REF_R_AS_REF_L	USE_PIN_REF_L_AS_REF_R
	ANA_DIV_VCCIOF		ANA_DIV_VCCIO		PWM_IN_FLT		REF_FLT	
	HALL_FLT		ENC_FLT		INV_PIN_RXD	INV_PIN_PWM_IN	INV_PIN_SCK	INV_PIN_SDI
	INV_PIN_REF_R	INV_PIN_REF_L	INV_PIN_HALL_W	INV_PIN_HALL_V	INV_PIN_HALL_U	INV_PIN_ENC_N	INV_PIN_ENC_B	INV_PIN_ENC_A

ADDRESS & NAME	FIELDS				MSB (TOP LEFT) TO LSB (BOTTOM RIGHT)			
0x009 **CHIP.STATUS_FLAGS**	GDRV_ON_STATE	SYS_READY_STATE	OVERTEMP_WARN_STATUS	OV_VM_LIMIT_FAIL_STATUS	VM_TEMP_CLIPPED_STATUS	AIN_CLIPPED_STATUS	I_CLIPPED_STATUS	CURRENT_OVERLOAD_STATUS
	HALL_FAIL_STATUS	ABN_FAIL_STATUS	RAMP_REF_H_STATUS	RAMP_REF_LR_STATUS	RAMP_V_ZERO_STATUS	RAMP_V_REACHED_STATUS	RAMP_TARGET_REACHED_STATUS	RAMP_STALL_FAIL_STATUS
	IO_CONTROLLER_STATUS	DIFF_ENC_FAIL_STATUS	OT_FAIL_STATUS	SHRT_FAIL_STATUS	VEL_FAIL_STATUS	TEMP_LIMIT_FAIL_STATUS	ADC_FAIL_STATUS	EXT_RES_FAIL_STATUS
	CLK_PLL_FAIL_STATUS	POWER_FAIL_STATUS	UV_VM_STATUS	CP_FAIL_STATUS	VCC_IOF_FAIL_STATUS	UART_FAIL_STATUS	SPI_FAIL_STATUS	PWRUP_FAIL_STATUS
0x00A **CHIP.EVENTS**	GDRV_ON_EVENT	SYS_READY_EVENT	OVERTEMP_WARN_EVENT	OV_VM_LIMIT_FAIL_EVENT	VM_TEMP_CLIPPED_EVENT	AIN_CLIPPED_EVENT	I_CLIPPED_EVENT	CURRENT_OVERLOAD_EVENT
	HALL_FAIL_EVENT	ABN_FAIL_EVENT	RAMP_REF_H_EVENT	RAMP_REF_LR_EVENT	RAMP_V_ZERO_EVENT	RAMP_V_REACHED_EVENT	RAMP_TARGET_REACHED_EVENT	RAMP_STALL_FAIL_EVENT
	IO_CONTROLLER_EVENT	DIFF_ENC_FAIL_EVENT	OT_FAIL_EVENT	SHRT_FAIL_EVENT	VEL_FAIL_EVENT	TEMP_LIMIT_FAIL_EVENT	ADC_FAIL_EVENT	EXT_RES_FAIL_EVENT
	CLK_PLL_FAIL_EVENT	POWER_FAIL_EVENT	UV_VM_EVENT	CP_FAIL_EVENT	VCCIOF_FAIL_EVENT	UART_FAIL_EVENT	SPI_FAIL_EVENT	RST_EVENT
0x00B **CHIP.FAULTN_INT_MASK**	FAULTN_INT_MASK[31:24]							
	FAULTN_INT_MASK[23:16]							
	FAULTN_INT_MASK[15:8]							
	FAULTN_INT_MASK[7:0]							
0x00C **CHIP.SPI_STATUS_MASK**	SPI_STATUS_MASK[31:24]							
	SPI_STATUS_MASK[23:16]							
	SPI_STATUS_MASK[15:8]							
	SPI_STATUS_MASK[7:0]							
0x040 **CLK_CTRL.CONFIG**								
				CLOCK_DIVIDER				
			CLK_FSM_EN	PWM_CLK_EN	ADC_CLK_EN	PLL_EN	PLL_SRC	COMMIT
0x041 **CLK_CTRL.STATUS**	CLK_FSM_FLG							
	CLK_FSM_ERR_RPT							
		PLL_READY	PLL_LOCK_ON	PLL_ERR	CLK_STUCK	CLK_FSM_ERR	CLK_1M0_TIMEOUT	CLK_1M0_OK
0x080 **ADC.CONFIG**								
					CSA_AZ_FLT_EXP			
					NRST_ADC_1	NRST_ADC_0		

ADDRESS & NAME	FIELDS							MSB (TOP LEFT) TO LSB (BOTTOM RIGHT)
0x081 **ADC.ADC_VERSION**	VERSION_NUMBER[31:24]							
	VERSION_NUMBER[23:16]							
	VERSION_NUMBER[15:8]							
	VERSION_NUMBER[7:0]							
0x082 **ADC.I2_I1_RAW**	I2[15:8]							
	I2[7:0]							
	I1[15:8]							
	I1[7:0]							
0x083 **ADC.VM_I3_RAW**	VM[15:8]							
	VM[7:0]							
	I3[15:8]							
	I3[7:0]							
0x084 **ADC.TEMP_RAW**	TEMP_INT[15:8]							
	TEMP_INT[7:0]							
	TEMP_EXT[15:8]							
	TEMP_EXT[7:0]							
0x085 **ADC.** **AIN_V_AIN_U_RAW**	AIN_V[15:8]							
	AIN_V[7:0]							
	AIN_U[15:8]							
	AIN_U[7:0]							
0x086 **ADC.** **AIN_AIN_W_RAW**	AIN[15:8]							
	AIN[7:0]							
	AIN_W[15:8]							
	AIN_W[7:0]							
0x08A **ADC.STATUS**	I123_FAIL	NO_ACK_ ADC			USE_ADC_ EXT_ VALID_EXT	ADC_EXT_ VALID_EXT	USE_ADC_ I_VALID_ EXT	ADC_I_ VALID_EXT
	ADC_EXT_ VALID	ADC_I_ VALID	AIN_DONE	AIN_W_ DONE	AIN_V_ DONE	AIN_U_ DONE	I3_DONE	I2_DONE
	I1_DONE	TEMP_ EXT_DONE	TEMP_ EXT_ CLIPPED	TEMP_INT_ DONE	TEMP_INT_ CLIPPED	VM_DONE	VM_ CLIPPED	AIN_ CLIPPED
	AIN_W_ CLIPPED	AIN_V_ CLIPPED	AIN_U_ CLIPPED	I3_CLIPPED	I2_CLIPPED	I1_CLIPPED	ADC_1_ READY	ADC_0_ READY
0x08C **ADC.I123**	I123_LIMIT[15:8]							
	I123_LIMIT[7:0]							
	I123[15:8]							
	I123[7:0]							
0x0C0 **MCC_ADC.** **I_GEN_CONFIG**								
			STATUS					
								MEAS_ MODE
0x0C1 **MCC_ADC.IW_IU**	IW[15:8]							
	IW[7:0]							
	IU[15:8]							
	IU[7:0]							
0x0C2 **MCC_ADC.IV**								
	IV[15:8]							
	IV[7:0]							

ADDRESS & NAME	FIELDS				MSB (TOP LEFT) TO LSB (BOTTOM RIGHT)			
0x0C3 **MCC_ADC.CSA_GAIN**								
	DYN_GAIN_HYST							
		DYN_GAIN_4X_EN	DYN_GAIN_3X_EN	DYN_GAIN_2X_EN	DYN_GAIN_ACT		CSA_GAIN	
0x0C4 **MCC_ADC.EVENTS**								
		CURRENT_OVERLOAD_STATUS	TEMP_INT_LIMIT_EXCEEDED	TEMP_EXT_LIMIT_EXCEEDED			MEAS_DONE	ADC_CLIPPED
0x0C5 **MCC_ADC.DYN_GAIN_LIMITS_4X_3X**	DYN_GAIN_LIMIT_4X[15:8]							
	DYN_GAIN_LIMIT_4X[7:0]							
	DYN_GAIN_LIMIT_3X[15:8]							
	DYN_GAIN_LIMIT_3X[7:0]							
0x0C6 **MCC_ADC.DYN_GAIN_LIMIT_2X**								
	DYN_GAIN_LIMIT_2X[15:8]							
	DYN_GAIN_LIMIT_2X[7:0]							
0x0C7 **MCC_ADC.TEMP_LIMITS**	TEMP_INT_LIMIT[15:8]							
	TEMP_INT_LIMIT[7:0]							
	TEMP_EXT_LIMIT[15:8]							
	TEMP_EXT_LIMIT[7:0]							
0x0C8 **MCC_ADC.CURRENT_OVERLOAD**								
	CURRENT_OVERLOAD[15:8]							
	CURRENT_OVERLOAD[7:0]							
0x100 **MCC_CONFIG.MOTOR_MOTION**								
					VELOCITY_FF_EN	TORQUE_FF_VISC_FRIC_EN	TORQUE_FF_COULOMB FRICTION_EN	TORQUE_FF_ACC_EN
	RAMP_USE_PHI_E	RAMP_EN	RAMP_MODE	MOTION_MODE				MOTOR_TYPE[1]
	MOTOR_TYPE[0]	N_POLE_PAIRS						
0x101 **MCC_CONFIG.GDRV**	CHARGE_PUMP_EN			LP_MODE_EN				USE_INTERNAL_R_REF
				OVERTEMP_LATCH				DRV_EN_BIT
			OCP_AUTORETRY		OCP_DEGLITCH		OCP_DETECTION_MODE	
			LS_RES_ON				SLEW_RATE	
0x102 **MCC_CONFIG.PWM**	FLAT_BOTTOM_OFFSET[15:8]							
	FLAT_BOTTOM_OFFSET[7:0]							
			SV_MODE			CHOP		
0x103 **MCC_CONFIG.PWM_PERIOD**	AUTO_KIRCHOFF_LIM[15:8]							
	AUTO_KIRCHOFF_LIM[7:0]							
	MAX_COUNT[15:8]							
	MAX_COUNT[7:0]							

ADDRESS & NAME	FIELDS							MSB (TOP LEFT) TO LSB (BOTTOM RIGHT)
0x104 MCC_CONFIG. BRAKE_CHOPPER_ LIMITS	UPPER_SWITCH_LIM[15:8]							
	UPPER_SWITCH_LIM[7:0]							
	LOWER_SWITCH_LIM[15:8]							
	LOWER_SWITCH_LIM[7:0]							
0x105 MCC_CONFIG. MCC_STATUS								OVERTEMP_CH3_EVENT
	OVERTEMP_CH2_EVENT	OVERTEMP_CH1_EVENT	SHRT2V_CH3_EVENT	SHRT2V_CH2_EVENT	SHRT2V_CH1_EVENT	SHRT2G_CH3_EVENT	SHRT2G_CH2_EVENT	SHRT2G_CH1_EVENT
								OVERTEMP_CH3_STATUS
	OVERTEMP_CH2_STATUS	OVERTEMP_CH1_STATUS	SHRT2V_CH3_STATUS	SHRT2V_CH2_STATUS	SHRT2V_CH1_STATUS	SHRT2G_CH3_STATUS	SHRT2G_CH2_STATUS	SHRT2G_CH1_STATUS
0x106 MCC_CONFIG. TORQUE_FF_ACC_ CONFIG								
						RAMP_ACC_SHIFT		
	RAMP_ACC_GAIN[15:8]							
	RAMP_ACC_GAIN[7:0]							
0x107 MCC_CONFIG. TORQUE_FF_VISC_ FRIC_CONFIG								
						RAMP_VEL_SHIFT		
	RAMP_VEL_GAIN[15:8]							
	RAMP_VEL_GAIN[7:0]							
0x108 MCC_CONFIG. TORQUE_FF_ COLOUMB_FRIC								
	TORQUE_FF_COLOUMB_FRIC[15:8]							
	TORQUE_FF_COLOUMB_FRIC[7:0]							
0x109 MCC_CONFIG. TORQUE_ FEEDFORWARD								
	TORQUE_FEEDFORWARD[15:8]							
	TORQUE_FEEDFORWARD[7:0]							
0x140 FOC.PID_CONFIG								
		POSITION_SAMPLING						
	VELOCITY_SHIFT				POSITION_NORM_I		POSITION_NORM_P	
	VELOCITY_NORM_I		VELOCITY_NORM_P		CURRENT_NORM_I	CURRENT_NORM_P	OVERWRITE_TARGET	KEEP_POS_TARGET
0x141 FOC.PID_U_S_MAX								
	U_S_MAX[15:8]							
	U_S_MAX[7:0]							
0x142 FOC. PID_FLUX_COEFF	FLUX_P[15:8]							
	FLUX_P[7:0]							
	FLUX_I[15:8]							
	FLUX_I[7:0]							
0x143 FOC. PID_TORQUE_COEFF	TORQUE_P[15:8]							
	TORQUE_P[7:0]							
	TORQUE_I[15:8]							
	TORQUE_I[7:0]							
0x144 FOC. PID_FIELDWEAK_ COEFF	FIELDWEAK_P[15:8]							
	FIELDWEAK_P[7:0]							
	FIELDWEAK_I[15:8]							
	FIELDWEAK_I[7:0]							

ADDRESS & NAME	FIELDS				MSB (TOP LEFT) TO LSB (BOTTOM RIGHT)			
0x145 FOC. PID_VELOCITY_COEFF	VELOCITY_P[15:8]							
	VELOCITY_P[7:0]							
	VELOCITY_I[15:8]							
	VELOCITY_I[7:0]							
0x146 FOC. PID_POSITION_COEFF	POSITION_P[15:8]							
	POSITION_P[7:0]							
	POSITION_I[15:8]							
	POSITION_I[7:0]							
0x147 FOC. PID_POSITION_ TOLERANCE		PID_POSITION_TOLERANCE[30:24]						
	PID_POSITION_TOLERANCE[23:16]							
	PID_POSITION_TOLERANCE[15:8]							
	PID_POSITION_TOLERANCE[7:0]							
0x148 FOC. PID_POSITION_ TOLERANCE_DELAY								
	PID_POSITION_TOLERANCE_DELAY[15:8]							
	PID_POSITION_TOLERANCE_DELAY[7:0]							
0x149 FOC. PID_UQ_UD_LIMITS								
	PID_UQ_UD_LIMITS[15:8]							
	PID_UQ_UD_LIMITS[7:0]							
0x14A FOC. PID_TORQUE_FLUX_ LIMITS		PID_TORQUE_LIMIT[14:8]						
	PID_TORQUE_LIMIT[7:0]							
		PID_FLUX_LIMIT[14:8]						
	PID_FLUX_LIMIT[7:0]							
0x14B FOC. PID_VELOCITY_LIMIT		PID_VELOCITY_LIMIT[30:24]						
	PID_VELOCITY_LIMIT[23:16]							
	PID_VELOCITY_LIMIT[15:8]							
	PID_VELOCITY_LIMIT[7:0]							
0x14C FOC. PID_POSITION_LIMIT_ LOW	PID_POSITION_LIMIT_LOW[31:24]							
	PID_POSITION_LIMIT_LOW[23:16]							
	PID_POSITION_LIMIT_LOW[15:8]							
	PID_POSITION_LIMIT_LOW[7:0]							
0x14D FOC. PID_POSITION_LIMIT_ HIGH	PID_POSITION_LIMIT_HIGH[31:24]							
	PID_POSITION_LIMIT_HIGH[23:16]							
	PID_POSITION_LIMIT_HIGH[15:8]							
	PID_POSITION_LIMIT_HIGH[7:0]							
0x14E FOC. PID_TORQUE_FLUX_ TARGET	PID_TORQUE_TARGET[15:8]							
	PID_TORQUE_TARGET[7:0]							
	PID_FLUX_TARGET[15:8]							
	PID_FLUX_TARGET[7:0]							
0x14F FOC. PID_TORQUE_FLUX_ OFFSET	PID_TORQUE_OFFSET[15:8]							
	PID_TORQUE_OFFSET[7:0]							
	PID_FLUX_OFFSET[15:8]							
	PID_FLUX_OFFSET[7:0]							
0x150 FOC. PID_VELOCITY_ TARGET	PID_VELOCITY_TARGET[31:24]							
	PID_VELOCITY_TARGET[23:16]							
	PID_VELOCITY_TARGET[15:8]							
	PID_VELOCITY_TARGET[7:0]							
0x151 FOC. PID_VELOCITY_ OFFSET	PID_VELOCITY_OFFSET[31:24]							
	PID_VELOCITY_OFFSET[23:16]							
	PID_VELOCITY_OFFSET[15:8]							
	PID_VELOCITY_OFFSET[7:0]							

ADDRESS & NAME	FIELDS							MSB (TOP LEFT) TO LSB (BOTTOM RIGHT)
0x152 FOC. PID_POSITION_TARGET	PID_POSITION_TARGET[31:24]							
	PID_POSITION_TARGET[23:16]							
	PID_POSITION_TARGET[15:8]							
	PID_POSITION_TARGET[7:0]							
0x153 FOC. PID_TORQUE_FLUX_ACTUAL	PID_TORQUE_ACTUAL[15:8]							
	PID_TORQUE_ACTUAL[7:0]							
	PID_FLUX_ACTUAL[15:8]							
	PID_FLUX_ACTUAL[7:0]							
0x154 FOC. PID_VELOCITY_ACTUAL	PID_VELOCITY_ACTUAL[31:24]							
	PID_VELOCITY_ACTUAL[23:16]							
	PID_VELOCITY_ACTUAL[15:8]							
	PID_VELOCITY_ACTUAL[7:0]							
0x155 FOC. PID_POSITION_ACTUAL	PID_POSITION_ACTUAL[31:24]							
	PID_POSITION_ACTUAL[23:16]							
	PID_POSITION_ACTUAL[15:8]							
	PID_POSITION_ACTUAL[7:0]							
0x156 FOC. PID_POSITION_ACTUAL_OFFSET	PID_POSITION_ACTUAL_OFFSET[31:24]							
	PID_POSITION_ACTUAL_OFFSET[23:16]							
	PID_POSITION_ACTUAL_OFFSET[15:8]							
	PID_POSITION_ACTUAL_OFFSET[7:0]							
0x157 FOC. PID_TORQUE_ERROR								
	PID_TORQUE_ERROR[15:8]							
	PID_TORQUE_ERROR[7:0]							
0x158 FOC. PID_FLUX_ERROR								
	PID_FLUX_ERROR[15:8]							
	PID_FLUX_ERROR[7:0]							
0x159 FOC. PID_VELOCITY_ERROR	PID_VELOCITY_ERROR[31:24]							
	PID_VELOCITY_ERROR[23:16]							
	PID_VELOCITY_ERROR[15:8]							
	PID_VELOCITY_ERROR[7:0]							
0x15A FOC. PID_VELOCITY_ERROR_MAX		PID_VELOCITY_ERROR_MAX[30:24]						
	PID_VELOCITY_ERROR_MAX[23:16]							
	PID_VELOCITY_ERROR_MAX[15:8]							
	PID_VELOCITY_ERROR_MAX[7:0]							
0x15B FOC. PID_POSITION_ERROR	PID_POSITION_ERROR[31:24]							
	PID_POSITION_ERROR[23:16]							
	PID_POSITION_ERROR[15:8]							
	PID_POSITION_ERROR[7:0]							
0x15C FOC. PID_POSITION_ERROR_MAX		PID_POSITION_ERROR_MAX[30:24]						
	PID_POSITION_ERROR_MAX[23:16]							
	PID_POSITION_ERROR_MAX[15:8]							
	PID_POSITION_ERROR_MAX[7:0]							
0x15D FOC. PID_TORQUE_INTEGRATOR	PID_TORQUE_INTEGRATOR[31:24]							
	PID_TORQUE_INTEGRATOR[23:16]							
	PID_TORQUE_INTEGRATOR[15:8]							
	PID_TORQUE_INTEGRATOR[7:0]							
0x15E FOC. PID_FLUX_INTEGRATOR	PID_FLUX_INTEGRATOR[31:24]							
	PID_FLUX_INTEGRATOR[23:16]							
	PID_FLUX_INTEGRATOR[15:8]							
	PID_FLUX_INTEGRATOR[7:0]							

ADDRESS & NAME	FIELDS MSB (TOP LEFT) TO LSB (BOTTOM RIGHT)
0x15F FOC.PID_VELOCITY_INTEGRATOR	PID_VELOCITY_INTEGRATOR[31:24]
	PID_VELOCITY_INTEGRATOR[23:16]
	PID_VELOCITY_INTEGRATOR[15:8]
	PID_VELOCITY_INTEGRATOR[7:0]
0x160 FOC.PID_POSITION_INTEGRATOR	PID_POSITION_INTEGRATOR[31:24]
	PID_POSITION_INTEGRATOR[23:16]
	PID_POSITION_INTEGRATOR[15:8]
	PID_POSITION_INTEGRATOR[7:0]
0x161 FOC.PIDIN_TORQUE_FLUX_TARGET	PIDIN_TORQUE_TARGET[15:8]
	PIDIN_TORQUE_TARGET[7:0]
	PIDIN_FLUX_TARGET[15:8]
	PIDIN_FLUX_TARGET[7:0]
0x162 FOC.PIDIN_VELOCITY_TARGET	PIDIN_VELOCITY_TARGET[31:24]
	PIDIN_VELOCITY_TARGET[23:16]
	PIDIN_VELOCITY_TARGET[15:8]
	PIDIN_VELOCITY_TARGET[7:0]
0x163 FOC.PIDIN_POSITION_TARGET	PIDIN_POSITION_TARGET[31:24]
	PIDIN_POSITION_TARGET[23:16]
	PIDIN_POSITION_TARGET[15:8]
	PIDIN_POSITION_TARGET[7:0]
0x164 FOC.PIDIN_TORQUE_FLUX_TARGET_LIMITED	PIDIN_TORQUE_TARGET_LIMITED[15:8]
	PIDIN_TORQUE_TARGET_LIMITED[7:0]
	PIDIN_FLUX_TARGET_LIMITED[15:8]
	PIDIN_FLUX_TARGET_LIMITED[7:0]
0x165 FOC.PIDIN_VELOCITY_TARGET_LIMITED	PIDIN_VELOCITY_TARGET_LIMITED[31:24]
	PIDIN_VELOCITY_TARGET_LIMITED[23:16]
	PIDIN_VELOCITY_TARGET_LIMITED[15:8]
	PIDIN_VELOCITY_TARGET_LIMITED[7:0]
0x166 FOC.PIDIN_POSITION_TARGET_LIMITED	PIDIN_POSITION_TARGET_LIMITED[31:24]
	PIDIN_POSITION_TARGET_LIMITED[23:16]
	PIDIN_POSITION_TARGET_LIMITED[15:8]
	PIDIN_POSITION_TARGET_LIMITED[7:0]
0x167 FOC.FOC_IBETA_IALPHA	IBETA[15:8]
	IBETA[7:0]
	IALPHA[15:8]
	IALPHA[7:0]
0x168 FOC.FOC_IQ_ID	IQ[15:8]
	IQ[7:0]
	ID[15:8]
	ID[7:0]
0x169 FOC.FOC_UQ_UD	UQ[15:8]
	UQ[7:0]
	UD[15:8]
	UD[7:0]
0x16A FOC.FOC_UQ_UD_LIMITED	UQ_LIMITED[15:8]
	UQ_LIMITED[7:0]
	UD_LIMITED[15:8]
	UD_LIMITED[7:0]
0x16B FOC.FOC_UBETA_UALPHA	UBETA[15:8]
	UBETA[7:0]
	UALPHA[15:8]
	UALPHA[7:0]

ADDRESS & NAME	FIELDS				MSB (TOP LEFT) TO LSB (BOTTOM RIGHT)			
0x16C FOC.FOC_UW_UU	UW[15:8]							
	UW[7:0]							
	UU[15:8]							
	UU[7:0]							
0x16D FOC.FOC_UV								
	UV[15:8]							
	UV[7:0]							
0x16E FOC.PWM_V_U	PWM_V[15:8]							
	PWM_V[7:0]							
	PWM_U[15:8]							
	PWM_U[7:0]							
0x16F FOC.PWM_W								
	PWM_W[15:8]							
	PWM_W[7:0]							
0x170 FOC.FOC_STATUS								
					FOC_STATUS			
0x171 FOC.U_S_ACTUAL_I_S_ACTUAL	U_S_ACTUAL[15:8]							
	U_S_ACTUAL[7:0]							
	I_S_ACTUAL[15:8]							
	I_S_ACTUAL[7:0]							
0x172 FOC.P_MOTOR	P_MOTOR[31:24]							
	P_MOTOR[23:16]							
	P_MOTOR[15:8]							
	P_MOTOR[7:0]							
0x173 FOC.I_T_ACTUAL								
	I_T_ACTUAL[15:8]							
	I_T_ACTUAL[7:0]							
0x174 FOC.PRBS_AMPLITUDE	PRBS_AMPLITUDE[31:24]							
	PRBS_AMPLITUDE[23:16]							
	PRBS_AMPLITUDE[15:8]							
	PRBS_AMPLITUDE[7:0]							
0x175 FOC.PRBS_DOWN_SAMPLING_RATIO								
	PRBS_DOWN_SAMPLING_RATIO							
0x180 BIQUAD.BIQUAD_EN								
							TORQUE_EN	VELOCITY_EN
0x181 BIQUAD.VELOCITY_A1								
	COEFF_VELOCITY_A1[23:16]							
	COEFF_VELOCITY_A1[15:8]							
	COEFF_VELOCITY_A1[7:0]							
0x182 BIQUAD.VELOCITY_A2								
	COEFF_VELOCITY_A2[23:16]							
	COEFF_VELOCITY_A2[15:8]							
	COEFF_VELOCITY_A2[7:0]							

ADDRESS & NAME	FIELDS					MSB (TOP LEFT) TO LSB (BOTTOM RIGHT)		
0x183 BIQUAD.VELOCITY_B0								
	COEFF_VELOCITY_B0[23:16]							
	COEFF_VELOCITY_B0[15:8]							
	COEFF_VELOCITY_B0[7:0]							
0x184 BIQUAD.VELOCITY_B1								
	COEFF_VELOCITY_B1[23:16]							
	COEFF_VELOCITY_B1[15:8]							
	COEFF_VELOCITY_B1[7:0]							
0x185 BIQUAD.VELOCITY_B2								
	COEFF_VELOCITY_B2[23:16]							
	COEFF_VELOCITY_B2[15:8]							
	COEFF_VELOCITY_B2[7:0]							
0x186 BIQUAD.TORQUE_A1								
	COEFF_TORQUE_A1[23:16]							
	COEFF_TORQUE_A1[15:8]							
	COEFF_TORQUE_A1[7:0]							
0x187 BIQUAD.TORQUE_A2								
	COEFF_TORQUE_A2[23:16]							
	COEFF_TORQUE_A2[15:8]							
	COEFF_TORQUE_A2[7:0]							
0x188 BIQUAD.TORQUE_B0								
	COEFF_TORQUE_B0[23:16]							
	COEFF_TORQUE_B0[15:8]							
	COEFF_TORQUE_B0[7:0]							
0x189 BIQUAD.TORQUE_B1								
	COEFF_TORQUE_B1[23:16]							
	COEFF_TORQUE_B1[15:8]							
	COEFF_TORQUE_B1[7:0]							
0x18A BIQUAD.TORQUE_B2								
	COEFF_TORQUE_B2[23:16]							
	COEFF_TORQUE_B2[15:8]							
	COEFF_TORQUE_B2[7:0]							
0x1C0 RAMPER.TIME_CONFIG	T_VMAX[15:8]							
	T_VMAX[7:0]							
	T_ZEROWAIT[15:8]							
	T_ZEROWAIT[7:0]							
0x1C1 RAMPER.SWITCH_MODE								
						STOP_ON_V_DEVIATION	STOP_ON_POS_DEVIATION	FORCE_HARD_STOP
	SOFT_STOP_EN	STALL_STOP_EN		LATCH_REF_H_INACTIVE	LATCH_REF_H_ACTIVE	LATCH_REF_R_INACTIVE	LATCH_REF_R_ACTIVE	LATCH_REF_L_INACTIVE
	LATCH_REF_L_ACTIVE	SWAP_RAMP_REF_LR	RAMP_REF_H_POL	RAMP_REF_R_POL	RAMP_REF_L_POL	RAMP_REF_H_STOP_EN	RAMP_REF_R_STOP_EN	RAMP_REF_L_STOP_EN
0x1C3 RAMPER.PHI_E	PHI_E_OFFSET[15:8]							
	PHI_E_OFFSET[7:0]							
	PHI_E[15:8]							
	PHI_E[7:0]							
0x1C4 RAMPER.A1								
		A1[22:16]						
	A1[15:8]							
	A1[7:0]							

ADDRESS & NAME	FIELDS					MSB (TOP LEFT) TO LSB (BOTTOM RIGHT)		
0x1C5 RAMPER.A2								
		A2[22:16]						
	A2[15:8]							
	A2[7:0]							
0x1C6 RAMPER.A_MAX								
		A_MAX[22:16]						
	A_MAX[15:8]							
	A_MAX[7:0]							
0x1C7 RAMPER.D1								
		D1[22:16]						
	D1[15:8]							
	D1[7:0]							
0x1C8 RAMPER.D2								
		D2[22:16]						
	D2[15:8]							
	D2[7:0]							
0x1C9 RAMPER.D_MAX								
		D_MAX[22:16]						
	D_MAX[15:8]							
	D_MAX[7:0]							
0x1CA RAMPER.V_START								
		V_START[22:16]						
	V_START[15:8]							
	V_START[7:0]							
0x1CB RAMPER.V1						V1[26:24]		
	V1[23:16]							
	V1[15:8]							
	V1[7:0]							
0x1CC RAMPER.V2						V2[26:24]		
	V2[23:16]							
	V2[15:8]							
	V2[7:0]							
0x1CD RAMPER.V_STOP								
		V_STOP[22:16]						
	V_STOP[15:8]							
	V_STOP[7:0]							
0x1CE RAMPER.V_MAX						V_MAX[26:24]		
	V_MAX[23:16]							
	V_MAX[15:8]							
	V_MAX[7:0]							
0x1CF RAMPER. ACCELERATION								
	ACCELERATION[23:16]							
	ACCELERATION[15:8]							
	ACCELERATION[7:0]							
0x1D0 RAMPER.V_ACTUAL					V_ACTUAL[27:24]			
	V_ACTUAL[23:16]							
	V_ACTUAL[15:8]							
	V_ACTUAL[7:0]							
0x1D1 RAMPER.POSITION	POSITION[31:24]							
	POSITION[23:16]							
	POSITION[15:8]							
	POSITION[7:0]							

ADDRESS & NAME	FIELDS							MSB (TOP LEFT) TO LSB (BOTTOM RIGHT)
0x1D2 RAMPER. POSITION_LATCH	POSITION_LATCH[31:24]							
	POSITION_LATCH[23:16]							
	POSITION_LATCH[15:8]							
	POSITION_LATCH[7:0]							
0x1D3 RAMPER. POSITION_ACTUAL_ LATCH	POSITION_ACTUAL_LATCH[31:24]							
	POSITION_ACTUAL_LATCH[23:16]							
	POSITION_ACTUAL_LATCH[15:8]							
	POSITION_ACTUAL_LATCH[7:0]							
0x1D4 RAMPER.STATUS								
							STALL_IN_POSITION_ERR	STALL_IN_V_ERR
		T_ZEROWAIT_ACTIVE	V_ZERO	POSITION_REACHED_STATUS	V_REACHED_STATUS			RAMP_REF_H_STOP_STATUS
	RAMP_REF_R_STOP_STATUS	RAMP_REF_L_STOP_STATUS				RAMP_REF_H_STATUS	RAMP_REF_R_STATUS	RAMP_REF_L_STATUS
0x1D5 RAMPER.EVENTS								
			LATCH_REF_R_READY	LATCH_REF_L_READY	LATCH_REF_H_READY	SECOND_MOVE_EVENT	STALL_STOP_EVENT	POSITION_REACHED_EVENT
0x200 EXT_CTRL.VOLTAGE	UQ[15:8]							
	UQ[7:0]							
	UD[15:8]							
	UD[7:0]							
0x202 EXT_CTRL.PWM_V_U	PWM_V[15:8]							
	PWM_V[7:0]							
	PWM_U[15:8]							
	PWM_U[7:0]							
0x203 EXT_CTRL.PWM_W								
	PWM_W[15:8]							
	PWM_W[7:0]							
0x240 FEEDBACK. CONF_CH_A					SRC_SEL_A			
	CPR_INV_A[23:16]							
	CPR_INV_A[15:8]							
	CPR_INV_A[7:0]							
0x241 FEEDBACK. CONF_CH_B					SRC_SEL_B			
	CPR_INV_B[23:16]							
	CPR_INV_B[15:8]							
	CPR_INV_B[7:0]							
0x242 FEEDBACK. PHI_E_OFFSET								
	PHI_E_OFFSET[15:8]							
	PHI_E_OFFSET[7:0]							

ADDRESS & NAME	FIELDS				MSB (TOP LEFT) TO LSB (BOTTOM RIGHT)			
0x243 FEEDBACK.LUT	RDATA							
	ADDR							
		LOOKUP_B_GAIN				LOOKUP_A_GAIN		
						LOOKUP_ B_EN	LOOKUP_ A_EN	SPLIT_ MODE_EN
0x244 FEEDBACK. VELOCITY_FRQ_CONF								
	VELOCITY_SCALING[15:8]							
	VELOCITY_SCALING[7:0]							
	VELOCITY_SAMPLING							VELOCITY_ SYNC_SRC
0x245 FEEDBACK. VELOCITY_PER_CONF	POS_DEV_TIMER[15:8]							
	POS_DEV_TIMER[7:0]							
		POS_DEV_MIN[14:8]						
	POS_DEV_MIN[7:0]							
0x246 FEEDBACK. VELOCITY_PER_FILTER								
						FILTER_WIDTH		
0x247 FEEDBACK. PHI_CONVERTED	PHI_CONVERTED_B[15:8]							
	PHI_CONVERTED_B[7:0]							
	PHI_CONVERTED_A[15:8]							
	PHI_CONVERTED_A[7:0]							
0x248 FEEDBACK.CH_A	PHI_EXTRAPOLATED_AB[15:8]							
	PHI_EXTRAPOLATED_AB[7:0]							
	PHI_LOOKUP_A[15:8]							
	PHI_LOOKUP_A[7:0]							
0x249 FEEDBACK.CH_B								
	PHI_LOOKUP_B[15:8]							
	PHI_LOOKUP_B[7:0]							
0x24A FEEDBACK. VELOCITY_FRQ	VELOCITY_FRQ[31:24]							
	VELOCITY_FRQ[23:16]							
	VELOCITY_FRQ[15:8]							
	VELOCITY_FRQ[7:0]							
0x24B FEEDBACK. VELOCITY_PER	VELOCITY_PER[31:24]							
	VELOCITY_PER[23:16]							
	VELOCITY_PER[15:8]							
	VELOCITY_PER[7:0]							
0x24C FEEDBACK. LUT_WDATA	WDATA[31:24]							
	WDATA[23:16]							
	WDATA[15:8]							
	WDATA[7:0]							
0x24D FEEDBACK. PHI_EXT_A								
	PHI_EXT_A[23:16]							
	PHI_EXT_A[15:8]							
	PHI_EXT_A[7:0]							
0x24E FEEDBACK. PHI_EXT_B								
	PHI_EXT_B[23:16]							
	PHI_EXT_B[15:8]							
	PHI_EXT_B[7:0]							

ADDRESS & NAME	FIELDS				MSB (TOP LEFT) TO LSB (BOTTOM RIGHT)			
0x24F FEEDBACK.VELOCITY_EXT	VELOCITY_EXT[31:24]							
	VELOCITY_EXT[23:16]							
	VELOCITY_EXT[15:8]							
	VELOCITY_EXT[7:0]							
0x250 FEEDBACK.OUTPUT_CONF								
	VELOCITY_SELECTION		VELOCITY_SRC	POSITION_SRC			PHI_E_SRC	
	PHI_E_MUL_FACTOR							
0x251 FEEDBACK.PHI_E								
	PHI_E_FOC[15:8]							
	PHI_E_FOC[7:0]							
0x252 FEEDBACK.PHI_DIFF_LIMIT								
	PHI_DIFF_LIMIT[15:8]							
	PHI_DIFF_LIMIT[7:0]							
0x280 ABN.CONFIG	CPR[23:16]							
	CPR[15:8]							
	CPR[7:0]							
					INV_DIR		CLN	COMBINED_N
0x281 ABN.COUNT								
	COUNT[23:16]							
	COUNT[15:8]							
	COUNT[7:0]							
0x282 ABN.COUNT_N_CAPTURE								
	COUNT_N_CAPTURE[23:16]							
	COUNT_N_CAPTURE[15:8]							
	COUNT_N_CAPTURE[7:0]							
0x283 ABN.COUNT_N_WRITE								
	COUNT_N_WRITE[23:16]							
	COUNT_N_WRITE[15:8]							
	COUNT_N_WRITE[7:0]							
0x284 ABN.EVENTS								
							N_EVENT	INVALID_SIGNAL_EVENT
0x2C0 ABN2.CONFIG	CPR[23:16]							
	CPR[15:8]							
	CPR[7:0]							
					INV_DIR		CLN	COMBINED_N
0x2C1 ABN2.COUNT								
	COUNT[23:16]							
	COUNT[15:8]							
	COUNT[7:0]							
0x2C2 ABN2.COUNT_N_CAPTURE								
	COUNT_N_CAPTURE[23:16]							
	COUNT_N_CAPTURE[15:8]							
	COUNT_N_CAPTURE[7:0]							

ADDRESS & NAME	FIELDS				MSB (TOP LEFT) TO LSB (BOTTOM RIGHT)			
0x2C3 ABN2.COUNT_N_WRITE								
	COUNT_N_WRITE[23:16]							
	COUNT_N_WRITE[15:8]							
	COUNT_N_WRITE[7:0]							
0x2C4 ABN2.EVENTS								
							N_EVENT	INVALID_SIGNAL_EVENT
0x300 HALL.MAP_CONFIG								
		ANA_HALL_MAP				HALL_MAP		
0x301 HALL.DIG_COUNT								
						COUNT		
0x302 HALL.ANA_CONFIG								
		ADC_CLIPPED	N_PULSE_OUT	USE_N_PULSE	TWO_CYCLE_MODE_EN	N_PULSE_POL	N_PULSE_EDGE	ANA_MODE
	N_PULSE_THRESHOLD[15:8]							
	N_PULSE_THRESHOLD[7:0]							
0x303 HALL.ANA_UX_CONFIG	UX_SCALE[15:8]							
	UX_SCALE[7:0]							
	UX_OFFSET[15:8]							
	UX_OFFSET[7:0]							
0x304 HALL.ANA_VN_CONFIG	VN_SCALE[15:8]							
	VN_SCALE[7:0]							
	VN_OFFSET[15:8]							
	VN_OFFSET[7:0]							
0x305 HALL.ANA_WY_CONFIG	WY_SCALE[15:8]							
	WY_SCALE[7:0]							
	WY_OFFSET[15:8]							
	WY_OFFSET[7:0]							
0x306 HALL.ANA_UX_OUT								
	UX_OUT[15:8]							
	UX_OUT[7:0]							
0x307 HALL.ANA_VN_OUT								
	VN_OUT[15:8]							
	VN_OUT[7:0]							
0x308 HALL.ANA_WY_OUT								
	WY_OUT[15:8]							
	WY_OUT[7:0]							
0x309 HALL.ANA_OUT	ANA_PHI[15:8]							
	ANA_PHI[7:0]							
	ANA_PHI_N_CAPTURE[15:8]							
	ANA_PHI_N_CAPTURE[7:0]							

ADDRESS & NAME	FIELDS				MSB (TOP LEFT) TO LSB (BOTTOM RIGHT)			
0x30A HALL.DIG_EVENTS								
								INVALID_SIGNAL_EVENT
0x340 UART.CONTROL	RTMI_SAMPLING							
		RTMI_EN	RTMI_CRC_EN	NORMAL_CRC_EN	RX_FILTER_EN	AUTOBAUD_EN		TIMEOUT_PRE_DIVIDER_EN
				MANTISSA_LIMIT[12:8]				
	MANTISSA_LIMIT[7:0]							
0x341 UART.TIMEOUT	COUNTER[15:8]							
	COUNTER[7:0]							
	LIMIT[15:8]							
	LIMIT[7:0]							
0x342 UART.STATUS								
				AUTOBAUD_VALID		TX_ACTIVE		RX_ACTIVE
0x343 UART.EVENTS								
	INVALID_CRC_EVENT	INVALID_STOP_BIT_EVENT	INVALID_START_BIT_EVENT	RX_IDLE_TIMEOUT_EVENT	RTMI_INTERRUPT_EVENT	NOISY_DATA_EVENT		AUTOBAUD_SYNC_FAIL_EVENT
0x344 UART.RTMI_CH_0								
								CH_0_EN
							CH_0_ADDR[9:8]	
	CH_0_ADDR[7:0]							
0x345 UART.RTMI_CH_1								
								CH_1_EN
							CH_1_ADDR[9:8]	
	CH_1_ADDR[7:0]							
0x346 UART.RTMI_CH_2								
								CH_2_EN
							CH_2_ADDR[9:8]	
	CH_2_ADDR[7:0]							
0x347 UART.RTMI_CH_3								
								CH_3_EN
							CH_3_ADDR[9:8]	
	CH_3_ADDR[7:0]							
0x348 UART.RTMI_CH_4								
								CH_4_EN
							CH_4_ADDR[9:8]	
	CH_4_ADDR[7:0]							
0x349 UART.RTMI_CH_5								
								CH_5_EN
							CH_5_ADDR[9:8]	
	CH_5_ADDR[7:0]							

ADDRESS & NAME	FIELDS							MSB (TOP LEFT) TO LSB (BOTTOM RIGHT)
0x34A UART.RTMI_CH_6								
								CH_6_EN
							CH_6_ADDR[9:8]	
	CH_6_ADDR[7:0]							
0x34B UART.RTMI_CH_7								
								CH_7_EN
							CH_7_ADDR[9:8]	
	CH_7_ADDR[7:0]							
0x380 IO_CONTROLLER. CONTROL								
								RESET
0x381 IO_CONTROLLER. COMMAND	COMMAND[31:24]							
	COMMAND[23:16]							
	COMMAND[15:8]							
	COMMAND[7:0]							
0x382 IO_CONTROLLER. RESPONSE_0	RESPONSE_0[31:24]							
	RESPONSE_0[23:16]							
	RESPONSE_0[15:8]							
	RESPONSE_0[7:0]							
0x383 IO_CONTROLLER. RESPONSE_1	RESPONSE_1[31:24]							
	RESPONSE_1[23:16]							
	RESPONSE_1[15:8]							
	RESPONSE_1[7:0]							
0x384 IO_CONTROLLER. RESPONSE_2	RESPONSE_2[31:24]							
	RESPONSE_2[23:16]							
	RESPONSE_2[15:8]							
	RESPONSE_2[7:0]							
0x385 IO_CONTROLLER. RESPONSE_3	RESPONSE_3[31:24]							
	RESPONSE_3[23:16]							
	RESPONSE_3[15:8]							
	RESPONSE_3[7:0]							

Register Detailed Description

A detailed explanation of all registers of the TMC6460 is given below. The accessibility register field types are as follows: R (read-only), RW (read/write), W1C (write ‘1’ to clear), W1S (write ‘1’ to set).

0x000: CHIP.ID

Default: 0x36343630

Die ID number.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:0] ID	R, unsigned 0x36343630	Chip ID

0x001: CHIP.VARIANT

Default: 0x00000000

Die variant.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:0] VARIANT	R, unsigned 0x00000000	Chip variant

0x002: CHIP.REVISION

Default: 0x00000013

Die Revision.

BITS & NAME	TYPE & RESET	DESCRIPTION
[30:0] REVISION	R, unsigned 0x00000013	IC revision

0x004: CHIP.INPUTS_RAW

Default: 0x00000000

Input signals for PWM_IN, DIR_IN, reference switch inputs, digital Hall inputs, digital ABN encoder inputs, I/O Controller inputs and communication pin inputs. Signals available for direct read out during system setup and validation. The raw signals are the actual input levels read at the physical pins. The non raw signals are the logical level once the pin signal gets routed through the matrix, and goes through the input processing step.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31] PWM_IN	R 0x0	Internal assigned PWM_IN input value
[30] DIR_IN	R 0x0	Internal assigned Direction input value (forwarded to I/O controller)
[29] DIRECT_IN1	R 0x0	Direct Input Value 1: Internal assigned input 1 (forwarded to I/O controller) input value
[28] DIRECT_IN0	R 0x0	Direct Input Value 0: Internal assigned input 0 (forwarded to I/O controller) input value
[27] REF_H	R 0x0	Internal assigned reference value REF_H
[26] REF_R	R 0x0	Internal assigned reference value REF_R
[25] REF_L	R 0x0	Internal assigned reference value REF_L
[24] HALL_W	R 0x0	Internal assigned hall signal W
[23] HALL_V	R 0x0	Internal assigned hall signal V

BITS & NAME	TYPE & RESET	DESCRIPTION
[22] HALL_U	R 0x0	Internal assigned hall signal U
[21] ENC2_N	R 0x0	Internal assigned encoder signal N for ABN2 module
[20] ENC2_B	R 0x0	Internal assigned encoder signal B for ABN2 module
[19] ENC2_A	R 0x0	Internal assigned encoder signal A for ABN2 module
[18] ENC_N	R 0x0	Internal assigned encoder signal N for ABN module
[17] ENC_B	R 0x0	Internal assigned encoder signal B for ABN module
[16] ENC_A	R 0x0	Internal assigned encoder signal A for ABN module
[15] PIN_DRV_EN	R 0x0	Unprocessed input signal at pin DRV_EN
[13] PIN_UART_RXD_RAW	R 0x0	Unprocessed input signal at pin UART_RXD
[12] PIN_PWM_IN_RAW	R 0x0	Unprocessed input signal at pin PWM_IN
[11] PIN_CSN_RAW	R 0x0	Unprocessed Com input signal at pin CSN
[10] PIN_SCK_RAW	R 0x0	Unprocessed input signal at pin SCK
[9] PIN_SDO_RAW	R 0x0	Unprocessed input signal at pin SDO
[8] PIN_SDI_RAW	R 0x0	Unprocessed input signal at pin SDI
[7] PIN_REF_R_RAW	R 0x0	Unprocessed input signal at pin REF_R
[6] PIN_REF_L_RAW	R 0x0	Unprocessed input signal at pin REF_L
[5] PIN_HALL_W_RAW	R 0x0	Unprocessed input signal at pin HALL_W
[4] PIN_HALL_V_RAW	R 0x0	Unprocessed input signal at pin HALL_V
[3] PIN_HALL_U_RAW	R 0x0	Unprocessed input signal at pin HALL_U
[2] PIN_ENC_N_RAW	R 0x0	Unprocessed input signal at pin ENC_N
[1] PIN_ENC_B_RAW	R 0x0	Unprocessed input signal at pin ENC_B
[0] PIN_ENC_A_RAW	R 0x0	Unprocessed input signal at pin ENC_A

0x005: CHIP.OUTPUTS_RAW

Default: 0x00000000

Logic level for FAULTN, INT and PWM internal output signals, as well as error flags for digital differential ABN encoder signals.

BITS & NAME	TYPE & RESET	DESCRIPTION
[26] INVALID_DIFF_ENC_N	R 0x0	Digital Differential Encoder fail at ENC_N inputs
[25] INVALID_DIFF_ENC_B	R 0x0	Digital Differential Encoder fail at ENC_B inputs
[24] INVALID_DIFF_ENC_A	R 0x0	Digital Differential Encoder fail at ENC_A inputs
[7] FAULTN_OUT	R 0x0	Internal FAULTN output value
[6] INT_OUT	R 0x0	Internal INT output value
[5] PWM_W_H	R 0x0	Internal digital PWM output value at high side of OUT3
[4] PWM_W_L	R 0x0	Internal digital PWM output value at low side of OUT3
[3] PWM_V_H	R 0x0	Internal digital PWM output value at high side of OUT2
[2] PWM_V_L	R 0x0	Internal digital PWM output value at low side of OUT2
[1] PWM_U_H	R 0x0	Internal digital PWM output value at high side of OUT1
[0] PWM_U_L	R 0x0	Internal digital PWM output value at low side of OUT1

0x006: CHIP.IO_MATRIX

Default: 0x00000000

Alternate function assignment for the all the IO pins. Note that the pin logic might change from input to output for some of the alternate functions related to the I/O Controller.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:30] PIN_FAULTN	RW 0x0	Pin FAULTN function assignment
	0: FAULTN_OD	Main function: Digital Output FAULTN, open drain; controlled by CHIP.STATUS_FLAGS combined with CHIP.FAULTN_INT_MASK.
	1: INT_OD	Alternative function 1: Digital Output INT, open drain; controlled by CHIP.EVENTS combined with CHIP.FAULTN_INT_MASK.
	2: FAULTN_PP	Alternative function 2: Digital Output FAULTN, push pull; controlled by CHIP.STATUS_FLAGS combined with CHIP.FAULTN_INT_MAS
	3: INT_PP	Alternative function 3: Digital Output INT, push pull; controlled by CHIP.EVENTS combined with CHIP.FAULTN_INT_MASK.
[29:28] PIN_UART_TXD	RW 0x0	Pin UART_TXD function assignment
	0: UART_TXD	Main function: Digital Output TXD; UART communication interface.
	1: BRAKE	Alternate function 1: Digital BRAKE Output.
	2: DIRECT_OUT0	Alternate function 2: Digital Output 0 of I/O controller.
	3: DIRECT_OUT1	Alternate function 3: Digital Output 1 of I/O controller.
[27:26] PIN_UART_RXD	RW 0x0	Pin UART_RXD function assignment
	0: UART_RXD	Main function: Digital Input RXD; UART communication interface.
	1: DIR_IN	Alternate function 1: Digital Input DIR_IN for I/O controller.
	3: DIRECT_IN0	Alternate function 3: Digital Input 0 of I/O controller.

BITS & NAME	TYPE & RESET	DESCRIPTION
[25:24] PIN_PWM_IN	RW 0x0	Pin PWM_IN function assignment
	0: PWM_IN	Main function: Digital Input PWM_IN for I/O controller.
	1: AIN	Alternate function 1: Analog Input for AIN value.
	3: DIRECT_IN1	Alternate function 3: Digital Input 1 of I/O controller.
[23] PIN_SCK	RW 0x0	Pin SCK function assignment
	0: SCK	Main function: Digital Input SCK; SPI communication interface.
	1: DIRECT_IN1	Alternate function 1: Digital Input 1 of I/O controller.
[22:21] PIN_SDO	RW 0x0	Pin SDO function assignment
	0: SDO	Main function: Digital Output SDO; SPI communication interface.
	1: BRAKE	Alternate function 1: Digital BRAKE Output.
	3: DIRECT_OUT0	Alternate function 3: Digital Output 0 of I/O controller.
[20:19] PIN_SDI	RW 0x0	Pin SDI function assignment
	0: SDI	Main function: Digital Input SDI; SPI communication interface.
	1: DIR_IN	Alternate function 1: Digital Input DIR_IN for I/O controller.
	3: DIRECT_IN0	Alternate function 3: Digital Input 0 of I/O controller.
[18:17] PIN_REF_R	RW 0x0	Pin REF_R function assignment.
	0: REF_R	Main function: Digital Input REF_R.
	1: REF_H	Alternate function 1: Digital Input REF_H.
	2: BRAKE	Alternate function 2: Digital BRAKE Output.
	3: DIRECT_OUT1	Alternate function 3: Digital Output 1 of I/O controller.
[16:15] PIN_REF_L	RW 0x0	Pin REF_L function assignment
	0: REF_L	Main function: Digital Input REF_L.
	1: REF_H	Alternate function 1: Digital Input REF_H.
	3: DIRECT_IN0	Alternate function 3: Digital Input 0 of I/O controller.
[14:12] PIN_HALL_W	RW 0x0	Pin HALL_W function assignment
	0: HALL_W	Main function: Digital Hall Input W.
	1: AINP_W	Alternate function 1: Positive Analog Input for Analog Hall W or SinCos Y.
	2: ENC2_N	Alternate function 2: Digital 2nd Encoder Input N.
	3: DIRECT_OUT2	Alternate function 3: Digital Output 2 of I/O controller.
	4: REF_H	Alternate function 4: Digital REF_H input.
	6: DIG_DIFF_ENC_N_MONITOR	Alternate function 6: Negative Differential Digital Input for ENC_N; Monitoring only.
	7: DIG_DIFF_ENC_N_N	Alternate function 7: Negative Differential Digital Input for ENC_N.
[11:9] PIN_HALL_V	RW 0x0	Pin HALL_V function assignment
	0: HALL_V	Main function: Digital Hall Input V.
	1: AINP_V	Alternate function 1: Positive Analog Input for Analog Hall V or SinCos N.
	2: ENC2_B	Alternate function 2: Digital 2nd Encoder Input B.
	3: SCL	Alternate function 3: Digital Output of I/O controller; SCL of I2C communication interface; push pull.
	4: BRAKE	Alternate function 4: Digital BRAKE Output.
	5: SCL_OD	Alternate function 5: Digital IO of I/O controller; SCL of I2C communication interface; open drain.
	6: DIG_DIFF_ENC_B_MONITOR	Alternate function 6: Negative Differential Digital Input for ENC_B; Monitoring only.
	7: DIG_DIFF_ENC_B_N	Alternate function 7: Negative Differential Digital Input for ENC_B.

BITS & NAME	TYPE & RESET	DESCRIPTION
[8:6] PIN_HALL_U	RW 0x0	Pin HALL_U function assignment
	0: HALL_U 1: AINP_U 2: ENC2_A 3: SDA 6: DIG_DIFF_ENC_A_MONITOR 7: DIG_DIFF_ENC_A_N	Main function: Digital Hall Input U. Alternate function 1: Positive Analog Input for Analog Hall U or SinCos X. Alternate function 2: Digital 2nd Encoder Input A. Alternate function 3: Digital IO of I/O controller; SDA of I2C communication interface. Alternate function 6: Negative Differential Digital Input for ENC_A; Monitoring only. Alternate function 7: Negative Differential Digital Input for ENC_A.
[5:4] PIN_ENC_N	RW 0x0	Pin ENC_N function assignment
	0: ENC_N 1: AINN_W 2: HALL_W 3: DIRECT_IN1_OUT1	Main function: Digital Encoder Input N. Alternate function 1: Negative Analog Input for Analog Hall W or SinCos Y. Alternate function 2: Digital Hall Input W. Alternate function 3: Digital Input 1 or Output 1 of I/O controller. I/O assignment is defined by the I/O Controller assignment.
[3:2] PIN_ENC_B	RW 0x0	Pin ENC_B function assignment
	0: ENC_B 1: AINN_V 2: HALL_V 3: DIRECT_IN0	Main function: Digital Encoder Input B. Alternate function 1: Negative Analog Input for Analog Hall V or SinCos N. Alternate function 2: Digital Hall Input V. Alternate function 3: Digital Input 0 of I/O controller.
[1:0] PIN_ENC_A	RW 0x0	Pin ENC_A function assignment
	0: ENC_A 1: AINN_U 2: HALL_U 3: DIRECT_OUT0	Main function: Digital Encoder Input A. Alternate function 1: Negative Analog Input for Analog Hall U or SinCos X. Alternate function 2: Digital Hall Input U. Alternate function 3: Digital Output 0 of I/O controller.

0x007: CHIP.IO_PU_PD

Default: 0x075800FF

Enable or disable pull-up and pull-down resistors. When writing this register, default value of not used bits must be kept.

BITS & NAME	TYPE & RESET	DESCRIPTION
[27] PIN_UART_RXD_PD	RW 0x0	Enables Pull Down at pin UART_RXD
[26] PIN_PWM_IN_PD	RW 0x1	Enables Pull Down at pin PWM_IN
[25] PIN_SCK_PD	RW 0x1	Enables Pull Down at pin SCK
[24] PIN_SDI_PD	RW 0x1	Enables Pull Down at pin SDI
[22] PIN_FAULTN_PU	RW 0x1	Enables Pull Up at pin FAULTN, only active during open drain assignment.
[20] PIN_SDO_PU	RW 0x1	Enables Pull Up at pin SDO; only active if CSN='1' (SPI communication) or if output enable of I/O controller is not active for DIRECT_OUT0
[19] PIN_UART_RXD_PU	RW 0x1	Enables Pull Up at pin UART_RXD
[18] PIN_PWM_IN_PU	RW 0x0	Enables Pull Up at pin PWM_IN
[17] PIN_SCK_PU	RW 0x0	Enables Pull Up at pin SCK

BITS & NAME	TYPE & RESET	DESCRIPTION
[16] PIN_SDI_PU	RW 0x0	Enables Pull Up at pin SDI
[15] PIN_REF_R_PD	RW 0x0	Enables Pull Down at pin REF_R
[14] PIN_REF_L_PD	RW 0x0	Enables Pull Down at pin REF_L
[13] PIN_HALL_W_PD	RW 0x0	Enables Pull Down at pin HALL_W
[12] PIN_HALL_V_PD	RW 0x0	Enables Pull Down at pin HALL_V
[11] PIN_HALL_U_PD	RW 0x0	Enables Pull Down at pin HALL_U
[10] PIN_ENC_N_PD	RW 0x0	Enables Pull Down at pin ENC_N
[9] PIN_ENC_B_PD	RW 0x0	Enables Pull Down at pin ENC_B
[8] PIN_ENC_A_PD	RW 0x0	Enables Pull Down at pin ENC_A
[7] PIN_REF_R_PU	RW 0x1	Enables Pull Up at pin REF_R
[6] PIN_REF_L_PU	RW 0x1	Enables Pull Up at pin REF_L
[5] PIN_HALL_W_PU	RW 0x1	Enables Pull Up at pin HALL_W
[4] PIN_HALL_V_PU	RW 0x1	Enables Pull Up at pin HALL_V
[3] PIN_HALL_U_PU	RW 0x1	Enables Pull Up at pin HALL_U
[2] PIN_ENC_N_PU	RW 0x1	Enables Pull Up at pin ENC_N
[1] PIN_ENC_B_PU	RW 0x1	Enables Pull Up at pin ENC_B
[0] PIN_ENC_A_PU	RW 0x1	Enables Pull Up at pin ENC_A

0x008: CHIP.IO_CONFIG

Default: 0x70055000

Enable and configure additional input processing capability for some pins. For instance selecting the voltage divider for the analog inputs, inverting digital input levels, or enabling digital filters to remove spurious signals.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:28] DIFF_DELAY	RW, unsigned 0x7	Blank time [clockCycles] for digital differential inputs: During this time span no faults is created after a level change at the inputs during DIG_DIFF_ monitoring
[25] USE_PIN_REF_R_AS_REF_L	RW 0x0	Pin REF_R input signal is also assigned to internal assigned REF_L signal. Use to make both REF_R and REF_L signals available with just one physical pin REF_R, e.g. to stop both direction with one pin
[24] USE_PIN_REF_L_AS_REF_R	RW 0x0	Pin REF_L input signal is also assigned to internal assigned REF_R signal. Use to make both REF_R and REF_L signals available with just one physical pin REF_L, e.g. to stop both direction with one pin

BITS & NAME	TYPE & RESET	DESCRIPTION
[23:22] ANA_DIV_VCCIOF	RW 0x0	Divider for analog inputs (ENC_x, HALL_x pins) in VCC_IOF domain
	0: DIV_BY_4P5 1: DIV_BY_3P0 2: DIV_BY_2P2 3: DIV_BY_1P0	AnalogInput divided by 4.5 AnalogInput divided by 3.0 AnalogInput divided by 2.2 AnalogInput divided by 1.0
[21:20] ANA_DIV_VCCIO	RW 0x0	Divider for analog inputs (PWM_IN, TEMP pins) in VCC_IO domain
	0: DIV_BY_4P5 1: DIV_BY_3P0 2: DIV_BY_2P2 3: DIV_BY_1P0	AnalogInput divided by 4.5 AnalogInput divided by 3.0 AnalogInput divided by 2.2 AnalogInput divided by 1.0
[19:18] PWM_IN_FLT	RW 0x1	Digital Filter Settings assigned to PWM_IN input
	0: NO_FLT 1: FLT_1M 2: FLT_500K 3: FLT_100K	No filter Digital filter of 1 MHz Digital filter of 500 kHz Digital filter of 100 kHz
[17:16] REF_FLT	RW 0x1	Digital Filter Settings for Reference input pins
	0: NO_FLT 1: FLT_1M 2: FLT_500K 3: FLT_100K	No filter Digital filter of 1 MHz Digital filter of 500 kHz Digital filter of 100 kHz
[15:14] HALL_FLT	RW 0x1	Digital Filter Settings for Hall Inputs
	0: NO_FLT 1: FLT_1M 2: FLT_500K 3: FLT_100K	No filter Digital filter of 1 MHz Digital filter of 500 kHz Digital filter of 100 kHz
[13:12] ENC_FLT	RW 0x1	Digital Filter for Encoder Inputs
	0: NO_FLT 1: FLT_1M 2: FLT_500K 3: FLT_100K	No filter Digital filter of 1 MHz Digital filter of 500 kHz Digital filter of 100 kHz
[11] INV_PIN_RXD	RW 0x0	Inverts incoming PIN_RXD_RAW input signal for further processing.
[10] INV_PIN_PWM_IN	RW 0x0	Inverts incoming PIN_PWM_IN_RAW input signal for further processing.
[9] INV_PIN_SCK	RW 0x0	Inverts incoming PIN_SCK_RAW input signal for further processing. Not applicable for the SPI interface signal.
[8] INV_PIN_SDI	RW 0x0	Inverts incoming PIN_SDI_RAW input signal for further processing. Not applicable for the SPI interface signal.
[7] INV_PIN_REF_R	RW 0x0	Inverts incoming PIN_REF_R_RAW input signal for further processing.
[6] INV_PIN_REF_L	RW 0x0	Inverts incoming PIN_REF_L_RAW input signal for further processing.
[5] INV_PIN_HALL_W	RW 0x0	Inverts incoming PIN_HALL_W_RAW input signal for further processing.
[4] INV_PIN_HALL_V	RW 0x0	Inverts incoming PIN_HALL_V_RAW input signal for further processing.
[3] INV_PIN_HALL_U	RW 0x0	Inverts incoming PIN_HALL_U_RAW input signal for further processing.

BITS & NAME	TYPE & RESET	DESCRIPTION
[2] INV_PIN_ENC_N	RW 0x0	Inverts incoming PIN_ENC_N_RAW input signal for further processing.
[1] INV_PIN_ENC_B	RW 0x0	Inverts incoming PIN_ENC_B_RAW input signal for further processing.
[0] INV_PIN_ENC_A	RW 0x0	Inverts incoming PIN_ENC_A_RAW input signal for further processing.

0x009: CHIP.STATUS_FLAGS

Default: 0x00000000

Selection of individual or merged error condition and status bits. This register is the base for the CHIP.EVENTS register, which latches the activated state of the corresponding status bits.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31] GDRV_ON_STATE	R 0x0	GDRV_ON Status Bit: Gate driver is active. In case certain error conditions are met, this state will be left to SYS_READY.
[30] SYS_READY_STATE	R 0x0	SYS_READY Status Bit: Basic ready state of the chip is active. This state is reached after power-up before gate driver stage activation (therefore, set DRV_EN pin ='1' and DRV_EN_BIT='1') and in case GDRV_ON state is left automatically due to certain error conditions.
[29] OVERTEMP_WARN_STATUS	R 0x0	In- or externally acquired temperature exceeds register based Overtemperature limit.
[28] OV_VM_LIMIT_FAIL_STATUS	R 0x0	Acquired VM voltage exceeds register based overvoltage limit (UPPER_SWITCH_LIMIT).
[27] VM_TEMP_CLIPPED_STATUS	R 0x0	VM and/or external/internal Temperature value are clipped during last acquisition cycle.
[26] AIN_CLIPPED_STATUS	R 0x0	Analog Input Current Value Clipped Status Bit
[25] I_CLIPPED_STATUS	R 0x0	Any Phase Current Value is Clipped during last current acquisition cycle.
[24] CURRENT_OVERLOAD_STATUS	R 0x0	Actual Current surpasses register based Current Overlaod Limit
[23] HALL_FAIL_STATUS	R 0x0	Hall Fail Status Bit: All 3 Hall signals (U, V, W) are equal.
[22] ABN_FAIL_STATUS	R 0x0	ABN Fail Status Bit, combined for ABN and ABN2. Refer to particular ABN.EVENTS register.
[21] RAMP_REF_H_STATUS	R 0x0	Active REF_H condition of ramper module
[20] RAMP_REF_LR_STATUS	R 0x0	Active REF_L or REF_R condition of ramper module
[19] RAMP_V_ZERO_STATUS	R 0x0	Ramper V_ZERO Status Bit: Velocity value equals 0.
[18] RAMP_V_REACHED_STATUS	R 0x0	Ramper Velocity Reached Status Bit: Velocity value matches actual postion value.
[17] RAMP_TARGET_REACHED_STATUS	R 0x0	Ramper Target Reached Status Bit: Target value matches actual postion value.
[16] RAMP_STALL_FAIL_STATUS	R 0x0	Active Ramper Stall condition: Position and/or velocity value exceeds assigned limits.
[15] IO_CONTROLLER_STATUS	R 0x0	I/O controller is disabled, or I/O controller generated status bit is active
[14] DIFF_ENC_FAIL_STATUS	R 0x0	Differential Encoder Fail Status Bit: Digital (and differentially assigned) encoder inputs are not differential

BITS & NAME	TYPE & RESET	DESCRIPTION
[13] OT_FAIL_STATUS	R 0x0	OverTemperature Fail Status Bit: Internal measured temperature in any phase exceeds limit. Driver output stage will be disabled as long as this over-temperature failure is active
[12] SHRT_FAIL_STATUS	R 0x0	Short Fail Status Bit: Short to GND/VS is monitored on any phase output.
[11] VEL_FAIL_STATUS	R 0x0	Output frequency exceeds 600Hz limit.
[10] TEMP_LIMIT_FAIL_STATUS	R 0x0	Die Thermal Shutdown Fail Status Bit: Internally monitored temperature exceeds limit.
[9] ADC_FAIL_STATUS	R 0x0	ADC Fail Status Bit. Refer to ADC.STATUS register
[8] EXT_RES_FAIL_STATUS	R 0x0	External Resistor Fail Status Bit: External resistor value is not in expected range
[7] CLK_PLL_FAIL_STATUS	R 0x0	Clk/PLL Fail Status Bit
[6] POWER_FAIL_STATUS	R 0x0	Voltage Fail Status Bit: VM and/or ChargePump not properly started
[5] UV_VM_STATUS	R 0x0	Undervoltage VM Fail Status Bit: VM voltage level is too low.
[4] CP_FAIL_STATUS	R 0x0	ChargePump Fail Status Bit
[3] VCC_IOF_FAIL_STATUS	R 0x0	Voltage level at VCCIOF is too low.
[2] UART_FAIL_STATUS	R 0x0	UART Fail Status Bit: Any UART communcation error is active
[1] SPI_FAIL_STATUS	R 0x0	SPI Fail Status: Timeout or Interrupt Failure is active
[0] PWRUP_FAIL_STATUS	R 0x0	Power-up process fails

0x00A: CHIP.EVENTS

Default: 0x00000001

Latched state of the corresponding activated status bits in CHIP.STATUS_FLAGS register. Individual events remain set until they are actively cleared via register access. Clearing a set bit is achieved by writing a '1' to the particular bit.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31] GDRV_ON_EVENT	RW, W1C 0x0	GDRV_ON Event, based on particular status bit
[30] SYS_READY_EVENT	RW, W1C 0x0	SYS_READY Event, based on particular status bit
[29] OVERTEMP_WARN_EVENT	RW, W1C 0x0	Internal or External Overtemperature Limit Fail Event, based on particular status bit.
[28] OV_VM_LIMIT_FAIL_EVENT	RW, W1C 0x0	Overvoltage (VM) Limit Fail Event, based on particular status bit.
[27] VM_TEMP_CLIPPED_EVENT	RW, W1C 0x0	VM or Temperature Value Clipped Event, based on particular status bit
[26] AIN_CLIPPED_EVENT	RW, W1C 0x0	Analog Input Current Value Clipped Event, based on particular status bit.
[25] I_CLIPPED_EVENT	RW, W1C 0x0	Phase Current Value Clipped Event, based on particular status bit.
[24] CURRENT_OVERLOAD_EVENT	RW, W1C 0x0	Register based Current Overlaod Limit Exceeded
[23] HALL_FAIL_EVENT	RW, W1C 0x0	Hall Fail Event: Clear in HALL.DIG_EVENTS register before clearing this field.

BITS & NAME	TYPE & RESET	DESCRIPTION
[22] ABN_FAIL_EVENT	RW, W1C 0x0	ABN Fail Event, combined for ABN and ABN2, based on particular status bit. Clear in particular ABN.EVENTS register before clearing this field.
[21] RAMP_REF_H_EVENT	RW, W1C 0x0	Ramper Home Event, based on particular status bit
[20] RAMP_REF_LR_EVENT	RW, W1C 0x0	Ramper Reference Event, based on particular status bit
[19] RAMP_V_ZERO_EVENT	RW, W1C 0x0	Ramper V_ZERO Event, based on particular status bit
[18] RAMP_V_REACHED_EVENT	RW, W1C 0x0	Ramper Velocity Reached Event, based on particular status bit.
[17] RAMP_TARGET_REACHED_EVENT	RW, W1C 0x0	Ramper Target Reached Event, based on particular status bit
[16] RAMP_STALL_FAIL_EVENT	RW, W1C 0x0	Ramper Stall Event, based on particular status bit
[15] IO_CONTROLLER_EVENT	RW, W1C 0x0	I/O controller Event, based on particular status bit
[14] DIFF_ENC_FAIL_EVENT	RW, W1C 0x0	Differential Encoder Fail Event, based on particular status bit
[13] OT_FAIL_EVENT	RW, W1C 0x0	OverTemperature Fail Event, based on particular status bit.
[12] SHRT_FAIL_EVENT	RW, W1C 0x0	Short Fail Event, based on particular status bit
[11] VEL_FAIL_EVENT	RW, W1C 0x0	Velocity Fail Event, based on particular status bit.
[10] TEMP_LIMIT_FAIL_EVENT	RW, W1C 0x0	Die Thermal Shutdown Fail Event
[9] ADC_FAIL_EVENT	RW, W1C 0x0	ADC Fail Event, based on particular status bit.
[8] EXT_RES_FAIL_EVENT	RW, W1C 0x0	External Resistor Fail Event, based on particular status bit
[7] CLK_PLL_FAIL_EVENT	RW, W1C 0x0	Clk/PLL Fail Event, based on particular status bit.
[6] POWER_FAIL_EVENT	RW, W1C 0x0	Voltage Fail Event, based on particular status bit
[5] UV_VM_EVENT	RW, W1C 0x0	Undervoltage VM Fail Event, based on particular status bit
[4] CP_FAIL_EVENT	RW, W1C 0x0	ChargePump Fail Event, based on particular status bit
[3] VCCIOF_FAIL_EVENT	RW, W1C 0x0	VCCIOF Fail Event, based on particular status bit
[2] UART_FAIL_EVENT	RW, W1C 0x0	UART Fail Event
[1] SPI_FAIL_EVENT	RW, W1C 0x0	SPI Fail Event, based on particular status bit
[0] RST_EVENT	RW, W1C 0x1	Reset Event: Reset has been triggered

0x00B: CHIP.FAULTN_INT_MASK

Default: 0x00000000

Mask to select which bits of CHIP.STATUS_FLAGS or CHIP.EVENTS registers will drive the FAULTN pin. The value in the CHIP.IO_MATRIX.PIN_FAULTN field determines whether the status flags bits (FAULTN_x options) or the events bits (INT_x options) are used to drive the output.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:0] FAULTN_INT_MASK	RW, unsigned 0x00000000	Mask for selection of bits to drive the FAULTN output pin. Bits set to '1' in the mask will select the corresponding bit from CHIP.STATUS_FLAGS or CHIP.EVENTS registers. Which register is chosen is determined by the value in the CHIP.IO_MATRIX.PIN_FAULTN field. If more than one bit is set in the mask, all the corresponding bits are OR wired.

0x00C: CHIP.SPI_STATUS_MASK

Default: 0x00000000

Mask to select which bits of CHIP.EVENTS registers will be included with each SPI command response. At most 6 bits can be selected.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:0] SPI_STATUS_MASK	RW, unsigned 0x00000000	Mask for selection of bits to include in each SPI command response. Bits set to '1' in the mask will select the corresponding bit from the CHIP.EVENTS register. At most 6 bits can be selected. If more bits are selected the lower bit numbers are prior.

0x040: CLK_CTRL.CONFIG

Default: 0x00000E3C

System clocks configuration.

BITS & NAME	TYPE & RESET	DESCRIPTION
[12:8] CLOCK_DIVIDER	RW, unsigned 0x0E	Clock divider for the source clock of the PLL. The input frequency for the PLL must be 1 MHz. The default value of 14 is configured for the internal oscillator (15MHz). pll_input_freq = source_clk_freq / (CLOCK_DIVIDER + 1)
[5] CLK_FSM_EN	RW 0x1	Enable the clock and PLL controller FSM. If the FSM is stuck at a certain point due to an incorrect configuration, the user must manually clear this bit to reset the FSM.
[4] PWM_CLK_EN	RW 0x1	Enable 120MHz PWM clock.
[3] ADC_CLK_EN	RW 0x1	Enable 60MHz ADC clock.
[2] PLL_EN	RW 0x1	Select the internal oscillator clock (15MHz) or the PLL output clock (120MHz) as system master clock.
	0: INT_OSC	Use the internal oscillator clock (15MHz) as source for the system clock, PWM clock and ADC clock.
	1: PLL	Use the PLL output clock (120MHz) as source for the system clock, PWM clock and ADC clock. System clock frequency = 60MHz ADC clock frequency = 60MHz PWM clock frequency = 120MHz
[1] PLL_SRC	RW 0x0	Select the internal oscillator clock (15MHz) or the external clock as input source for the PLL.
	0: INT_CLK	Intern oscillator clock (15MHz) as PLL input clock.
	1: EXT_CLK	External clock as PLL input clock.
[0] COMMIT	RW, W1S 0x0	Commit the clock and PLL configuration by writing 1 to this bit. After the clock configuration is finished this bit is cleared automatically.

0x041: CLK_CTRL.STATUS

Default: 0x00000000

Status bits and status fields related to the some of the system clocks.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:24] CLK_FSM_FLG	R, unsigned 0x00	This bitfield holds clk FSM current status.
[23:16] CLK_FSM_ERR_RPT	R 0x00	In case of CLK_FSM_ERR this fields will report the error condition.
	0: NO_ERR_RPT	No error happened.
	1: NOT_USED	
	2: DET_ERR_RPT	1MHz for the PLL input clock was no longer detected while the external clock was used as the source.
	4: STK_ERR_RPT	PLL output clock is stuck.
	8: PLL_ERR_RPT	PLL analog error occurred.
	255: UNK_ERR_RPT	Unknown error.
[6] PLL_READY	R 0x0	This bit is set to 1 after the PLL configuration in finished. The 120MHz and 60MHz output clocks are generated and ready to use. If this bit is 0 the output clock of the PLL is not used. The chip is using the 15MHz internal oscillator clock.
[5] PLL_LOCK_ON	R 0x0	PLL locked during start-up procedure.
[4] PLL_ERR	R 0x0	Analog PLL error occurred.
[3] CLK_STUCK	R 0x0	PLL output clock is stuck. Watchdog is monitoring the PLL output clock with the internal oscillator clock.
[2] CLK_FSM_ERR	R 0x0	Error detected by clock controller FSM. Error code can be read out at CLK_FSM_ERR_RPT. If this bit is set, the PLL is turned off and the clock controller awaits a new COMMIT.
[1] CLK_1M0_TIMEOUT	R 0x0	Timeout expired while waiting for the 1MHz clock to enable the PLL.
[0] CLK_1M0_OK	R 0x0	1MHz for PLL input clock is detected.

0x080: ADC.CONFIG

Default: 0x00000400

ADC configuration. When writing this register, the default value of the not used bits must be kept.

BITS & NAME	TYPE & RESET	DESCRIPTION
[11:8] CSA_AZ_FLT_EXP	RW, unsigned 0x4	Filter length exponent of digital filter (moving average style) for all raw AZ (Auto-zero) values.
[3] NRST_ADC_1	RW 0x0	Low active software reset of ADC1. Reset I/O controller after resetting any ADC.
	0: RESET	ADC1 is reset.
	1: ENABLE	ADC1 is enabled.
[2] NRST_ADC_0	RW 0x0	Low active software reset of ADC0. Reset I/O controller after resetting any ADC.
	0: RESET	ADC0 is reset.
	1: ENABLE	ADC0 is enabled.

0x081: ADC.ADC_VERSION

Default: 0x00000002

ADC version no.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:0] VERSION_NUMBER	R, unsigned 0x00000002	ADC version no.

0x082: ADC.I2_I1_RAW

Default: 0x00000000

Raw phase current I2 and I1 values.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:16] I2	RW, signed 0x0000	Raw Phase current I2.
[15:0] I1	RW, signed 0x0000	Raw Phase current I1.

0x083: ADC.VM_I3_RAW

Default: 0x00000000

Raw supply voltage and phase current I3 values.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:16] VM	RW, signed 0x0000	Raw Analog Voltage Monitoring value.
[15:0] I3	RW, signed 0x0000	Raw Phase current I3.

0x084: ADC.TEMP_RAW

Default: 0x00000000

Raw internal die temperature value and raw voltage value at TEMP analog input pin.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:16] TEMP_INT	RW, signed 0x0000	Raw Internal Die Temperature.
[15:0] TEMP_EXT	RW, signed 0x0000	Raw voltage value for external temperature measurement at TEMP analog input pin.

0x085: ADC.AIN_V_AIN_U_RAW

Default: 0x00000000

Raw voltage value for position feedback analog sensors.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:16] AIN_V	RW, signed 0x0000	Raw analog input for pin HALL_V (with alternate function) (optionally the raw analog differential input between pins HALL_V and ENC_B)
[15:0] AIN_U	RW, signed 0x0000	Raw analog input for pin HALL_U (with alternate function) (optionally the raw analog differential input between pins HALL_U and ENC_A)

0x086: ADC.AIN_AIN_W_RAW

Default: 0x00000000

Raw voltage value for position feedback analog sensors.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:16] AIN	RW, signed 0x0000	Raw analog input AIN
[15:0] AIN_W	RW, signed 0x0000	Raw analog input for pin HALL_W (with alternate function) (optionally the raw analog differential input between pins HALL_W and ENC_N)

0x08A: ADC.STATUS

Default: 0x00000000

ADC status bits

BITS & NAME	TYPE & RESET	DESCRIPTION
[31] I123_FAIL	R 0x0	Sum of all raw phase currents exceeds the absolute limit set in ADC.I123.I123_LIMIT register field.
[30] NO_ACK_ADC	RW, W1C 0x0	ADC acknowledged not correctly.
[27] USE_ADC_EXT_VALID_EXT	RW 0x0	ADC_EXT_VALID_EXT will be used for feedback engine, not internally generated ADC_EXT_VALID.
[26] ADC_EXT_VALID_EXT	RW 0x0	Externally generated ADC_EXT_VALID signal.
[25] USE_ADC_I_VALID_EXT	RW 0x0	ADC_I_VALID_EXT will be used for FOC engine, not internally generated ADC_I_VALID.
[24] ADC_I_VALID_EXT	RW 0x0	Externally generated ADC_I_VALID signal.
[23] ADC_EXT_VALID	R 0x0	Indication that external ADC measurement is completed.
[22] ADC_I_VALID	R 0x0	Indication that CSA measurements have been completed.
[21] AIN_DONE	RW, W1C 0x0	AIN channel measurement is done.
[20] AIN_W_DONE	RW, W1C 0x0	Analog measurement at pin HALL_W is done.
[19] AIN_V_DONE	RW, W1C 0x0	Analog measurement at pin HALL_V is done.
[18] AIN_U_DONE	RW, W1C 0x0	Analog measurement at pin HALL_U is done.
[17] I3_DONE	RW, W1C 0x0	ADC measurement for I3 done.
[16] I2_DONE	RW, W1C 0x0	ADC measurement for I2 done.
[15] I1_DONE	RW, W1C 0x0	ADC measurement for I1 done.
[14] TEMP_EXT_DONE	RW, W1C 0x0	Raw analog value measurement of external temperature input pin TEMP is done.
[13] TEMP_EXT_CLIPPED	RW, W1C 0x0	Raw analog value for external temperature input pin TEMP is clipped.
[12] TEMP_INT_DONE	RW, W1C 0x0	Raw Internal Die Temperature measurement is done.
[11] TEMP_INT_CLIPPED	RW, W1C 0x0	Raw Internal Die Temperature value is clipped.
[10] VM_DONE	RW, W1C 0x0	Raw Analog Voltage Monitoring measurement is done.

BITS & NAME	TYPE & RESET	DESCRIPTION
[9] VM_CLIPPED	RW, W1C 0x0	Raw Analog Voltage Monitoring value is clipped.
[8] AIN_CLIPPED	RW, W1C 0x0	AIN channel value is clipped.
[7] AIN_W_CLIPPED	RW, W1C 0x0	Analog measurement at pin HALL_W clipped.
[6] AIN_V_CLIPPED	RW, W1C 0x0	Analog measurement at pin HALL_V clipped.
[5] AIN_U_CLIPPED	RW, W1C 0x0	Analog measurement at pin HALL_U clipped.
[4] I3_CLIPPED	RW, W1C 0x0	ADC value for I3 is clipped.
[3] I2_CLIPPED	RW, W1C 0x0	ADC value for I2 is clipped.
[2] I1_CLIPPED	RW, W1C 0x0	ADC value for I1 is clipped.
[1] ADC_1_READY	R 0x0	ADC1 ready status bit
	0: ONGOING 1: READY	ADC_1 calibration ongoing. ADC_1 is calibrated and ready for normal operation.
[0] ADC_0_READY	R 0x0	ADC0 ready status bit
	0: ONGOING 1: READY	ADC_0 calibration ongoing. ADC_0 is calibrated and ready for normal operation.

0x08C: ADC.I123

Default: 0xFFFF0000

Sum of all current values

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:16] I123_LIMIT	RW, unsigned 0xFFFF	Limit for the sum of all raw phase currents' absolute value before triggering the ADC.STATUS.I123_FAIL register bit.
[15:0] I123	R, signed 0x0000	All Raw Phase currents added.

0x0C0: MCC_ADC.I_GEN_CONFIG

Default: 0x00000000

General Configuration of the current ADCs. When writing this register, the default value of the not used bits must be kept.

BITS & NAME	TYPE & RESET	DESCRIPTION
[13:12] STATUS	R, unsigned 0x0	Autokirchhoff calculation status: 0= no current calculation is replaced. If this value is greater than 0, the particular phase no., which current measurement is replaced by Autkirchhoff feature, is forwarded (1= IU, 2= IV, 3= IW).
[0] MEAS_MODE	RW 0x0	Current measurement mode
	0: AUTOMATIC_SWITCH	Auto Kirchhoff feature is enabled: Current measurement with highest PWM duty cycle may be skipped (dependent on AUTO_KIRCHHOFF_LIM) and calculated with the help of the two other phase current values.
	1: NO_SWITCH	Auto Kirchhoff feature is disabled: All current measurement values are directly transferred to the corresponding FOC input values.

0x0C1: MCC_ADC.IW_IU

Default: 0x00000000

Scaled and offset compensated ADC measurements for U and W phase currents, as sent to the FOC module.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:16] IW	R, signed 0x0000	Scaled and offset compensated ADC measurement for the W phase current.
[15:0] IU	R, signed 0x0000	Scaled and offset compensated ADC measurement for the U phase current.

0x0C2: MCC_ADC.IV

Default: 0x00000000

Scaled and offset compensated ADC measurement for the V phase current, as sent to the FOC module.

BITS & NAME	TYPE & RESET	DESCRIPTION
[15:0] IV	R, signed 0x0000	Scaled and offset compensated ADC measurement for the V phase current.

0x0C3: MCC_ADC.CSA_GAIN

Default: 0x00000000

CSA gain configurations.

BITS & NAME	TYPE & RESET	DESCRIPTION
[15:8] DYN_GAIN_HYST	RW, unsigned 0x00	Hysteresis of Dynamic Gain Selection
[6] DYN_GAIN_4X_EN	RW 0x0	Enables dynamic gain factor 4x.
[5] DYN_GAIN_3X_EN	RW 0x0	Enables dynamic gain factor 3x.
[4] DYN_GAIN_2X_EN	RW 0x0	Enables dynamic gain factor 2x.
[3:2] DYN_GAIN_ACT	R, unsigned 0x0	Actual Gain Factor
[1:0] CSA_GAIN	RW 0x0	CSA gain value. This value is active when dynamic gain is not enabled.
	0: X1 1: X2 2: X3 3: X4	CSA Gain=1 CSA Gain=2 CSA Gain=3 CSA Gain=4

0x0C4: MCC_ADC.EVENTS

Default: 0x00000000

Status bits and events for some ADC measurements.

BITS & NAME	TYPE & RESET	DESCRIPTION
[6] CURRENT_OVERLOAD_STATUS	R 0x0	Status of current measurement comparison with the soft overload limit value set in MCC_ADC.CURRENT_OVERLOAD register.
[5] TEMP_INT_LIMIT_EXCEEDED	R 0x0	Status of internal temperature comparison with the soft limit value set in MCC_ADC.TEMP_LIMITS.TEMP_INT_LIMIT register field.
[4] TEMP_EXT_LIMIT_EXCEEDED	R 0x0	Status of external temperature comparison with the soft limit value set in MCC_ADC.TEMP_LIMITS.TEMP_EXT_LIMIT register field.
[1] MEAS_DONE	RW, W1C 0x0	Current Measurement Done flag (MCC_ADC processing). Write a '1' to clear the bit.
[0] ADC_CLIPPED	RW, W1C 0x0	This bit is set if any ADC measurement was clipped. Write a '1' to clear the bit.

0x0C5: MCC_ADC.DYN_GAIN_LIMITS_4X_3X

Default: 0x40008000

Current limits for automatic gain switching

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:16] DYN_GAIN_LIMIT_4X	RW, unsigned 0x4000	Current limit value after which a 4x gain is automatically switched to the next lower active dynamic gain.
[15:0] DYN_GAIN_LIMIT_3X	RW, unsigned 0x8000	Current limit value after which a 3x gain is automatically switched to the next lower active dynamic gain.

0x0C6: MCC_ADC.DYN_GAIN_LIMIT_2X

Default: 0x0000C000

Current limits for automatic gain switching

BITS & NAME	TYPE & RESET	DESCRIPTION
[15:0] DYN_GAIN_LIMIT_2X	RW, unsigned 0xC000	Current limit value after which a 2x gain is automatically switched to the next lower active dynamic gain.

0x0C7: MCC_ADC.TEMP_LIMITS

Default: 0xFFFFFFFF

Soft temperature ADC measurement limits. Values can be set as a threshold to trigger particular status bits.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:16] TEMP_INT_LIMIT	RW, unsigned 0xFFFF	Soft limit for the internal die temperature ADC measurement, which, if exceeded, triggers the MCC_ADC.EVENTS.TEMP_INT_LIMIT_EXCEEDED register bit.
[15:0] TEMP_EXT_LIMIT	RW, unsigned 0xFFFF	Soft limit for the TEMP pin ADC measurement, which, if exceeded, triggers the MCC_ADC.EVENTS.TEMP_EXT_LIMIT_EXCEEDED register bit.

0x0C8: MCC_ADC.CURRENT_OVERLOAD

Default: 0x0000FFFF

Soft current ADC measurement limits. A value can be set as a threshold to trigger a particular status bit.

BITS & NAME	TYPE & RESET	DESCRIPTION
[15:0] CURRENT_OVERLOAD	RW, unsigned 0xFFFF	Soft limit for ADC measurement for any of the phase currents(IU, IV, IW), which triggers the MCC_ADC.EVENTS.CURRENT_OVERLOAD_STATUS register bit if the limit is exceeded.

0x100: MCC_CONFIG.MOTOR_MOTION

Default: 0x00000000

General configurations for the motor type and the active motion mode.

BITS & NAME	TYPE & RESET	DESCRIPTION
[19] VELOCITY_FF_EN	RW 0x0	Enable feedforward of RAMPER.RAMPER_VELOCITY value to the velocity PID controller. Enable only while in MOTION_MODE=POSITION, must be disabled otherwise.
[18] TORQUE_FF_VISC_FRIC_EN	RW 0x0	Consider the torque feedforward viscous friction component (torque_ff_visc_fric) when calculating the value of the MCC_CONFIG.TORQUE_FEEDFORWARD register.
[17] TORQUE_FF_COULOMB_FRICTION_EN	RW 0x0	Consider the torque feedforward Coulomb friction component (torque_ff_coulomb_fric) when calculating the value of the MCC_CONFIG.TORQUE_FEEDFORWARD register.

BITS & NAME	TYPE & RESET	DESCRIPTION
[16] TORQUE_FF_ACC_EN	RW 0x0	Consider the torque feedforward acceleration component (torque_ff_acc) when calculating the value of the MCC_CONFIG.TORQUE_FEEDFORWARD register.
[15] RAMP_USE_PHI_E	RW 0x0	Enable this bit to use the value in RAMPER.PHI_E.PHI_E register field as phi_e value for the FOC controller, bypassing the selection in FEEDBACK.OUTPUT_CONF.PHI_E_SRC register field. When this bit is set, the controller operates in open loop mode.
[14] RAMP_EN	RW 0x0	Enable the Ramper module for generation of ramping movement profiles.
[13] RAMP_MODE	RW 0x0	Whether to generate velocity or positioning movement profiles with the ramper module. An appropriate motion mode must also be set in the MOTION_MODE register field. 0: RAMP_POSITION Position mode. 1: RAMP_VELOCITY Velocity mode.
[12:9] MOTION_MODE	RW 0x0	Motion Controller configuration setup. 0: PWM_OFF PWM output disabled. 1: PWM_ON PWM output enabled. 2: TORQUE Torque mode, user sets torque target value, only the torque PID is used. 3: VELOCITY Velocity mode, user sets velocity target, velocity -> torque cascade is used. 4: POSITION Position mode, user sets position target, full cascaded controller is used. 5: PRBS_UD PRBS as UD. 6: PRBS_FLUX PRBS as flux controller target. 7: PRBS_TORQUE PRBS as torque controller target. 8: PRBS_VELOCITY PRBS as velocity controller target. 9: PRBS_POSITION PRBS as position controller target. 10: PWM_EXT External PWM mode, to manually set the PWM duty cycle. 11: VOLTAGE_EXT External voltage mode, used for open loop control.
[8:7] MOTOR_TYPE	RW 0x0	Type of motor, influences usage of available half bridges. 0: NONE No motor. 1: DC Single phase DC motor, uses U and V half bridge outputs to drive single coil. 2: RESERVED 3: BLDC Three phase BLDC motor, drives three phase currents on U, V, W half bridge outputs.
[6:0] N_POLE_PAIRS	RW, unsigned 0x00	Number of BLDC motor pole pairs. This value is used to calculate the motor's mechanical angle with: phi_m = phi_e × N_POLE_PAIRS. Set a value of 1 when using DC motors.

0x101: MCC_CONFIG.GDRV

Default: 0x81003431

Gate drive configurations.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31] CHARGE_PUMP_EN	RW 0x1	Enable or disable the charge pump.
[28] LP_MODE_EN	RW 0x0	Low power mode.
[24] USE_INTERNAL_R_REF	RW 0x1	Use internal reference resistor.

BITS & NAME	TYPE & RESET	DESCRIPTION
[20] OVERTEMP_LATCH	RW 0x0	Overtemperature fail events are latched in the CHIP.EVENTS.OT_FAIL_EVENT register bit. This bit controls whether or not the event bit has to be cleared before the driver stage is enabled again.
	0: AUTO_RESTART	Automatic restart after overtemperature status bit is cleared.
	1: CLEAR	Overtemperature event bit must be cleared before reenabling the driver stage.
[16] DRV_EN_BIT	RW 0x0	Enable or disable the driver stage. Any motor motion operation requires the driver stage to be enabled (DRV_EN_BIT=1 and DRV_EN input high).
[13:12] OCP_AUTORETRY	RW 0x3	Overcurrent protection (OCP) retry time. When set, the gate driver is automatically restarted some time after an OCP event occurs indefinitely.
	0: RETRY_AFTER_1MS	Retry after 1 ms.
	1: RETRY_AFTER_5MS	Retry after 5 ms.
	2: RETRY_AFTER_50MS	Retry after 50 ms.
	3: NO_RETRY	No retry is executed.
[11:10] OCP_DEGLITCH	RW 0x1	Overcurrent protection (OCP) deglitch time. OCP events are triggered only if the overcurrent state is present this time after the overcurrent status is detected.
	0: DEGLITCH_300_NS	300 ns
	1: DEGLITCH_600_NS	600 ns
	2: DEGLITCH_1200_NS	1200 ns
	3: DEGLITCH_2400_NS	2400 ns
[9] OCP_DETECTION_MODE	RW 0x0	Overcurrent protection (OCP) detected by monitoring load current or output voltage
	0: LOAD_CURRENT	OCP checks load current.
	1: OUTPUT_VOLTAGE	OCP checks output voltage.
[5:4] LS_RES_ON	RW 0x3	Low side power resistor Ron selection.
	0: RES_200_MOHM	200 mOhm
	1: RES_112_MOHM	112 mOhm
	2: RES_78_MOHM	78 mOhm
	3: RES_50_MOHM	55 mOhm
[1:0] SLEW_RATE	RW 0x1	Control output slew rate.
	0: SR_100_V_PER_US	100 V/µs.
	1: SR_200_V_PER_US	200 V/µs.
	2: SR_400_V_PER_US	400 V/µs.
	3: SR_800_V_PER_US	800 V/µs.

0x102: MCC_CONFIG.PWM

Default: 0x00000017

General configurations for the PWM engine.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:16] FLAT_BOTTOM_OFFSET	RW, unsigned 0x0000	Offset for PWM duty cycle for flat bottom modulation with offset mode, used when SV_MODE=BOTTOM_OFFSET.
[5:4] SV_MODE	RW 0x1	Space Vector PWM Mode.
	0: DISABLED	Space Vector PWM disabled.
	1: HARMONIC	Third Harmonic Injection enabled.
	2: BOTTOM	Flat Bottom Modulation.
	3: BOTTOM_OFFSET	Flat Bottom Modulation with Offset.

<table>
<tr><th>BITS &amp; NAME</th><th>TYPE &amp; RESET</th><th colspan="2">DESCRIPTION</th></tr>
<tr><td rowspan="2">[2:0]
CHOP</td><td>RW
0x7</td><td colspan="2">Use CENTERED mode as on, and OFF as off.
Other modes are for testing purposes.</td></tr>
<tr><td>0: OFF_FREE
1: OFF_LSON
2: OFF_HSON
3: OFF_FREE2
4: OFF_FREE3
5: LSPWM_HSOFF
6: HSPWM_LSOFF
7: CENTERED</td><td colspan="2">PWM off, free running.
PWM off, Low Side (LS) permanently on.
PWM off, High Side (HS) permanent on.
PWM off, free running.
PWM off, free running.
Low side (LS) PWM, high side (HS) off.
High side (HS) PWM, low side (LS) off.
Centered PWM for FOC.</td></tr>
</table>

0x103: MCC_CONFIG.PWM_PERIOD

Default: 0xFFFF12C0

Configuration of the PWM frequency as well as the auto Kirchhoff duty cycle limit.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:16] AUTO_KIRCHOFF_LIM	RW, unsigned 0xFFFF	If the PWM duty cycle of any phase exceeds this limit, calculate the third current using Kirchhoff rule instead of the ADC measurement of that phase as it might be corrupted.
[15:0] MAX_COUNT	RW, unsigned 0x12C0	Number of 120 MHz clock cycles from which the PWM signal that drives the output stage is generated. Determines the PWM frequency as well as the FOC controller main update rate. Calculate with MAX_COUNT = 120MHz / f_pwm. The default value of 4800 results in a PWM frequency of 25 kHz.

0x104: MCC_CONFIG.BRAKE_CHOPPER_LIMITS

Default: 0xFFFF0000

Threshold values with which the internal measured supply voltage is compared to control the brake chopper output to be active as soon as a soft overvoltage limit is exceeded.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:16] UPPER_SWITCH_LIM	RW, unsigned 0xFFFF	Upper switching limit for brake chopper of VM measurement. Exceeding this limit activates OV_VM_LIMIT_FAIL_STATUS status bit resp. BRAKE output if this output is enabled by CHIP.IO_MATRIX.
[15:0] LOWER_SWITCH_LIM	RW, unsigned 0x0000	Lower switching limit for brake chopper of VM measurement (low hysteresis value). During overvoltage condition (active OV_VM_LIMIT_FAIL_STATUS bit), falling below this limit deactivates OV_VM_LIMIT_FAIL_STATUS bit resp. BRAKE output if this output is enabled by CHIP.IO_MATRIX.

0x105: MCC_CONFIG.MCC_STATUS

Default: 0x00000000

Status and event bits related to the driver stage.

BITS & NAME	TYPE & RESET	DESCRIPTION
[24] OVERTEMP_CH3_EVENT	RW, W1C 0x0	Overtemperature channel 3 detected
[23] OVERTEMP_CH2_EVENT	RW, W1C 0x0	Overtemperature channel 2 detected
[22] OVERTEMP_CH1_EVENT	RW, W1C 0x0	Overtemperature channel 1 detected
[21] SHRT2V_CH3_EVENT	RW, W1C 0x0	Shrt-to-VS channel 3 detected

BITS & NAME	TYPE & RESET	DESCRIPTION
[20] SHRT2V_CH2_EVENT	RW, W1C 0x0	Shrt-to-VS channel 2 detected
[19] SHRT2V_CH1_EVENT	RW, W1C 0x0	Shrt-to-VS channel 1 detected
[18] SHRT2G_CH3_EVENT	RW, W1C 0x0	Shrt-to-Ground channel 3 detected
[17] SHRT2G_CH2_EVENT	RW, W1C 0x0	Shrt-to-Ground channel 2 detected
[16] SHRT2G_CH1_EVENT	RW, W1C 0x0	Shrt-to-Ground channel 1 detected
[8] OVERTEMP_CH3_STATUS	R 0x0	Overtemperature channel 3
[7] OVERTEMP_CH2_STATUS	R 0x0	Overtemperature channel 2
[6] OVERTEMP_CH1_STATUS	R 0x0	Overtemperature channel 1
[5] SHRT2V_CH3_STATUS	R 0x0	Shrt-to-VS channel 3
[4] SHRT2V_CH2_STATUS	R 0x0	Shrt-to-VS channel 2
[3] SHRT2V_CH1_STATUS	R 0x0	Shrt-to-VS channel 1
[2] SHRT2G_CH3_STATUS	R 0x0	Shrt-to-Ground channel 3
[1] SHRT2G_CH2_STATUS	R 0x0	Shrt-to-Ground channel 2
[0] SHRT2G_CH1_STATUS	R 0x0	Shrt-to-Ground channel 1

0x106: MCC_CONFIG.TORQUE_FF_ACC_CONFIG

Default: 0x00060000

Configuration of the torque feedforward acceleration component.

BITS & NAME	TYPE & RESET	DESCRIPTION
[18:16] RAMP_ACC_SHIFT	RW 0x6	Shift factor to the ramper acceleration for torque feedforward acceleration component calculation. torque_ff_acc = (RAMPER.ACCELERATION × RAMP_ACC_GAIN) >> (RAMP_ACC_SHIFT × 4)
	0: NO_ACC_SHIFT 1: ACC_SHIFT_BY_4 2: ACC_SHIFT_BY_8 3: ACC_SHIFT_BY_12 4: ACC_SHIFT_BY_16 5: ACC_SHIFT_BY_20 6: ACC_SHIFT_BY_24	
[15:0] RAMP_ACC_GAIN	RW, signed 0x0000	Gain factor to scale the ramper acceleration for torque feedforward acceleration component calculation. torque_ff_acc = (RAMPER.ACCELERATION × RAMP_ACC_GAIN) >> (RAMP_ACC_SHIFT × 4)

0x107: MCC_CONFIG.TORQUE_FF_VISC_FRIC_CONFIG

Default: 0x00070000

Configuration of the torque feedforward viscous friction component.

BITS & NAME	TYPE & RESET	DESCRIPTION
[18:16] RAMP_VEL_SHIFT	RW 0x7	Shift factor to the ramper velocity for torque feedforward viscous friction component calculation. torque_ff_visc_fric = (RAMPER.V_ACTUAL × RAMP_VEL_GAIN) >> (RAMP_VEL_SHIFT × 4)
	0: NO_VEL_SHIFT 1: VEL_SHIFT_BY_4 2: VEL_SHIFT_BY_8 3: VEL_SHIFT_BY_12 4: VEL_SHIFT_BY_16 5: VEL_SHIFT_BY_20 6: VEL_SHIFT_BY_24 7: VEL_SHIFT_BY_28	
[15:0] RAMP_VEL_GAIN	RW, signed 0x0000	Gain factor to scale the ramper velocity for torque feedforward viscous friction component calculation. torque_ff_visc_fric = (RAMPER.V_ACTUAL × RAMP_VEL_GAIN) >> (RAMP_VEL_SHIFT × 4)

0x108: MCC_CONFIG.TORQUE_FF_COLOUMB_FRIC

Default: 0x00000000

Configuration of the torque feedforward Coulomb friction component.

BITS & NAME	TYPE & RESET	DESCRIPTION

[15:0]
TORQUE_FF_COLOUMB_FRIC
	RW, signed
	0x0000
		Feedforward value for the torque feedforward Coulomb friction component. The value in this field is used for
		torque_ff_coulomb_fric if RAMPER.V_ACTUAL > 0;
		the inverted value is used if RAMPER.V_ACTUAL < 0; and
		torque_ff_coulomb_fric = 0 otherwise.

0x109: MCC_CONFIG.TORQUE_FEEDFORWARD

Default: 0x00000000

Feedforward value for torque PID controller.

BITS & NAME	TYPE & RESET	DESCRIPTION
[15:0] TORQUE_FEEDFORWARD	R, signed 0x0000	Feedforward value for torque PID controller. Depending on the enabled torque feedforward components in MCC_CONFIG.MOTOR_MOTION register, this value results from: torque_ff_visc_fric + torque_ff_coulomb_fric + torque_ff_acc.

0x140: FOC.PID_CONFIG

Default: 0x0000855C

General configuration for the PID controller.

BITS & NAME	TYPE & RESET	DESCRIPTION
[22:16] POSITION_SAMPLING	RW, unsigned 0x00	Set the sampling frequency for the position controller, calculated as position_controller_sampling_freq = pwm_freq / (VELOCITY_SAMPLING + 1) / (POSITION_SAMPLING + 1). VELOCITY_SAMPLING is part of FEEDBACK.VELOCITY_FRQ_CONF.
[15:12] VELOCITY_SHIFT	RW, unsigned 0x8	Shift factor for the velocity controller output. The shifted value becomes the target of the torque controller when MOTION_MODE = VELOCITY or MOTION_MODE = POSITION. Shift right by VELOCITY_SHIFT bits.

BITS & NAME	TYPE & RESET	DESCRIPTION
[11:10] POSITION_NORM_I	RW 0x1	Normalization factor for the Ki coefficient of the position controller. Change the fixed-point interpretation of the POSITION_I field from default Q8.8 (POSITION_NORM_I = 0) by 8 bits right for each POSITON_NORM_I number increase. This effectively divides the coefficient I value in the register field by 256^(POSITION_NORM_I+1), keeping the fractional part of the result.
	0: I_SHIFT_BY_8	Position Shift Factor Ki: Shifts POSITION_I right by 8 bits
	1: I_SHIFT_BY_16	Position Shift Factor Ki: Shifts POSITION_I right by 16 bits
	2: I_SHIFT_BY_24	Position Shift Factor Ki: Shifts POSITION_I right by 24 bits
	3: I_SHIFT_BY_32	Position Shift Factor Ki: Shifts POSITION_I right by 32 bits
[9:8] POSITION_NORM_P	RW 0x1	Normalization factor for the Kp coefficient of the position controller. Change the fixed-point interpretation of the POSITION_P field from default Q16.0 (POSITION_NORM_P = 0) by 8 bits right for each POSITON_NORM_P number increase. This effectively divides the coefficient P value in the register field by 256^POSITION_NORM_P, keeping the fractional part of the result.
	0: P_NO_SHIFT	Position Shift Factor Kp: No shift for POSITION_P
	1: P_SHIFT_BY_8	Position Shift Factor Kp: Shifts POSITION_P right by 8 bits
	2: P_SHIFT_BY_16	Position Shift Factor Kp: Shifts POSITION_P right by 16 bits
	3: P_SHIFT_BY_24	Position Shift Factor Kp: Shifts POSITION_P right by 24 bits
[7:6] VELOCITY_NORM_I	RW 0x1	Normalization factor for the Ki coefficient of the velocity controller. Change the fixed-point interpretation of the VELOCITY_I field from default Q8.8 (VELOCITY_NORM_I = 0) by 8 bits right for each VELOCITY_NORM_I number increase. This effectively divides the coefficient I value in the register field by 256^(VELOCITY_NORM_I+1), keeping the fractional part of the result.
	0: I_SHIFT_BY_8	Velocity Shift Factor Ki: Shifts VELOCITY_I right by 8 bits
	1: I_SHIFT_BY_16	Velocity Shift Factor Ki: Shifts VELOCITY_I right by 16 bits
	2: I_SHIFT_BY_24	Velocity Shift Factor Ki: Shifts VELOCITY_I right by 24 bits
	3: I_SHIFT_BY_32	Velocity Shift Factor Ki: Shifts VELOCITY_I right by 32 bits
[5:4] VELOCITY_NORM_P	RW 0x1	Normalization factor for the Kp coefficient of the velocity controller. Change the fixed-point interpretation of the VELOCITY_P field from default Q16.0 (VELOCITY_NORM_P = 0) by 8 bits right for each VELOCITY_NORM_P number increase. This effectively divides the coefficient I value in the register field by 256^VELOCITY_NORM_P, keeping the fractional part of the result.
	0: P_NO_SHIFT	Velocity Shift Factor Kp: No shift for VELOCITY_P
	1: P_SHIFT_BY_8	Velocity Shift Factor Kp: Shifts VELOCITY_P right by 8 bits
	2: P_SHIFT_BY_16	Velocity Shift Factor Kp: Shifts VELOCITY_P right by 16 bits
	3: P_SHIFT_BY_24	Velocity Shift Factor Kp: Shifts VELOCITY_P right by 24 bits
[3] CURRENT_NORM_I	RW 0x1	Normalization factor for the torque and flux controllers' Ki coefficients. Change the fixed-point interpretation of TORQUE_I and FLUX_I fields from Q16.0 (CURRENT_NORM_I = 0) by shifting the number of fractional bits by 8 bits right (=> Q8.8). This effectively divides the coefficient values in the register fields by 256^CURRENT_NORM_I, keeping the fractional part of the result.
	0: CUR_I_NO_SHIFT	Current Shift Factor Ki: No shift
	1: CUR_I_SHIFT_BY_8	Current Shift Factor Ki: Shifts FLUX_I and TORQUE_I right by 8 bits

BITS & NAME	TYPE & RESET	DESCRIPTION
[2] CURRENT_NORM_P	RW 0x1	Normalization factor for the torque and flux controllers' Kp coefficients. Change the fixed-point interpretation of TORQUE_P and FLUX_P fields from Q16.0 (CURRENT_NORM_P = 0) by shifting the number of fractional bits by 8 bits right (=> Q8.8). This effectively divides the coefficient values in the register fields by 256^CURRENT_NORM_P, keeping the fractional part of the result.
	0: CUR_P_NO_SHIFT 1: CUR_P_SHIFT_BY_8	Current Shift Factor Kp: No shift Current Shift Factor Kp: Shifts FLUX_P and TORQUE_P right by 8 bits
[1] OVERWRITE_TARGET	RW 0x0	When enabled, the controller targets (pid_velocity_target, pid_torque_target) and integrator (pid_velocity_integrator) are overwritten directly by the module outputs for the smooth motion mode transition.
[0] KEEP_POS_TARGET	RW 0x0	When set, PID_POSITION_TARGET is not automatically overwritten when manually writing PID_POSITION_ACTUAL register.

0x141: FOC.PID_U_S_MAX

Default: 0x00007FFF

Maximum permissible back-EMF motor voltage, beyond which the field-weakening controller is activated.

BITS & NAME	TYPE & RESET	DESCRIPTION
[15:0] U_S_MAX	RW, unsigned 0x7FFF	Maximum permissible back-EMF motor voltage, beyond which the field-weakening controller is activated.

0x142: FOC.PID_FLUX_COEFF

Default: 0x00000000

Configuration of the PI Flux controller gains.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:16] FLUX_P	RW, signed 0x0000	Proportional gain for the PI Flux controller.
[15:0] FLUX_I	RW, signed 0x0000	Integral gain for the PI Flux controller.

0x143: FOC.PID_TORQUE_COEFF

Default: 0x00000000

Configuration of the PI Torque controller gains.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:16] TORQUE_P	RW, signed 0x0000	Proportional gain for the PI Torque controller.
[15:0] TORQUE_I	RW, signed 0x0000	Integral gain for the PI Torque controller.

0x144: FOC.PID_FIELDWEAK_COEFF

Default: 0x00000000

Configuration of the PI Field-weakening controller gains.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:16] FIELDWEAK_P	RW, signed 0x0000	Proportional gain for the PI Field-weakening controller.
[15:0] FIELDWEAK_I	RW, signed 0x0000	Integral gain for the PI Field-weakening controller.

0x145: FOC.PID_VELOCITY_COEFF

Default: 0x00000000

Configuration of the PI Velocity controller gains.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:16] VELOCITY_P	RW, signed 0x0000	Proportional gain for the PI Velocity controller.
[15:0] VELOCITY_I	RW, signed 0x0000	Integral gain for the PI Velocity controller.

0x146: FOC.PID_POSITION_COEFF

Default: 0x00000000

Configuration of the PI Position controller gains.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:16] POSITION_P	RW, signed 0x0000	Proportional gain for the PI Position controller.
[15:0] POSITION_I	RW, signed 0x0000	Integral gain for the PI Position controller.

0x147: FOC.PID_POSITION_TOLERANCE

Default: 0x00000000

Configuration of the position tolerance of the controller.

BITS & NAME	TYPE & RESET	DESCRIPTION
[30:0] PID_POSITION_TOLERANCE	RW, unsigned 0x00000000	The position controller ignores position errors smaller than this value. Two conditions must be true for this to take effect: the position_reached_event interrupt signal must be set and the error must remain smaller for at least the configured PID_POSITION_TOLERANCE_DELAY PWM cycles.

0x148: FOC.PID_POSITION_TOLERANCE_DELAY

Default: 0x00000000

Configuration of the timings for the position tolerance of the controller.

BITS & NAME	TYPE & RESET	DESCRIPTION
[15:0] PID_POSITION_TOLERANCE_DELAY	RW, unsigned 0x0000	Number of PWM periods the absolute PID_POSITION_ERROR value must stay within configured PID_POSITION_TOLERANCE value for the position controller to ignore the errors.

0x149: FOC.PID_UQ_UD_LIMITS

Default: 0x00005A81

Configuration of limits for the target values of the voltage controller.

BITS & NAME	TYPE & RESET	DESCRIPTION
[15:0] PID_UQ_UD_LIMITS	RW, unsigned 0x5A81	Limit value for the absolute target values of the voltage controller.

0x14A: FOC.PID_TORQUE_FLUX_LIMITS

Default: 0x7FFF7FFF

Configuration of limits for the absolute target values of the torque and flux controllers.

BITS & NAME	TYPE & RESET	DESCRIPTION
[30:16] PID_TORQUE_LIMIT	RW, unsigned 0x7FFF	Limit for the absolute target value of the torque controller.
[14:0] PID_FLUX_LIMIT	RW, unsigned 0x7FFF	Limit for the absolute target value of the flux controller.

0x14B: FOC.PID_VELOCITY_LIMIT

Default: 0x7FFFFFFF

Configuration of limits for the absolute target value of the velocity controller.

BITS & NAME	TYPE & RESET	DESCRIPTION
[30:0] PID_VELOCITY_LIMIT	RW, unsigned 0x7FFFFFFF	Limit for the absolute target value of the velocity controller.

0x14C: FOC.PID_POSITION_LIMIT_LOW

Default: 0x80000001

Configuration of lower limit for the target value of the position controller.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:0] PID_POSITION_LIMIT_LOW	RW, signed 0x80000001	Lower limit for the target value of the position controller. The limit can be positive or negative, but must be lower than PID_POSITION_LIMIT_HIGH.

0x14D: FOC.PID_POSITION_LIMIT_HIGH

Default: 0x7FFFFFFF

Configuration of upper limit for the target value of the position controller.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:0] PID_POSITION_LIMIT_HIGH	RW, signed 0x7FFFFFFF	Upper limit for the target value of the position controller. The limit can be positive or negative, but must be higher than PID_POSITION_LIMIT_LOW.

0x14E: FOC.PID_TORQUE_FLUX_TARGET

Default: 0x00000000

Externally writable PID target values for the torque and flux controllers (MOTION_MODE = TORQUE).

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:16] PID_TORQUE_TARGET	RW, signed 0x0000	Externally writable PID target value for the torque controller, used when MOTION_MODE = TORQUE.
[15:0] PID_FLUX_TARGET	RW, signed 0x0000	Externally writable PID target value for the flux controller, used when MOTION_MODE = TORQUE.

0x14F: FOC.PID_TORQUE_FLUX_OFFSET

Default: 0x00000000

PID torque and flux offset.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:16] PID_TORQUE_OFFSET	RW, signed 0x0000	Torque offset for feed forward control.
[15:0] PID_FLUX_OFFSET	RW, signed 0x0000	Flux offset for feed forward control.

0x150: FOC.PID_VELOCITY_TARGET

Default: 0x00000000

Externally writable PID target value for the velocity controller (MOTION_MODE = VELOCITY).

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:0] PID_VELOCITY_TARGET	RW, signed 0x00000000	Externally writable PID target value for the velocity controller, used when MOTION_MODE = VELOCITY.

0x151: FOC.PID_VELOCITY_OFFSET

Default: 0x00000000

PID velocity offset for feed forward control.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:0] PID_VELOCITY_OFFSET	RW, signed 0x00000000	PID velocity offset for feed forward control.

0x152: FOC.PID_POSITION_TARGET

Default: 0x00000000

Externally writable PID target value for the position controller (MOTION_MODE = POSITION).

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:0] PID_POSITION_TARGET	RW, signed 0x00000000	Externally writable PID target value for the position controller, used when MOTION_MODE = POSITION.

0x153: FOC.PID_TORQUE_FLUX_ACTUAL

Default: 0x00000000

PID actual values for the torque and flux controllers.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:16] PID_TORQUE_ACTUAL	R, signed 0x0000	PID actual value for the torque controller.
[15:0] PID_FLUX_ACTUAL	R, signed 0x0000	PID actual value for the flux controller.

0x154: FOC.PID_VELOCITY_ACTUAL

Default: 0x00000000

PID actual value for the velocity controller.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:0] PID_VELOCITY_ACTUAL	R, signed 0x00000000	PID actual value for the velocity controller.

0x155: FOC.PID_POSITION_ACTUAL

Default: 0x00000000

PID actual value for the position controller.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:0] PID_POSITION_ACTUAL	RW, signed 0x00000000	PID actual value for the position controller. Unless KEEP_POS_TARGET bit is set, manually writing a value on this field writes the same value into PID_POSITION_TARGET to avoid unexpected motion.

0x156: FOC.PID_POSITION_ACTUAL_OFFSET

Default: 0x00000000

PID position actual offset.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:0] PID_POSITION_ACTUAL_OFFSET	RW, signed 0x00000000	PID position actual offset.

0x157: FOC.PID_TORQUE_ERROR

Default: 0x00000000

PID error value from the torque controller.

BITS & NAME	TYPE & RESET	DESCRIPTION
[15:0] PID_TORQUE_ERROR	R, signed 0x0000	PID error value from the torque controller.

0x158: FOC.PID_FLUX_ERROR

Default: 0x00000000

PID error value from the flux controller.

BITS & NAME	TYPE & RESET	DESCRIPTION
[15:0] PID_FLUX_ERROR	R, signed 0x0000	PID error value from the flux controller.

0x159: FOC.PID_VELOCITY_ERROR

Default: 0x00000000

PID error value from the velocity controller.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:0] PID_VELOCITY_ERROR	R, signed 0x00000000	PID error value from the velocity controller.

0x15A: FOC.PID_VELOCITY_ERROR_MAX

Default: 0x7FFFFFFF

Maximum permissible absolute PID velocity error value.

BITS & NAME	TYPE & RESET	DESCRIPTION
[30:0] PID_VELOCITY_ERROR_MAX	RW, unsigned 0x7FFFFFFF	Maximum permissible PID velocity error value, beyond which the RAMPER.STATUS.STALL_IN_V_ERR status bit is set.

0x15B: FOC.PID_POSITION_ERROR

Default: 0x00000000

PID error value from the position controller.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:0] PID_POSITION_ERROR	R, signed 0x00000000	PID error value from the position controller.

0x15C: FOC.PID_POSITION_ERROR_MAX

Default: 0x7FFFFFFF

Maximum permissible absolute PID velocity error value.

BITS & NAME	TYPE & RESET	DESCRIPTION
[30:0] PID_POSITION_ERROR_MAX	RW, unsigned 0x7FFFFFFF	Maximum permissible PID velocity error value, beyond which the RAMPER.STATUS.STALL_IN_POSITION_ERR status bit is set.

0x15D: FOC.PID_TORQUE_INTEGRATOR

Default: 0x00000000

Integral component of the PID torque controller.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:0] PID_TORQUE_INTEGRATOR	RW, signed 0x00000000	Integral component of the PID torque controller.

0x15E: FOC.PID_FLUX_INTEGRATOR

Default: 0x00000000

Integral component of the PID flux controller.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:0] PID_FLUX_INTEGRATOR	RW, signed 0x00000000	Integral component of the PID flux controller.

0x15F: FOC.PID_VELOCITY_INTEGRATOR

Default: 0x00000000

Integral component of the PID velocity controller.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:0] PID_VELOCITY_INTEGRATOR	RW, signed 0x00000000	Integral component of the PID velocity controller.

0x160: FOC.PID_POSITION_INTEGRATOR

Default: 0x00000000

Integral component of the PID position controller.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:0] PID_POSITION_INTEGRATOR	RW, signed 0x00000000	Integral component of the PID position controller.

0x161: FOC.PIDIN_TORQUE_FLUX_TARGET

Default: 0x00000000

PID torque and flux target value before any filter, offset, feedforward compensation or limits are applied.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:16] PIDIN_TORQUE_TARGET	R, signed 0x0000	PID torque target value before any filter, offset, feedforward compensation or limits are applied.
[15:0] PIDIN_FLUX_TARGET	R, signed 0x0000	PID flux target value before any filter, offset, feedforward compensation or limits are applied.

0x162: FOC.PIDIN_VELOCITY_TARGET

Default: 0x00000000

PID velocity target value velocity before any offset, feedforward compensation or limits are applied.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:0] PIDIN_VELOCITY_TARGET	R, signed 0x00000000	PID velocity target value velocity before any offset, feedforward compensation or limits are applied.

0x163: FOC.PIDIN_POSITION_TARGET

Default: 0x00000000

PID position target value before any limits are applied.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:0] PIDIN_POSITION_TARGET	R, signed 0x00000000	PID position target value before any limits are applied.

0x164: FOC.PIDIN_TORQUE_FLUX_TARGET_LIMITED

Default: 0x00000000

PID target value for the torque and flux controllers after the set filter, offset, feedforward compensation and limits are applied.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:16] PIDIN_TORQUE_TARGET_LIMITED	R, signed 0x0000	PID target value for the torque controller after the set filter, offset, feedforward compensation and limits are applied.
[15:0] PIDIN_FLUX_TARGET_LIMITED	R, signed 0x0000	PID target value for the flux controller after the set filter, offset, feedforward compensation and limits are applied.

0x165: FOC.PIDIN_VELOCITY_TARGET_LIMITED

Default: 0x00000000

PID target value for the velocity controller after the set offset, feedforward compensation and limits are applied.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:0] PIDIN_VELOCITY_TARGET_LIMITED	R, signed 0x00000000	PID target value for the velocity controller after the set offset, feedforward compensation and limits are applied.

0x166: FOC.PIDIN_POSITION_TARGET_LIMITED

Default: 0x00000000

PID target value for the position controller after the set limits are applied.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:0] PIDIN_POSITION_TARGET_LIMITED	R, signed 0x00000000	PID target value for the position controller after the set limits are applied.

0x167: FOC.FOC_IBETA_IALPHA

Default: 0x00000000

Interim result for Ibeta and Ialpha terms of the FOC module.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:16] IBETA	R, signed 0x0000	Interim result for Ibeta term of the FOC module.
[15:0] IALPHA	R, signed 0x0000	Interim result for Ialpha term of the FOC module.

0x168: FOC.FOC_IQ_ID

Default: 0x00000000

Interim result for Iq and Id terms of the FOC module.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:16] IQ	R, signed 0x0000	Interim result for Iq term of the FOC module.
[15:0] ID	R, signed 0x0000	Interim result for Id term of the FOC module.

0x169: FOC.FOC_UQ_UD

Default: 0x00000000

Interim result for Uq and Ud terms of the FOC module before any limits are applied.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:16] UQ	R, signed 0x0000	Interim result for Uq term of the FOC module before any limits are applied.
[15:0] UD	R, signed 0x0000	Interim result for Ud term of the FOC module before any limits are applied.

0x16A: FOC.FOC_UQ_UD_LIMITED

Default: 0x00000000

Interim result for Uq and Ud terms of the FOC module after the set limits are applied.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:16] UQ_LIMITED	R, signed 0x0000	Interim result for Uq term of the FOC module after the set limits are applied.
[15:0] UD_LIMITED	R, signed 0x0000	Interim result for Ud term of the FOC module after the set limits are applied.

0x16B: FOC.FOC_UBETA_UALPHA

Default: 0x00000000

Interim result for Ubeta and Ualpha terms of the FOC module.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:16] UBETA	R, signed 0x0000	Interim result for Ubeta term of the FOC module.
[15:0] UALPHA	R, signed 0x0000	Interim result for Ualpha term of the FOC module.

0x16C: FOC.FOC_UW_UU

Default: 0x00000000

Interim result for UW and UU terms of the FOC module.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:16] UW	R, signed 0x0000	Interim result for UW term of the FOC module.
[15:0] UU	R, signed 0x0000	Interim result for UU term of the FOC module.

0x16D: FOC.FOC_UV

Default: 0x00000000

Interim result for the UV term of the FOC module.

BITS & NAME	TYPE & RESET	DESCRIPTION
[15:0] UV	R, signed 0x0000	Interim result for the UV term of the FOC module.

0x16E: FOC.PWM_V_U

Default: 0x00000000

PWM duty cycle for the V and U phases.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:16] PWM_V	R, unsigned 0x0000	PWM duty cycle for the V phase.
[15:0] PWM_U	R, unsigned 0x0000	PWM duty cycle for the U phase.

0x16F: FOC.PWM_W

Default: 0x00000000

PWM duty cycle for the W phase.

BITS & NAME	TYPE & RESET	DESCRIPTION
[15:0] PWM_W	R, unsigned 0x0000	PWM duty cycle for the W phase.

0x170: FOC.FOC_STATUS

Default: 0x00000000

Current state of the FOC module.

BITS & NAME	TYPE & RESET	DESCRIPTION
[3:0] FOC_STATUS	R, unsigned 0x0	Current state of the FOC module.

0x171: FOC.U_S_ACTUAL_I_S_ACTUAL

Default: 0x00000000

Actual motor voltage and current values.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:16] U_S_ACTUAL	R, unsigned 0x0000	Actual motor voltage value, calculated as U_S_ACTUAL = sqrt(u_d^2 + u_q^2).
[15:0] I_S_ACTUAL	R, unsigned 0x0000	Actual motor current value, calculated as I_S_ACTUAL = sqrt(i_d^2 + i_q^2).

0x172: FOC.P_MOTOR

Default: 0x00000000

Actual power applied to the motor.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:0] P_MOTOR	R, unsigned 0x00000000	Actual power applied to the motor, calculated as P_MOTOR = U_S_ACTUAL × I_S_ACTUAL.

0x173: FOC.I_T_ACTUAL

Default: 0x00000000

Actual target current value.

BITS & NAME	TYPE & RESET	DESCRIPTION
[15:0] I_T_ACTUAL	R, unsigned 0x0000	Actual target current value, calculated as I_T_ACTUAL = sqrt(i_d_target^2 + i_q_target^2).

0x174: FOC.PRBS_AMPLITUDE

Default: 0x00000000

Amplitude of PRBS signal, used for system identification.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:0] PRBS_AMPLITUDE	RW, unsigned 0x00000000	Amplitude of PRBS signal.

0x175: FOC.PRBS_DOWN_SAMPLING_RATIO

Default: 0x00000000

Down sampling rate of PRBS signal, used for system identification.

BITS & NAME	TYPE & RESET	DESCRIPTION
[7:0] PRBS_DOWN_SAMPLING_RATIO	RW, unsigned 0x00	Down sampling rate of PRBS signal, prbs_freq = pwm_freq / (PRBS_DOWN_SAMPLING_RATIO + 1).

0x180: BIQUAD.BIQUAD_EN

Default: 0x00000001

Enable signals for the velocity and torque biquad filters. Note that the velocity filter is used for velocity measurement, while the torque filter is used for the torque controller's target value.

BITS & NAME	TYPE & RESET	DESCRIPTION
[1] TORQUE_EN	RW 0x0	Enable the biquad filter for the torque controller's target value.
[0] VELOCITY_EN	RW 0x1	Enable the biquad filter for the velocity actual measurement.

0x181: BIQUAD.VELOCITY_A1

Default: 0x001C376B

Velocity biquad filter coefficient a1.

BITS & NAME	TYPE & RESET	DESCRIPTION
[23:0] COEFF_VELOCITY_A1	RW, signed 0x1C376B	Velocity biquad filter coefficient a1

0x182: BIQUAD.VELOCITY_A2

Default: 0x00F38F52

Velocity biquad filter coefficient a2.

BITS & NAME	TYPE & RESET	DESCRIPTION
[23:0] COEFF_VELOCITY_A2	RW, signed 0xF38F52	Velocity biquad filter coefficient a2

0x183: BIQUAD.VELOCITY_B0

Default: 0x00000E51

Velocity biquad filter coefficient b0.

BITS & NAME	TYPE & RESET	DESCRIPTION
[23:0] COEFF_VELOCITY_B0	RW, signed 0x000E51	Velocity biquad filter coefficient b0

0x184: BIQUAD.VELOCITY_B1

Default: 0x00001CA1

Velocity biquad filter coefficient b1.

BITS & NAME	TYPE & RESET	DESCRIPTION
[23:0] COEFF_VELOCITY_B1	RW, signed 0x001CA1	Velocity biquad filter coefficient b1

0x185: BIQUAD.VELOCITY_B2

Default: 0x00000E51

Velocity biquad filter coefficient b2.

BITS & NAME	TYPE & RESET	DESCRIPTION
[23:0] COEFF_VELOCITY_B2	RW, signed 0x000E51	Velocity biquad filter coefficient b2

0x186: BIQUAD.TORQUE_A1

Default: 0x00000000

Torque biquad filter coefficient a1.

BITS & NAME	TYPE & RESET	DESCRIPTION
[23:0] COEFF_TORQUE_A1	RW, signed 0x000000	Torque biquad filter coefficient a1

0x187: BIQUAD.TORQUE_A2

Default: 0x00000000

Torque biquad filter coefficient a2.

BITS & NAME	TYPE & RESET	DESCRIPTION
[23:0] COEFF_TORQUE_A2	RW, signed 0x000000	Torque biquad filter coefficient a2

0x188: BIQUAD.TORQUE_B0

Default: 0x00100000

Torque biquad filter coefficient b0.

BITS & NAME	TYPE & RESET	DESCRIPTION
[23:0] COEFF_TORQUE_B0	RW, signed 0x100000	Torque biquad filter coefficient b0

0x189: BIQUAD.TORQUE_B1

Default: 0x00000000

Torque biquad filter coefficient b1.

BITS & NAME	TYPE & RESET	DESCRIPTION
[23:0] COEFF_TORQUE_B1	RW, signed 0x000000	Torque biquad filter coefficient b1

0x18A: BIQUAD.TORQUE_B2

Default: 0x00000000

Torque biquad filter coefficient b2.

BITS & NAME	TYPE & RESET	DESCRIPTION
[23:0] COEFF_TORQUE_B2	RW, signed 0x000000	Torque biquad filter coefficient b2

0x1C0: RAMPER.TIME_CONFIG

Default: 0x00000000

Configuration of Ramp timing parameters.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:16] T_VMAX	RW, unsigned 0x0000	Minimum time at VMAX before starting deceleration.
[15:0] T_ZEROWAIT	RW, unsigned 0x0000	Minimum time after finishing a ramp positioning movement before starting a new set movement.

0x1C1: RAMPER.SWITCH_MODE

Default: 0x00000000

Configuration of Ramp reference switches and stop signals.

BITS & NAME	TYPE & RESET	DESCRIPTION
[18] STOP_ON_V_DEVIATION	RW 0x0	Allow the STALL_IN_V_ERR signal to trigger a stall stop event for the Ramp controller. STALL_STOP_EN must also be set to generate the stop event.
[17] STOP_ON_POS_DEVIATION	RW 0x0	Allow the STALL_IN_POSITION_ERR signal to trigger a stall stop event for the Ramp controller. STALL_STOP_EN must also be set to generate the stop event.
[16] FORCE_HARD_STOP	RW 0x0	Manually trigger a stall stop event for the Ramper controller. STALL_STOP_EN must also be set to generate the stop event.
[15] SOFT_STOP_EN	RW 0x0	Use a soft stop sequence instead of a hard stop for the reference switch stop events. The RAMP_REF_H_STOP_EN, RAMP_REF_R_STOP_EN, RAMP_REF_L_STOP_EN, and/or STALL_STOP_EN bits must be set for the corresponding reference switch to trigger a stop event. The soft stop uses the deceleration ramp settings DMAX, D2, D1, V2, V1, VSTOP and T_ZEROWAIT to stop the motor. Hard stops also use T_ZEROWAIT before the motor is released.
[14] STALL_STOP_EN	RW 0x0	Enable stall stop events from STOP_ON_V_DEVIATION, STOP_ON_POS_DEVIATION and FORCE_HARD_STOP to stop the motor through the Ramper controller. When disabled, the EVENTS.STALL_STOP_EVENT bit is automatically cleared.
[12] LATCH_REF_H_INACTIVE	RW 0x0	Activate latching of the position to POSITION_LATCH and POSITION_ACTUAL_LATCH upon an inactive going edge on the home reference switch input. The active level's polarity is defined by RAMP_REF_H_POL. Use LATCH_REF_H_READY event flag to monitor if the latched position can be retrieved.

BITS & NAME	TYPE & RESET	DESCRIPTION
[11] LATCH_REF_H_ACTIVE	RW 0x0	Activate latching of the position to POSITION_LATCH and POSITION_ACTUAL_LATCH upon an active going edge on the home reference switch input. The active level's polarity is defined by RAMP_REF_H_POL. Use LATCH_REF_H_READY event flag to monitor if the latched position can be retrieved.
[10] LATCH_REF_R_INACTIVE	RW 0x0	Activate latching of the position to POSITION_LATCH and POSITION_ACTUAL_LATCH upon an inactive going edge on the right reference switch input. The active level's polarity is defined by RAMP_REF_R_POL. Use LATCH_REF_R_READY event flag to monitor if the latched position can be retrieved.
[9] LATCH_REF_R_ACTIVE	RW 0x0	Activate latching of the position to POSITION_LATCH and POSITION_ACTUAL_LATCH upon an active going edge on the right reference switch input. The active level's polarity is defined by RAMP_REF_R_POL. Use LATCH_REF_R_READY event flag to monitor if the latched position can be retrieved.
[8] LATCH_REF_L_INACTIVE	RW 0x0	Activate latching of the position to POSITION_LATCH and POSITION_ACTUAL_LATCH upon an inactive going edge on the left reference switch input. The active level's polarity is defined by RAMP_REF_L_POL. Use LATCH_REF_L_READY event flag to monitor if the latched position can be retrieved.
[7] LATCH_REF_L_ACTIVE	RW 0x0	Activate latching of the position to POSITION_LATCH and POSITION_ACTUAL_LATCH upon an active going edge on the left reference switch input. The active level's polarity is defined by RAMP_REF_L_POL. Use LATCH_REF_L_READY event flag to monitor if the latched position can be retrieved.
[6] SWAP_RAMP_REF_LR	RW 0x0	Swap the left and the right reference switch input REF_L and REF_R.
[5] RAMP_REF_H_POL	RW 0x0	Sets the active polarity of the home reference switch input.
	0: NON_INVERTED 1: INVERTED	Non-inverted, high active. Inverted, low active.
[4] RAMP_REF_R_POL	RW 0x0	Sets the active polarity of the right reference switch input.
	0: NON_INVERTED 1: INVERTED	Non-inverted, high active. Inverted, low active.
[3] RAMP_REF_L_POL	RW 0x0	Sets the active polarity of the left reference switch input.
	0: NON_INVERTED 1: INVERTED	Non-inverted, high active. Inverted, low active.
[2] RAMP_REF_H_STOP_EN	RW 0x0	Enable automatic motor stop through the Ramper controller during active home reference switch input. The status of the stop signal can be retrieved from RAMP_REF_H_STOP_STATUS bit.
[1] RAMP_REF_R_STOP_EN	RW 0x0	Enable automatic motor stop through the Ramper controller during active right reference switch input. The stop signal is triggered if the velocity has a positive sign when activating the switch input. The status of the stop signal can be retrieved from RAMP_REF_R_STOP_STATUS bit.
[0] RAMP_REF_L_STOP_EN	RW 0x0	Enable automatic motor stop through the Ramper controller during active left reference switch input. The stop signal is triggered if the velocity has a negative sign when activating the switch input. The status of the stop signal can be retrieved from RAMP_REF_L_STOP_STATUS bit.

0x1C3: RAMPER.PHI_E

Default: 0x00000000

Ramper phi_e based on ramper position.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:16] PHI_E_OFFSET	RW, unsigned 0x0000	Offset for Ramp phi_e calculation.
[15:0] PHI_E	R, unsigned 0x0000	Open loop phi_e, calculated from POSITION.POSITION+PHI_E_OFFSET. Used when MCC_CONFIG.MOTOR_MOTION.RAMP_USE_PHI_E is set.

0x1C4: RAMPER.A1

Default: 0x00010000

EightPoint Ramp acceleration from V_START to V1.

BITS & NAME	TYPE & RESET	DESCRIPTION
[22:0] A1	RW, unsigned 0x010000	EightPoint Ramp acceleration from V_START to V1.

0x1C5: RAMPER.A2

Default: 0x00010000

EightPoint Ramp acceleration from V1 to V2.

BITS & NAME	TYPE & RESET	DESCRIPTION
[22:0] A2	RW, unsigned 0x010000	EightPoint Ramp acceleration from V1 to V2.

0x1C6: RAMPER.A_MAX

Default: 0x00010000

EightPoint Ramp acceleration from V2 to V_MAX.

BITS & NAME	TYPE & RESET	DESCRIPTION
[22:0] A_MAX	RW, unsigned 0x010000	EightPoint Ramp acceleration from V2 to V_MAX.

0x1C7: RAMPER.D1

Default: 0x00010000

EightPoint Ramp deceleration from V1 to V_STOP.

BITS & NAME	TYPE & RESET	DESCRIPTION
[22:0] D1	RW, unsigned 0x010000	EightPoint Ramp deceleration from V1 to V_STOP.

0x1C8: RAMPER.D2

Default: 0x00010000

EightPoint Ramp deceleration from V2 to V1.

BITS & NAME	TYPE & RESET	DESCRIPTION
[22:0] D2	RW, unsigned 0x010000	EightPoint Ramp deceleration from V2 to V1.

0x1C9: RAMPER.D_MAX

Default: 0x00010000

EightPoint Ramp deceleration from V_MAX to V2.

BITS & NAME	TYPE & RESET	DESCRIPTION
[22:0] D_MAX	RW, unsigned 0x010000	EightPoint Ramp deceleration from V_MAX to V2.

0x1CA: RAMPER.V_START

Default: 0x00000000

Starting velocity of the EightPoint Ramp.

BITS & NAME	TYPE & RESET	DESCRIPTION
[22:0] V_START	RW, unsigned 0x000000	Starting velocity of the EightPoint Ramp.

0x1CB: RAMPER.V1

Default: 0x00000000

First switch velocity of the EightPoint Ramp.

BITS & NAME	TYPE & RESET	DESCRIPTION
[26:0] V1	RW, unsigned 0x0000000	First velocity of the EightPoint Ramp.

0x1CC: RAMPER.V2

Default: 0x00000000

Second switch velocity of the EightPoint Ramp.

BITS & NAME	TYPE & RESET	DESCRIPTION
[26:0] V2	RW, unsigned 0x0000000	Second velocity of the EightPoint Ramp.

0x1CD: RAMPER.V_STOP

Default: 0x00000100

Stopping velocity of the EightPoint Ramp.

BITS & NAME	TYPE & RESET	DESCRIPTION
[22:0] V_STOP	RW, unsigned 0x000100	Stopping velocity of the EightPoint Ramp.

0x1CE: RAMPER.V_MAX

Default: 0x07FFFFFF

Maximum (third) velocity of the EightPoint Ramp.

BITS & NAME	TYPE & RESET	DESCRIPTION
[26:0] V_MAX	RW, unsigned 0x7FFFFFF	Maximum (third) velocity of the EightPoint Ramp.

0x1CF: RAMPER.ACCELERATION

Default: 0x00000000

Current acceleration output value of Ramp controller.

BITS & NAME	TYPE & RESET	DESCRIPTION
[23:0] ACCELERATION	R, signed 0x000000	Current acceleration output value of Ramp controller.

0x1D0: RAMPER.V_ACTUAL

Default: 0x00000000

Current velocity output value of Ramp controller.

BITS & NAME	TYPE & RESET	DESCRIPTION
[27:0] V_ACTUAL	R, signed 0x0000000	Current velocity output value of Ramp controller.

0x1D1: RAMPER.POSITION

Default: 0x00000000

Current multi-turn position output value of Ramp controller.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:0] POSITION	RW, signed 0x00000000	Current multi-turn position output value of Ramp controller.

0x1D2: RAMPER.POSITION_LATCH

Default: 0x00000000

Latched RAMPER.POSITION on reference switch trigger. The trigger must be configured in RAMPER.SWITCH_MODE.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:0] POSITION_LATCH	R, signed 0x00000000	Latched RAMPER.POSITION on reference switch trigger. The trigger must be configured in RAMPER.SWITCH_MODE.

0x1D3: RAMPER.POSITION_ACTUAL_LATCH

Default: 0x00000000

Latched FOC.PID_POSITION_ACTUAL on reference switch trigger. The trigger must be configured in RAMPER.SWITCH_MODE.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:0] POSITION_ACTUAL_LATCH	R, signed 0x00000000	Latched FOC.PID_POSITION_ACTUAL on reference switch trigger. The trigger must be configured in RAMPER.SWITCH_MODE.

0x1D4: RAMPER.STATUS

Default: 0x00000000

Status bits for the Ramp controller.

BITS & NAME	TYPE & RESET	DESCRIPTION
[17] STALL_IN_POSITION_ERR	R 0x0	This bit is set when the current absolute position deviation exceeds FOC.PID_POSITION_ERROR_MAX.
[16] STALL_IN_V_ERR	R 0x0	This bit is set when the current absolute velocity deviation exceeds FOC.PID_VELOCITY_ERROR_MAX.
[14] T_ZEROWAIT_ACTIVE	R 0x0	This bit is set when the T_ZEROWAIT portion of the Ramp is active, signaling that the motor is in standstill.

BITS & NAME	TYPE & RESET	DESCRIPTION
[13] V_ZERO	R 0x0	This bit is set when the value of V_ACTUAL is zero.
[12] POSITION_REACHED_STATUS	R 0x0	This bit is set while POSITION.POSITION and FOC.PID_POSITION_TARGET match.
[11] V_REACHED_STATUS	R 0x0	Is active during RAMP_MODE = RAMP_POSITION and V_ACTUAL matches V_MAX, or during RAMP_MODE = RAMP_VELOCITY and V_ACTUAL matches FOC.PID_VELOCITY_TARGET. Note that RAMP_MODE is part of the MCC_CONFIG.MOTOR_MOTION register.
[8] RAMP_REF_H_STOP_STATUS	R 0x0	Signals that a stop condition on the home reference switch is active. If soft stop is enabled, the condition will become active not after the soft stop sequence is finished, but as soon as the switch is detected.
[7] RAMP_REF_R_STOP_STATUS	R 0x0	Signals that a stop condition on the right reference switch is active. If soft stop is enabled, the condition will become active not after the soft stop sequence is finished, but as soon as the switch is detected. The interrupt and stop conditions can be cleared by commanding a motion into the opposite direction.
[6] RAMP_REF_L_STOP_STATUS	R 0x0	Signals that a stop condition on the left reference switch is active. If soft stop is enabled, the condition will become active not after the soft stop sequence is finished, but as soon as the switch is detected. The interrupt and stop conditions can be cleared by commanding a motion into the opposite direction.
[2] RAMP_REF_H_STATUS	R 0x0	Status of the home reference switch as configured in the Ramp controller.
[1] RAMP_REF_R_STATUS	R 0x0	Status of the right reference switch as configured in the Ramp controller.
[0] RAMP_REF_L_STATUS	R 0x0	Status of the left reference switch as configured in the Ramp controller.

0x1D5: RAMPER.EVENTS

Default: 0x00000000

Event flags for the Ramp controller.

BITS & NAME	TYPE & RESET	DESCRIPTION
[5] LATCH_REF_R_READY	RW, W1C 0x0	Signals that a position value was latched after a right reference switch event. Position latching must be enabled for the desired edge in the in corresponding bit of SWITCH_MODE register. Write a '1' to clear the bit.
[4] LATCH_REF_L_READY	RW, W1C 0x0	Signals that a position value was latched after a left reference switch event. Position latching must be enabled for the desired edge in the in corresponding bit of SWITCH_MODE register. Write a '1' to clear the bit.
[3] LATCH_REF_H_READY	RW, W1C 0x0	Signals that a position value was latched after a home reference switch event. Position latching must be enabled for the desired edge in the in corresponding bit of SWITCH_MODE register. Write a '1' to clear the bit.
[2] SECOND_MOVE_EVENT	RW, W1C 0x0	Signals that the Ramp controller performed an automatic movement in the opposite direction, for example due to parameter changes while in motion. Write a '1' to clear the bit.
[1] STALL_STOP_EVENT	RW, W1C 0x0	Signals that stall stop event occurred. The STALL_STOP_EN as well as the desired trigger condition bits must be set in SWITCH_MODE register. Note that the motor motion may be resumed if this bit is manually cleared. Write a '1' to clear the bit and the corresponding interrupt condition.
[0] POSITION_REACHED_EVENT	RW, W1C 0x0	Signals that the target position has been reached, that is, that POSITION.POSITION and FOC.PID_POSITION_TARGET became equal. Write a '1' to clear the bit and the corresponding interrupt condition.

0x200: EXT_CTRL.VOLTAGE

Default: 0x00000000

Externally writable voltage component values, used for open loop voltage control if MOTION_MODE = VOLTAGE_EXT.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:16] UQ	RW, signed 0x0000	Externally writable voltage component Uq value, used for open loop voltage control if MOTION_MODE = VOLTAGE_EXT.
[15:0] UD	RW, signed 0x0000	Externally writable voltage component Ud value, used for open loop voltage control if MOTION_MODE = VOLTAGE_EXT.

0x202: EXT_CTRL.PWM_V_U

Default: 0x00000000

Externally writable PWM duty cycle values, used when MOTION_MODE=PWM_EXT.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:16] PWM_V	RW, unsigned 0x0000	Externally writable PWM duty cycle value for the motor's V phase, used when MOTION_MODE = PWM_EXT.
[15:0] PWM_U	RW, unsigned 0x0000	Externally writable PWM duty cycle value for the motor's U phase, used when MOTION_MODE = PWM_EXT.

0x203: EXT_CTRL.PWM_W

Default: 0x00000000

Externally writable PWM duty cycle values, used when MOTION_MODE = PWM_EXT.

BITS & NAME	TYPE & RESET	DESCRIPTION
[15:0] PWM_W	RW, unsigned 0x0000	Externally writable PWM duty cycle value for the motor's W phase, used when MOTION_MODE = PWM_EXT.

0x240: FEEDBACK.CONF_CH_A

Default: 0x00000000

Configuration for channel A of the feedback engine.

BITS & NAME	TYPE & RESET	DESCRIPTION
[27:24] SRC_SEL_A	RW 0x0	Angle source selection for the feedback engine's channel A. The input value is scaled with CPR_INV_A and then fed into the correction LUT module. 0: ABN_1 – Decoded single turn position from the ABN encoder connected to ENC_x pins. 1: ABN_1_FREE – "Multi turn" count of the ABN encoder connected to ENC_x pins. Select this option for velocity or position measurement only, not for phi_e calculation. 2: ABN_2 – Decoded single turn position from the ABN encoder connected to HALL_x pins (with alternate function). 3: ABN_2_FREE – "Multi turn" count of the ABN encoder connected to HALL_x pins (with alternate function). Select this option for velocity or position measurement only, not for phi_e calculation. 4: ANALOG_SENSOR – Decoded analog sensor position from Analog Hall or SinCos encoder connected to HALL_x pins (with alternate function). 5: HALL – Decoded position from the Hall sensor connected to HALL_x or ENC_x pins (ENC_x with alternate function). Select this option for phi_e calculation only, not for velocity or position measurement. 6: HALL_FREE – "Multi turn" count from the Hall sensor connected to HALL_x or ENC_x pins (ENC_x with alternate function). Select this option for velocity or position measurement only, not for phi_e calculation. 7: PHI_EXT_A – Use the value from FEEDBACK.PHI_EXT_A register. Select this option for encoders interfaced through the I/O Controller (e.g. SPI encoders) or when updating PHI_EXT_A externally through register access. 8: PHI_EXT_B – Use the value from FEEDBACK.PHI_EXT_B register. Select this option for encoders interfaced through the I/O Controller (e.g. SPI encoders) or when updating PHI_EXT_B externally through register access.
[23:0] CPR_INV_A	RW, unsigned 0x000000	Scaling factor for the 16-bit conversion operation. The converted output is obtained with converted_angle = (selected_source_angle × CPR_INV_A) >> 8. Set this value to a power of 2 equal or greater than 256 if selecting any of ABN_1_FREE, ABN_2_FREE or HALL_FREE options from SRC_SEL_A. For the rest of SRC_SEL_A options, set this value to 2^24/CPR of the selected feedback source.

0x241: FEEDBACK.CONF_CH_B

Default: 0x00000000

Configuration for channel B of the feedback engine.

BITS & NAME	TYPE & RESET	DESCRIPTION
[27:24] SRC_SEL_B	RW 0x0	Angle source selection for the feedback engine's channel B. The input value is scaled with CPR_INV_B and then fed into the correction LUT module.
	0: ABN_1	Decoded single turn position from the ABN encoder connected to ENC_x pins.
	1: ABN_1_FREE	"Multi turn" count of the ABN encoder connected to ENC_x pins. Select this option for velocity or position measurement only, not for phi_e calculation.
	2: ABN_2	Decoded single turn position from the ABN encoder connected to HALL_x pins (with alternate function).
	3: ABN_2_FREE	"Multi turn" count of the ABN encoder connected to HALL_x pins (with alternate function). Select this option for velocity or position measurement only, not for phi_e calculation.
	4: ANALOG_SENSOR	Decoded analog sensor position from Analog Hall or SinCos encoder connected to HALL_x pins (with alternate function).
	5: HALL	Decoded position from the Hall sensor connected to HALL_x or ENC_x pins (ENC_x with alternate function). Select this option for phi_e calculation only, not for velocity or position measurement.
	6: HALL_FREE	"Multi turn" count from the Hall sensor connected to HALL_x or ENC_x pins (ENC_x with alternate function). Select this option for velocity or position measurement only, not for phi_e calculation.
	7: PHI_EXT_A	Use the value from FEEDBACK.PHI_EXT_A register. Select this option for encoders interfaced through the I/O Controller (e.g. SPI encoders) or when updating PHI_EXT_A externally through register access.
	8: PHI_EXT_B	Use the value from FEEDBACK.PHI_EXT_B register. Select this option for encoders interfaced through the I/O Controller (e.g. SPI encoders) or when updating PHI_EXT_B externally through register access.
[23:0] CPR_INV_B	RW, unsigned 0x000000	Scaling factor for the 16-bit conversion operation. The converted output is obtained with converted_angle = (selected_source_angle × CPR_INV_B) >> 8. Set this value to a power of 2 equal or greater than 256 if selecting any of ABN_1_FREE, ABN_2_FREE or HALL_FREE options from SRC_SEL_B. For the rest of SRC_SEL_B options, set this value to 2^24/counts_per_rotation of the selected feedback source.

0x242: FEEDBACK.PHI_E_OFFSET

Default: 0x00000000

Offset for the feedback engine's calculated phi_e angle, applied after multiplying the angle of the selected channel by PHI_E_MUL_FACTOR. Use this offset to align the feedback engine's calculated phi_e with the internally generated phi_e.

BITS & NAME	TYPE & RESET	DESCRIPTION
[15:0] PHI_E_OFFSET	RW, unsigned 0x0000	The bitfield PHI_E_OFFSET holds the phi_e offset value that is added to the phi_e.

0x243: FEEDBACK.LUT

Default: 0x00000000

Configuration for the correction LUT module.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:24] RDATA	R, signed 0x00	Holds the read back value from a single entry of the LUT. The value in this field is updated after a new ADDR value is written.
[23:16] ADDR	RW, unsigned 0x00	Set the LUT address for writing/reading values to/from the lookup table. Since writing LUT values through the LUT_WDATA register is done four entries at a time, the address should be given in increments of four when populating the LUT, for example by setting ADDR = 0, 4, 8, 12, and so on. Reading LUT values to the RDATA field is done one entry at a time, so an ADDR value must be set for each entry of the LUT.
[14:12] LOOKUP_B_GAIN	RW 0x0	Gain value for the channel B of the correction LUT module.
	0: GAIN1 1: GAIN2 2: GAIN4 3: GAIN8 4: GAIN16	1x gain. 2x gain. 4x gain. 8x gain. 16x gain.
[10:8] LOOKUP_A_GAIN	RW 0x0	Gain value for the channel A of the correction LUT module.
	0: GAIN1 1: GAIN2 2: GAIN4 3: GAIN8 4: GAIN16	1x gain. 2x gain. 4x gain. 8x gain. 16x gain.
[2] LOOKUP_B_EN	RW 0x0	Set this bit to enable the channel B of the correction LUT module. Only one of channel A or B can be enabled if SPLIT_MODE_EN is not set.
[1] LOOKUP_A_EN	RW 0x0	Set this bit to enable the channel A of the correction LUT module. Only one of channel A or B can be enabled if SPLIT_MODE_EN is not set.
[0] SPLIT_MODE_EN	RW 0x0	Enables the split mode of the correction LUT. Both channel A and B should be enabled if the split mode is used.

0x244: FEEDBACK.VELOCITY_FRQ_CONF

Default: 0x00000000

Configuration for the frequency velocity meter.

BITS & NAME	TYPE & RESET	DESCRIPTION
[23:8] VELOCITY_SCALING	RW, unsigned 0x0000	Set the frequency velocity meter scaling factor, used so that its output and the that of the period velocity meter have the same scale, thus allowing to switch between them even during operation. Set VELOCITY_SCALING = 2^24 / 6E7 × velocity_update_rate. Refer to VELOCITY_SAMPLING field for a way to calculate velocity_update_rate.
[7:1] VELOCITY_SAMPLING	RW, unsigned 0x00	Set the velocity controller update rate and thus the frequency velocity meter sampling frequency, calculated as velocity_update_rate = pwm_freq / (VELOCITY_SAMPLING+1). Note that this field should be configured even if the period velocity meter is used instead of the frequency velocity meter, as the update rate of the velocity controller as a whole is configured through this field.
[0] VELOCITY_SYNC_SRC	RW 0x0	Select which trigger (relative to a single PWM cycle) to use as a sync signal for the velocity meters.
	0: PWM_Z 1: PWM_C	Use PWM_Z, the start of the PWM period. Use PWM_C, the center of the PWM period.

0x245: FEEDBACK.VELOCITY_PER_CONF

Default: 0xFFF00001

Configuration for the period velocity meter.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:16] POS_DEV_TIMER	RW, unsigned 0xFFF0	Set the timeout that the minimum angle change set in POS_DEV_MIN has to follow for the period velocity measurement to update. The timeout is calculated as POS_DEV_TIMER / f_CLKSYS. The default value results in a timeout of 1.092ms.
[14:0] POS_DEV_MIN	RW, unsigned 0x0001	Only update the period velocity measurement when the angle of the selected channel changes exceeds this value. The change must also happen within the time determined by POS_DEV_TIMER for the update to take place.

0x246: FEEDBACK.VELOCITY_PER_FILTER

Default: 0x00000000

Configuration for the moving average filter of the period velocity meter.

BITS & NAME	TYPE & RESET	DESCRIPTION
[2:0] FILTER_WIDTH	RW, unsigned 0x0	Set the moving average filter window width for the period velocity measurement. A wider window results in more smoothing, but also a more delayed output.

0x247: FEEDBACK.PHI_CONVERTED

Default: 0x00000000

Holds the feedback data for channel A and B after conversion/scaling to 16-bits according to configuration in CONF_CH_A and CONF_CH_B registers. This data is the input for the correction LUT module.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:16] PHI_CONVERTED_B	R, unsigned 0x0000	Holds the feedback data for channel B after conversion/scaling to 16-bits according to configuration in CONF_CH_B. This data is the input for the correction LUT module.
[15:0] PHI_CONVERTED_A	R, unsigned 0x0000	Holds the feedback data for channel A after conversion/scaling to 16-bits according to configuration in CONF_CH_A. This data is the input for the correction LUT.

0x248: FEEDBACK.CH_A

Default: 0x00000000

Holds the feedback data for channel A after the correction LUT module according to configuration in LUT register. This register also contains the data after the phi extrapolator module (for either channel A or B) according to configuration in PHI_E_SRC and PHI_DIFF_LIMIT fields.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:16] PHI_EXTRAPOLATED_AB	R, unsigned 0x0000	Holds the data after the phi extrapolator module (for either channel A or B) according to configuration in PHI_E_SRC and PHI_DIFF_LIMIT fields.
[15:0] PHI_LOOKUP_A	R, unsigned 0x0000	Holds the feedback data for channel A after the correction LUT module according to configuration in LUT register. Note that when LOOKUP_A_EN is not set, data from PHI_CONVERTED_A will remain unchanged.

0x249: FEEDBACK.CH_B

Default: 0x00000000

Holds the feedback data for channel B after the correction LUT module according to configuration in LUT register.

BITS & NAME	TYPE & RESET	DESCRIPTION
[15:0] PHI_LOOKUP_B	R, unsigned 0x0000	Holds the feedback data for channel B after the correction LUT module according to configuration in the LUT register. Note that when LOOKUP_B_EN is not set, data from PHI_CONVERTED_B will remain unchanged.

0x24A: FEEDBACK.VELOCITY_FRQ

Default: 0x00000000

Holds the output of the frequency velocity meter.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:0] VELOCITY_FRQ	R, signed 0x00000000	Holds the output of the frequency velocity meter.

0x24B: FEEDBACK.VELOCITY_PER

Default: 0x00000000

Holds the output of the period velocity meter.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:0] VELOCITY_PER	R, signed 0x00000000	Holds the output of the period velocity meter.

0x24C: FEEDBACK.LUT_WDATA

Default: 0x00000000

Holds the data to be written to the correction LUT.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:0] WDATA	RW, unsigned 0x00000000	Write a 32-bit value composed of four bytes. The LUT automatically writes bits [7:0] to the table entry pointed by ADDR field, bits [15:8] to the table entry at (ADDR+1), and so forth.

0x24D: FEEDBACK.PHI_EXT_A

Default: 0x00000000

Externally writable feedback engine input value, forwarded to the 16-bit conversion/scaling module for channel A or B.

BITS & NAME	TYPE & RESET	DESCRIPTION
[23:0] PHI_EXT_A	RW, unsigned 0x000000	Externally writable feedback engine input value, forwarded to the 16-bit conversion/scaling module for channel A or B when their corresponding source selection is configured with the PHI_EXT_A option. This register may also be written by the I/O controller when using an encoder executable, e.g. by activating the SPI encoder (ROM code feature).

0x24E: FEEDBACK.PHI_EXT_B

Default: 0x00000000

Externally writable feedback engine input value, forwarded to the 16-bit conversion/scaling module for channel A or B.

BITS & NAME	TYPE & RESET	DESCRIPTION
[23:0] PHI_EXT_B	RW, unsigned 0x000000	Externally writable feedback engine input value. Used as input for the 16-bit conversion/scaling module for channel A or B when their corresponding source selection is configured with the PHI_EXT_B option. This register may also be written by the I/O controller when using an encoder executable, e.g. by activating the SPI encoder (ROM code feature).

0x24F: FEEDBACK.VELOCITY_EXT

Default: 0x00000000

Externally writable velocity value. Used as input to the velocity controller if VELOCITY_SELECTION = VELOCITY_EXT.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:0] VELOCITY_EXT	RW, signed 0x00000000	Externally writable velocity value. Used as input to the velocity controller when VELOCITY_SELECTION = VELOCITY_EXT.

0x250: FEEDBACK.OUTPUT_CONF

Default: 0x00000001

Configuration for the different output channels of the feedback engine based on selected angle sources and configured A and B channels.

BITS & NAME	TYPE & RESET	DESCRIPTION
[23:22] VELOCITY_SELECTION	RW 0x0	Select the velocity source used for the velocity controller of the FOC module.
	0: VELOCITY_FRQ	Frequency velocity meter. Functions well for most applications. Best at medium to high velocities.
	1: VELOCITY_PER	Period velocity meter. This meter is better for slow speed applications. It works better for regulating 0 speed as well.
	2: VELOCITY_EXT	Measured speed is manually written into VELOCITY_EXT if an external controller handled the speed measurement or by using the I/O controller.
[21] VELOCITY_SRC	RW 0x0	Select the LUT channel to use as input for both frequency and period velocity meters.
	0: LOOKUP_A	LUT channel A.
	1: LOOKUP_B	LUT channel B.
[20] POSITION_SRC	RW 0x0	Select the LUT channel to use as input for the position multiturn decoder.
	0: LOOKUP_A	LUT channel A.
	1: LOOKUP_B	LUT channel B.
[17:16] PHI_E_SRC	RW 0x0	Select the source for the feedback engine's phi_e calculation module.
	0: LOOKUP_A	LUT channel A.
	1: LOOKUP_B	LUT channel B.
	2: EXTRAPOLATOR_A	Extrapolated angle channel A.
	3: EXTRAPOLATOR_B	Extrapolated angle channel B.
[7:0] PHI_E_MUL_FACTOR	RW, unsigned 0x01	Set the multiplication factor between the selected feedback angle and the phi_e angle. Set a value equal to the motor number of pole pairs for feedback sources that output the mechanical angle, for example ABN encoders, SinCos encoders, etc. Set a value of 1 for feedback sources that output the electrical angle, for example digital or analog Hall sensors.

0x251: FEEDBACK.PHI_E

Default: 0x00000000

Holds the feedback engine's calculated phi_e for closed loop operation. This angle is used as input to the FOC module when MCC_CONFIG.MOTOR_MOTION.RAMP_USE_PHI_E is not set.

BITS & NAME	TYPE & RESET	DESCRIPTION
[15:0] PHI_E_FOC	R, unsigned 0x0000	Holds the feedback engine's calculated phi_e for closed loop operation. Calculated from the angle of the channel selected in PHI_E_SRC by multiplying it with PHI_E_MUL_FACTOR and applying PHI_E_OFFSET. This angle is used as input to the FOC module when MCC_CONFIG.MOTOR_MOTION.RAMP_USE_PHI_E is not set.

0x252: FEEDBACK.PHI_DIFF_LIMIT

Default: 0x00004000

This register represents the maximum allowable difference between the input and the output of the extrapolator module, that is, the difference between the actual feedback angle and the predicted extrapolated angle.

BITS & NAME	TYPE & RESET	DESCRIPTION
[15:0] PHI_DIFF_LIMIT	RW, unsigned 0x4000	Holds the maximum allowable difference between the input and the output of the extrapolator module, that is, the difference between the actual feedback angle and the predicted extrapolated angle.

0x280: ABN.CONFIG

Default: 0xFFFFFF02

Configuration for the ABN encoder connected to ENC_x pins.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:8] CPR	RW, unsigned 0xFFFFFF	Holds the maximum value after which the COUNT has to be reset. Set to the encoder's CPR-1.
[3] INV_DIR	RW 0x0	Inverse the direction of the count.
[1] CLN	RW 0x1	Enable the transfer of COUNT_N_WRITE into COUNT upon N pulse detection.
	0: OFF	off.
	1: ON	on.
[0] COMBINED_N	RW 0x0	Use AND of all three signals A, B, N to determine the N pulse.
	0: ONLY_N	Ignore A and B, just use N pulse as Null signal.
	1: ALL	Use all three signals as Null signal.

0x281: ABN.COUNT

Default: 0x00000000

Holds the raw ABN decoder count.

BITS & NAME	TYPE & RESET	DESCRIPTION
[23:0] COUNT	RW, unsigned 0x000000	Holds the raw ABN decoder count.

0x282: ABN.COUNT_N_CAPTURE

Default: 0x00000000

Holds the ABN decoder count latched upon N pulse detection. The N_EVENT bit can be monitored to see if a N pulse event emerged. If CONFIG.CLN bit is set, then the latched value will simply match the value set in COUNT_N_WRITE.

BITS & NAME	TYPE & RESET	DESCRIPTION
[23:0] COUNT_N_CAPTURE	R, unsigned 0x000000	Holds the ABN decoder count latched upon N pulse detection. The N_EVENT bit can be monitored to see if a N pulse event emerged. If CONFIG.CLN bit is set, then the latched value will simply match the value set in COUNT_N_WRITE field.

0x283: ABN.COUNT_N_WRITE

Default: 0x00000000

Set the value that will be written to COUNT upon N pulse detection. This only valid if CONFIG.CLN bit is set.

BITS & NAME	TYPE & RESET	DESCRIPTION
[23:0] COUNT_N_WRITE	RW, unsigned 0x000000	Set the value that will be written to COUNT upon N pulse detection. The value is written only when the CONFIG.CLN bit is set.

0x284: ABN.EVENTS

Default: 0x00000000

Event bits of the ABN decoder.

BITS & NAME	TYPE & RESET	DESCRIPTION
[1] N_EVENT	RW, W1C 0x0	N pulse event emerged. Write a '1' to clear the bit.
[0] INVALID_SIGNAL_EVENT	RW, W1C 0x0	This bit is set if A and B change at the same time. Write a '1' to clear the bit.

0x2C0: ABN2.CONFIG

Default: 0xFFFFFF02

Configuration for the ABN encoder connected to HALL_x pins (with alternate functions).

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:8] CPR	RW, unsigned 0xFFFFFF	Holds the maximum count per revolution at which the decoder count has to be reset. Set this value to CPR-1.
[3] INV_DIR	RW 0x0	Inverse the direction of the count.
[1] CLN	RW 0x1	Enable the transfer of COUNT_N_WRITE into COUNT upon N pulse detection.
	0: OFF	off.
	1: ON	on.
[0] COMBINED_N	RW 0x0	Use AND of all three signals A, B, N to determine the N pulse.
	0: ONLY_N	Ignore A and B, just use N pulse as Null signal.
	1: ALL	Use all three signals as Null signal.

0x2C1: ABN2.COUNT

Default: 0x00000000

Holds the raw ABN2 decoder count.

BITS & NAME	TYPE & RESET	DESCRIPTION
[23:0] COUNT	RW, unsigned 0x000000	Holds the raw ABN2 decoder count.

0x2C2: ABN2.COUNT_N_CAPTURE

Default: 0x00000000

Holds the ABN2 decoder count latched upon N pulse detection. The N_EVENT bit can be monitored to see if a N pulse event emerged. If CONFIG.CLN bit is set, then the latched value will simply match the value set in COUNT_N_WRITE.

BITS & NAME	TYPE & RESET	DESCRIPTION
[23:0] COUNT_N_CAPTURE	R, unsigned 0x000000	Holds the ABN2 decoder count latched upon N pulse detection. The N_EVENT bit can be monitored to see if a N pulse event emerged. If CONFIG.CLN bit is set, then the latched value will simply match the value set in COUNT_N_WRITE field.

0x2C3: ABN2.COUNT_N_WRITE

Default: 0x00000000

Set the value that will be written to COUNT upon N pulse detection. This only valid if CONFIG.CLN bit is set.

BITS & NAME	TYPE & RESET	DESCRIPTION
[23:0] COUNT_N_WRITE	RW, unsigned 0x000000	Set the value that will be written to COUNT upon N pulse detection. The value is written only when the CONFIG.CLN bit is set.

0x2C4: ABN2.EVENTS

Default: 0x00000000

Event bits for the ABN2 decoder.

BITS & NAME	TYPE & RESET	DESCRIPTION
[1] N_EVENT	RW, W1C 0x0	N pulse event emerged. Write a '1' to clear the bit.
[0] INVALID_SIGNAL_EVENT	RW, W1C 0x0	This bit is set if A and B change at the same time. Write a '1' to clear the bit.

0x300: HALL.MAP_CONFIG

Default: 0x00000000

Configurations for digital and analog hall signal assignment.

BITS & NAME	TYPE & RESET	DESCRIPTION
[6:4] ANA_HALL_MAP	RW 0x0	Alternative mapping for the U/X, V/N and W/Y analog sensor signals.
	0: ANA_UVW	Analog sensor U/X, V/N, W/Y from AIN_U, AIN_V, AIN_W resp.
	1: ANA_UWV	Analog sensor U/X, V/N, W/Y from AIN_U, AIN_W, AIN_V resp.
	2: ANA_VUW	Analog sensor U/X, V/N, W/Y from AIN_V, AIN_U, AIN_W resp.
	3: ANA_WUV	Analog sensor U/X, V/N, W/Y from AIN_W, AIN_U, AIN_V resp.
	4: ANA_VWU	Analog sensor U/X, V/N, W/Y from AIN_V, AIN_W, AIN_U resp.
	5: ANA_WVU	Analog sensor U/X, V/N, W/Y from AIN_W, AIN_V, AIN_U resp.
[2:0] HALL_MAP	RW 0x0	Alternative mapping for the Hall UVW signals.
	0: UVW	Hall sensor U, V, W from PIN_HALL_U_RAW, PIN_HALL_V_RAW, PIN_HALL_W_RAW resp.
	1: UWV	Hall sensor U, V, W from PIN_HALL_U_RAW, PIN_HALL_W_RAW, PIN_HALL_V_RAW resp.
	2: VUW	Hall sensor U, V, W from PIN_HALL_V_RAW, PIN_HALL_U_RAW, PIN_HALL_W_RAW resp.
	3: WUV	Hall sensor U, V, W from PIN_HALL_W_RAW, PIN_HALL_U_RAW, PIN_HALL_V_RAW resp.
	4: VWU	Hall sensor U, V, W from PIN_HALL_V_RAW, PIN_HALL_W_RAW, PIN_HALL_U_RAW resp.
	5: WVU	Hall sensor U, V, W from PIN_HALL_W_RAW, PIN_HALL_V_RAW, PIN_HALL_U_RAW resp.

0x301: HALL.DIG_COUNT

Default: 0x00000000

Holds the digital Hall decoder count, that is, the current Hall sensor position.

BITS & NAME	TYPE & RESET	DESCRIPTION
[2:0] COUNT	R, unsigned 0x0	Holds the digital Hall decoder count, that is, the current Hall sensor position.

0x302: HALL.ANA_CONFIG

Default: 0x00000000

Configuration and status flags for the analog feedback sensors, including analog Hall and SinCos encoder.

BITS & NAME	TYPE & RESET	DESCRIPTION
[22] ADC_CLIPPED	R 0x0	Indicates that at least one of the analog sensor values was clipped during angle calculation.
[21] N_PULSE_OUT	R 0x0	Holds the value of the generated N pulse from the V/N analog sensor signal according to N_PULSE_POL and N_PULSE_THRESHOLD.
[20] USE_N_PULSE	RW 0x0	If activated, V/N serves as N channel for the SinCos encoder. N_PULSE_THRESHOLD must be set also to define the analog value that represents the N event.
[19] TWO_CYCLE_MODE_EN	RW 0x0	Set this bit for analog sensors that utilizes two cycles per revolution.
[18] N_PULSE_POL	RW 0x0	Select the polarity of the N pulse. This determines whether N_PULSE_OUT is set when the V/N signal value is greater than N_PULSE_THRESHOLD (non-inverted) or smaller than it (inverted).
	0: NON_INVERTED 1: INVERTED	Non-inverted, high active. Inverted, low active.
[17] N_PULSE_EDGE	RW 0x0	Select the edge of the N_PULSE_OUT bit that will trigger a latching of the current analog sensor angle if USE_N_PULSE is set.
	0: RISING 1: FALLING	Low to use rising edge of n_pulse. Falling edge of the N pulse.
[16] ANA_MODE	RW 0x0	Select the analog sensor mode. Use UVW mode for analog Hall sensors and XY mode for SinCos encoders.
	0: XY 1: UVW	XY mode. UVW mode.
[15:0] N_PULSE_THRESHOLD	RW, signed 0x0000	Write the threshold value for the V/N analog sensor signal to generate the N pulse in N_PULSE_OUT. Note that the threshold is checked against the V/N signal before offsetting and scaling.

0x303: HALL.ANA_UX_CONFIG

Default: 0x00000000

Set the offset and scale for the U/X analog sensor input.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:16] UX_SCALE	RW, signed 0x0000	Set the value that is used to scale the U/X analog sensor input after adding the offset.
[15:0] UX_OFFSET	RW, signed 0x0000	Set the value to be added to the U/X analog sensor input before scaling.

0x304: HALL.ANA_VN_CONFIG

Default: 0x00000000

Set the offset and scale for the V/N analog sensor input.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:16] VN_SCALE	RW, signed 0x0000	Set the value that is used to scale the V/N analog sensor input after adding the offset.
[15:0] VN_OFFSET	RW, signed 0x0000	Set the value to be added to the V/N analog sensor input before scaling.

0x305: HALL.ANA_WY_CONFIG

Default: 0x00000000

Set the offset and scale for the W/Y analog sensor input.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:16] WY_SCALE	RW, signed 0x0000	Set the value that is used to scale the W/Y analog sensor input after adding the offset.
[15:0] WY_OFFSET	RW, signed 0x0000	Set the value to be added to the W/Y analog sensor input before scaling.

0x306: HALL.ANA_UX_OUT

Default: 0x00000000

Holds the U/X analog sensor value after offsetting and scaling.

BITS & NAME	TYPE & RESET	DESCRIPTION
[15:0] UX_OUT	R, signed 0x0000	Holds the U/X analog sensor value after offsetting and scaling.

0x307: HALL.ANA_VN_OUT

Default: 0x00000000

Holds the V/N analog sensor value after offsetting and scaling.

BITS & NAME	TYPE & RESET	DESCRIPTION
[15:0] VN_OUT	R, signed 0x0000	Holds the V/N analog sensor value after offsetting and scaling.

0x308: HALL.ANA_WY_OUT

Default: 0x00000000

Holds the W/Y analog sensor value after offsetting and scaling.

BITS & NAME	TYPE & RESET	DESCRIPTION
[15:0] WY_OUT	R, signed 0x0000	Holds the W/Y analog sensor value after offsetting and scaling.

0x309: HALL.ANA_OUT

Default: 0x00000000

Holds the calculated analog sensor angle and the latched value on the optional N pulse detection.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:16] ANA_PHI	R, unsigned 0x0000	Holds the analog sensor angle obtained from the arc tangent calculation of the inputs.
[15:0] ANA_PHI_N_CAPTURE	R, unsigned 0x0000	Holds the analog sensor angle captured upon N pulse detection. Configure the N pulse detection in ANA_CONFIG register. The USE_N_PULSE bit must be set for the capture to occur.

0x30A: HALL.DIG_EVENTS

Default: 0x00000000

Event bits for the digital Hall decoder.

BITS & NAME	TYPE & RESET	DESCRIPTION
[0] INVALID_SIGNAL_EVENT	RW, W1C 0x0	This bit is set when the digital Hall signals are all zero or all one. Write a '1' to clear the bit.

0x340: UART.CONTROL

Default: 0x00040000

UART settings

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:24] RTMI_SAMPLING	RW, unsigned 0x00	Downsampling value to trigger RTMI block. RTMI trigger rate = f_PWM / (RTMI_SAMPLING+1)
[22] RTMI_EN	RW 0x0	Enables RTMI feature.
[21] RTMI_CRC_EN	RW 0x0	Defines if CRC evaluation resp. generation is enabled for the RTMI read/write datagrams and regular write response datagrams.
[20] NORMAL_CRC_EN	RW 0x0	Defines if CRC evaluation resp. generation is enabled for the normal read datagrams and write requests.
[19] RX_FILTER_EN	RW 0x0	If input filter is enabled, received UART_RXD data is sampled three times at the expected center of each bit period. The final bit value is determined using a majority voting mechanism, where two out of the three defines the final bit value. This helps to ensure reliable data reception and mitigate the effects of noise or timing variations.
[18] AUTOBAUD_EN	RW 0x1	Enable automatic baud rate detection, ignoring MANTISSA_LIMIT settings.
[16] TIMEOUT_PRE_DIVIDER_EN	RW 0x0	Enable system clock divided by 4096 pre-divider for the timeout counter.
[12:0] MANTISSA_LIMIT	RW, unsigned 0x0000	Defines the baud rate = system_clock / MANTISSA_LIMIT. The minimum value is 5.

0x341: UART.TIMEOUT

Default: 0x0000FFFF

UART timeout limit and timeout count

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:16] COUNTER	R, unsigned 0x0000	Starts counting after stop bit received while RX line is idle. Counter always counts upwards until timeout limit is reached using the system clock or the system clock divided by 4096 (refer to TIMEOUT_PRE_DIVIDER_EN).
[15:0] LIMIT	RW, unsigned 0xFFFF	Timeout counter upper limit. Timeout flag will be set when counter reaches this limit while RX line is idle. For counting either the system clock will be used or the system clock divided by 4096 depending on the TIMEOUT_PRE_DIVIDER_EN field. In case this upper limit is set to zero, the timeout counter will be deactivated.

0x342: UART.STATUS

Default: 0x00000000

UART status bits

BITS & NAME	TYPE & RESET	DESCRIPTION
[4] AUTOBAUD_VALID	RW 0x0	Is active when autobaud is successfully detected after receiving the data.
[2] TX_ACTIVE	R 0x0	Is active if UART_TXD is sending data.
[0] RX_ACTIVE	R 0x0	Is active if UART_RXD is receiving data.

0x343: UART.EVENTS

Default: 0x00000000

UART failure events. All events are cleared by writing a ‘1’ to the particular event bit.

BITS & NAME	TYPE & RESET	DESCRIPTION
[7] INVALID_CRC_EVENT	RW, W1C 0x0	Is activated if CRC is enabled and the calculated CRC differs from the one attached to the datagram. In this case datagram is also discarded.
[6] INVALID_STOP_BIT_EVENT	RW, W1C 0x0	Is activated if the stop bit of any datagram bytes is corrupted. Write a '1' to clear the bit.
[5] INVALID_START_BIT_EVENT	RW, W1C 0x0	Is activated if the start bit of any datagram bytes is corrupted. Write a '1' to clear the bit.
[4] RX_IDLE_TIMEOUT_EVENT	RW, W1C 0x0	Is activated if the UART_RXD pin is idle between the datagram bytes for more than specified timeout period. Write a '1' to clear the bit.
[3] RTMI_INTERRUPT_EVENT	RW, W1C 0x0	This flag indicates if the transmission of the RTMI block is interrupted in case a new PWM cycles starts again (because PWM cycle is expired) while still processing an output stream. Write a '1' to clear.
[2] NOISY_DATA_EVENT	RW, W1C 0x0	Is activated all the three sampling points are not equal (either 000_b or 111_b) if UART filtering is active. UART datagram is processed anyway as the final bit value is determined using a majority voting mechanism during UART filtering. Write a '1' to clear the bit.
[0] UART_SYNC_FAIL_EVENT	RW, W1C 0x0	Is activated if the UART module detects any disruption of bits or the module have not received bits properly (in case autobaud is enabled) or when the datagram is sent with the wrong sync bits (in the byte 0) and autobaud is disabled. Write a '1' to clear the bit.

0x344: UART.RTMI_CH_0

Default: 0x00000000

RTMI channel 0 settings.

BITS & NAME	TYPE & RESET	DESCRIPTION
[16] CH_0_EN	RW 0x0	Enables channel 0 data transmission via UART_TXD if RTMI is enabled.
[9:0] CH_0_ADDR	RW, unsigned 0x000	The RTMI channel 0 address mapped to the register bank address.

0x345: UART.RTMI_CH_1

Default: 0x00000000

RTMI channel 1 settings.

BITS & NAME	TYPE & RESET	DESCRIPTION
[16] CH_1_EN	RW 0x0	Enables channel 1 data transmission via UART_TXD if RTMI is enabled.
[9:0] CH_1_ADDR	RW, unsigned 0x000	The RTMI channel 1 address mapped to the register bank address.

0x346: UART.RTMI_CH_2

Default: 0x00000000

RTMI channel 2 settings.

BITS & NAME	TYPE & RESET	DESCRIPTION
[16] CH_2_EN	RW 0x0	Enables channel 2 data transmission via UART_TXD if RTMI is enabled.
[9:0] CH_2_ADDR	RW, unsigned 0x000	The RTMI channel 2 address mapped to the register bank address.

0x347: UART.RTMI_CH_3

Default: 0x00000000

RTMI channel 3 settings.

BITS & NAME	TYPE & RESET	DESCRIPTION
[16] CH_3_EN	RW 0x0	Enables channel 3 data transmission via UART_TXD if RTMI is enabled.
[9:0] CH_3_ADDR	RW, unsigned 0x000	The RTMI channel 3 address mapped to the register bank address.

0x348: UART.RTMI_CH_4

Default: 0x00000000

RTMI channel 4 settings.

BITS & NAME	TYPE & RESET	DESCRIPTION
[16] CH_4_EN	RW 0x0	Enables channel 4 data transmission via UART_TXD if RTMI is enabled.
[9:0] CH_4_ADDR	RW, unsigned 0x000	The RTMI channel 4 address mapped to the register bank address.

0x349: UART.RTMI_CH_5

Default: 0x00000000

RTMI channel 5 settings.

BITS & NAME	TYPE & RESET	DESCRIPTION
[16] CH_5_EN	RW 0x0	Enables channel 5 data transmission via UART_TXD if RTMI is enabled.
[9:0] CH_5_ADDR	RW, unsigned 0x000	The RTMI channel 5 address mapped to the register bank address.

0x34A: UART.RTMI_CH_6

Default: 0x00000000

RTMI channel 6 settings.

BITS & NAME	TYPE & RESET	DESCRIPTION
[16] CH_6_EN	RW 0x0	Enables channel 6 data transmission via UART_TXD if RTMI is enabled.
[9:0] CH_6_ADDR	RW, unsigned 0x000	The RTMI channel 6 address mapped to the register bank address.

0x34B: UART.RTMI_CH_7

Default: 0x00010000

RTMI channel 7 settings. Channel 7 must be also enabled to get responses for an UART write request.

BITS & NAME	TYPE & RESET	DESCRIPTION
[16] CH_7_EN	RW 0x1	Enables channel 7 data transmission via UART_TXD if RTMI is enabled. By default, this channel is enabled to send out replies for the normal write request datagrams when RTMI is disabled.
[9:0] CH_7_ADDR	RW, unsigned 0x000	The RTMI channel 7 address mapped to the register bank address if RTMI is enabled. When RTMI is disabled, RTMI channel 7 address is mapped to previous write address to fetch previous written value to the register bank.

0x380: IO_CONTROLLER.CONTROL

Default: 0x00000000

Control signals for the I/O Controller.

BITS & NAME	TYPE & RESET	DESCRIPTION
[0] RESET	RW 0x0	Set this bit to reset the I/O Controller. Activating during GDRV_ON state, will only reset to SYS_READY state. Otherwise if activated during SYS_READY state, I/O controller will be reset and the chip start-up process (incl. ADC reconfiguration) is executed; finishes also LowPower mode.

0x381: IO_CONTROLLER.COMMAND

Default: 0x00000000

Command register of the I/O Controller.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:0] COMMAND	RW, unsigned 0x00000000	Command register of the I/O Controller.

0x382: IO_CONTROLLER.RESPONSE_0

Default: 0x00000000

Response 0 register of the I/O Controller.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:0] RESPONSE_0	RW, unsigned 0x00000000	Response 0 register of the I/O Controller. The response is dependent on the requested command. Serves as handshake register for all ROM code features. In case RESPONSE_0 equals COMMAND register value, requested ROM code is finished for one-time requests or indicates valid response data (RESPONSE_1...3) for repeated requests.

0x383: IO_CONTROLLER.RESPONSE_1

Default: 0x00000000

Response 1 register of the I/O Controller.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:0] RESPONSE_1	RW, unsigned 0x00000000	Response 1 register of the I/O Controller. The response is dependent on the requested command.

0x384: IO_CONTROLLER.RESPONSE_2

Default: 0x00000000

Response 2 register of the I/O Controller.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:0] RESPONSE_2	RW, unsigned 0x00000000	Response 2 register of the I/O Controller. The response is dependent on the requested command.

0x385: IO_CONTROLLER.RESPONSE_3

Default: 0x00000000

Response 3 register of the I/O Controller.

BITS & NAME	TYPE & RESET	DESCRIPTION
[31:0] RESPONSE_3	RW, unsigned 0x00000000	Response 3 register of the I/O Controller. The response is dependent on the requested command.

Ordering Information

PART NUMBER	TEMP RANGE	PIN-PACKAGE
TMC6460ATU+*	-40°C to +125°C	38 TQFN 5mm x 7mm
TMC6460ATU+T*	-40°C to +125°C	38 TQFN 5mm x 7mm

*Potential future product.

+ Denotes a lead(Pb)-free/RoHS-compliant package.

T Denotes tape-and-reel.

Revision History

REVISION NUMBER	REVISION DATE	DESCRIPTION	PAGES CHANGED
PrA	03/26	Preliminary release	—

Information furnished by Analog Devices is believed to be accurate and reliable. However, no responsibility is assumed by Analog Devices for its use, nor for any infringements of patents or other rights of third parties that may result from its use. Specifications subject to change without notice. No license is granted by implication or otherwise under any patent or patent rights of Analog Devices. Trademarks and registered trademarks are the property of their respective owners. All Analog Devices products contained herein are subject to release and availability.