

Motor Control Toolkit for SPC56ELxx and SPC560Pxx platforms

Introduction

This manual describes the SPC5 Motor Control ToolKit (SPC5-MCTK) designed for and to be used with SPC5 Automotive MCUs microcontrollers. The software library implements the Field Oriented Control (FOC) drive of 3-phase Permanent Magnet Synchronous Motors (PMSM), both Surface Mounted (SM-PMSM) and Internal (I-PMSM). The library exploit a sensed and sensorless techniques that, in conjunction with motor characteristic, is able to extend the range of allowed speed to zero.

The SPC5 family of 32-bit automotive microcontrollers build on Power Architecture® technology offering scalable solution for single- and multi-core MCUs features, addressing from cost-sensitive to highly-advanced automotive applications thanks to hardware and software compatibility. The portfolio is intended for body, powertrain, chassis and safety applications.

These microcontrollers combine high performance with first-class peripherals that make them suitable for performing three-phase motors FOC in automotive application.

The PMSM FOC library can be used to quickly evaluate SPC5 microcontrollers and complete ST application platforms, and to save time when developing Motor Control algorithms to be run on SPC5 microcontrollers. It is written in C language, and implements the core Motor Control algorithms as well as sensor reading/decoding algorithms and a sensorless algorithm for rotor position reconstruction.

The library can be customized to suit user application parameters (motor, sensors, power stage, control stage, pin-out assignment) and provides a ready-to-use Application Programming Interface (API). A user project has been implemented to demonstrate how to interact with the Motor Control API.

This project provides CAN and UART User Interface, thus representing a convenient realtime fine-tuning and remote control tool. A PC live monitor and the SPC5Studio motor control component, allows a complete and easy customization of the PMSM FOC library. In a very short time the user can run a PMSM motor. A set of ready-to-use examples are provided to explain the usage of the motor control API and its most common features.

Microcontrollers supported:

- SPC560P (Pictus),
- SPC56EL (Leopard)

Contents

1	SPC5 Motor Control Tool Kit	1
2	SPC5 PMSM MC Application	3
2.1	MCInterface class private methods	3
2.1.1	Function Documentation	3
2.1.1.1	MCI_NewObject	3
2.1.1.2	MCI_Init	4
2.1.1.3	MCI_ExecBufferedCommands	4
2.2	MCInterface class private types	6
2.2.1	Typedef Documentation	6
2.2.1.1	Params_t	6
2.3	MCTuning class exported types	7
2.3.1	Typedef Documentation	8
2.3.1.1	CMCT	8
2.3.1.2	MCTuningParams_t	8
2.3.1.3	COL	8
2.3.1.4	CPI	8
2.3.1.5	CPID_PI	8
2.3.1.6	CPWMC	8
2.3.1.7	CRUC	8
2.3.1.8	CSPD	9
2.3.1.9	CSTO_SPD	9
2.3.1.10	CSTO_CR_SPD	9
2.3.1.11	CSTC	9
2.3.1.12	CTSM	9
2.3.1.13	CTSNS	9
2.3.1.14	CVBS	9
2.3.1.15	CDOUT	9
2.3.1.16	CMPM	9
2.3.1.17	CFW	10
2.3.1.18	CFF	10

2.3.1.19	CHFI_FP	10
2.4	MCTuning class private methods	11
2.4.1	Function Documentation	11
2.4.1.1	MCT_NewObject	11
2.4.1.2	MCT_Init	11
2.5	MCTuning class private types	12
2.5.1	Typedef Documentation	12
2.5.1.1	Params_t	12
2.6	MC exported types	13
2.7	Motor control protocol registers	14
2.8	MC exported methods	15
2.8.1	Function Documentation	15
2.8.1.1	GetMCI	15
2.8.1.2	GetMCT	16
2.8.1.3	MC_RequestRegularConv	16
2.8.1.4	MC_GetRegularConv	16
2.8.1.5	MC_GetMultipleRegularConv	17
2.8.1.6	MC-RegularConvState	17
2.8.1.7	MC_SetDAC	17
2.8.1.8	MC_SetUserDAC	18
2.9	MCInterface class exported types	19
2.9.1	Typedef Documentation	19
2.9.1.1	CMCI	19
2.9.1.2	MCInterfaceParams_t	19
2.9.2	Enumeration Type Documentation	19
2.9.2.1	CommandState_t	19
2.9.2.2	UserCommands_t	20
2.10	MCInterface class exported methods	21
2.10.1	Function Documentation	23
2.10.1.1	MCI_ExecSpeedRamp	23
2.10.1.2	MCI_ExecTorqueRamp	23
2.10.1.3	MCI_SetCurrentReferences	24
2.10.1.4	MCI_StartMotor	24
2.10.1.5	MCI_StopMotor	25
2.10.1.6	MCI_FaultAcknowledged	25
2.10.1.7	MCI_EncoderAlign	26
2.10.1.8	MCI_IsCommandAcknowledged	27
2.10.1.9	MCI_GetSTMState	27
2.10.1.10	MCI_GetOccurredFaults	28
2.10.1.11	MCI_GetCurrentFaults	28

2.10.1.12 MCI_GetTorqueRef	29
2.10.1.13 MCI_GetTorque	29
2.10.1.14 MCI_GetControlMode	29
2.10.1.15 MCI_GetImposedMotorDirection	30
2.10.1.16 MCI_GetLastRampFinalSpeed	30
2.10.1.17 MCI_RampCompleted	30
2.10.1.18 MCI_GetSpdSensorReliability	31
2.10.1.19 MCI_GetAvrgMecSpeed01Hz	32
2.10.1.20 MCI_GetMecSpeedRef01Hz	32
2.10.1.21 MCI_Getlab	33
2.10.1.22 MCI_Getlalphabet	33
2.10.1.23 MCI_Getlqd	33
2.10.1.24 MCI_GetlqdHF	34
2.10.1.25 MCI_Getlqdref	34
2.10.1.26 MCI_GetVqd	34
2.10.1.27 MCI_GetValphabet	35
2.10.1.28 MCI_GetElAngledpp	35
2.10.1.29 MCI_GetTeref	35
2.10.1.30 MCI_GetPhaseCurrentAmplitude	35
2.10.1.31 MCI_GetPhaseVoltageAmplitude	36
2.10.1.32 MCI_SetIdref	36
2.10.1.33 MCI_Clear_Iqdref	37
2.11 MCTasks exported types	38
2.12 MCTasks exported methods	39
2.12.1 Function Documentation	39
2.12.1.1 MCboot	39
2.12.1.2 MC_Scheduler	40
2.12.1.3 TSK_SafetyTask	40
2.12.1.4 TSK_HighFrequencyTask	40
2.12.1.5 TSK_DualDriveFIFOUpdate	41
2.12.1.6 TSK_HardwareFaultTask	42
2.13 MCTuning class exported methods	43
2.13.1 Function Documentation	45
2.13.1.1 MCT_GetFluxWeakeningCtrl	45
2.13.1.2 MCT_GetFeedForwardCtrl	45
2.13.1.3 MCT_GetHFIctrl	46
2.13.1.4 MCT_GetSpeedLoopPID	46
2.13.1.5 MCT_GetIqLoopPID	47
2.13.1.6 MCT_GetIdLoopPID	47
2.13.1.7 MCT_GetFluxWeakeningLoopPID	48

2.13.1.8 MCT_GetPWMnCurrFdbk	49
2.13.1.9 MCT_GetRevupCtrl	49
2.13.1.10 MCT_GetSpeednPosSensorMain	50
2.13.1.11 MCT_GetSpeednPosSensorAuxiliary	50
2.13.1.12 MCT_GetSpeednPosSensorVirtual	51
2.13.1.13 MCT_GetSpeednTorqueController	51
2.13.1.14 MCT_GetStateMachine	52
2.13.1.15 MCT_GetTemperatureSensor	52
2.13.1.16 MCT_GetBusVoltageSensor	53
2.13.1.17 MCT_GetBrakeResistor	53
2.13.1.18 MCT_GetNTCRelay	54
2.13.1.19 MCT_GetMotorPowerMeasurement	54
2.14 FluxWeakeningCtrl class exported methods	56
2.14.1 Function Documentation	56
2.14.1.1 FW_SetVref	56
2.14.1.2 FW_GetVref	57
2.14.1.3 FW_GetAvVAmplitude	57
2.14.1.4 FW_GetAvVPercentage	57
2.15 FeedForwardCtrl class exported methods	59
2.15.1 Function Documentation	59
2.15.1.1 FF_SetFFConstants	59
2.15.1.2 FF_GetFFConstants	59
2.15.1.3 FF_GetVqdff	60
2.15.1.4 FF_GetVqdAvPlout	60
2.16 HFIctrl class exported methods	61
2.16.1 Function Documentation	61
2.16.1.1 HFI_FP_GetRotorAngleLock	61
2.16.1.2 HFI_FP_GetSaturationDifference	61
2.16.1.3 HFI_FP_GetCurrent	62
2.16.1.4 HFI_FP_GetPITrack	62
2.16.1.5 HFI_FP_SetMinSaturationDifference	62
2.17 OLCtrl class exported methods	63
2.17.1 Function Documentation	63
2.17.1.1 OL_UpdateVoltage	63
2.18 PI regulator class exported	64
2.18.1 Detailed Description	64
2.18.2 Function Documentation	64
2.18.2.1 PI_SetKP	64
2.18.2.2 PI_SetKI	65
2.18.2.3 PI_GetKP	65

2.18.2.4	PI_GetKI	65
2.18.2.5	PI_GetDefaultKP	65
2.18.2.6	PI_GetDefaultKI	66
2.18.2.7	PI_SetIntegralTerm	66
2.19	PID class exported methods	67
2.19.1	Function Documentation	67
2.19.1.1	PID_SetPrevError	67
2.19.1.2	PID_SetKD	67
2.19.1.3	PID_GetKD	68
2.20	PWMnCurrFdbk class exported methods	69
2.20.1	Function Documentation	69
2.20.1.1	PWMC_ExecRegularConv	69
2.20.1.2	PWMC_GetMultipleRegularConv	70
2.20.1.3	PWMC_ADC_SetSamplingTime	70
2.21	RevupCtrl class exported methods	71
2.21.1	Function Documentation	71
2.21.1.1	RUC_SetPhaseDurationms	71
2.21.1.2	RUC_SetPhaseFinalMecSpeed01Hz	72
2.21.1.3	RUC_SetPhaseFinalTorque	72
2.21.1.4	RUC_GetPhaseDurationms	72
2.21.1.5	RUC_GetPhaseFinalMecSpeed01Hz	73
2.21.1.6	RUC_GetPhaseFinalTorque	73
2.21.1.7	RUC_GetNumberOfPhases	73
2.22	SpeednPosFdbk class exported methods	75
2.22.1	Function Documentation	75
2.22.1.1	SPD_GetEIAngle	75
2.22.1.2	SPD_GetMecAngle	76
2.22.1.3	SPD_GetAvrgMecSpeed01Hz	76
2.22.1.4	SPD_GetEISpeedDpp	77
2.22.1.5	SPD_Check	77
2.22.1.6	SPD_GetS16Speed	78
2.23	STO class exported methods	80
2.23.1	Function Documentation	80
2.23.1.1	STO_GetEstimatedBemf	80
2.23.1.2	STO_GetEstimatedCurrent	81
2.23.1.3	STO_GetObserverGains	81
2.23.1.4	STO_SetObserverGains	81
2.23.1.5	STO_SetPLLGains	82
2.23.1.6	STO_SetPLLGains	82
2.23.1.7	STO_ResetPLL	82

2.23.1.8	STO_GetEstimatedBemfLevel	83
2.23.1.9	STO_GetObservedBemfLevel	83
2.23.1.10	STO_BemfConsistencyCheckSwitch	83
2.23.1.11	STO_IsBemfConsistent	84
2.24	STO_CORDIC class exported methods	85
2.24.1	Function Documentation	85
2.24.1.1	STO_CR_GetEstimatedBemf	85
2.24.1.2	STO_CR_GetEstimatedCurrent	85
2.24.1.3	STO_CR_GetObserverGains	86
2.24.1.4	STO_CR_SetObserverGains	86
2.24.1.5	STO_CR_GetEstimatedBemfLevel	86
2.24.1.6	STO_CR_GetObservedBemfLevel	87
2.24.1.7	STO_CR_BemfConsistencyCheckSwitch	87
2.24.1.8	STO_CR_IsBemfConsistent	87
2.25	VSS class exported methods	88
2.25.1	Function Documentation	88
2.25.1.1	VSPD_SetMecAcceleration	88
2.25.1.2	VSPD_GetLastRampFinalSpeed	88
2.26	SpeednTorqCtrl class exported methods	90
2.26.1	Function Documentation	90
2.26.1.1	STC_GetMecSpeedRef01Hz	90
2.26.1.2	STC_GetTorqueRef	90
2.26.1.3	STC_GetControlMode	91
2.26.1.4	STC_GetMaxAppPositiveMecSpeed01Hz	91
2.26.1.5	STC_GetMinAppNegativeMecSpeed01Hz	91
2.26.1.6	STC_GetDefaultIqdref	92
2.27	StateMachine class exported methods	93
2.27.1	Function Documentation	93
2.27.1.1	STM_GetState	93
2.27.1.2	STM_GetFaultState	93
2.28	Temperature sensor class	95
2.28.1	Detailed Description	95
2.28.2	Function Documentation	95
2.28.2.1	TSNS_GetAvTemp_C	95
2.28.2.2	TSNS_CheckTemp	96
2.29	BusVoltageSensor class exported methods	97
2.29.1	Function Documentation	97
2.29.1.1	VBS_GetAvBusVoltage_V	97
2.29.1.2	VBS_CheckVbus	97
2.30	DigitalOutput class exported methods	99

2.30.1	Function Documentation	99
2.30.1.1	DOUT_GetOutputState	99
2.31	MotorPowerMeasurement class exported methods	100
2.31.1	Function Documentation	100
2.31.1.1	MPM_GetElMotorPowerW	100
2.31.1.2	MPM_GetAvrgElMotorPowerW	100
2.32	MCInterface	101
2.33	SPC5_PMSM_MC_Application	102
2.34	MCTuning	103
2.35	MC	104
2.36	MCTasks	105
3	SPC5 PMSM MC Library	107
3.1	DigitalOutput class private types	107
3.1.1	Typedef Documentation	108
3.1.1.1	Params_t	108
3.2	DigitalOutput class exported constants	109
3.3	DigitalOutput class exported types	110
3.3.1	Typedef Documentation	110
3.3.1.1	CDOUT	110
3.4	DigitalOutput class exported methods	111
3.4.1	Function Documentation	111
3.4.1.1	DOUT_NewObject	111
3.4.1.2	DOUT_Init	111
3.4.1.3	DOUT_SetOutputState	112
3.4.1.4	DOUT_GetOutputState	112
3.5	EncAlignCtrl class private types	113
3.5.1	Typedef Documentation	113
3.5.1.1	Params_t	113
3.6	EncAlignCtrl class exported types	114
3.6.1	Typedef Documentation	114
3.6.1.1	CEAC	114
3.7	EncAlignCtrl class exported methods	115
3.7.1	Function Documentation	115
3.7.1.1	EAC_NewObject	115
3.7.1.2	EAC_Init	116
3.7.1.3	EAC_StartAlignment	116
3.7.1.4	EAC_Exec	117
3.7.1.5	EAC_IsAligned	117
3.7.1.6	EAC_SetRestartState	118

3.7.1.7	EAC_GetRestartState	118
3.8	CircleLimitation class private types	120
3.8.1	Typedef Documentation	120
3.8.1.1	Params_t	120
3.9	CircleLimitation class exported types	121
3.9.1	Typedef Documentation	121
3.9.1.1	CCLM	121
3.10	CircleLimitation class exported methods	122
3.10.1	Function Documentation	122
3.10.1.1	CLM_NewObject	122
3.10.1.2	Circle_Limitation	122
3.11	MCIRQ_Handler module exported definitions	124
3.11.1	Define Documentation	124
3.11.1.1	MC_IRQ_SPEEDNPOSFDBK_1	124
3.11.1.2	MC_IRQ_SPEEDNPOSFDBK_HS	124
3.11.1.3	MC_IRQ_ICS	124
3.11.1.4	MC_IRQ_R1	124
3.12	MCIRQ Handler module exported functions	125
3.12.1	Function Documentation	125
3.12.1.1	Exec_IRQ_Handler	125
3.13	MotorPowerMeasurement class private defines	126
3.13.1	Define Documentation	126
3.13.1.1	MPM_BUFFER_LENGTHT	126
3.14	MotorPowerMeasurement class private types	127
3.15	MotorPowerMeasurement class methods exported to private implementation of derived classes	128
3.15.1	Function Documentation	128
3.15.1.1	MPM_NewObject	128
3.16	MotorPowerMeasurement class exported types	129
3.16.1	Typedef Documentation	129
3.16.1.1	CMPM	129
3.16.1.2	pMPMInitStruct_t	129
3.17	MotorPowerMeasurement class exported methods	130
3.17.1	Function Documentation	130
3.17.1.1	MPM_Init	130
3.17.1.2	MPM_Clear	131
3.17.1.3	MPM_CalcElMotorPower	131
3.17.1.4	MPM_GetElMotorPowerW	131
3.17.1.5	MPM_GetAvrgElMotorPowerW	132
3.18	PQD private types	133
3.18.1	Typedef Documentation	133

3.18.1.1	DParams_t	133
3.19	PQD class exported types	134
3.19.1	Typedef Documentation	134
3.19.1.1	CPQD_MPM	134
3.20	PQD class exported methods	135
3.20.1	Function Documentation	135
3.20.1.1	PQD_NewObject	135
3.21	PQD class private methods	136
3.22	OpenLoop class private types	137
3.22.1	Typedef Documentation	137
3.22.1.1	Params_t	137
3.23	OpenLoop class exported types	138
3.23.1	Typedef Documentation	138
3.23.1.1	COL	138
3.24	OpenLoop class exported methods	139
3.24.1	Function Documentation	139
3.24.1.1	OL_NewObject	139
3.24.1.2	OL_Init	139
3.24.1.3	OL_VqdConditioning	140
3.24.1.4	OL_UpdateVoltage	140
3.24.1.5	OL_Calc	140
3.24.1.6	OL_VF	141
3.25	PID private types	142
3.25.1	Typedef Documentation	142
3.25.1.1	DParams_t	142
3.26	PID regulator class exported types	143
3.26.1	Typedef Documentation	143
3.26.1.1	CPID_PI	143
3.27	PID class exported methods	144
3.27.1	Function Documentation	144
3.27.1.1	PID_NewObject	144
3.27.1.2	PID_ObjectInit	145
3.27.1.3	PID_SetPrevError	145
3.27.1.4	PID_SetKD	145
3.27.1.5	PID_GetKD	146
3.27.1.6	PID_GetKDDivisor	146
3.27.1.7	PID_Controller	146
3.28	PI regulator class private types	148
3.28.1	Typedef Documentation	148
3.28.1.1	Params_t	148

3.29 PI regulator class exported types	149
3.29.1 Typedef Documentation	149
3.29.1.1 CPI	149
3.30 PI regulator class exported methods	150
3.30.1 Function Documentation	151
3.30.1.1 PI_NewObject	151
3.30.1.2 PI_ObjectInit	151
3.30.1.3 PI_Controller	151
3.30.1.4 PI_SetKP	151
3.30.1.5 PI_SetKI	152
3.30.1.6 PI_GetKP	152
3.30.1.7 PI_GetKI	152
3.30.1.8 PI_GetKPDvisor	153
3.30.1.9 PI_GetKIDvisor	153
3.30.1.10 PI_SetIntegralTerm	153
3.30.1.11 PI_SetLowerIntegralTermLimit	153
3.30.1.12 PI_SetUpperIntegralTermLimit	154
3.30.1.13 PI_SetLowerOutputLimit	154
3.30.1.14 PI_SetUpperOutputLimit	154
3.30.1.15 PI_GetDefaultKP	155
3.30.1.16 PI_GetDefaultKI	155
3.31 FeedForwardCtrl class private types	156
3.31.1 Typedef Documentation	156
3.31.1.1 Params_t	156
3.32 FeedForwardCtrl class exported types	157
3.32.1 Typedef Documentation	157
3.32.1.1 CFF	157
3.33 FeedForwardCtrl class exported methods	158
3.33.1 Function Documentation	158
3.33.1.1 FF_NewObject	158
3.33.1.2 FF_Init	159
3.33.1.3 FF_Clear	159
3.33.1.4 FF_VqdffComputation	159
3.33.1.5 FF_VqdConditioning	159
3.33.1.6 FF_DataProcess	160
3.33.1.7 FF_InitFOCAdditionalMethods	160
3.33.1.8 FF_SetFFConstants	160
3.33.1.9 FF_GetFFConstants	160
3.33.1.10 FF_GetVqdff	161
3.33.1.11 FF_GetVqdAvPlout	161

3.34 MTPACtrl class private types	162
3.34.1 Typedef Documentation	162
3.34.1.1 Params_t	162
3.35 MTPACtrl class exported types	163
3.35.1 Typedef Documentation	163
3.35.1.1 CMTPA	163
3.36 MTPACtrl class exported methods	164
3.36.1 Function Documentation	164
3.36.1.1 MTPA_NewObject	164
3.36.1.2 MTPA_CalcCurrRef	164
3.37 FluxWeakeningCtrl class private types	165
3.37.1 Typedef Documentation	165
3.37.1.1 Params_t	165
3.38 FluxWeakeningCtrl class exported types	166
3.38.1 Typedef Documentation	166
3.38.1.1 CFW	166
3.39 FluxWeakeningCtrl class exported methods	167
3.39.1 Function Documentation	167
3.39.1.1 FW_NewObject	167
3.39.1.2 FW_Init	168
3.39.1.3 FW_Clear	168
3.39.1.4 FW_CalcCurrRef	169
3.39.1.5 FW_DataProcess	170
3.39.1.6 FW_SetVref	170
3.39.1.7 FW_GetVref	170
3.39.1.8 FW_GetAvVAmplitude	170
3.39.1.9 FW_GetAvVPercentage	171
3.40 PWMnCurrFdbk class exported types	172
3.40.1 Typedef Documentation	172
3.40.1.1 CPWMC	172
3.40.2 Enumeration Type Documentation	172
3.40.2.1 LowSideOutputsFunction_t	172
3.40.2.2 CRCAction_t	173
3.41 PWMnCurrFdbk class exported methods	174
3.41.1 Function Documentation	174
3.41.1.1 PWMC_Init	174
3.41.1.2 PWMC_GetPhaseCurrents	175
3.41.1.3 PWMC_SetPhaseVoltage	175
3.41.1.4 PWMC_SwitchOffPWM	175
3.41.1.5 PWMC_SwitchOnPWM	176

3.41.1.6	PWMC_CurrentReadingCalibr	176
3.41.1.7	PWMC_TurnOnLowSides	177
3.41.1.8	PWMC_ExecRegularConv	177
3.41.1.9	PWMC_ADC_SetSamplingTime	178
3.41.1.10	PWMC_CheckOverCurrent	178
3.41.1.11	PWMC_GetTurnOnLowSidesAction	178
3.42	ICS class Description	180
3.43	ICS_class exported types	184
3.43.1	Typedef Documentation	184
3.43.1.1	CICS_PWMC	184
3.43.2	Enumeration Type Documentation	184
3.43.2.1	ICS_ShuntPosition_t	184
3.43.2.2	ICS_SensingSelection_t	185
3.44	ICS class exported	186
3.44.1	Detailed Description	186
3.44.2	Function Documentation	186
3.44.2.1	ICS_NewObject	186
3.45	R1 class Description	187
3.46	R1_class exported types	191
3.46.1	Typedef Documentation	191
3.46.1.1	CR1_PWMC	191
3.47	R1 class exported methods	192
3.47.1	Function Documentation	192
3.47.1.1	R1_NewObject	192
3.48	RevupCtrl class private types	193
3.48.1	Typedef Documentation	193
3.48.1.1	Params_t	193
3.49	RevupCtrl class exported types	194
3.49.1	Typedef Documentation	194
3.49.1.1	CRUC	194
3.50	RevupCtrl class exported methods	195
3.50.1	Function Documentation	196
3.50.1.1	RUC_NewObject	196
3.50.1.2	RUC_Init	196
3.50.1.3	RUC_Clear	196
3.50.1.4	RUC_Exec	197
3.50.1.5	RUC_SetPhaseDurationms	198
3.50.1.6	RUC_SetPhaseFinalMecSpeed01Hz	198
3.50.1.7	RUC_SetPhaseFinalTorque	199
3.50.1.8	RUC_GetPhaseDurationms	199

3.50.1.9 RUC_GetPhaseFinalMecSpeed01Hz	199
3.50.1.10 RUC_GetPhaseFinalTorque	200
3.50.1.11 RUC_GetNumberOfPhases	200
3.50.1.12 RUC_FirstAccelerationStageReached	200
3.51 Encoder Description	202
3.52 ENCODER class exported types	204
3.52.1 Typedef Documentation	204
3.52.1.1 CENC_SPD	204
3.53 ENCODER class exported methods	205
3.53.1 Function Documentation	205
3.53.1.1 Enc_NewObject	205
3.54 HALL Sensor Description	206
3.55 HALL class exported types	210
3.55.1 Typedef Documentation	210
3.55.1.1 CHALL_SPD	210
3.56 HALL class exported methods	211
3.56.1 Function Documentation	211
3.56.1.1 HALL_NewObject	211
3.57 RESOLVER Sensor Description	212
3.58 RESOLVER class exported types	215
3.58.1 Typedef Documentation	215
3.58.1.1 CRES_SPD	215
3.59 RESOLVER class exported methods	216
3.59.1 Function Documentation	216
3.59.1.1 Res_NewObject	216
3.60 SpeednPosFdbk class exported types	217
3.60.1 Typedef Documentation	217
3.60.1.1 CSPD	217
3.61 SpeednPosFdbk class exported methods	218
3.61.1 Function Documentation	219
3.61.1.1 SPD_NewObject	219
3.61.1.2 SPD_Init	219
3.61.1.3 SPD_GetEIAngle	220
3.61.1.4 SPD_GetMecAngle	220
3.61.1.5 SPD_GetAvrgMecSpeed01Hz	221
3.61.1.6 SPD_GetEISpeedDpp	221
3.61.1.7 SPD_SetMecAngle	222
3.61.1.8 SPD_Clear	222
3.61.1.9 SPD_Check	223
3.61.1.10 SPD_CalcAngle	223

3.61.1.11 SPD_CalcAvrgMecSpeed01Hz	224
3.61.1.12 SPD_GetS16Speed	224
3.62 STO class exported types	226
3.62.1 Typedef Documentation	226
3.62.1.1 CSTO_SPD	226
3.63 STO class exported methods	227
3.63.1 Function Documentation	228
3.63.1.1 STO_NewObject	228
3.63.1.2 STO_IsObserverConverged	228
3.63.1.3 STO_GetEstimatedBemf	229
3.63.1.4 STO_GetEstimatedCurrent	229
3.63.1.5 STO_GetObserverGains	229
3.63.1.6 STO_SetObserverGains	230
3.63.1.7 STO_GetPLLGains	230
3.63.1.8 STO_SetPLLGains	231
3.63.1.9 STO_ResetPLL	231
3.63.1.10 STO_SetPLL	232
3.63.1.11 STO_CalcAvrgEISpeedDpp	232
3.63.1.12 STO_GetEstimatedBemfLevel	233
3.63.1.13 STO_GetObservedBemfLevel	233
3.63.1.14 STO_BemfConsistencyCheckSwitch	233
3.63.1.15 STO_IsBemfConsistent	234
3.64 VirtualSpeedSensor class exported types	235
3.64.1 Typedef Documentation	235
3.64.1.1 CVSS_SPD	235
3.65 VirtualSpeedSensor class exported methods	236
3.65.1 Function Documentation	236
3.65.1.1 VSS_NewObject	236
3.65.1.2 VSPD_SetMecAcceleration	237
3.65.1.3 VSPD_GetLastRampFinalSpeed	237
3.65.1.4 VSPD_SetStartTransition	237
3.66 VirtualSpeedSensor class private methods	239
3.66.1 Function Documentation	239
3.66.1.1 VSPD_SetMecAcceleration	239
3.66.1.2 VSPD_GetLastRampFinalSpeed	239
3.66.1.3 VSPD_SetStartTransition	240
3.67 SpeednTorqCtrl class private types	241
3.67.1 Typedef Documentation	241
3.67.1.1 Params_t	241
3.68 SpeednTorqCtrl class exported types	242

3.68.1 Typedef Documentation	242
3.68.1.1 CSTC	242
3.69 SpeednTorqCtrl class exported methods	243
3.69.1 Function Documentation	244
3.69.1.1 STC_NewObject	244
3.69.1.2 STC_Init	244
3.69.1.3 STC_Clear	244
3.69.1.4 STC_GetMecSpeedRef01Hz	245
3.69.1.5 STC_GetTorqueRef	245
3.69.1.6 STC_SetControlMode	246
3.69.1.7 STC_GetControlMode	246
3.69.1.8 STC_ExecRamp	246
3.69.1.9 STC_StopRamp	247
3.69.1.10 STC_CalcTorqueReference	247
3.69.1.11 STC_GetMecSpeedRef01HzDefault	248
3.69.1.12 STC_GetMaxAppPositiveMecSpeed01Hz	248
3.69.1.13 STC_GetMinAppNegativeMecSpeed01Hz	249
3.69.1.14 STC_RampCompleted	249
3.69.1.15 STC_SetSpeedSensor	249
3.69.1.16 STC_GetSpeedSensor	250
3.69.1.17 STC_GetDefaultIqdref	250
3.70 StateMachine class private types	251
3.71 StateMachine class exported constants	252
3.72 StateMachine class exported types	253
3.72.1 Typedef Documentation	253
3.72.1.1 CSTM	253
3.73 StateMachine class exported methods	254
3.73.1 Function Documentation	254
3.73.1.1 STM_NewObject	254
3.73.1.2 STM_Init	255
3.73.1.3 STM_NextState	255
3.73.1.4 STM_FaultProcessing	256
3.73.1.5 STM_GetState	256
3.73.1.6 STM_FaultAcknowledged	257
3.73.1.7 STM_GetFaultState	257
3.74 NTC private types	258
3.74.1 Typedef Documentation	258
3.74.1.1 DParams_t	258
3.75 NTC class exported types	259
3.75.1 Typedef Documentation	259

3.75.1.1	CNTC_TSNS	259
3.76	NTC class exported methods	260
3.76.1	Function Documentation	260
3.76.1.1	NTC_NewObject	260
3.77	TemperatureSensor class private types	262
3.77.1	Typedef Documentation	262
3.77.1.1	Params_t	262
3.78	TempSensor class methods exported to private implementation of derived classes	263
3.78.1	Function Documentation	263
3.78.1.1	TSNS_NewObject	263
3.79	Temperature Sensor class exported types	264
3.79.1	Typedef Documentation	264
3.79.1.1	CTSNS	264
3.80	Temperature sensor class exported methods	265
3.80.1	Function Documentation	265
3.80.1.1	TSNS_Init	265
3.80.1.2	TSNS_Clear	265
3.80.1.3	TSNS_CalcAvTemp	266
3.80.1.4	TSNS_GetAvTemp_d	266
3.80.1.5	TSNS_GetAvTemp_C	266
3.80.1.6	TSNS_CheckTemp	267
3.81	Virtual temperature sensor private types	268
3.81.1	Typedef Documentation	268
3.81.1.1	DParams_t	268
3.82	Virtual Temperature sensor class exported types	269
3.82.1	Typedef Documentation	269
3.82.1.1	CVTS_TSNS	269
3.83	Virtual Temperature sensor class exported methods	270
3.83.1	Function Documentation	270
3.83.1.1	VTS_NewObject	270
3.84	Virtual Temperature sensor class private methods	272
3.85	BusVoltageSensor class private types	273
3.85.1	Typedef Documentation	273
3.85.1.1	Params_t	273
3.86	BusVoltageSensor class methods exported to private implementation of derived classes	274
3.86.1	Function Documentation	274
3.86.1.1	VBS_NewObject	274
3.87	BusVoltageSensor class exported types	275
3.87.1	Typedef Documentation	275
3.87.1.1	CVBS	275

3.88 BusVoltageSensor class exported methods	276
3.88.1 Function Documentation	276
3.88.1.1 VBS_Init	276
3.88.1.2 VBS_Clear	277
3.88.1.3 VBS_CalcAvVbus	277
3.88.1.4 VBS_GetBusVoltage_d	277
3.88.1.5 VBS_GetAvBusVoltage_d	278
3.88.1.6 VBS_GetAvBusVoltage_V	278
3.88.1.7 VBS_CheckVbus	278
3.89 Virtual Vbus sensor private types	280
3.89.1 Typedef Documentation	280
3.89.1.1 DParams_t	280
3.90 Virtual Vbus sensor class exported types	281
3.90.1 Typedef Documentation	281
3.90.1.1 CVVBS_VBS	281
3.91 Virtual Vbus sensor class exported methods	282
3.91.1 Function Documentation	282
3.91.1.1 VVBS_NewObject	282
3.92 Virtual Vbus sensor class private methods	283
3.93 Rdivider private types	284
3.93.1 Typedef Documentation	284
3.93.1.1 DParams_t	284
3.94 Rdivider class exported types	285
3.94.1 Typedef Documentation	285
3.94.1.1 CRVBS_VBS	285
3.95 Rdivider class exported methods	286
3.95.1 Function Documentation	286
3.95.1.1 RVBS_NewObject	286
3.96 Rdivider class private methods	287
3.97 SPC5_PMSM_MC_Library	288
3.98 DigitalOutput	290
3.99 EncAlignCtrl	291
3.100CircleLimitation	292
3.101MCIRQ_Handler	293
3.102MotorPowerMeasurement	294
3.103MotorPowerMeasurement_PQD	295
3.104OpenLoop	296
3.105PI_regulator_PID	297
3.106PI_regulator	298
3.107FeedForwardCtrl	299

3.108MTPACtrl	300
3.109FluxWeakeningCtrl	301
3.110PWMMnCurrFdbk	302
3.111PWMMnCurrFdbk_ICS	303
3.112PWMMnCurrFdbk_R1	304
3.113RevupCtrl	305
3.114SpeednPosFdbk_ENCODER	306
3.115SpeednPosFdbk_HALL	307
3.116SpeednPosFdbk_RESOLVER	308
3.117SpeednPosFdbk	309
3.118SpeednPosFdbk_STO	310
3.119SpeednPosFdbk_VirtualSpeedSensor	311
3.120SpeednTorqCtrl	312
3.121StateMachine	313
3.122TemperatureSensor_NTC	314
3.123TemperatureSensor	315
3.124TemperatureSensor_Virtual	316
3.125BusVoltageSensor	317
3.126BusVoltageSensor_Virtual	318
3.127BusVoltageSensor_Rdivider	319
4 SPC5 PMSM UI Library	321
4.1 UserInterface class exported types	321
4.1.1 Define Documentation	321
4.1.1.1 UI_IRQ_USART	321
4.1.1.2 UI_IRQ_CAN	322
4.1.2 Typedef Documentation	322
4.1.2.1 CUI	322
4.1.2.2 UserInterfaceParams_t	322
4.2 UserInterface class exported methods	323
4.2.1 Function Documentation	324
4.2.1.1 UI_NewObject	324
4.2.1.2 UI_Init	324
4.2.1.3 UI_SelectMC	325
4.2.1.4 UI_GetSelectedMC	325
4.2.1.5 UI_GetSelectedMCConfig	325
4.2.1.6 UI_SetReg	326
4.2.1.7 UI_GetReg	326
4.2.1.8 UI_GetCurrentMCT	327
4.2.1.9 UI_ExecCmd	327

4.2.1.10	UI_ExecSpeedRamp	328
4.2.1.11	UI_ExecTorqueRamp	329
4.2.1.12	UI_GetRevupData	329
4.2.1.13	UI_SetRevupData	330
4.2.1.14	UI_SetCurrentReferences	331
4.2.1.15	UI_DACInit	332
4.2.1.16	UI_DACExec	332
4.2.1.17	UI_SetDAC	332
4.2.1.18	UI_GetDAC	333
4.2.1.19	UI_SetUserDAC	333
4.2.1.20	UI_LCDInit	334
4.2.1.21	UI_LCDExec	334
4.2.1.22	UI_LCDUpdateAll	334
4.2.1.23	UI_LCDUpdateMeasured	334
4.3	UserInterface class private types	336
4.3.1	Typedef Documentation	336
4.3.1.1	Params_t	336
4.4	UserInterface sensor code	338
4.4.1	Define Documentation	338
4.4.1.1	UI_SCODE_HALL	338
4.4.1.2	UI_SCODE_ENC	338
4.4.1.3	UI_SCODE_RES	338
4.4.1.4	UI_SCODE_STO_PLL	338
4.4.1.5	UI_SCODE_STO_CR	339
4.4.1.6	UI_SCODE_HFINJ	339
4.5	UserInterface configuration option	340
4.5.1	Define Documentation	340
4.5.1.1	UI_CFGOPT_NONE	340
4.5.1.2	UI_CFGOPT_FW	340
4.5.1.3	UI_CFGOPT_SPEED_KD	340
4.5.1.4	UI_CFGOPT_Iq_KD	340
4.5.1.5	UI_CFGOPT_Id_KD	341
4.5.1.6	UI_CFGOPT_DAC	341
4.5.1.7	UI_CFGOPT_SETIDINSPDMODE	341
4.5.1.8	UI_CFGOPT_PLLTUNING	341
4.5.1.9	UI_CFGOPT_PFC	341
4.5.1.10	UI_CFGOPT_PFC_I_KD	341
4.5.1.11	UI_CFGOPT_PFC_V_KD	341
4.6	Motor Control Protocol class exported types	342
4.7	Motor Control Protocol class exported methods	343

4.7.1	Function Documentation	343
4.7.1.1	MCP_NewObject	343
4.8	Frame Communication Protocol class exported types	344
4.9	Frame Communication Protocol class exported methods	345
4.9.1	Function Documentation	345
4.9.1.1	FCP_NewObject	345
4.10	Physical Layer Communication class exported types	346
4.11	Physical Layer Communication class exported methods	347
4.11.1	Function Documentation	347
4.11.1.1	COM_NewObject	347
4.11.1.2	COM_StartReceive	347
4.11.1.3	COM_StartTransmit	348
4.12	USART physical layer communication class exported types	349
4.12.1	Define Documentation	349
4.12.1.1	_I	349
4.12.1.2	_O	349
4.12.1.3	_IO	349
4.12.1.4	USART_CR1_RE	350
4.12.1.5	USART_CR1_TE	350
4.12.2	Typedef Documentation	350
4.12.2.1	CUSART_COM	350
4.13	USART physical layer communication class exported methods	351
4.13.1	Function Documentation	351
4.13.1.1	USART_NewObject	351
4.13.1.2	USART_ITConfig	352
4.13.1.3	USART_SendData	352
4.14	Can Physical Layer Communication class Description	354
4.15	CAN physical layer communication class exported types	357
4.15.1	Typedef Documentation	357
4.15.1.1	CCAN_COM	357
4.16	CAN physical layer communication class exported methods	358
4.16.1	Function Documentation	358
4.16.1.1	CAN_NewObject	358
4.16.1.2	CAN_ITConfig	358
4.16.1.3	CAN_IRQHandlerRX	359
4.17	SPC5_PMSM_UI_Library	360
4.18	UserInterface_IRQHandler	361
4.19	UserInterface	362
4.20	MotorControlProtocol	363
4.21	FrameCommunicationProtocol	364

4.22 PhysicalLayerCommunication	365
4.23 USART_PhysicalCommunicationLayer	366
4.24 CAN_PhysicalCommunicationLayer	367
5 Data Structure Documentation	369
5.1 _CMCI_t Struct Reference	369
5.1.1 Detailed Description	369
5.1.2 Field Documentation	370
5.1.2.1 Vars_str	370
5.1.2.2 pParams_str	370
5.2 _CMCT_t Struct Reference	371
5.2.1 Detailed Description	371
5.2.2 Field Documentation	372
5.2.2.1 Vars_str	372
5.2.2.2 pParams_str	372
5.3 MCTuningInitStruct_t Struct Reference	372
5.3.1 Detailed Description	372
5.4 Vars_t Struct Reference	372
5.4.1 Detailed Description	372
5.4.2 Field Documentation	374
5.4.2.1 oSTM	374
5.4.2.2 oSTC	375
5.4.2.3 pFOCVars	375
5.4.2.4 lastCommand	375
5.4.2.5 hFinalSpeed	375
5.4.2.6 hFinalTorque	375
5.4.2.7 lqdref	375
5.4.2.8 hDurationms	376
5.4.2.9 CommandState	376
5.4.2.10 LastModalitySetByUser	376
5.4.2.11 oVSS	376
5.4.2.12 oENC	376
5.4.2.13 hRemainingTicks	376
5.4.2.14 EncAligned	376
5.4.2.15 EncRestart	377
5.4.2.16 hMeasBuffer	377
5.4.2.17 hNextMeasBufferIndex	377
5.4.2.18 hLastMeasBufferIndex	377
5.4.2.19 hAvrgEIMotorPowerW	377
5.4.2.20 hVoltage	377

5.4.2.21	oVSS	377
5.4.2.22	VFMode	377
5.4.2.23	Vqdff	378
5.4.2.24	VqdPlout	378
5.4.2.25	VqdAvPlout	378
5.4.2.26	wConstant_1D	378
5.4.2.27	wConstant_1Q	378
5.4.2.28	wConstant_2	378
5.4.2.29	oVBS	378
5.4.2.30	oPI_q	378
5.4.2.31	oPI_d	378
5.4.2.32	oFluxWeakeningPI	379
5.4.2.33	oSpeedPI	379
5.4.2.34	hFW_V_Ref	379
5.4.2.35	AvVolt_qd	379
5.4.2.36	AvVoltAmpl	379
5.4.2.37	hldRefOffset	379
5.4.2.38	hT_Sqrt3	379
5.4.2.39	hSector	379
5.4.2.40	bTurnOnLowSidesAction	380
5.4.2.41	hOffCalibrWaitTimeCounter	380
5.4.2.42	bMotor	380
5.4.2.43	hPhaseRemainingTicks	380
5.4.2.44	pPhaseParams	380
5.4.2.45	hDirection	380
5.4.2.46	bStageCnt	380
5.4.2.47	ParamsData	381
5.4.2.48	bPhaseNbr	381
5.4.2.49	bMode	381
5.4.2.50	hTargetFinal	381
5.4.2.51	wSpeedRef01HzExt	381
5.4.2.52	wTorqueRef	381
5.4.2.53	hRampRemainingStep	381
5.4.2.54	oPISpeed	382
5.4.2.55	oSPD	382
5.4.2.56	wIncDecAmount	382
5.4.2.57	bState	382
5.4.2.58	hFaultNow	382
5.4.2.59	hFaultOccurred	382
5.4.2.60	hFaultState	382

5.4.2.61	hLatestConv	383
5.4.2.62	bDriveNum	383
5.4.2.63	pMCI	383
5.4.2.64	pMCT	383
5.4.2.65	pUICfg	383
5.4.2.66	bSelectedDrive	383
5.5	_CCLM_t Struct Reference	383
5.5.1	Detailed Description	383
5.5.2	Field Documentation	384
5.5.2.1	pParams_str	384
5.6	_CDOUT_t Struct Reference	384
5.6.1	Detailed Description	384
5.6.2	Field Documentation	385
5.6.2.1	Vars_str	385
5.6.2.2	pParams_str	385
5.7	_CEAC_t Struct Reference	386
5.7.1	Detailed Description	386
5.7.2	Field Documentation	387
5.7.2.1	Vars_str	387
5.7.2.2	pParams_str	387
5.8	_CFF_t Struct Reference	387
5.8.1	Detailed Description	387
5.8.2	Field Documentation	388
5.8.2.1	Vars_str	388
5.8.2.2	pParams_str	388
5.9	_CFW_t Struct Reference	389
5.9.1	Detailed Description	389
5.9.2	Field Documentation	390
5.9.2.1	Vars_str	390
5.9.2.2	pParams_str	390
5.10	_CMCIRQ_t Struct Reference	390
5.10.1	Detailed Description	390
5.11	_CMPM_t Struct Reference	390
5.11.1	Detailed Description	390
5.11.2	Field Documentation	391
5.11.2.1	Methods_str	391
5.11.2.2	Vars_str	392
5.11.2.3	DerivedClass	392
5.12	_CMTPA_t Struct Reference	392
5.12.1	Detailed Description	392

5.12.2 Field Documentation	392
5.12.2.1 pParams_str	392
5.13 _CNTC_TSNS_t Struct Reference	392
5.13.1 Detailed Description	392
5.13.2 Field Documentation	393
5.13.2.1 DVars_str	393
5.13.2.2 pDParams_str	394
5.14 _COL_t Struct Reference	394
5.14.1 Detailed Description	394
5.14.2 Field Documentation	395
5.14.2.1 Vars_str	395
5.14.2.2 pParams_str	395
5.15 _CPI_t Struct Reference	395
5.15.1 Detailed Description	395
5.15.2 Field Documentation	396
5.15.2.1 Vars_str	396
5.15.2.2 pParams_str	397
5.15.2.3 DerivedClass	397
5.16 _CPWMC_t Struct Reference	397
5.16.1 Detailed Description	397
5.16.2 Field Documentation	398
5.16.2.1 Methods_str	398
5.16.2.2 Vars_str	399
5.16.2.3 pParams_str	399
5.16.2.4 DerivedClass	399
5.17 _CRUC_t Struct Reference	399
5.17.1 Detailed Description	399
5.17.2 Field Documentation	400
5.17.2.1 Vars_str	400
5.17.2.2 pParams_str	400
5.18 _CRVBS_VBS_t Struct Reference	401
5.18.1 Detailed Description	401
5.18.2 Field Documentation	402
5.18.2.1 DVars_str	402
5.18.2.2 pDParams_str	403
5.19 _CSPD_t Struct Reference	403
5.19.1 Detailed Description	403
5.19.2 Field Documentation	404
5.19.2.1 Methods_str	404
5.19.2.2 Vars_str	405

5.19.2.3	pParams_str	405
5.19.2.4	DerivedClass	405
5.20	_CSTC_t Struct Reference	405
5.20.1	Detailed Description	405
5.20.2	Field Documentation	406
5.20.2.1	Vars_str	406
5.20.2.2	pParams_str	406
5.21	_CSTM_t Struct Reference	407
5.21.1	Detailed Description	407
5.21.2	Field Documentation	408
5.21.2.1	Vars_str	408
5.22	_CTSNS_t Struct Reference	408
5.22.1	Detailed Description	408
5.22.2	Field Documentation	409
5.22.2.1	Methods_str	409
5.22.2.2	Vars_str	409
5.22.2.3	pParams_str	409
5.22.2.4	DerivedClass	409
5.23	_CVBS_t Struct Reference	409
5.23.1	Detailed Description	409
5.23.2	Field Documentation	410
5.23.2.1	Methods_str	410
5.23.2.2	Vars_str	411
5.23.2.3	pParams_str	411
5.23.2.4	DerivedClass	411
5.24	_CVTS_TSNS_t Struct Reference	411
5.24.1	Detailed Description	411
5.24.2	Field Documentation	412
5.24.2.1	pDParams_str	412
5.25	_CVVBS_VBS_t Struct Reference	412
5.25.1	Detailed Description	412
5.25.2	Field Documentation	412
5.25.2.1	pDParams_str	412
5.26	_DCENC_SPD_t Struct Reference	413
5.26.1	Detailed Description	413
5.26.2	Field Documentation	414
5.26.2.1	DVars_str	414
5.26.2.2	pDParams_str	414
5.27	_DCHALL_SPD_t Struct Reference	414
5.27.1	Detailed Description	414

5.27.2 Field Documentation	415
5.27.2.1 DVars_str	415
5.27.2.2 pDParams_str	416
5.28 _DCICS_PWMC_t Struct Reference	416
5.28.1 Detailed Description	416
5.28.2 Field Documentation	417
5.28.2.1 DVars_str	417
5.28.2.2 pDParams_str	418
5.29 _DCPID_t Struct Reference	418
5.29.1 Detailed Description	418
5.30 _DCPQD_MPM_t Struct Reference	419
5.30.1 Detailed Description	419
5.30.2 Field Documentation	420
5.30.2.1 DVars_str	420
5.30.2.2 pDParams_str	421
5.31 _DCR1_PWMC_t Struct Reference	421
5.31.1 Detailed Description	421
5.31.2 Field Documentation	422
5.31.2.1 DVars_str	422
5.31.2.2 pDParams_str	423
5.32 _DCRES_SPD_t Struct Reference	423
5.32.1 Detailed Description	423
5.33 _DCSTO_SPD_t Struct Reference	424
5.33.1 Detailed Description	424
5.33.2 Field Documentation	425
5.33.2.1 DVars_str	425
5.33.2.2 pDParams_str	426
5.34 _DCVSS_SPD_t Struct Reference	426
5.34.1 Detailed Description	426
5.34.2 Field Documentation	427
5.34.2.1 DVars_str	427
5.34.2.2 pDParams_str	428
5.35 BusVoltageSensorParams_t Struct Reference	428
5.35.1 Detailed Description	428
5.35.2 Field Documentation	428
5.35.2.1 bSensorType	428
5.35.2.2 hConversionFactor	428
5.36 CircleLimitationParams_t Struct Reference	428
5.36.1 Detailed Description	428
5.36.2 Field Documentation	429

5.36.2.1	hMaxModule	429
5.36.2.2	hCircle_limit_table	429
5.36.2.3	bStart_index	429
5.37	DigitalOutputParams_t Struct Reference	429
5.37.1	Detailed Description	429
5.37.2	Field Documentation	429
5.37.2.1	hDOOutputPort	429
5.37.2.2	hDOOutputPin	429
5.37.2.3	bDOOutputPolarity	430
5.38	DVars_t Struct Reference	430
5.38.1	Detailed Description	430
5.38.2	Field Documentation	433
5.38.2.1	pFOCVars	433
5.38.2.2	oVBS	433
5.38.2.3	hPhaseAOOffset	433
5.38.2.4	hPhaseBOffset	433
5.38.2.5	Half_PWMPeriod	434
5.38.2.6	hRegConv	434
5.38.2.7	CurrentCTUFifoBuffer	434
5.38.2.8	VBusCTUFifoBuffer	434
5.38.2.9	RegularCTUFifoBuffer	434
5.38.2.10	hPhaseOffset	434
5.38.2.11	hCntSmp1	434
5.38.2.12	hCntSmp2	434
5.38.2.13	sampCur1	434
5.38.2.14	sampCur2	435
5.38.2.15	hCurrAOld	435
5.38.2.16	hCurrBOld	435
5.38.2.17	hCurrCOld	435
5.38.2.18	blInverted_pwm_new	435
5.38.2.19	hFlags	435
5.38.2.20	hTimerOverflowNb	435
5.38.2.21	SensorIsReliable	435
5.38.2.22	hPreviousCapture	436
5.38.2.23	wDeltaCapturesBuffer	436
5.38.2.24	wU32MAXdivPulseNumber	436
5.38.2.25	hSpeedSamplingFreqHz	436
5.38.2.26	TimerOverflowError	436
5.38.2.27	e_angle_first_index_value	436
5.38.2.28	e_angle_current_index_value	436

5.38.2.29 direction	436
5.38.2.30 bls_First_Measurement	436
5.38.2.31 blIndex_Signal	437
5.38.2.32 hMeasuredElAngle	437
5.38.2.33 hPrev_MeasuredElAngle	437
5.38.2.34 h_u_angle	437
5.38.2.35 speed_buffer	437
5.38.2.36 Hall_current_speed	437
5.38.2.37 h_polar_pairs	437
5.38.2.38 bSpeed	437
5.38.2.39 hAvrElSpeedDpp	437
5.38.2.40 bHallState	438
5.38.2.41 pole_pair_ranges	438
5.38.2.42 ResCTUFifoBuffer	438
5.38.2.43 hC1	438
5.38.2.44 hC2	438
5.38.2.45 hC3	438
5.38.2.46 hC4	438
5.38.2.47 hC5	438
5.38.2.48 hC6	439
5.38.2.49 hF1	439
5.38.2.50 hF2	439
5.38.2.51 hF3	439
5.38.2.52 hF3POW2	439
5.38.2.53 oPIRegulator	439
5.38.2.54 wlalpha_est	439
5.38.2.55 wlbeta_est	439
5.38.2.56 wBemf_alpha_est	440
5.38.2.57 wBemf_beta_est	440
5.38.2.58 hBemf_alpha_est	440
5.38.2.59 hBemf_beta_est	440
5.38.2.60 hSpeed_Buffer	440
5.38.2.61 bSpeed_Buffer_Index	440
5.38.2.62 blsSpeedReliable	440
5.38.2.63 bConsistencyCounter	440
5.38.2.64 bReliabilityCounter	441
5.38.2.65 blsAlgorithmConverged	441
5.38.2.66 blsBemfConsistent	441
5.38.2.67 wObs_Bemf_Level	441
5.38.2.68 wEst_Bemf_Level	441

5.38.2.69 bEnableDualCheck	441
5.38.2.70 wDppBufferSum	441
5.38.2.71 hSpeedBufferOldestEl	441
5.38.2.72 wElAccDppP32	442
5.38.2.73 wElSpeedDpp32	442
5.38.2.74 hRemainingStep	442
5.38.2.75 hFinalMecSpeed01Hz	442
5.38.2.76 bTransitionStarted	442
5.38.2.77 bTransitionEnded	442
5.38.2.78 hTransitionRemainingSteps	442
5.38.2.79 hElAngleAccu	442
5.38.2.80 bTransitionLocked	443
5.38.2.81 IsOverTempFaultActive	443
5.38.2.82 bChannel_variable	443
5.38.2.83 hUserValue	443
5.38.2.84 comON	443
5.38.2.85 bChannel	443
5.38.2.86 bChTransmitted	443
5.38.2.87 bByteTransmitted	443
5.38.2.88 wBuffer	444
5.38.2.89 bChByteNum	444
5.38.2.90 bChNum	444
5.39 EncAlignCtrlParams_t Struct Reference	444
5.39.1 Detailed Description	444
5.39.2 Field Documentation	444
5.39.2.1 hEACFrequencyHz	444
5.39.2.2 hFinalTorque	444
5.39.2.3 hElAngle	445
5.39.2.4 hDurationms	445
5.39.2.5 bElToMecRatio	445
5.40 ENCODERParams_t Struct Reference	445
5.40.1 Detailed Description	445
5.40.2 Field Documentation	446
5.40.2.1 IRQnb	446
5.40.2.2 hPulseNumber	446
5.40.2.3 RevertSignal	446
5.40.2.4 hSpeedSamplingFreq01Hz	446
5.40.2.5 bSpeedBufferSize	446
5.40.2.6 hInpCaptFilter_chA_B	447
5.40.2.7 hInpCaptFilter_index_ch	447

5.40.2.8 eTimerModule	447
5.40.2.9 quad_eTimer_ch	447
5.40.2.10 CH_A_eTimer_ch	447
5.40.2.11 CH_B_eTimer_ch	447
5.40.2.12 index_enable	447
5.40.2.13 position_eTimer_ch	447
5.40.2.14 index_counter_eTimer_ch	447
5.40.2.15 index_input_capture_eTimer_ch	448
5.40.2.16 Overflow_eTimer_ch	448
5.40.2.17 position_acq_pwm	448
5.41 FeedForwardCtrlParams_t Struct Reference	448
5.41.1 Detailed Description	448
5.41.2 Field Documentation	448
5.41.2.1 hVqdLowPassFilterBW	448
5.41.2.2 wDefConstant_1D	448
5.41.2.3 wDefConstant_1Q	449
5.41.2.4 wDefConstant_2	449
5.41.2.5 hVqdLowPassFilterBWLOG	449
5.42 FFInit_t Struct Reference	449
5.42.1 Detailed Description	449
5.43 FluxWeakeningCtrlParams_t Struct Reference	449
5.43.1 Detailed Description	449
5.43.2 Field Documentation	449
5.43.2.1 hMaxModule	449
5.43.2.2 hDefaultFW_V_Ref	450
5.43.2.3 hDemagCurrent	450
5.43.2.4 wNominalSqCurr	450
5.43.2.5 hVqdLowPassFilterBW	450
5.43.2.6 hVqdLowPassFilterBWLOG	450
5.44 FWInit_t Struct Reference	450
5.44.1 Detailed Description	450
5.45 HALLParams_t Struct Reference	450
5.45.1 Detailed Description	450
5.45.2 Field Documentation	451
5.45.2.1 IRQnb	451
5.45.2.2 bSensorPlacement	451
5.45.2.3 hPhaseShift	452
5.45.2.4 hSpeedSamplingFreqHz	452
5.45.2.5 hSpeedThreshold	452
5.45.2.6 hPwmFrequency	452

5.45.2.7	hFocUpdate	452
5.45.2.8	hInpCaptFilter_ch1_2_3	452
5.45.2.9	eTimerModule	452
5.45.2.10	S1_eTimer_ch	452
5.45.2.11	S2_eTimer_ch	452
5.45.2.12	S3_eTimer_ch	453
5.45.2.13	ctu_trig_eTimer_ch	453
5.46	ICS_Params_t Struct Reference	453
5.46.1	Detailed Description	453
5.46.2	Field Documentation	455
5.46.2.1	hlaChannel	455
5.46.2.2	hlbChannel	455
5.46.2.3	AdcPwrDownDelay	455
5.46.2.4	DmaChannel	456
5.46.2.5	EnableVbusConversion	456
5.46.2.6	AdcVbusUnit	456
5.46.2.7	AdcVbusChannel	456
5.46.2.8	DmaVBusChannel	456
5.46.2.9	EnableUserConversion	456
5.46.2.10	EnableMultipleUserConversion	456
5.46.2.11	AdcUserUnit	456
5.46.2.12	AdcUserChannel	456
5.46.2.13	DmaUserChannel	457
5.46.2.14	IRQnb	457
5.46.2.15	FlexPWM_Clock_Divider	457
5.46.2.16	FlexPWMModule	457
5.46.2.17	hDeadTime	457
5.46.2.18	PWMLoadFreq	457
5.46.2.19	hCh1Polarity	457
5.46.2.20	hCh1IdleState	457
5.46.2.21	hCh2Polarity	458
5.46.2.22	hCh2IdleState	458
5.46.2.23	hCh3Polarity	458
5.46.2.24	hCh3IdleState	458
5.46.2.25	LowSideOutputs	458
5.46.2.26	ShuntPosition	458
5.46.2.27	SensingSelection	458
5.46.2.28	hCh1NPolarity	458
5.46.2.29	hCh1NIdleState	458
5.46.2.30	hCh2NPolarity	459

5.46.2.31 hCh2NIdleState	459
5.46.2.32 hCh3NPolarity	459
5.46.2.33 hCh3NIdleState	459
5.46.2.34 EmergencyStop	459
5.46.2.35 FaultPIN	459
5.46.2.36 FaultPolarity	459
5.47 Methods_t Struct Reference	459
5.47.1 Detailed Description	459
5.48 MTPACtrlParams_t Struct Reference	460
5.48.1 Detailed Description	460
5.48.2 Field Documentation	460
5.48.2.1 hSegDiv	460
5.48.2.2 wAngCoeff	460
5.48.2.3 wOffset	460
5.49 NTCParams_t Struct Reference	460
5.49.1 Detailed Description	460
5.49.2 Field Documentation	461
5.49.2.1 bTsensADCModule	461
5.49.2.2 bTsensADChannel	461
5.49.2.3 bTsensSamplingTime	461
5.49.2.4 hLowPassFilterBW	461
5.49.2.5 hOverTempThreshold	461
5.49.2.6 hOverTempDeactThreshold	461
5.49.2.7 hSensitivity	461
5.49.2.8 wV0	462
5.49.2.9 hT0	462
5.50 Observer_Inputs_t Struct Reference	462
5.50.1 Detailed Description	462
5.51 OpenLoopParams_t Struct Reference	462
5.51.1 Detailed Description	462
5.51.2 Field Documentation	463
5.51.2.1 VFMode	463
5.51.2.2 hVFSlope	463
5.52 PIDParams_t Struct Reference	463
5.52.1 Detailed Description	463
5.52.2 Field Documentation	463
5.52.2.1 hDefKdGain	463
5.52.2.2 hKdDivisor	463
5.52.2.3 hKdDivisorPOW2	464
5.53 PIParams_t Struct Reference	464

5.53.1	Detailed Description	464
5.53.2	Field Documentation	464
5.53.2.1	hDefKpGain	464
5.53.2.2	hDefKiGain	464
5.53.2.3	hKpDivisor	464
5.53.2.4	hKiDivisor	465
5.53.2.5	wDefMaxIntegralTerm	465
5.53.2.6	wDefMinIntegralTerm	465
5.53.2.7	hDefMaxOutput	465
5.53.2.8	hDefMinOutput	465
5.53.2.9	hKpDivisorPOW2	465
5.53.2.10	hKiDivisorPOW2	465
5.54	PQD_MPMLInitStruct_t Struct Reference	465
5.54.1	Detailed Description	465
5.54.2	Field Documentation	466
5.54.2.1	pFOCVars	466
5.54.2.2	oVBS	467
5.55	PQDParams_t Struct Reference	467
5.55.1	Detailed Description	467
5.56	PWMnCurrFdbkParams_t Struct Reference	467
5.56.1	Detailed Description	467
5.56.2	Field Documentation	467
5.56.2.1	hPWMperiod	467
5.56.2.2	hOffCalibrWaitTicks	467
5.57	R1_Params_t Struct Reference	467
5.57.1	Detailed Description	467
5.57.2	Field Documentation	470
5.57.2.1	hIChannel	470
5.57.2.2	AdcPwrDownDelay	470
5.57.2.3	DmaChannel	470
5.57.2.4	EnableVbusConversion	470
5.57.2.5	AdcVbusUnit	470
5.57.2.6	AdcVbusChannel	470
5.57.2.7	DmaVBusChannel	471
5.57.2.8	EnableUserConversion	471
5.57.2.9	EnableMultipleUserConversion	471
5.57.2.10	AdcUserUnit	471
5.57.2.11	AdcUserChannel	471
5.57.2.12	DmaUserChannel	471
5.57.2.13	IRQnb	471

5.57.2.14 hDeadTime	471
5.57.2.15 hTaftter	471
5.57.2.16 hTbefore	472
5.57.2.17 hTMin	472
5.57.2.18 hHTMin	472
5.57.2.19 hTSample	472
5.57.2.20 FlexPWM_Clock_Divider	472
5.57.2.21 FlexPWMMModule	472
5.57.2.22 PWMLoadFreq	472
5.57.2.23 hCh1Polarity	472
5.57.2.24 hCh1IdleState	472
5.57.2.25 hCh2Polarity	473
5.57.2.26 hCh2IdleState	473
5.57.2.27 hCh3Polarity	473
5.57.2.28 hCh3IdleState	473
5.57.2.29 LowSideOutputs	473
5.57.2.30 hCh1NPolarity	473
5.57.2.31 hCh1NIdleState	473
5.57.2.32 hCh2NPolarity	473
5.57.2.33 hCh2NIdleState	474
5.57.2.34 hCh3NPolarity	474
5.57.2.35 hCh3NIdleState	474
5.57.2.36 EmergencyStop	474
5.57.2.37 FaultPIN	474
5.57.2.38 FaultPolarity	474
5.58 RdividerParams_t Struct Reference	474
5.58.1 Detailed Description	474
5.58.2 Field Documentation	475
5.58.2.1 bVbusADCModule	475
5.58.2.2 bVbusADChannel	475
5.58.2.3 bVbusSamplingTime	475
5.58.2.4 hLowPassFilterBW	475
5.58.2.5 hOverVoltageThreshold	475
5.58.2.6 hUnderVoltageThreshold	475
5.59 RESOLVERParams_t Struct Reference	475
5.59.1 Detailed Description	475
5.59.2 Field Documentation	476
5.59.2.1 eTimerModule	476
5.59.2.2 RES_Excitation_eTimer_ch	477
5.59.2.3 RES_Filtering_shift	477

5.59.2.4	RES_Excitation_Signal_freq	477
5.59.2.5	Res_SIN_ADC_Module	477
5.59.2.6	Res_SIN_ADC_Channel	477
5.59.2.7	Res_COS_ADC_Module	477
5.59.2.8	Res_COS_ADC_Channel	477
5.59.2.9	hSpeedSamplingFreqHz	477
5.59.2.10	DmaChannel	477
5.59.2.11	hPhaseShift	478
5.59.2.12	hResSinPhaseChannelB	478
5.59.2.13	hResCosPhaseChannelB	478
5.59.2.14	RES_Angle_Reading_Type	478
5.59.2.15	hResPole2MotorDegreesMultiplier	478
5.59.2.16	hResMinADCChannelA	478
5.59.2.17	hResMaxADCChannelA	478
5.59.2.18	hResMinADCChannelB	478
5.59.2.19	hResMaxADCChannelB	479
5.60	RevupCtrlParams_t Struct Reference	479
5.60.1	Detailed Description	479
5.60.2	Field Documentation	479
5.60.2.1	hRUCFrequencyHz	479
5.60.2.2	hStartingMecAngle	480
5.60.2.3	pPhaseParam	480
5.61	RUCPhasesParams_t Struct Reference	480
5.61.1	Detailed Description	480
5.61.2	Field Documentation	480
5.61.2.1	hDurationms	480
5.61.2.2	hFinalMecSpeed01Hz	480
5.61.2.3	hFinalTorque	480
5.61.2.4	pNext	481
5.62	SpeednPosFdbkParams_t Struct Reference	481
5.62.1	Detailed Description	481
5.62.2	Field Documentation	481
5.62.2.1	bElToMecRatio	481
5.62.2.2	hMaxReliableMecSpeed01Hz	481
5.62.2.3	hMinReliableMecSpeed01Hz	481
5.62.2.4	bMaximumSpeedErrorsNumber	482
5.62.2.5	hMaxReliableMecAccel01HzP	482
5.62.2.6	hMeasurementFrequency	482
5.63	SpeednTorqCtrlParams_t Struct Reference	482
5.63.1	Detailed Description	482

5.63.2 Field Documentation	482
5.63.2.1 hSTCFrequencyHz	482
5.63.2.2 hMaxAppPositiveMecSpeed01Hz	483
5.63.2.3 hMinAppPositiveMecSpeed01Hz	483
5.63.2.4 hMaxAppNegativeMecSpeed01Hz	483
5.63.2.5 hMinAppNegativeMecSpeed01Hz	483
5.63.2.6 hMaxPositiveTorque	483
5.63.2.7 hMinNegativeTorque	483
5.63.2.8 bModeDefault	483
5.63.2.9 hMecSpeedRef01HzDefault	483
5.63.2.10 hTorqueRefDefault	483
5.63.2.11 hldrefDefault	484
5.64 STOPParams_t Struct Reference	484
5.64.1 Detailed Description	484
5.64.2 Field Documentation	485
5.64.2.1 hC1	485
5.64.2.2 hC2	485
5.64.2.3 hC3	485
5.64.2.4 hC4	485
5.64.2.5 hC5	485
5.64.2.6 hF1	486
5.64.2.7 hF2	486
5.64.2.8 PIParams	486
5.64.2.9 bSpeedBufferSize01Hz	486
5.64.2.10 bSpeedBufferSizedpp	486
5.64.2.11 hVariancePercentage	486
5.64.2.12 bSpeedValidationBand_H	486
5.64.2.13 bSpeedValidationBand_L	486
5.64.2.14 hMinStartUpValidSpeed	486
5.64.2.15 bStartUpConsistThreshold	487
5.64.2.16 bReliability_hysteresys	487
5.64.2.17 bBemfConsistencyCheck	487
5.64.2.18 bBemfConsistencyGain	487
5.64.2.19 hMaxAppPositiveMecSpeed01Hz	487
5.64.2.20 hF1LOG	487
5.64.2.21 hF2LOG	487
5.64.2.22 bSpeedBufferSizedppLOG	487
5.65 TempSensorParams_t Struct Reference	487
5.65.1 Detailed Description	488
5.65.2 Field Documentation	488

5.65.2.1	bSensorType	488
5.66	Vars_t Struct Reference	488
5.66.1	Detailed Description	488
5.66.2	Field Documentation	490
5.66.2.1	oSTM	490
5.66.2.2	oSTC	491
5.66.2.3	pFOCVars	491
5.66.2.4	lastCommand	491
5.66.2.5	hFinalSpeed	491
5.66.2.6	hFinalTorque	491
5.66.2.7	lqdref	491
5.66.2.8	hDurationms	492
5.66.2.9	CommandState	492
5.66.2.10	LastModalitySetByUser	492
5.66.2.11	oVSS	492
5.66.2.12	oENC	492
5.66.2.13	hRemainingTicks	492
5.66.2.14	EncAligned	492
5.66.2.15	EncRestart	493
5.66.2.16	hMeasBuffer	493
5.66.2.17	hNextMeasBufferIndex	493
5.66.2.18	hLastMeasBufferIndex	493
5.66.2.19	hAvgElMotorPowerW	493
5.66.2.20	hVoltage	493
5.66.2.21	oVSS	493
5.66.2.22	VFMode	493
5.66.2.23	Vqdff	494
5.66.2.24	VqdPlout	494
5.66.2.25	VqdAvPlout	494
5.66.2.26	wConstant_1D	494
5.66.2.27	wConstant_1Q	494
5.66.2.28	wConstant_2	494
5.66.2.29	oVBS	494
5.66.2.30	oPI_q	494
5.66.2.31	oPI_d	494
5.66.2.32	oFluxWeakeningPI	495
5.66.2.33	oSpeedPI	495
5.66.2.34	hFW_V_Ref	495
5.66.2.35	AvVolt_qd	495
5.66.2.36	AvVoltAmpl	495

5.66.2.37 hIdRefOffset	495
5.66.2.38 hT_Sqrt3	495
5.66.2.39 hSector	495
5.66.2.40 bTurnOnLowSidesAction	496
5.66.2.41 hOffCalibrWaitTimeCounter	496
5.66.2.42 bMotor	496
5.66.2.43 hPhaseRemainingTicks	496
5.66.2.44 pPhaseParams	496
5.66.2.45 hDirection	496
5.66.2.46 bStageCnt	496
5.66.2.47 ParamsData	497
5.66.2.48 bPhaseNbr	497
5.66.2.49 bMode	497
5.66.2.50 hTargetFinal	497
5.66.2.51 wSpeedRef01HzExt	497
5.66.2.52 wTorqueRef	497
5.66.2.53 hRampRemainingStep	497
5.66.2.54 oPISpeed	498
5.66.2.55 oSPD	498
5.66.2.56 wIncDecAmount	498
5.66.2.57 bState	498
5.66.2.58 hFaultNow	498
5.66.2.59 hFaultOccurred	498
5.66.2.60 hFaultState	498
5.66.2.61 hLatestConv	499
5.66.2.62 bDriveNum	499
5.66.2.63 pMCI	499
5.66.2.64 pMCT	499
5.66.2.65 pUICfg	499
5.66.2.66 bSelectedDrive	499
5.67 VirtualParams_t Struct Reference	499
5.67.1 Detailed Description	499
5.67.2 Field Documentation	500
5.67.2.1 hExpectedVbus_d	500
5.68 VirtualSpeedSensorParams_t Struct Reference	500
5.68.1 Detailed Description	500
5.68.2 Field Documentation	500
5.68.2.1 hSpeedSamplingFreqHz	500
5.69 VirtualTParams_t Struct Reference	500
5.69.1 Detailed Description	500

5.69.2 Field Documentation	501
5.69.2.1 hExpectedTemp_d	501
5.69.2.2 hExpectedTemp_C	501
5.70 _CUI_t Struct Reference	501
5.70.1 Detailed Description	501
5.70.2 Field Documentation	502
5.70.2.1 Methods_str	502
5.70.2.2 Vars_str	503
5.70.2.3 pParams_str	503
5.70.2.4 DerivedClass	503
5.71 CANParams_t Struct Reference	503
5.71.1 Detailed Description	503
5.72 FCPPParams_t Struct Reference	503
5.72.1 Detailed Description	503
5.73 MCPParams_t Struct Reference	503
5.73.1 Detailed Description	503
5.74 Methods_t Struct Reference	504
5.74.1 Detailed Description	504
5.75 USART_TypeDef Struct Reference	504
5.75.1 Detailed Description	504
5.76 USARTParams_t Struct Reference	504
5.76.1 Detailed Description	504
5.77 Vars_t Struct Reference	505
5.77.1 Detailed Description	505
5.77.2 Field Documentation	507
5.77.2.1 oSTM	507
5.77.2.2 oSTC	508
5.77.2.3 pFOCVars	508
5.77.2.4 lastCommand	508
5.77.2.5 hFinalSpeed	508
5.77.2.6 hFinalTorque	508
5.77.2.7 Iqdref	508
5.77.2.8 hDurationms	509
5.77.2.9 CommandState	509
5.77.2.10 LastModalitySetByUser	509
5.77.2.11 oVSS	509
5.77.2.12 oENC	509
5.77.2.13 hRemainingTicks	509
5.77.2.14 EncAligned	509
5.77.2.15 EncRestart	510

5.77.2.16 hMeasBuffer	510
5.77.2.17 hNextMeasBufferIndex	510
5.77.2.18 hLastMeasBufferIndex	510
5.77.2.19 hAvgElMotorPowerW	510
5.77.2.20 hVoltage	510
5.77.2.21 oVSS	510
5.77.2.22 VFMode	510
5.77.2.23 Vqdff	511
5.77.2.24 VqdPlout	511
5.77.2.25 VqdAvPlout	511
5.77.2.26 wConstant_1D	511
5.77.2.27 wConstant_1Q	511
5.77.2.28 wConstant_2	511
5.77.2.29 oVBS	511
5.77.2.30 oPI_q	511
5.77.2.31 oPI_d	511
5.77.2.32 oFluxWeakeningPI	512
5.77.2.33 oSpeedPI	512
5.77.2.34 hFW_V_Ref	512
5.77.2.35 AvVolt_qd	512
5.77.2.36 AvVoltAmpl	512
5.77.2.37 hIdRefOffset	512
5.77.2.38 hT_Sqrt3	512
5.77.2.39 hSector	512
5.77.2.40 bTurnOnLowSidesAction	513
5.77.2.41 hOffCalibrWaitTimeCounter	513
5.77.2.42 bMotor	513
5.77.2.43 hPhaseRemainingTicks	513
5.77.2.44 pPhaseParams	513
5.77.2.45 hDirection	513
5.77.2.46 bStageCnt	513
5.77.2.47 ParamsData	514
5.77.2.48 bPhaseNbr	514
5.77.2.49 bMode	514
5.77.2.50 hTargetFinal	514
5.77.2.51 wSpeedRef01HzExt	514
5.77.2.52 wTorqueRef	514
5.77.2.53 hRampRemainingStep	514
5.77.2.54 oPISpeed	515
5.77.2.55 oSPD	515

5.77.2.56 wIncDecAmount	515
5.77.2.57 bState	515
5.77.2.58 hFaultNow	515
5.77.2.59 hFaultOccurred	515
5.77.2.60 hFaultState	515
5.77.2.61 hLatestConv	516
5.77.2.62 bDriveNum	516
5.77.2.63 pMCI	516
5.77.2.64 pMCT	516
5.77.2.65 pUICfg	516
5.77.2.66 bSelectedDrive	516

Chapter 1

SPC5 Motor Control Tool Kit

What is a SPC5 Motor Control Component?

This component is composed by four header configuration files that are automatically generated in accordance with the SPC5Studio interface setting; it is also composed from a software library that include file.c and file.h that allow implementing the FOC algorithm, testing and characterizing the Motor.

The software library implements the Field Oriented Control (FOC) drive of 3-phase Permanent Magnet Synchronous Motors (PMSM), both Surface Mounted (SM-PMSM) and Internal (I-PMSM).

The library exploit a sensorless technique which start in open loop.

The PMSM FOC library can be used to quickly evaluate ST microcontrollers and complete ST application platforms, and to save time when developing Motor Control algorithms to be run on ST microcontrollers. It is written in C language, and implements the core Motor Control algorithms as well as sensor reading/decoding algorithms and a sensorless algorithm for rotor position reconstruction.

Features (in bold are already implemented)

- Single or simultaneous Dual **PMSM FOC sensorless / sensored**.
- **Speed feedback**
- Sensorless:
 - High Frequency Injection HFI (Patent pending) plus B-EMF State Observer, PLL rotor speed/angle computation from B-EMF.
 - **B-EMF State Observer, PLL rotor speed/angle computation from B-EMF.**
 - B-EMF State Observer, CORDIC rotor angle computation from B-EMF.
- 60 or **120 degrees displaced Hall sensors decoding, rising/falling edge responsiveness**.
- **Quadrature incremental encoder**.
- Resolver (Available but not fully tested).
- Current sampling methods:
 - **Two shunts (selectable on the Legs or on the Phases)**.
 - **Single, common DC-link shunt resistor (ST patented)**.
 - Three-shunt resistors placed on the bottom of the three inverter legs.
- **Torque control mode, speed control mode**.
- Brake strategies (optional):
 - Dissipative DC link brake resistor handling.

- Optimized I-PMSM and SM-PMSM drive.
- Programmable speed ramps (parameters duration and final target).
- Programmable torque ramps (parameters duration and final target).
- Real-time fine tuning of:
 - PID regulators.
 - Flux weakening algorithm.
 - Start-up procedure (in case of Encoder or sensorless).
- Fault conditions management:
 - Over-current.
 - Over and Under-voltage.
 - Over-temperature.
 - Speed feedback reliability error.
 - FOC algorithm execution overrun.
- User Interface: Serial and CAN Communication support.
- Easy customization of options, pin-out assignments, CPU clock frequency through SPC5Studio GUI.
- C language code:
 - MISRA-C 2012 rules check.
 - Conforms strictly with ISO/ANSI.
 - Object-oriented programming architecture.

Chapter 2

SPC5 PMSM MC Application

2.1 MCInterface class private methods

Collaboration diagram for MCInterface class private methods:



Functions

- **CMCI MCI_NewObject (pMCInterfaceParams_t pMCInterfaceParams)**
Creates an object of the class MCInterface.
- void **MCI_Init (CMCI this, CSTM oSTM, CSTC oSTC, pFOCVars_t pFOCVars)**
Initializes all the object variables, usually it has to be called once right after object creation. It is also used to assign the state machine object, the speed and torque controller, and the FOC drive object to be used by MC Interface.
- void **MCI_ExecBufferedCommands (CMCI this)**
This is usually a method managed by task. It must be called periodically in order to check the status of the related oSTM object and eventually to execute the buffered command if the condition occurs.

2.1.1 Function Documentation

2.1.1.1 CMCI MCI_NewObject (pMCInterfaceParams_t pMCInterfaceParams)

Creates an object of the class MCInterface.

Parameters

<i>pMCInterface- Params</i>	pointer to an MCInterface parameters structure
---------------------------------	--

Return values

<i>CMCI</i>	new instance of MCInterface object
-------------	------------------------------------

Definition at line 63 of file MCInterfaceClass.c.

References MC_NULL, and _CMCI_t::pParams_str.

2.1.1.2 void MCI_Init (CMCI *this*, CSTM *oSTM*, CSTC *oSTC*, pFOCVars_t *pFOCVars*)

Initializes all the object variables, usually it has to be called once right after object creation. It is also used to assign the state machine object, the speed and torque controller, and the FOC drive object to be used by MC Interface.

Parameters

<i>this</i>	related object of class CMCI.
<i>oSTM</i>	the state machine object used by the MCI.
<i>oSTC</i>	the speed and torque controller used by the MCI.
<i>pFOCVars</i>	pointer to FOC vars to be used by MCI.

Return values

<i>none.</i>

Definition at line 96 of file MCInterfaceClass.c.

References Vars_t::CommandState, Vars_t::hDurationms, Vars_t::hFinalSpeed, Vars_t::hFinalTorque, Vars_t::lastCommand, MCI_BUFFER_EMPTY, MCI_NOCOMMANDSYET, Vars_t::oSTC, Vars_t::oSTM, and Vars_t::pFOCVars.

2.1.1.3 void MCI_ExecBufferedCommands (CMCI *this*)

This is usually a method managed by task. It must be called periodically in order to check the status of the related oSTM object and eventually to execute the buffered command if the condition occurs.

Parameters

<i>this</i>	related object of class CMCI.
-------------	-------------------------------

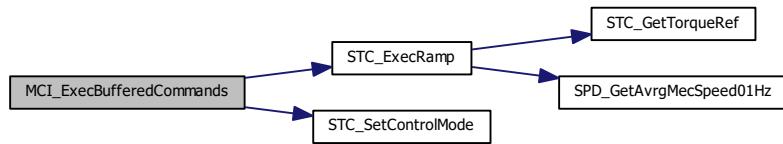
Return values

<i>none.</i>

Definition at line 282 of file MCInterfaceClass.c.

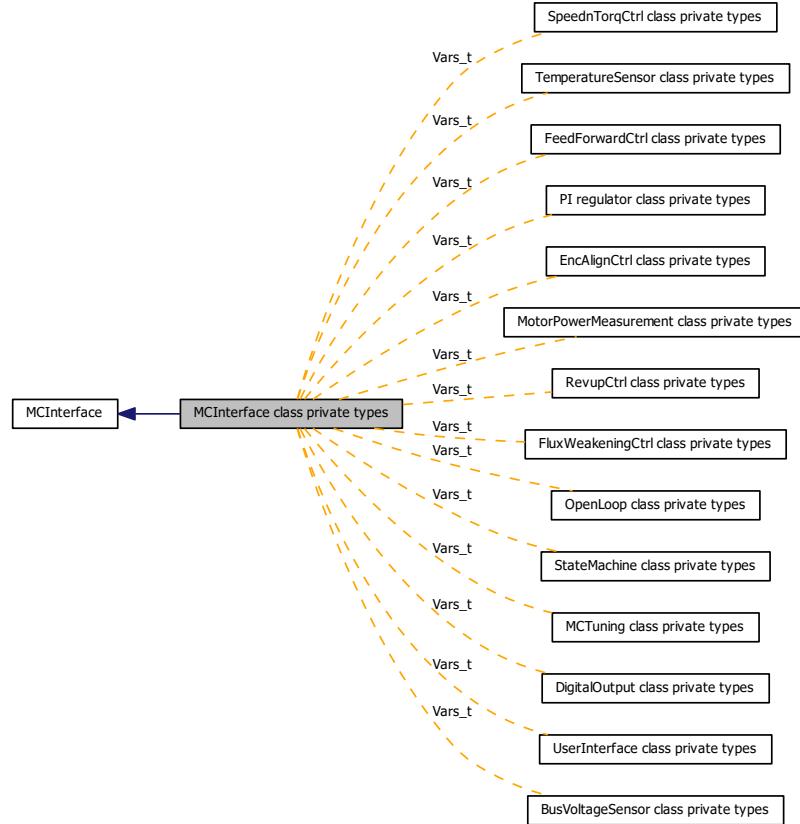
References FOCVars_t::bDriveInput, Vars_t::CommandState, Vars_t::hDurationms, Vars_t::hFinalSpeed, Vars_t::hFinalTorque, Vars_t::lqdref, FOCVars_t::lqdref, Vars_t::lastCommand, MCI_COMMAND_EXECUTED_SUCCESSFULLY, MCI_COMMAND_EXECUTED_UNSUCCESSFULLY, MCI_COMMAND_NOT_ALREADY_EXECUTED, MCI_EXECSPEEDRAMP, MCI_EXECTORQUERAMP, MCI_SETCURRENTREFERENCES, Vars_t::oSTC, Vars_t::pFOCVars, STC_ExecRamp(), STC_SetControlMode(), STC_SPEED_MODE, and STC_TORQUE_MODE.

Here is the call graph for this function:



2.2 MCInterface class private types

Collaboration diagram for MCInterface class private types:



Data Structures

- struct [Vars_t](#)
MCInterface class members definition.
- struct [_CMCI_t](#)
Private MCInterface class definition.

TypeDefs

- typedef [MCInterfaceParams_t Params_t](#)
Redefinition of parameter structure.

2.2.1 Typedef Documentation

2.2.1.1 typedef MCInterfaceParams_t Params_t

Redefinition of parameter structure.

Definition at line 73 of file MCInterfacePrivate.h.

2.3 MCTuning class exported types

Collaboration diagram for MCTuning class exported types:



Data Structures

- struct [MCTuningInitStruct_t](#)
MC tuning internal objects initialization structure type;

Typedefs

- typedef struct CMCT_t * [CMCT](#)
Public MCTuning class definition.
- typedef const void [MCTuningParams_t](#)
MCTuning class parameters definition.
- typedef struct COL_t * [COL](#)
Public OpenLoop class definition.
- typedef struct CPI_t * [CPI](#)
Public PI regulator class definition.
- typedef struct CPID_PI_t * [CPID_PI](#)
Public PID class definition.
- typedef struct CPWMC_t * [CPWMC](#)
Public PWMnCurrFdbk class definition.
- typedef struct CRUC_t * [CRUC](#)
Public RevupCtrl class definition.
- typedef struct CSPD_t * [CSPD](#)
Public SpeednPosFdbk class definition.
- typedef struct CSTO_SPD_t * [CSTO_SPD](#)
Public STO class definition.
- typedef struct CSTO_CR_SPD_t * [CSTO_CR_SPD](#)
Public STO_CORDIC class definition.
- typedef struct CSTC_t * [CSTC](#)
Public SpeednTorqCtrl class definition.
- typedef struct CSTM_t * [CSTM](#)
Public StateMachine class definition.
- typedef struct CTSNS_t * [CTSNS](#)
Public temperature sensor class definition.
- typedef struct CVBS_t * [CVBS](#)
Public BusVoltageSensor class definition.
- typedef struct CDOUT_t * [CDOUT](#)
Public DigitalOutput class definition.

- **typedef struct CMPM_t * CMPM**
Public MotorPowerMeasurement class definition.
- **typedef struct CFW_t * CFW**
Public FluxWeakeningCtrl class definition.
- **typedef struct CFF_t * CFF**
Public FeedForwardCtrl class definition.
- **typedef struct CHFI_FP_t * CHFI_FP**
Public HiFreqInj_FPU_Ctrl class definition.

2.3.1 Typedef Documentation

2.3.1.1 **typedef struct CMCT_t* CMCT**

Public MCTuning class definition.

Definition at line 54 of file MCTuningClass.h.

2.3.1.2 **typedef const void MCTuningParams_t**

MCTuning class parameters definition.

Definition at line 59 of file MCTuningClass.h.

2.3.1.3 **typedef struct COL_t* COL**

Public OpenLoop class definition.

Definition at line 65 of file MCTuningClass.h.

2.3.1.4 **typedef struct CPI_t* CPI**

Public PI regulator class definition.

Definition at line 72 of file MCTuningClass.h.

2.3.1.5 **typedef struct CPID_PI_t* CPID_PI**

Public PID class definition.

Definition at line 78 of file MCTuningClass.h.

2.3.1.6 **typedef struct CPWMC_t* CPWMC**

Public PWMnCurrFdbk class definition.

Definition at line 85 of file MCTuningClass.h.

2.3.1.7 **typedef struct CRUC_t* CRUC**

Public RevupCtrl class definition.

Definition at line 92 of file MCTuningClass.h.

2.3.1.8 **typedef struct CSPD_t* **CSPD****

Public SpeednPosFdbk class definition.

Definition at line 99 of file MCTuningClass.h.

2.3.1.9 **typedef struct CSTO_SPD_t* **CSTO_SPD****

Public STO class definition.

Definition at line 106 of file MCTuningClass.h.

2.3.1.10 **typedef struct CSTO_CR_SPD_t* **CSTO_CR_SPD****

Public STO_CORDIC class definition.

Definition at line 113 of file MCTuningClass.h.

2.3.1.11 **typedef struct CSTC_t* **CSTC****

Public SpeednTorqCtrl class definition.

Definition at line 120 of file MCTuningClass.h.

2.3.1.12 **typedef struct CSTM_t* **CSTM****

Public StateMachine class definition.

Definition at line 127 of file MCTuningClass.h.

2.3.1.13 **typedef struct CTSNS_t* **CTSNS****

Public temperature sensor class definition.

Definition at line 134 of file MCTuningClass.h.

2.3.1.14 **typedef struct CVBS_t* **CVBS****

Public BusVoltageSensor class definition.

Definition at line 141 of file MCTuningClass.h.

2.3.1.15 **typedef struct CDOUT_t* **CDOUT****

Public DigitalOutput class definition.

Definition at line 148 of file MCTuningClass.h.

2.3.1.16 **typedef struct CMPM_t* **CMPM****

Public MotorPowerMeasurement class definition.

Definition at line 155 of file MCTuningClass.h.

2.3.1.17 `typedef struct CFW_t* CFW`

Public FluxWeakeningCtrl class definition.

Definition at line 162 of file MCTuningClass.h.

2.3.1.18 `typedef struct CFF_t* CFF`

Public FeedForwardCtrl class definition.

Definition at line 169 of file MCTuningClass.h.

2.3.1.19 `typedef struct CHFI_FP_t* CHFI_FP`

Public HiFreqInj_FPU_Ctrl class definition.

Definition at line 176 of file MCTuningClass.h.

2.4 MCTuning class private methods

Collaboration diagram for MCTuning class private methods:



Functions

- [CMCT MCT_NewObject](#) (`pMCTuningParams_t pMCTuningParams`)
Creates an object of the class MCTuning.
- `void MCT_Init (CMCT this, MCTuningInitStruct_t MCTuningInitStruct)`
Example of public method of the class MCTuning.

2.4.1 Function Documentation

2.4.1.1 CMCT MCT_NewObject (`pMCTuningParams_t pMCTuningParams`)

Creates an object of the class MCTuning.

Parameters

<code>pMCTuning- Params</code>	pointer to an MCTuning parameters structure
------------------------------------	---

Return values

<code>CMCT</code>	new instance of MCTuning object
-------------------	---------------------------------

Definition at line 50 of file MCTuningClass.c.

References `MC_NULL`, and `_CMCT_t::pParams_str`.

2.4.1.2 void MCT_Init (`CMCT this, MCTuningInitStruct_t MCTuningInitStruct`)

Example of public method of the class MCTuning.

Parameters

<code>this</code>	related object of class CMCT
-------------------	------------------------------

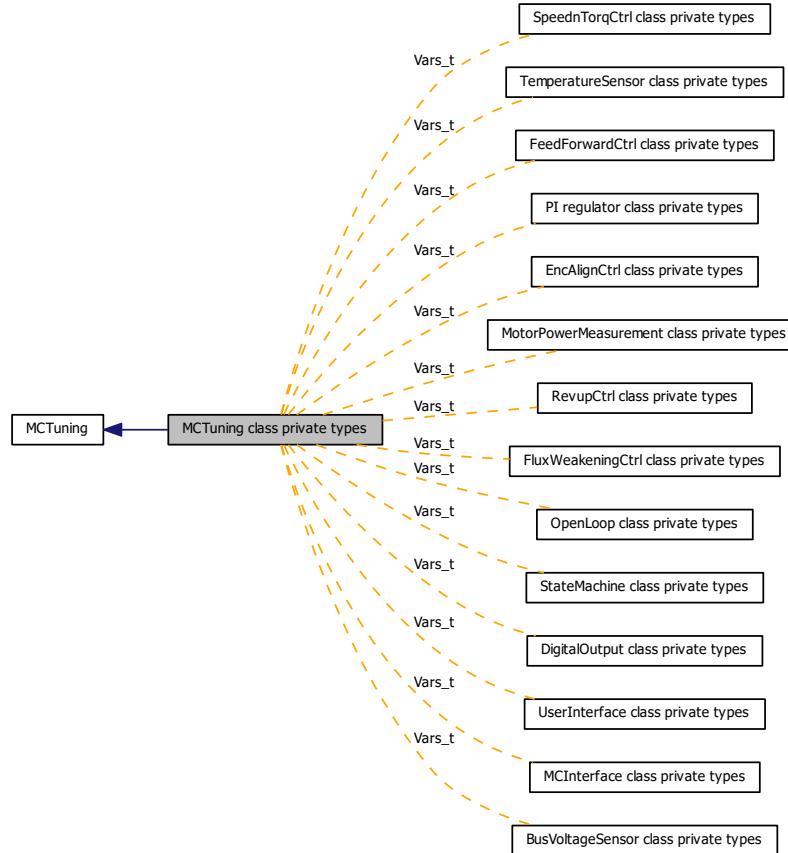
Return values

<code>none</code>

Definition at line 77 of file MCTuningClass.c.

2.5 MCTuning class private types

Collaboration diagram for MCTuning class private types:



Data Structures

- struct `Vars_t`
MCInterface class members definition.
- struct `_CMCT_t`
Private MCTuning class definition.

TypeDefs

- typedef `MCTuningParams_t Params_t`
Redefinition of parameter structure.

2.5.1 Typedef Documentation

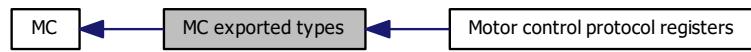
2.5.1.1 typedef `MCTuningParams_t Params_t`

Redefinition of parameter structure.

Definition at line 58 of file `MCTuningPrivate.h`.

2.6 MC exported types

Collaboration diagram for MC exported types:



Modules

- Motor control protocol registers

2.7 Motor control protocol registers

Collaboration diagram for Motor control protocol registers:



2.8 MC exported methods

Collaboration diagram for MC exported methods:



Functions

- **CMCI GetMCI** (uint8_t bMotor)

This function returns the reference of the MCInterface relative to the selected drive.

- **CMCT GetMCT** (uint8_t bMotor)

This function returns the reference of the MCTuning relative to the selected drive.

- void **MC_RequestRegularConv** (uint8_t bAdc_mod, uint8_t bChannel, uint8_t bSamplTime)

This function requests a user-defined regular conversion. All user defined conversion requests must be performed inside routines with the same priority level. If previous regular conversion request is pending this function has no effect, for this reason is better to call the MC-RegularConvState and check if the state is UDRC_STATE_IDLE before to call MC_RequestRegularConv.

- uint16_t **MC_GetRegularConv** (void)

Get the last user-defined regular conversion.

- uint16_t **MC_GetMultipleRegularConv** (uint8_t ADC_channel)

Get the ADC channel user-defined regular conversion. The ADC conversion data is stored in buffer with size equal to Number of conversion. The ADC User conversion shall have a period of PWMfreq/Number of conversion.

- UDRC_State_t **MC-RegularConvState** (void)

Use this function to know the status of the last requested regular conversion.

- void **MC_SetDAC** (DAC_Channel_t bChannel, MC_Protocol_REG_t bVariable)

This method is used to set up the DAC outputs. The selected variables will be provided in the related output channels.

- void **MC_SetUserDAC** (DAC_UserChannel_t bUserChNumber, int16_t hValue)

This method is used to set the value of the "User DAC channel".

2.8.1 Function Documentation

2.8.1.1 CMCI GetMCI (uint8_t bMotor)

This function returns the reference of the MCInterface relative to the selected drive.

Parameters

bMotor	Motor reference number defined here
--------	---

Return values

CMCI	Reference to MCInterface relative to the selected drive. Note: it can be MC_NULL if MCInterface of selected drive is not allocated.
------	---

Definition at line 2548 of file MCTasks.c.

References MC_NULL.

2.8.1.2 CMCT GetMCT (uint8_t bMotor)

This function returns the reference of the MCTuning relative to the selected drive.

Parameters

bMotor	Motor reference number defined here
--------	---

Return values

CMCI	Reference to MCInterface relative to the selected drive. Note: it can be MC_NULL if MCInterface of selected drive is not allocated.
------	---

Definition at line 2567 of file MCTasks.c.

References MC_NULL.

2.8.1.3 void MC_RequestRegularConv (uint8_t bAdc_mod, uint8_t bChannel, uint8_t bSampTime)

This function requests a user-defined regular conversion. All user defined conversion requests must be performed inside routines with the same priority level. If previous regular conversion request is pending this function has no effect, for this reason is better to call the MC-RegularConvState and check if the state is UDRC_STATE_IDLE before to call MC_RequestRegularConv.

Parameters

bAdc_mod	ADC module used for the regular conversion.
bChannel	ADC channel used for the regular conversion.
bSampTime	Sampling time selection, ADC_SampleTime_nCycles.

Definition at line 2252 of file MCTasks.c.

References ADConv_t::Adc_Unit, ADConv_t::hiChannel, ADConv_t::hSamplingTime, and PWMC_ADC_SetSamplingTime().

Here is the call graph for this function:



2.8.1.4 uint16_t MC_GetRegularConv (void)

Get the last user-defined regular conversion.

Return values

uint16_t	It returns converted value or 0xFFFF for conversion error. This function returns a valid result if the state returned by MC-RegularConvState is UDRC_STATE_EOC.
----------	---

Definition at line 2273 of file MCTasks.c.

2.8.1.5 `uint16_t MC_GetMultipleRegularConv (uint8_t ADC_channel)`

Get the ADC channel user-defined regular conversion. The ADC conversion data is stored in buffer with size equal to Number of conversion. The ADC User conversion shall have a period of PWMfreq/Number of conversion.

Parameters

<code>ADC_channel</code>	ADC channel used for the regular conversion.
--------------------------	--

Return values

<code>uint16_t</code>	It returns converted value or 0xFFFF for conversion error.
-----------------------	--

Definition at line 2295 of file MCTasks.c.

References PWMC_GetMultipleRegularConv().

Here is the call graph for this function:



2.8.1.6 `UDRC_State_t MC-RegularConvState (void)`

Use this function to know the status of the last requested regular conversion.

Return values

<code>UDRC_State_t</code>	The state of the last user-defined regular conversion. It can be one of the following values: UDRC_STATE_IDLE no regular conversion request pending. UDRC_STATE_REQUESTED regular conversion has been requested and not completed. UDRC_STATE_EOC regular conversion has been completed but not readed from the user.
---------------------------	--

Definition at line 2312 of file MCTasks.c.

2.8.1.7 `void MC_SetDAC (DAC_Channel_t bChannel, MC_Protocol_REG_t bVariable)`

This method is used to set up the DAC outputs. The selected variables will be provided in the related output channels.

Parameters

<code>bChannel</code>	the DAC channel to be programmed. It must be one of the exported channels decribed here. E.g. DAC_CH0.
<code>bVariable</code>	the variables to be provided in out through the selected channel. It must be one of the exported UI register described here . E.g. MC_PROTOCOL_REG_I_A.

Return values

<i>none.</i>

Definition at line 324 of file UITask.c.

References UI_SetDAC().

Here is the call graph for this function:



2.8.1.8 void MC_SetUserDAC (DAC_UserChannel_t bUserChNumber, int16_t hValue)

This method is used to set the value of the "User DAC channel".

Parameters

<i>bUserChNumber</i>	the "User DAC channel" to be programmed. One of the channels described here can be used. E.g. DAC_USER1.
<i>hValue</i>	16bit signed value to be put in output.

Return values

<i>none.</i>

Definition at line 329 of file UITask.c.

References UI_SetUserDAC().

Here is the call graph for this function:



2.9 MCInterface class exported types

Collaboration diagram for MCInterface class exported types:



TypeDefs

- `typedef struct CMCI_t * CMCI`
Public MCInterface class definition.
- `typedef const void MCInterfaceParams_t`
MCInterface class parameters definition.

Enumerations

- `enum CommandState_t { MCI_BUFFER_EMPTY, MCI_COMMAND_NOT_ALREADY_EXECUTED, MCI_COMMAND_EXECUTED_SUCCESFULLY, MCI_COMMAND_EXECUTED_UNSUCCESFULLY }`
- `enum UserCommands_t { MCI_NOCOMMANDSYET, MCI_EXECSPEEDRAMP, MCI_EXECTORQUERAMP, MCI_SETCURRENTREFERENCES }`

2.9.1 Typedef Documentation

2.9.1.1 `typedef struct CMCI_t* CMCI`

Public MCInterface class definition.

Definition at line 71 of file MCInterfaceClass.h.

2.9.1.2 `typedef const void MCInterfaceParams_t`

MCInterface class parameters definition.

Definition at line 76 of file MCInterfaceClass.h.

2.9.2 Enumeration Type Documentation

2.9.2.1 `enum CommandState_t`

Enumerator:

`MCI_BUFFER_EMPTY` If no buffered command has been called.

`MCI_COMMAND_NOT_ALREADY_EXECUTED` If the buffered command condition hasn't already occurred.

`MCI_COMMAND_EXECUTED_SUCCESFULLY` If the buffered command has been executed successfully.

`MCI_COMMAND_EXECUTED_UNSUCCESFULLY` If the buffered command has been executed unsuccessfully.

Definition at line 50 of file MCInterfaceClass.h.

2.9.2.2 enum UserCommands_t

Enumerator:

MCI_NOCOMMANDSYET No command has been set by the user.

MCI_EXECSPEEDRAMP ExecSpeedRamp command coming from the user.

MCI_EXECUTORQUERAMP ExecTorqueRamp command coming from the user.

MCI_SETCURRENTREFERENCES SetCurrentReferences command coming from the user.

Definition at line 61 of file MCInterfaceClass.h.

2.10 MCInterface class exported methods

Collaboration diagram for MCInterface class exported methods:



Functions

- void [MCI_ExecSpeedRamp](#) ([CMCI](#) this, int16_t hFinalSpeed, uint16_t hDurationms)

This is a buffered command to set a motor speed ramp. This commands don't become active as soon as it is called but it will be executed when the oSTM state is START_RUN or RUN. User can check the status of the command calling the MCI_IsCommandAcknowledged method.

- void [MCI_ExecTorqueRamp](#) ([CMCI](#) this, int16_t hFinalTorque, uint16_t hDurationms)

This is a buffered command to set a motor torque ramp. This commands don't become active as soon as it is called but it will be executed when the oSTM state is START_RUN or RUN. User can check the status of the command calling the MCI_IsCommandAcknowledged method.

- void [MCI_SetCurrentReferences](#) ([CMCI](#) this, [Curr_Components](#) lqdref)

This is a buffered command to set directly the motor current references Iq and Id. This commands don't become active as soon as it is called but it will be executed when the oSTM state is START_RUN or RUN. User can check the status of the command calling the MCI_IsCommandAcknowledged method.

- bool [MCI_StartMotor](#) ([CMCI](#) this)

This is a user command used to begin the start-up procedure. If the state machine is in IDLE state the command is executed instantaneously otherwise the command is discarded. User must take care of this possibility by checking the return value. Before calling MCI_StartMotor it is mandatory to execute one of these commands:

[MCI_ExecSpeedRamp](#)
[MCI_ExecTorqueRamp](#)
[MCI_SetCurrentReferences](#)

Otherwise the behaviour in run state will be unpredictable.

Note: The MCI_StartMotor command is used just to begin the start-up procedure moving the state machine from IDLE state to IDLE_START. The command MCI_StartMotor is not blocking the execution of project until the motor is really running; to do this, the user have to check the state machine and verify that the RUN state (or any other state) has been reached.

- bool [MCI_StopMotor](#) ([CMCI](#) this)

This is a user command used to begin the stop motor procedure. If the state machine is in RUN or START states the command is executed instantaneously otherwise the command is discarded. User must take care of this possibility by checking the return value.

Note: The MCI_StopMotor command is used just to begin the stop motor procedure moving the state machine to ANY_STOP. The command MCI_StopMotor is not blocking the execution of project until the motor is really stopped; to do this, the user have to check the state machine and verify that the IDLE state has been reached again.

- bool [MCI_FaultAcknowledged](#) ([CMCI](#) this)

This is a user command used to indicate that the user has seen the error condition. If is possible, the command is executed instantaneously otherwise the command is discarded. User must take care of this possibility by checking the return value.

- bool [MCI_EncoderAlign](#) ([CMCI](#) this)

This is a user command used to begin the encoder alignment procedure. If the state machine is in IDLE state the command is executed instantaneously otherwise the command is discarded. User must take care of this possibility by checking the return value.

Note: The MCI_EncoderAlign command is used just to begin the encoder alignment procedure moving the state machine from IDLE state to IDLE_ALIGNMENT. The command MCI_EncoderAlign is not blocking the execution of

project until the encoder is really calibrated; to do this, the user have to check the state machine and verify that the IDLE state has been reached again.

- **CommandState_t MCI_IsCommandAcknowledged (CMCI this)**

It returns information about the state of the last buffered command.

- **State_t MCI_GetSTMState (CMCI this)**

It returns information about the state of the related oSTM object.

- **uint16_t MCI_GetOccurredFaults (CMCI this)**

It returns a 16 bit fields containing information about faults historically occurred since the state machine has been moved into FAULT_NOW state.

Returned error codes are listed here .

- **uint16_t MCI_GetCurrentFaults (CMCI this)**

It returns a 16 bit fields containing information about faults currently present.

Returned error codes are listed here .

- **int16_t MCI_GetTorqueRef (CMCI this)**

*It returns information about the current motor torque reference. This value represents actually the Iq current reference expressed in digit. To convert current expressed in digit to current expressed in Amps is possible to use the formula:
Current(Amp) = [Current(digit) * Vdd micro] / [65536 * Rshunt * Aop].*

- **int16_t MCI_GetTorque (CMCI this)**

*It returns information about current motor measured torque. This value represents actually the Iq current expressed in digit. To convert current expressed in digit to current expressed in Amps is possible to use the formula: Current(Amp) = [Current(digit) * Vdd micro] / [65536 * Rshunt * Aop].*

- **STC_Modality_t MCI_GetControlMode (CMCI this)**

It returns the modality of the speed and torque controller.

- **int16_t MCI_GetImposedMotorDirection (CMCI this)**

It returns the motor direction imposed by the last command (MCI_ExecSpeedRamp, MCI_ExecTorqueRamp or MCI_SetCurrentReferences).

- **int16_t MCI_GetLastRampFinalSpeed (CMCI this)**

It returns information about the last ramp final speed sent by the user expressed in tenths of HZ.

- **bool MCI_RampCompleted (CMCI this)**

Check if the settled speed or torque ramp has been completed.

- **bool MCI_GetSpdSensorReliability (CMCI this)**

It returns speed sensor reliability with reference to the sensor actually used for reference frame transformation.

- **int16_t MCI_GetAvrgMecSpeed01Hz (CMCI this)**

It returns the last computed average mechanical speed, expressed in 01Hz (tenth of Hertz) and related to the sensor actually used by FOC algorithm.

- **int16_t MCI_GetMecSpeedRef01Hz (CMCI this)**

Get the current mechanical rotor speed reference expressed in tenths of HZ.

- **Curr_Components MCI_Getlab (CMCI this)**

It returns stator current lab in Curr_Components format.

- **Curr_Components MCI_Getlalphabet (CMCI this)**

It returns stator current lalphabet in Curr_Components format.

- **Curr_Components MCI_Getlqd (CMCI this)**

It returns stator current lqd in Curr_Components format.

- **Curr_Components MCI_GetlqdHF (CMCI this)**

It returns stator current lqdHF in Curr_Components format.

- **Curr_Components MCI_Getlqdref (CMCI this)**

It returns stator current lqdref in Curr_Components format.

- **Volt_Components MCI_GetVqd (CMCI this)**

It returns stator current Vqd in Volt_Components format.

- **Volt_Components MCI_GetValphabet (CMCI this)**

It returns stator current Valphabet in Volt_Components format.

- **int16_t MCI_GetElAngledpp (CMCI this)**

It returns the rotor electrical angle actually used for reference frame transformation.

- int16_t **MCI_GetTeref** (**CMCI** this)

*It returns the reference electrical torque, fed to derived class for *Iqref* and *Idref* computation.*
- int16_t **MCI_GetPhaseCurrentAmplitude** (**CMCI** this)

*It returns the motor phase current amplitude (0-to-peak) in s16A To convert s16A into Ampere following formula must be used: Current(Amp) = [Current(s16A) * Vdd micro] / [65536 * Rshunt * Aop].*
- int16_t **MCI_GetPhaseVoltageAmplitude** (**CMCI** this)

*It returns the applied motor phase voltage amplitude (0-to-peak) in s16V. To convert s16V into Volts following formula must be used: PhaseVoltage(V) = [PhaseVoltage(s16A) * Vbus(V)] / [sqrt(3) * 32767].*
- void **MCI_SetIdref** (**CMCI** this, int16_t hNewIdref)

*When *bDriveInput* is set to INTERNAL, *Idref* should be normally managed by *FOC_CalcCurrRef*. Nevertheless, this method allows forcing changing *Idref* value. Method call has no effect when either flux weakening region is entered or MTPA is enabled.*
- void **MCI_Clear_Iqdref** (**CMCI** this)

*It re-initializes *Iqdref* variables with their default values.*

2.10.1 Function Documentation

2.10.1.1 void **MCI_ExecSpeedRamp** (**CMCI** this, int16_t hFinalSpeed, uint16_t hDurationms)

This is a buffered command to set a motor speed ramp. This commands don't become active as soon as it is called but it will be executed when the oSTM state is START_RUN or RUN. User can check the status of the command calling the **MCI_IsCommandAcknowledged** method.

Parameters

<i>this</i>	related object of class CMCI.
<i>hFinalSpeed</i>	is the value of mechanical rotor speed reference at the end of the ramp expressed in tenths of HZ.
<i>hDurationms</i>	the duration of the ramp expressed in milliseconds. It is possible to set 0 to perform an instantaneous change in the value.

Return values

<i>none.</i>

Definition at line 125 of file MCInterfaceClass.c.

References *Vars_t::CommandState*, *Vars_t::hDurationms*, *Vars_t::hFinalSpeed*, *Vars_t::lastCommand*, *Vars_t::LastModalitySetByUser*, **MCI_COMMAND_NOT_ALREADY_EXECUTED**, **MCI_EXECSPEEDRAMP**, and **STC_SPEED_MODE**.

Referenced by **UI_ExecCmd()**, **UI_ExecSpeedRamp()**, and **UI_SetReg()**.

2.10.1.2 void **MCI_ExecTorqueRamp** (**CMCI** this, int16_t hFinalTorque, uint16_t hDurationms)

This is a buffered command to set a motor torque ramp. This commands don't become active as soon as it is called but it will be executed when the oSTM state is START_RUN or RUN. User can check the status of the command calling the **MCI_IsCommandAcknowledged** method.

Parameters

<i>this</i>	related object of class CMCI.
<i>hFinalTorque</i>	is the value of motor torque reference at the end of the ramp. This value represents actually the <i>Iq</i> current expressed in digit. To convert current expressed in Amps to current expressed in digit is possible to use the formula: Current (digit) = [Current(Amp) * 65536 * Rshunt * Aop] / Vdd micro.
<i>hDurationms</i>	the duration of the ramp expressed in milliseconds. It is possible to set 0 to perform an instantaneous change in the value.

Return values

<i>none.</i>

Definition at line 153 of file MCInterfaceClass.c.

References `Vars_t::CommandState`, `Vars_t::hDurationms`, `Vars_t::hFinalTorque`, `Vars_t::lastCommand`, `Vars_t::LastModalitySetByUser`, `MCI_COMMAND_NOT_ALREADY_EXECUTED`, `MCI_EXECTORQUERAMP`, and `STC_TORQUE_MODE`.

Referenced by `UI_ExecTorqueRamp()`, and `UI_SetReg()`.

2.10.1.3 void MCI_SetCurrentReferences (CMCI *this*, Curr_Components *lqdref*)

This is a buffered command to set directly the motor current references `Iq` and `Id`. This commands don't become active as soon as it is called but it will be executed when the oSTM state is `START_RUN` or `RUN`. User can check the status of the command calling the `MCI_IsCommandAcknowledged` method.

Parameters

<i>this</i>	related object of class CMCI.
<i>lqdref</i>	current references on <code>qd</code> reference frame in <code>Curr_Components</code> format.

Return values

<i>none.</i>

Definition at line 175 of file MCInterfaceClass.c.

References `Vars_t::CommandState`, `Vars_t::lqdref`, `Vars_t::lastCommand`, `Vars_t::LastModalitySetByUser`, `MCI_COMMAND_NOT_ALREADY_EXECUTED`, `MCI_SETCURRENTREFERENCES`, and `STC_TORQUE_MODE`.

Referenced by `UI_SetCurrentReferences()`, and `UI_SetReg()`.

2.10.1.4 bool MCI_StartMotor (CMCI *this*)

This is a user command used to begin the start-up procedure. If the state machine is in `IDLE` state the command is executed instantaneously otherwise the command is discarded. User must take care of this possibility by checking the return value. Before calling `MCI_StartMotor` it is mandatory to execute one of these commands:

`MCI_ExecSpeedRamp`

`MCI_ExecTorqueRamp`

`MCI_SetCurrentReferences`

Otherwise the behaviour in run state will be unpredictable.

Note: The `MCI_StartMotor` command is used just to begin the start-up procedure moving the state machine from `IDLE` state to `IDLE_START`. The command `MCI_StartMotor` is not blocking the execution of project until the motor is really running; to do this, the user have to check the state machine and verify that the `RUN` state (or any other state) has been reached.

Parameters

<i>this</i>	related object of class CMCI.
-------------	-------------------------------

Return values

<i>bool</i>	It returns <code>TRUE</code> if the command is successfully executed otherwise it return <code>FALSE</code> .
-------------	---

Definition at line 207 of file MCInterfaceClass.c.

References Vars_t::CommandState, IDLE_START, MCI_COMMAND_NOT_ALREADY_EXECUTED, and STM_NextState().

Referenced by UI_ExecCmd().

Here is the call graph for this function:



2.10.1.5 bool MCI_StopMotor (CMCI this)

This is a user command used to begin the stop motor procedure. If the state machine is in RUN or START states the command is executed instantaneously otherwise the command is discarded. User must take care of this possibility by checking the return value.

Note: The MCI_StopMotor command is used just to begin the stop motor procedure moving the state machine to ANY_STOP. The command MCI_StopMotor is not blocking the execution of project until the motor is really stopped; to do this, the user have to check the state machine and verify that the IDLE state has been reached again.

Parameters

this	related object of class CMCI.
------	-------------------------------

Return values

bool	It returns TRUE if the command is successfully executed otherwise it return FALSE.
------	--

Definition at line 235 of file MCInterfaceClass.c.

References ANY_STOP, and STM_NextState().

Referenced by UI_ExecCmd().

Here is the call graph for this function:



2.10.1.6 bool MCI_FaultAcknowledged (CMCI this)

This is a user command used to indicate that the user has seen the error condition. If is possible, the command is executed instantaneously otherwise the command is discarded. User must take care of this possibility by checking the return value.

Parameters

<i>this</i>	related object of class CMCI.
-------------	-------------------------------

Return values

<i>bool</i>	It returns TRUE if the command is successfully executed otherwise it return FALSE.
-------------	--

Definition at line 249 of file MCInterfaceClass.c.

References STM_FaultAcknowledged().

Referenced by UI_ExecCmd().

Here is the call graph for this function:

**2.10.1.7 bool MCI_EncoderAlign (CMCI *this*)**

This is a user command used to begin the encoder alignment procedure. If the state machine is in IDLE state the command is executed instantaneously otherwise the command is discarded. User must take care of this possibility by checking the return value.

Note: The MCI_EncoderAlign command is used just to begin the encoder alignment procedure moving the state machine from IDLE state to IDLE_ALIGNMENT. The command MCI_EncoderAlign is not blocking the execution of project until the encoder is really calibrated; to do this, the user have to check the state machine and verify that the IDLE state has been reached again.

Parameters

<i>this</i>	related object of class CMCI.
-------------	-------------------------------

Return values

<i>bool</i>	It returns TRUE if the command is successfully executed otherwise it return FALSE.
-------------	--

Definition at line 269 of file MCInterfaceClass.c.

References IDLE_ALIGNMENT, and STM_NextState().

Referenced by UI_ExecCmd().

Here is the call graph for this function:



2.10.1.8 CommandState_t MCI_IsCommandAcknowledged (CMCI *this*)

It returns information about the state of the last buffered command.

Parameters

<i>this</i>	related object of class CMCI.
-------------	-------------------------------

Return values

<i>CommandState_t</i>	<p>It can be one of the following codes:</p> <ul style="list-style-type: none"> • MCI_BUFFER_EMPTY if no buffered command has been called. • MCI_COMMAND_NOT_ALREADY_EXECUTED if the buffered command condition hasn't already occurred. • MCI_COMMAND_EXECUTED_SUCCESSFULLY if the buffered command has been executed successfully. In this case calling this function reset the command state to BC_BUFFER_EMPTY. • MCI_COMMAND_EXECUTED_UNSUCCESSFULLY if the buffered command has been executed unsuccessfully. In this case calling this function reset the command state to BC_BUFFER_EMPTY.
-----------------------	--

Definition at line 346 of file MCInterfaceClass.c.

References Vars_t::CommandState, MCI_BUFFER_EMPTY, MCI_COMMAND_EXECUTED_SUCCESSFULLY, and MCI_COMMAND_EXECUTED_UNSUCCESSFULLY.

2.10.1.9 State_t MCI_GetSTMState (CMCI *this*)

It returns information about the state of the related oSTM object.

Parameters

<i>this</i>	related object of class CMCI.
-------------	-------------------------------

Return values

<i>State_t</i>	It returns the current state of the related oSTM object.
----------------	--

Definition at line 364 of file MCInterfaceClass.c.

References STM_GetState().

Referenced by UI_ExecCmd().

Here is the call graph for this function:



2.10.1.10 uint16_t MCI_GetOccurredFaults (CMCI this)

It returns a 16 bit fields containing information about faults historically occurred since the state machine has been moved into FAULT_NOW state.

[Returned error codes are listed here](#).

Parameters

<i>this</i>	object of class CSTM.
-------------	-----------------------

Return values

<i>uint16_t</i>	16 bit fields with information about the faults historically occurred since the state machine has been moved into FAULT_NOW state. Returned error codes are listed here
-----------------	--

Definition at line 380 of file MCInterfaceClass.c.

References STM_GetFaultState().

Here is the call graph for this function:



2.10.1.11 uint16_t MCI_GetCurrentFaults (CMCI this)

It returns a 16 bit fields containing information about faults currently present.

[Returned error codes are listed here](#).

Parameters

<i>this</i>	object of class CSTM.
-------------	-----------------------

Return values

<i>uint16_t</i>	16 bit fields with information about currently present faults. Returned error codes are listed here
-----------------	--

Definition at line 394 of file MCInterfaceClass.c.

References STM_GetFaultState().

Here is the call graph for this function:



2.10.1.12 int16_t MCI_GetTorqueRef (CMCI this)

It returns information about the current motor torque reference. This value represents actually the Iq current reference expressed in digit. To convert current expressed in digit to current expressed in Amps is possible to use the formula: Current(Amp) = [Current(digit) * Vdd micro] / [65536 * Rshunt * Aop].

Parameters

<i>this</i>	related object of class CMCI.
-------------	-------------------------------

Return values

<i>int16_t</i>	current motor torque reference. This value represents actually the Iq ref current expressed in digit.
----------------	---

2.10.1.13 int16_t MCI_GetTorque (CMCI this)

It returns information about current motor measured torque. This value represents actually the Iq current expressed in digit. To convert current expressed in digit to current expressed in Amps is possible to use the formula: Current(-Amp) = [Current(digit) * Vdd micro] / [65536 * Rshunt * Aop].

Parameters

<i>this</i>	related object of class CMCI.
-------------	-------------------------------

Return values

<i>int16_t</i>	current motor measured torque. This value represents actually the Iq ref current expressed in digit.
----------------	--

2.10.1.14 STC_Modality_t MCI_GetControlMode (CMCI this)

It returns the modality of the speed and torque controller.

Parameters

<i>this</i>	related object of class CMCI.
-------------	-------------------------------

Return values

<i>STC_Modality_t</i>	It returns the modality of STC. It can be one of these two values: STC_TORQUE_MODE or STC_SPEED_MODE.
-----------------------	---

Definition at line 405 of file MCInterfaceClass.c.

References Vars_t::LastModalitySetByUser.

Referenced by UI_GetReg().

2.10.1.15 int16_t MCI_GetImposedMotorDirection (CMCI *this*)

It returns the motor direction imposed by the last command (MCI_ExecSpeedRamp, MCI_ExecTorqueRamp or MCI_SetCurrentReferences).

Parameters

<i>this</i>	related object of class CMCI.
-------------	-------------------------------

Return values

<i>int16_t</i>	It returns 1 or -1 according the sign of hFinalSpeed, hFinalTorque or lqdref.ql_-Component1 of the last command.
----------------	--

Definition at line 419 of file MCInterfaceClass.c.

References Vars_t::hFinalSpeed, Vars_t::hFinalTorque, Vars_t::lqdref, Vars_t::lastCommand, MCI_EXECSPEEDRAMP, MCI_EXECTORQUERAMP, and MCI_SETCURRENTREFERENCES.

2.10.1.16 int16_t MCI_GetLastRampFinalSpeed (CMCI *this*)

It returns information about the last ramp final speed sent by the user expressed in tenths of HZ.

Parameters

<i>this</i>	related object of class CMCI.
-------------	-------------------------------

Return values

<i>int16_t</i>	last ramp final speed sent by the user expressed in tehts of HZ.
----------------	--

Definition at line 457 of file MCInterfaceClass.c.

References Vars_t::hFinalSpeed, Vars_t::lastCommand, and MCI_EXECSPEEDRAMP.

Referenced by UI_GetReg().

2.10.1.17 bool MCI_RampCompleted (CMCI *this*)

Check if the settled speed or torque ramp has been completed.

Parameters

<i>this</i>	related object of class CMCI.
-------------	-------------------------------

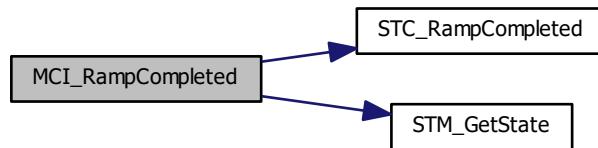
Return values

<i>bool</i>	It returns TRUE if the ramp is completed, FALSE otherwise.
-------------	--

Definition at line 474 of file MCInterfaceClass.c.

References Vars_t::oSTC, RUN, STC_RampCompleted(), and STM_GetState().

Here is the call graph for this function:

**2.10.1.18 bool MCI_GetSpdSensorReliability (CMCI this)**

It returns speed sensor reliability with reference to the sensor actually used for reference frame transformation.

Parameters

<i>this</i>	related object of class CMCI.
-------------	-------------------------------

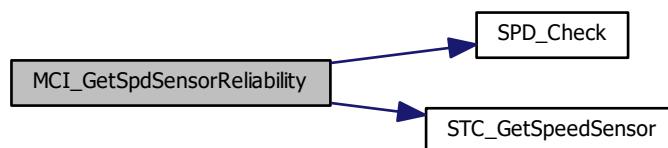
Return values

<i>bool</i>	It returns TRUE if the speed sensor utilized for reference frame transformation and (in speed control mode) for speed regulation is reliable, FALSE otherwise
-------------	---

Definition at line 494 of file MCInterfaceClass.c.

References Vars_t::oSTC, SPD_Check(), and STC_GetSpeedSensor().

Here is the call graph for this function:



2.10.1.19 int16_t MCI_GetAvrgMecSpeed01Hz (CMCI *this*)

It returns the last computed average mechanical speed, expressed in 01Hz (tenth of Hertz) and related to the sensor actually used by FOC algorithm.

Parameters

<i>this</i>	related object of class CMCI.
-------------	-------------------------------

Return values

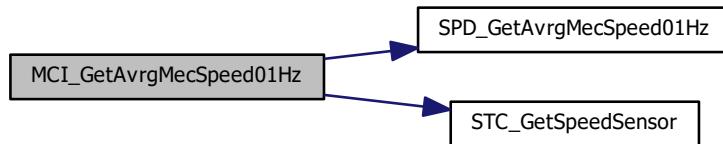
<i>int16_t</i>	rotor average mechanical speed (01Hz)
----------------	---------------------------------------

Definition at line 508 of file MCInterfaceClass.c.

References Vars_t::oSTC, SPD_GetAvrgMecSpeed01Hz(), and STC_GetSpeedSensor().

Referenced by UI_ExecCmd(), and UI_GetReg().

Here is the call graph for this function:



2.10.1.20 int16_t MCI_GetMecSpeedRef01Hz (CMCI *this*)

Get the current mechanical rotor speed reference expressed in tenths of HZ.

Parameters

<i>this</i>	related object of class CMCI.
-------------	-------------------------------

Return values

<i>int16_t</i>	current mechanical rotor speed reference expressed in tenths of HZ.
----------------	---

Definition at line 522 of file MCInterfaceClass.c.

References Vars_t::oSTC, and STC_GetMecSpeedRef01Hz().

Referenced by UI_GetReg(), and UI_SetReg().

Here is the call graph for this function:



2.10.1.21 Curr_Components MCI_Getlab (CMCI *this*)

It returns stator current lab in [Curr_Components](#) format.

Parameters

<i>this</i>	related object of class CMCI.
-------------	-------------------------------

Return values

Curr_Components	Stator current lab
---------------------------------	--------------------

Definition at line 534 of file MCInterfaceClass.c.

References FOCVars_t::lab, and Vars_t::pFOCVars.

Referenced by UI_GetReg().

2.10.1.22 Curr_Components MCI_Getlalphabet (CMCI *this*)

It returns stator current lalphabet in [Curr_Components](#) format.

Parameters

<i>this</i>	related object of class CMCI.
-------------	-------------------------------

Return values

Curr_Components	Stator current lalphabet
---------------------------------	--------------------------

Definition at line 545 of file MCInterfaceClass.c.

References FOCVars_t::lalphabet, and Vars_t::pFOCVars.

Referenced by UI_GetReg().

2.10.1.23 Curr_Components MCI_Getlqd (CMCI *this*)

It returns stator current lqd in [Curr_Components](#) format.

Parameters

<i>this</i>	related object of class CMCI.
-------------	-------------------------------

Return values

<i>Curr_Components</i>	Stator current Iqd
------------------------	--------------------

Definition at line 556 of file MCInterfaceClass.c.

References FOCVars_t::lqd, and Vars_t::pFOCVars.

Referenced by UI_GetReg().

2.10.1.24 Curr_Components MCI_GetIqdHF (CMCI *this*)

It returns stator current IqdHF in *Curr_Components* format.

Parameters

<i>this</i>	related object of class CMCI.
-------------	-------------------------------

Return values

<i>Curr_Components</i>	Stator current IqdHF if HFI is selected as main sensor. Otherwise it returns { 0, 0 }.
------------------------	--

Definition at line 568 of file MCInterfaceClass.c.

References FOCVars_t::lqdHF, and Vars_t::pFOCVars.

2.10.1.25 Curr_Components MCI_GetIqdref (CMCI *this*)

It returns stator current Iqdref in *Curr_Components* format.

Parameters

<i>this</i>	related object of class CMCI.
-------------	-------------------------------

Return values

<i>Curr_Components</i>	Stator current Iqdref
------------------------	-----------------------

Definition at line 579 of file MCInterfaceClass.c.

References FOCVars_t::lqdref, and Vars_t::pFOCVars.

Referenced by UI_GetReg(), and UI_SetReg().

2.10.1.26 Volt_Components MCI_GetVqd (CMCI *this*)

It returns stator current Vqd in *Volt_Components* format.

Parameters

<i>this</i>	related object of class CMCI.
-------------	-------------------------------

Return values

<i>Curr_Components</i>	Stator current Vqd
------------------------	--------------------

Definition at line 590 of file MCInterfaceClass.c.

References Vars_t::pFOCVars, and FOCVars_t::Vqd.

Referenced by UI_SetReg().

2.10.1.27 Volt_Components MCI_GetValphabeta (CMCI *this*)

It returns stator current Valphabeta in [Volt_Components](#) format.

Parameters

<i>this</i>	related object of class CMCI.
-------------	-------------------------------

Return values

Curr_Components	Stator current Valphabeta
---------------------------------	---------------------------

Definition at line 601 of file MCInterfaceClass.c.

References Vars_t::pFOCVars, and FOCVars_t::Valphabeta.

Referenced by UI_SetReg().

2.10.1.28 int16_t MCI_GetElAngledpp (CMCI *this*)

It returns the rotor electrical angle actually used for reference frame transformation.

Parameters

<i>this</i>	related object of class CMCI.
-------------	-------------------------------

Return values

<i>int16_t</i>	Rotor electrical angle in dpp format
----------------	--------------------------------------

Definition at line 613 of file MCInterfaceClass.c.

References FOCVars_t::hElAngle, and Vars_t::pFOCVars.

2.10.1.29 int16_t MCI_GetTeref (CMCI *this*)

It returns the reference elelctrical torque, fed to derived class for Iqref and Idref computation.

Parameters

<i>this</i>	related object of class CMCI.
-------------	-------------------------------

Return values

<i>int16_t</i>	Teref
----------------	-------

Definition at line 625 of file MCInterfaceClass.c.

References FOCVars_t::hTeref, and Vars_t::pFOCVars.

Referenced by UI_SetReg().

2.10.1.30 int16_t MCI_GetPhaseCurrentAmplitude (CMCI *this*)

It returns the motor phase current amplitude (0-to-peak) in s16A To convert s16A into Ampere following formula must be used: Current(Amp) = [Current(s16A) * Vdd micro] / [65536 * Rshunt * Aop].

Parameters

<i>this</i>	related object of class CMCI.
-------------	-------------------------------

Return values

<i>int16_t</i>	Motor phase current (0-to-peak) in s16A
----------------	---

Definition at line 638 of file MCInterfaceClass.c.

References FOCVars_t::lalphabet, MCM_Sqrt(), and Vars_t::pFOCVars.

Here is the call graph for this function:

**2.10.1.31 int16_t MCI_GetPhaseVoltageAmplitude (CMCI this)**

It returns the applied motor phase voltage amplitude (0-to-peak) in s16V. To convert s16V into Volts following formula must be used: PhaseVoltage(V) = [PhaseVoltage(s16A) * Vbus(V)] / [sqrt(3) *32767].

Parameters

<i>this</i>	related object of class CMCI.
-------------	-------------------------------

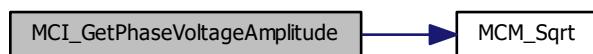
Return values

<i>int16_t</i>	Motor phase voltage (0-to-peak) in s16V
----------------	---

Definition at line 666 of file MCInterfaceClass.c.

References MCM_Sqrt(), Vars_t::pFOCVars, and FOCVars_t::Valphabet.

Here is the call graph for this function:

**2.10.1.32 void MCI_SetIdref (CMCI this, int16_t hNewIdref)**

When bDriveInput is set to INTERNAL, Idref should be normally managed by FOC_CalcCurrRef. Nevertheless, this method allows forcing changing Idref value. Method call has no effect when either flux weakening region is entered

or MTPA is enabled.

Parameters

<i>this</i>	related object of class CMCI.
<i>int16_t</i>	New target Id value

Return values

<i>none</i>

Definition at line 696 of file MCInterfaceClass.c.

References FOCVars_t::lqdref, and Vars_t::pFOCVars.

Referenced by UI_SetReg().

2.10.1.33 void MCI_Clear_Iqdref (CMCI *this*)

It re-initializes lqdref variables with their default values.

Parameters

<i>this</i>	related object of class CMCI.
-------------	-------------------------------

Return values

<i>none</i>

Definition at line 707 of file MCInterfaceClass.c.

References FOCVars_t::lqdref, Vars_t::oSTC, Vars_t::pFOCVars, and STC_GetDefaultIqdref().

Referenced by UI_ExecCmd().

Here is the call graph for this function:



2.11 MCTasks exported types

Collaboration diagram for MCTasks exported types:



2.12 MCTasks exported methods

Collaboration diagram for MCTasks exported methods:



Functions

- void **MCboot** (**CMCI oMCIList[], CMCT oMCTList[]**)
It initializes the whole MC core according to user defined parameters.
- void **MC_Scheduler** (**void**)
It executes MC tasks: safety task and medium frequency for all drive instances. It have to be clocked with Systick frequency.
- void **TSK_SafetyTask** (**void**)
It executes safety checks (e.g. bus voltage and temperature) for all drive instances. Faults flags are also here updated.
- uint8_t **TSK_HighFrequencyTask** (**void**)
Accordingly with the present state(s) of the state machine(s), it executes those motor control duties requiring a high frequency rate and a precise timing (e.g. FOC current control loop)
- void **TSK_DualDriveFIFOUpdate** (**void *oDrive**)
This function is called by TIMx_UP_IRQHandler in case of dual MC and it allows to reserve half PWM period in advance the FOC execution on ADC ISR.
- void **TSK_HardwareFaultTask** (**void**)
It is executed when a general hardware failure has been detected by the microcontroller and is used to put the system in safety condition.

2.12.1 Function Documentation

2.12.1.1 void **MCboot** (**CMCI oMCIList[], CMCT oMCTList[]**)

It initializes the whole MC core according to user defined parameters.

Parameters

oMCIList	pointer to the vector of MCInterface objects that will be created and initialized. The vector must have length equal to the number of motor drives.
oMCTList	pointer to the vector of MCTuning objects that will be created and initialized. The vector must have length equal to the number of motor drives.

Return values

None	
-------------	--

2.12.1.2 void MC_Scheduler(void)

It executes MC tasks: safety task and medium frequency for all drive instances. It have to be clocked with Systick frequency.

Parameters

<i>None</i>

Return values

<i>None</i>

Definition at line 782 of file MCTasks.c.

Referenced by TB_Scheduler().

2.12.1.3 void TSK_SafetyTask(void)

It executes safety checks (e.g. bus voltage and temperature) for all drive instances. Faults flags are also here updated.

Parameters

<i>None</i>

Return values

<i>None</i>

Definition at line 2323 of file MCTasks.c.

References PWMC_ExecRegularConv().

Referenced by TB_Scheduler().

Here is the call graph for this function:



2.12.1.4 uint8_t TSK_HighFrequencyTask(void)

Accordingly with the present state(s) of the state machine(s), it executes those motor control duties requiring a high frequency rate and a precise timing (e.g. FOC current control loop)

Parameters

<i>None</i>

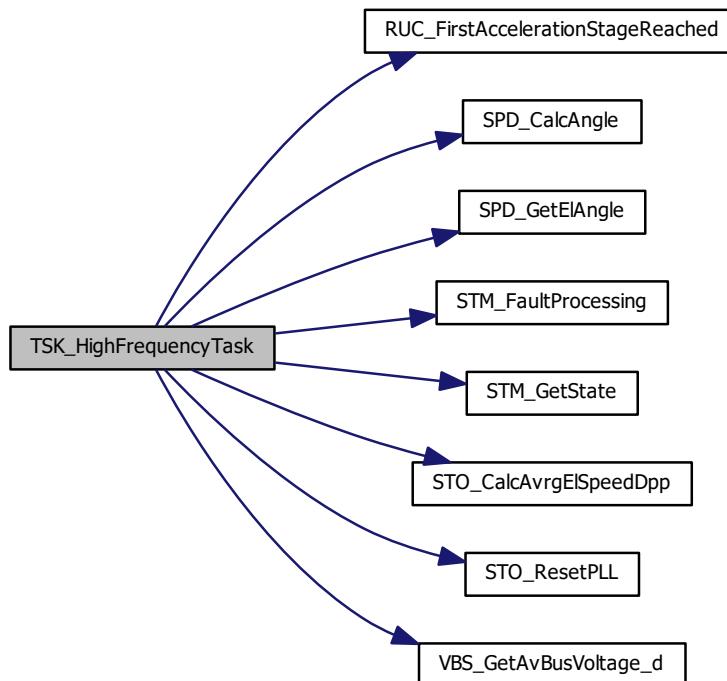
Return values

<code>uint8_t</code>	It return the motor instance number of last executed FOC.
----------------------	---

Definition at line 1814 of file MCTasks.c.

References `FOCVars_t::lalphabetas`, `M1`, `MC_FOC_DURATION`, `MC_NULL`, `RUC_FirstAccelerationStageReached()`, `SPD_CalcAngle()`, `SPD_GetElAngle()`, `START`, `START_RUN`, `STM_FaultProcessing()`, `STM_GetState()`, `STO_CalcAvgElSpeedDpp()`, `STO_ResetPLL()`, `FOCVars_t::Valphabetas`, and `VBS_GetAvBusVoltage_d()`.

Here is the call graph for this function:

**2.12.1.5 void TSK_DualDriveFIFOUpdate(void * oDrive)**

This function is called by `TIMx_UP_IRQHandler` in case of dual MC and it allows to reserve half PWM period in advance the FOC execution on ADC ISR.

Parameters

<code>oDrive</code>	pointer to a CFOC object
---------------------	--------------------------

Return values

<code>None</code>

2.12.1.6 void TSK_HardwareFaultTask (void)

It is executed when a general hardware failure has been detected by the microcontroller and is used to put the system in safety condition.

Parameters

None

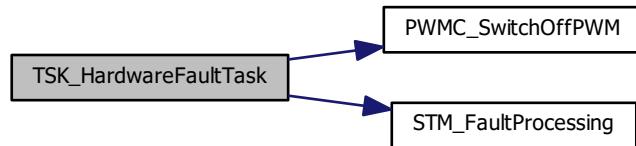
Return values

None

Definition at line 2584 of file MCTasks.c.

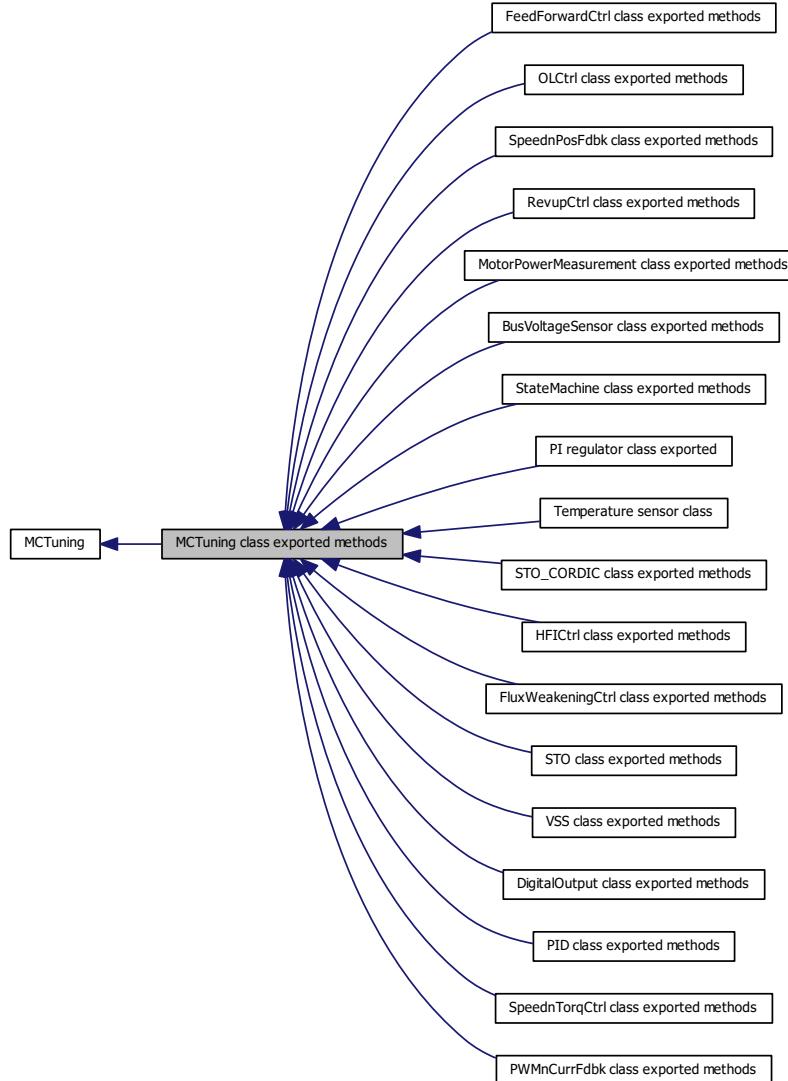
References MC_SW_ERROR, PWMC_SwitchOffPWM(), and STM_FaultProcessing().

Here is the call graph for this function:



2.13 MCTuning class exported methods

Collaboration diagram for MCTuning class exported methods:



Modules

- FluxWeakeningCtrl class exported methods
- FeedForwardCtrl class exported methods
- HFICtrl class exported methods
- OLCtrl class exported methods
- PI regulator class exported
- PID class exported methods
- PWMnCurrFdbk class exported methods
- RevupCtrl class exported methods
- SpeednPosFdbk class exported methods
- STO class exported methods

- STO_CORDIC class exported methods
- VSS class exported methods
- SpeednTorqCtrl class exported methods
- StateMachine class exported methods
- Temperature sensor class
- BusVoltageSensor class exported methods
- DigitalOutput class exported methods
- MotorPowerMeasurement class exported methods

Functions

- CFW MCT_GetFluxWeakeningCtrl (CMCT this)

*It returns the FluxWeakeningCtrl object
Methods listed here can be applied .*
- CFF MCT_GetFeedForwardCtrl (CMCT this)

*It returns the FeedForwardCtrl object
Methods listed here can be applied .*
- CHFI_FP MCT_GetHFICtrl (CMCT this)

*It returns the HFICtrl object
Methods listed here can be applied .*
- CPI MCT_GetSpeedLoopPID (CMCT this)

*It returns the speed control loop PI(D) object
Methods listed here can be applied
Methods listed here can be applied if the object belongs to derived class PID .*
- CPI MCT_GetIqLoopPID (CMCT this)

*It returns the Iq current control loop PI(D) object
Methods listed here can be applied
Methods listed here can be applied if the object belongs to derived class PID .*
- CPI MCT_GetIdLoopPID (CMCT this)

*It returns the Id current control loop PI(D) object
Methods listed here can be applied
Methods listed here can be applied if the object belongs to derived class PID .*
- CPI MCT_GetFluxWeakeningLoopPID (CMCT this)

*It returns the Flux Weakening control loop PI(D) object
Methods listed here can be applied
Methods listed here can be applied if the object belongs to derived class PID .*
- CPWMC MCT_GetPWMnCurrFdbk (CMCT this)

*It returns the PWMnCurrFdbk object
Methods listed here can be applied .*
- CRUC MCT_GetRevupCtrl (CMCT this)

*It returns the Rev-up controller object
Methods listed here can be applied .*
- CSPD MCT_GetSpeednPosSensorMain (CMCT this)

*It returns the Main Speed'n Position sensor object. Main position sensor is considered the one used to execute FOC
Methods listed here can be applied
Methods listed here can be applied if the object belongs to derived class STO (StateObserver) .*
- CSPD MCT_GetSpeednPosSensorAuxiliary (CMCT this)

*It returns the Auxiliary Speed'n Position sensor object. Auxiliary position sensor is considered the one used to backup/tune the main one
Methods listed here can be applied
Methods listed here can be applied if the object belongs to derived class STO (StateObserver) .*
- CSPD MCT_GetSpeednPosSensorVirtual (CMCT this)

*It returns the Virtual Speed'n Position sensor object. Virtual position sensor is considered the one used to rev-up the motor during the start-up procedure required by the state-observer sensorless algorithm
Methods listed here can be applied .*

- **CSTC MCT_GetSpeednTorqueController (CMCT this)**
It returns the Speed'n Torque Controller object.
Methods listed here can be applied .
- **CTSM MCT_GetStateMachine (CMCT this)**
It returns the State Machine object.
Methods listed here can be applied .
- **CTSNS MCT_GetTemperatureSensor (CMCT this)**
It returns the Temperature sensor object.
Methods listed here can be applied .
- **CVBS MCT_GetBusVoltageSensor (CMCT this)**
It returns the Bus Voltage sensor object
Methods listed here can be applied .
- **CDOUT MCT_GetBrakeResistor (CMCT this)**
It returns the Brake resistor object
Methods listed here can be applied .
- **CDOUT MCT_GetNTCRelay (CMCT this)**
It returns the NTC Relay object
Methods listed here can be applied .
- **CMPM MCT_GetMotorPowerMeasurement (CMCT this)**
It returns the MotorPowerMeasurement object
Methods listed here can be applied .

2.13.1 Function Documentation

2.13.1.1 CFW MCT_GetFluxWeakeningCtrl (CMCT this)

It returns the FluxWeakeningCtrl object

Methods listed here can be applied .

Parameters

<i>this</i>	related object of class CMCT
-------------	------------------------------

Return values

<i>CFW,instance</i>	of FluxWeakeningCtrl Class
---------------------	----------------------------

It returns the FluxWeakeningCtrl object

Methods listed here can be applied .

Parameters

<i>this</i>	related object of class CMCT
-------------	------------------------------

Return values

<i>CFW,instance</i>	of FluxWeakeningCtrl Class
---------------------	----------------------------

Definition at line 87 of file MCTuningClass.c.

Referenced by UI_GetReg(), and UI_SetReg().

2.13.1.2 CFF MCT_GetFeedForwardCtrl (CMCT this)

It returns the FeedForwardCtrl object

Methods listed here can be applied .

Parameters

<i>this</i>	related object of class CMCT
-------------	------------------------------

Return values

<i>CFF,instance</i>	of FeedForwardCtrl Class
---------------------	--------------------------

It returns the FeedForwardCtrl object

Methods listed here can be applied .

Parameters

<i>this</i>	related object of class CMCT
-------------	------------------------------

Return values

<i>CFF,instance</i>	of FeedForwardCtrl Class
---------------------	--------------------------

Definition at line 97 of file MCTuningClass.c.

Referenced by UI_GetReg(), and UI_SetReg().

2.13.1.3 CHFI_FP MCT_GetHFICtrl (CMCT *this*)

It returns the HFICtrl object

Methods listed here can be applied .

Parameters

<i>this</i>	related object of class CMCT
-------------	------------------------------

Return values

<i>CHFI_FP,instance</i>	of HFICtrl Class
-------------------------	------------------

Definition at line 108 of file MCTuningClass.c.

Referenced by UI_GetReg(), and UI_SetReg().

2.13.1.4 CPI MCT_GetSpeedLoopPID (CMCT *this*)

It returns the speed control loop PI(D) object

Methods listed here can be applied

Methods listed here can be applied if the object belongs to derived class PID .

Parameters

<i>this</i>	related object of class CMCT
-------------	------------------------------

Return values

<i>CPI,instance</i>	of PIRegulator Class
---------------------	----------------------

It returns the speed control loop PI(D) object

Methods listed here can be applied

Methods listed here can be applied if the object belongs to derived class PID .

Parameters

<i>this</i>	related object of class CMCT
-------------	------------------------------

Return values

<i>none</i>

Definition at line 118 of file MCTuningClass.c.

Referenced by UI_GetReg(), and UI_SetReg().

2.13.1.5 CPI MCT_GetIqLoopPID (CMCT *this*)

It returns the Iq current control loop PI(D) object

Methods listed here can be applied

Methods listed here can be applied if the object belongs to derived class PID .

Parameters

<i>this</i>	related object of class CMCT
-------------	------------------------------

Return values

<i>CPI,instance</i>	of PIRegulator Class
---------------------	----------------------

It returns the Iq current control loop PI(D) object

Methods listed here can be applied

Methods listed here can be applied if the object belongs to derived class PID .

Parameters

<i>this</i>	related object of class CMCT
-------------	------------------------------

Return values

<i>none</i>

Definition at line 128 of file MCTuningClass.c.

Referenced by UI_GetReg(), and UI_SetReg().

2.13.1.6 CPI MCT_GetIdLoopPID (CMCT *this*)

It returns the Id current control loop PI(D) object

Methods listed here can be applied

Methods listed here can be applied if the object belongs to derived class PID .

Parameters

<i>this</i>	related object of class CMCT
-------------	------------------------------

Return values

<i>CPI,instance</i>	of PIRegulator Class
---------------------	----------------------

It returns the Id current control loop PI(D) object

[Methods listed here can be applied](#)

[Methods listed here can be applied if the object belongs to derived class PID .](#)

Parameters

<i>this</i>	related object of class CMCT
-------------	------------------------------

Return values

<i>none</i>

Definition at line 138 of file MCTuningClass.c.

Referenced by UI_GetReg(), and UI_SetReg().

2.13.1.7 CPI MCT_GetFluxWeakeningLoopPID (CMCT *this*)

It returns the Flux Weakening control loop PI(D) object

[Methods listed here can be applied](#)

[Methods listed here can be applied if the object belongs to derived class PID .](#)

Parameters

<i>this</i>	related object of class CMCT
-------------	------------------------------

Return values

<i>CPI,instance</i>	of PI RegulatorClass; MC_NULL if Flux Weakening is not enabled
---------------------	--

It returns the Flux Weakening control loop PI(D) object

[Methods listed here can be applied](#)

[Methods listed here can be applied if the object belongs to derived class PID .](#)

Parameters

<i>this</i>	related object of class CMCT
-------------	------------------------------

Return values

<i>none</i>

Definition at line 148 of file MCTuningClass.c.

Referenced by UI_GetReg(), and UI_SetReg().

2.13.1.8 CPWMC MCT_GetPWMnCurrFdbk (CMCT *this*)

It returns the PWMnCurrFdbk object

Methods listed here can be applied .

Parameters

<i>this</i>	related object of class CMCT
-------------	------------------------------

Return values

<i>CPWMC,instance</i>	of PWMnCurrFdbk Class
-----------------------	-----------------------

It returns the PWMnCurrFdbk object

Methods listed here can be applied .

Parameters

<i>this</i>	related object of class CMCT
-------------	------------------------------

Return values

<i>CPWMC,instance</i>	of PWMnCurrFdbkClass
-----------------------	----------------------

Definition at line 158 of file MCTuningClass.c.

2.13.1.9 CRUC MCT_GetRevupCtrl (CMCT *this*)

It returns the Rev-up controller object

Methods listed here can be applied .

Parameters

<i>this</i>	related object of class CMCT
-------------	------------------------------

Return values

<i>CRUC,instance</i>	of RevupCtrl Class
----------------------	--------------------

It returns the Rev-up controller object

Methods listed here can be applied .

Parameters

<i>this</i>	related object of class CMCT
-------------	------------------------------

Return values

<i>CRUC,instance</i>	of RevupCtrlClass
----------------------	-------------------

Definition at line 168 of file MCTuningClass.c.

Referenced by UI_GetReg(), UI_GetRevupData(), and UI_SetRevupData().

2.13.1.10 CSPD MCT_GetSpeednPosSensorMain (CMCT *this*)

It returns the Main Speed'n Position sensor object. Main position sensor is considered the one used to execute FOC

Methods listed here can be applied

Methods listed here can be applied if the object belongs to derived class STO (StateObserver) .

Parameters

<i>this</i>	related object of class CMCT
-------------	------------------------------

Return values

<i>CSPD,instance</i>	of SpeednPosFdbk Class
----------------------	------------------------

It returns the Main Speed'n Position sensor object. Main position sensor is considered the one used to execute FOC

Methods listed here can be applied

Methods listed here can be applied if the object belongs to derived class STO (StateObserver) .

Parameters

<i>this</i>	related object of class CMCT
-------------	------------------------------

Return values

<i>CSPD,instance</i>	of SpeednPosFdbkClass
----------------------	-----------------------

Definition at line 179 of file MCTuningClass.c.

Referenced by UI_GetReg(), and UI_SetReg().

2.13.1.11 CSPD MCT_GetSpeednPosSensorAuxiliary (CMCT *this*)

It returns the Auxiliary Speed'n Position sensor object. Auxiliary position sensor is considered the one used to backup/tune the main one

Methods listed here can be applied

Methods listed here can be applied if the object belongs to derived class STO (StateObserver) .

Parameters

<i>this</i>	related object of class CMCT
-------------	------------------------------

Return values

<i>CSPD,instance</i>	of SpeednPosFdbk Class; MC_NULL if no auxiliary sensor is active
----------------------	--

It returns the Auxiliary Speed'n Position sensor object. Auxiliary position sensor is considered the one used to backup/tune the main one

Methods listed here can be applied

Methods listed here can be applied if the object belongs to derived class STO (StateObserver) .

Parameters

<i>this</i>	related object of class CMCT
-------------	------------------------------

Return values

<i>CSPD,instance</i>	of SpeednPosFdbkClass; MC_NULL if no auxiliary sensor is active
----------------------	---

Definition at line 192 of file MCTuningClass.c.

Referenced by UI_GetReg(), and UI_SetReg().

2.13.1.12 CSPD MCT_GetSpeednPosSensorVirtual (CMCT *this*)

It returns the Virtual Speed'n Position sensor object. Virtual position sensor is considered the one used to rev-up the motor during the start-up procedure required by the state-observer sensorless algorithm

[Methods listed here can be applied .](#)

Parameters

<i>this</i>	related object of class CMCT
-------------	------------------------------

Return values

<i>CSPD,instance</i>	of SpeednPosFdbk Class; MC_NULL if no auxiliary sensor is active
----------------------	--

It returns the Virtual Speed'n Position sensor object. Virtual position sensor is considered the one used to rev-up the motor during the start-up procedure required by the state-observer sensorless algorithm

[Methods listed here can be applied .](#)

Parameters

<i>this</i>	related object of class CMCT
-------------	------------------------------

Return values

<i>CSPD,instance</i>	of SpeednPosFdbkClass; MC_NULL if no auxiliary sensor is active
----------------------	---

Definition at line 206 of file MCTuningClass.c.

2.13.1.13 CSTC MCT_GetSpeednTorqueController (CMCT *this*)

It returns the Speed'n Torque Controller object.

[Methods listed here can be applied .](#)

Parameters

<i>this</i>	related object of class CMCT
-------------	------------------------------

Return values

<i>CSTC,instance</i>	of SpeednTorqCtrl Class
----------------------	-------------------------

It returns the Speed'n Torque Controller object.

[Methods listed here can be applied .](#)

Parameters

<i>this</i>	related object of class CMCT
-------------	------------------------------

Return values

<i>CSTC,instance</i>	of SpeednTorqCtrlClass
----------------------	------------------------

Definition at line 216 of file MCTuningClass.c.

Referenced by [UI_GetReg\(\)](#).

2.13.1.14 CSTM MCT_GetStateMachine (CMCT *this*)

It returns the State Machine object.

[Methods listed here can be applied](#) .

Parameters

<i>this</i>	related object of class CMCT
-------------	------------------------------

Return values

<i>CSTM,instance</i>	of StateMachine Class
----------------------	-----------------------

It returns the State Machine object.

[Methods listed here can be applied](#) .

Parameters

<i>this</i>	related object of class CMCT
-------------	------------------------------

Return values

<i>CSTM,instance</i>	of StateMachineClass
----------------------	----------------------

Definition at line 226 of file MCTuningClass.c.

Referenced by [UI_GetReg\(\)](#).

2.13.1.15 CTSNS MCT_GetTemperatureSensor (CMCT *this*)

It returns the Temperature sensor object.

[Methods listed here can be applied](#) .

Parameters

<i>this</i>	related object of class CMCT
-------------	------------------------------

Return values

<i>CTSNS,instance</i>	of TemperatureSensor Class
-----------------------	----------------------------

It returns the Temperature sensor object.

[Methods listed here can be applied](#) .

Parameters

<i>this</i>	related object of class CMCT
-------------	------------------------------

Return values

<i>CTSNS,instance</i>	of TemperatureSensorClass
-----------------------	---------------------------

Definition at line 236 of file MCTuningClass.c.

Referenced by [UI_GetReg\(\)](#).

2.13.1.16 CVBS MCT_GetBusVoltageSensor (CMCT *this*)

It returns the Bus Voltage sensor object

[Methods listed here can be applied](#) .

Parameters

<i>this</i>	related object of class CMCT
-------------	------------------------------

Return values

<i>CVBS,instance</i>	of BusVoltageSensor Class
----------------------	---------------------------

It returns the Bus Voltage sensor object

[Methods listed here can be applied](#) .

Parameters

<i>this</i>	related object of class CMCT
-------------	------------------------------

Return values

<i>CVBS,instance</i>	of BusVoltageSensorClass
----------------------	--------------------------

Definition at line 246 of file MCTuningClass.c.

Referenced by [UI_GetReg\(\)](#).

2.13.1.17 CDOUT MCT_GetBrakeResistor (CMCT *this*)

It returns the Brake resistor object

[Methods listed here can be applied](#) .

Parameters

<i>this</i>	related object of class CMCT
-------------	------------------------------

Return values

<i>CDOUT,instance</i>	of DigitalOutput Class; MC_NULL if no Brake resistor is active
-----------------------	--

It returns the Brake resistor object

[Methods listed here can be applied](#) .

Parameters

<i>this</i>	related object of class CMCT
-------------	------------------------------

Return values

<i>CDOU,instance</i>	of DigitalOutput Class; MC_NULL if no Brake resistor is active
----------------------	--

Definition at line 257 of file MCTuningClass.c.

2.13.1.18 CDOU MCT_GetNTCRelay (CMCT *this*)

It returns the NTC Relay object

[Methods listed here can be applied .](#)

Parameters

<i>this</i>	related object of class CMCT
-------------	------------------------------

Return values

<i>CDOU,instance</i>	of DigitalOutput Class; MC_NULL if no Relay is active
----------------------	---

It returns the NTC Relay object

[Methods listed here can be applied .](#)

Parameters

<i>this</i>	related object of class CMCT
-------------	------------------------------

Return values

<i>CDOU,instance</i>	of DigitalOutput Class; MC_NULL if no Relay is active
----------------------	---

Definition at line 268 of file MCTuningClass.c.

2.13.1.19 CMPM MCT_GetMotorPowerMeasurement (CMCT *this*)

It returns the MotorPowerMeasurement object

[Methods listed here can be applied .](#)

Parameters

<i>this</i>	related object of class CMCT
-------------	------------------------------

Return values

<i>CMPM,instance</i>	of MotorPowerMeasurement Class; MC_NULL if no MPM is active
----------------------	---

It returns the MotorPowerMeasurement object

[Methods listed here can be applied .](#)

Parameters

<i>this</i>	related object of class CMCT
-------------	------------------------------

Return values

<i>CMPM,instance</i>	of MotorPowerMeasurement Class; MC_NULL if no MPM is active
----------------------	---

Definition at line 279 of file MCTuningClass.c.

Referenced by UI_GetReg().

2.14 FluxWeakeningCtrl class exported methods

Collaboration diagram for FluxWeakeningCtrl class exported methods:



Functions

- void **FW_SetVref** (**CFW** this, uint16_t hNewVref)

Use this method to set a new value for the voltage reference used by flux weakening algorithm.
- uint16_t **FW_GetVref** (**CFW** this)

It returns the present value of target voltage used by flux weakening algorihtm.
- int16_t **FW_GetAvVAmplitude** (**CFW** this)

It returns the present value of voltage actually used by flux weakening algorihtm.
- uint16_t **FW_GetAvVPercentage** (**CFW** this)

It returns the measure of present voltage actually used by flux weakening algorihtm as percentage of available voltage.

2.14.1 Function Documentation

2.14.1.1 void FW_SetVref (CFW this, uint16_t hNewVref)

Use this method to set a new value for the voltage reference used by flux weakening algorithm.

Parameters

<i>this</i>	related object of class CFW
<i>uint16_t</i>	New target voltage value, expressend in tenth of percentage points of available voltage

Return values

<i>none</i>

Parameters

<i>this</i>	related object of class CFW.
<i>uint16_t</i>	New target voltage value, expressend in tenth of percentage points of available voltage.

Return values

<i>none</i>

Definition at line 264 of file FluxWeakeningCtrlClass.c.

Referenced by UI_SetReg().

2.14.1.2 uint16_t FW_GetVref (CFW this)

It returns the present value of target voltage used by flux weakening algorihtm.

Parameters

<i>this</i>	related object of class CFW
-------------	-----------------------------

Return values

<i>int16_t</i>	Present target voltage value expressed in tenth of percentage points of available voltage
----------------	---

Parameters

<i>this</i>	related object of class CFW.
-------------	------------------------------

Return values

<i>int16_t</i>	Present target voltage value expressed in tenth of percentage points of available voltage.
----------------	--

Definition at line 277 of file FluxWeakeningCtrlClass.c.

Referenced by UI_GetReg().

2.14.1.3 int16_t FW_GetAvVAmplitude (CFW this)

It returns the present value of voltage actually used by flux weakening algorihtm.

Parameters

<i>this</i>	related object of class CFW
-------------	-----------------------------

Return values

<i>int16_t</i>	Present averaged phase stator voltage value, expressed in s16V (0-to-peak), where PhaseVoltage(V) = [PhaseVoltage(s16A) * Vbus(V)] /[sqrt(3) *32767]
----------------	---

Parameters

<i>this</i>	related object of class CFW.
-------------	------------------------------

Return values

<i>int16_t</i>	Present averaged phase stator voltage value, expressed in s16V (0-to-peak), where PhaseVoltage(V) = [PhaseVoltage(s16A) * Vbus(V)] /[sqrt(3) *32767].
----------------	--

Definition at line 291 of file FluxWeakeningCtrlClass.c.

2.14.1.4 uint16_t FW_GetAvVPercentage (CFW this)

It returns the measure of present voltage actually used by flux weakening algorihtm as percentage of available voltage.

Parameters

<i>this</i>	related object of class CFW
-------------	-----------------------------

Return values

<i>uint16_t</i>	Present averaged phase stator voltage value, expressed in tenth of percentage points of available voltage.
-----------------	--

Parameters

<i>this</i>	related object of class CFW.
-------------	------------------------------

Return values

<i>uint16_t</i>	Present averaged phase stator voltage value, expressed in tenth of percentage points of available voltage.
-----------------	--

Definition at line 304 of file FluxWeakeningCtrlClass.c.

Referenced by UI_GetReg().

2.15 FeedForwardCtrl class exported methods

Collaboration diagram for FeedForwardCtrl class exported methods:



Functions

- void **FF_SetFFConstants (CFF this, FF_TuningStruct_t sNewConstants)**
Use this method to set new values for the constants utilized by feed-forward algorithm.
- **FF_TuningStruct_t FF_GetFFConstants (CFF this)**
Use this method to get present values for the constants utilized by feed-forward algorithm.
- **Volt_Components FF_GetVqdff (CFF this)**
Use this method to get present values for the Vqd feed-forward components.
- **Volt_Components FF_GetVqdAvPlout (CFF this)**
Use this method to get values of the averaged output of qd axes currents PI regulators.

2.15.1 Function Documentation

2.15.1.1 void FF_SetFFConstants (CFF this, FF_TuningStruct_t sNewConstants)

Use this method to set new values for the constants utilized by feed-forward algorithm.

Parameters

<i>this</i>	related object of class CFF
<i>IMFF_TuningStruct_t</i>	New values for the constants utilized by feed-forward algorithm

Return values

<i>none</i>	
-------------	--

Referenced by UI_SetReg().

2.15.1.2 FF_TuningStruct_t FF_GetFFConstants (CFF this)

Use this method to get present values for the constants utilized by feed-forward algorithm.

Parameters

<i>this</i>	related object of class CFF
-------------	-----------------------------

Return values

<i>IMFF_TuningStruct_t</i>	new values for the constants utilized by feed-forward algorithm
----------------------------	---

Referenced by UI_GetReg(), and UI_SetReg().

2.15.1.3 Volt_Components FF_GetVqdff (CFF *this*)

Use this method to get present values for the Vqd feed-forward components.

Parameters

<i>this</i>	related object of class CFF
-------------	-----------------------------

Return values

Volt_Components	Vqd feed-forward components
---------------------------------	-----------------------------

Referenced by UI_GetReg().

2.15.1.4 Volt_Components FF_GetVqdAvPlout (CFF *this*)

Use this method to get values of the averaged output of qd axes currents PI regulators.

Parameters

<i>this</i>	related object of class CFF
-------------	-----------------------------

Return values

Volt_Components	Averaged output of qd axes currents PI regulators
---------------------------------	---

Referenced by UI_GetReg().

2.16 HFICtrl class exported methods

Collaboration diagram for HFICtrl class exported methods:



Functions

- `int16_t HFI_FP_GetRotorAngleLock (CHFI_FP this)`
It returns the rotor angle lock value.
- `int16_t HFI_FP_GetSaturationDifference (CHFI_FP this)`
It returns the saturation difference measured during the last north/south identification stage.
- `int16_t HFI_FP_GetCurrent (CHFI_FP this)`
It return the quantity that shall be put in the DAC to tune the HFI.
- `CPI HFI_FP_GetPITrack (CHFI_FP this)`
It returns the Track PI.
- `void HFI_FP_SetMinSaturationDifference (CHFI_FP this, int16_t hMinSaturationDifference)`
It set the min saturation difference used to validate the north/south identification stage.

2.16.1 Function Documentation

2.16.1.1 `int16_t HFI_FP_GetRotorAngleLock (CHFI_FP this)`

It returns the rotor angle lock value.

Parameters

<code>this</code>	related object of class CHFI_FP
-------------------	---------------------------------

Return values

<code>int16_t</code>	Rotor angle lock value
----------------------	------------------------

Referenced by UI_GetReg().

2.16.1.2 `int16_t HFI_FP_GetSaturationDifference (CHFI_FP this)`

It returns the saturation difference measured during the last north/south identification stage.

Parameters

<code>this</code>	related object of class CHFI_FP
-------------------	---------------------------------

Return values

<code>int16_t</code>	Saturation difference measured during the last north/south identification stage.
----------------------	--

Referenced by UI_GetReg().

2.16.1.3 int16_t HFI_FP_GetCurrent (CHFI_FP *this*)

It return the quantity that shall be put in the DAC to tune the HFI.

Parameters

<i>this</i>	related object of class CHFI_FP
-------------	---------------------------------

Return values

<i>int16_t</i>	HFI current
----------------	-------------

Referenced by UI_GetReg().

2.16.1.4 CPI HFI_FP_GetPITrack (CHFI_FP *this*)

It returns the Track PI.

Parameters

<i>this</i>	related object of class CHFI_FP
-------------	---------------------------------

Return values

<i>CPI</i>	Track PI
------------	----------

Referenced by UI_GetReg(), and UI_SetReg().

2.16.1.5 void HFI_FP_SetMinSaturationDifference (CHFI_FP *this*, int16_t *hMinSaturationDifference*)

It set the min saturation difference used to validate the north/south identification stage.

Parameters

<i>this</i>	related object of class CHFI_FP
<i>hMinSaturation-Difference</i>	Min Saturation difference used to validate the north/south identification stage. identification stage.

Return values

<i>none</i>

Referenced by UI_SetReg().

2.17 OLCtrl class exported methods

Collaboration diagram for OLCtrl class exported methods:



Functions

- void [OL_UpdateVoltage \(COL this, int16_t hNewVoltage\)](#)
It allows changing applied open loop phase voltage.

2.17.1 Function Documentation

2.17.1.1 void [OL_UpdateVoltage \(COL this, int16_t hNewVoltage \)](#)

It allows changing applied open loop phase voltage.

Parameters

<i>this</i>	related object of class COL
<i>hNewVoltage</i>	New voltage value to be applied by the open loop.

Return values

<i>None</i>

Definition at line 111 of file OpenLoopClass.c.

2.18 PI regulator class exported

Collaboration diagram for PI regulator class exported:



2.18.1 Detailed Description

methods

Functions

- void [PI_SetKP](#) (CPI this, int16_t hKpGain)
It updates the Kp gain.
- void [PI_SetKI](#) (CPI this, int16_t hKiGain)
It updates the Ki gain.
- int16_t [PI_GetKP](#) (CPI this)
It returns the Kp gain of the passed PI object.
- int16_t [PI_GetKI](#) (CPI this)
It returns the Ki gain of the passed PI object.
- int16_t [PI_GetDefaultKP](#) (CPI this)
It returns the Default Kp gain of the passed PI object.
- int16_t [PI_GetDefaultKI](#) (CPI this)
It returns the Default Ki gain of the passed PI object.
- void [PI_SetIntegralTerm](#) (CPI this, int32_t wIntegralTermValue)
It set a new value into the PI integral term.

2.18.2 Function Documentation

2.18.2.1 void PI_SetKP (CPI this, int16_t hKpGain)

It updates the Kp gain.

Parameters

CPI	PI object
int16_t	New Kp gain

Return values

None

Definition at line 100 of file PIRegulatorClass.c.

Referenced by [STO_SetPLLGains\(\)](#), and [UI_SetReg\(\)](#).

2.18.2.2 void PI_SetKI (CPI *this*, int16_t *hKiGain*)

It updates the Ki gain.

Parameters

<i>CPI</i>	PI object
<i>int16_t</i>	New Ki gain

Return values

<i>None</i>

Definition at line 111 of file PIRegulatorClass.c.

Referenced by STO_SetPLLGains(), and UI_SetReg().

2.18.2.3 int16_t PI_GetKP (CPI *this*)

It returns the Kp gain of the passed PI object.

Parameters

<i>CPI</i>	PI regulator object
------------	---------------------

Return values

<i>int16_t</i>	Kp gain
----------------	---------

Definition at line 121 of file PIRegulatorClass.c.

Referenced by STO_GetPLLGains(), and UI_GetReg().

2.18.2.4 int16_t PI_GetKI (CPI *this*)

It returns the Ki gain of the passed PI object.

Parameters

<i>CPI</i>	PI regulator object
------------	---------------------

Return values

<i>int16_t</i>	Ki gain
----------------	---------

Definition at line 131 of file PIRegulatorClass.c.

Referenced by STO_GetPLLGains(), and UI_GetReg().

2.18.2.5 int16_t PI_GetDefaultKP (CPI *this*)

It returns the Default Kp gain of the passed PI object.

Parameters

<i>CPI</i>	PI regulator object
------------	---------------------

Return values

<i>int16_t</i>	Kp gain
----------------	---------

Definition at line 141 of file PIRegulatorClass.c.

2.18.2.6 int16_t PI_GetDefaultKI (CPI *this*)

It returns the Default Ki gain of the passed PI object.

Parameters

<i>CPI</i>	PI regulator object
------------	---------------------

Return values

<i>int16_t</i>	Ki gain
----------------	---------

Definition at line 151 of file PIRegulatorClass.c.

2.18.2.7 void PI_SetIntegralTerm (CPI *this*, int32_t *wIntegralTermValue*)

It set a new value into the PI integral term.

Parameters

<i>CPI</i>	PI regulator object
<i>int32_t</i>	New integral term value

Return values

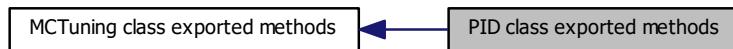
<i>None</i>

Definition at line 162 of file PIRegulatorClass.c.

Referenced by FW_Clear(), STC_Clear(), STO_ResetPLL(), and STO_SetPLL().

2.19 PID class exported methods

Collaboration diagram for PID class exported methods:



Functions

- void [PID_SetPrevError \(CPID_PI this, int32_t wPrevProcessVarError\)](#)
It set a new value into the PID Previous error variable required to compute derivative term.
- void [PID_SetKD \(CPID_PI this, int16_t hKdGain\)](#)
It updates the Kd gain.
- int16_t [PID_GetKD \(CPID_PI this\)](#)
It returns the Kd gain of the PID object passed.

2.19.1 Function Documentation

2.19.1.1 void PID_SetPrevError (CPID_PI this, int32_t wPrevProcessVarError)

It set a new value into the PID Previous error variable required to compute derivative term.

Parameters

<i>this</i>	regulator object
<i>wPrevProcess-VarError</i>	New integral term value

Return values

<i>None</i>

Parameters

<i>CPID_PI</i>	PID regulator object
<i>int32_t</i>	New integral term value

Return values

<i>None</i>

Definition at line 127 of file PID_PIRegulatorClass.c.

2.19.1.2 void PID_SetKD (CPID_PI this, int16_t hKdGain)

It updates the Kd gain.

Parameters

<i>this</i>	PID regulator object
<i>hKdGain</i>	New Kd gain

Return values

<i>None</i>

Parameters

<i>CPID_PI</i>	PID regulator object
<i>int16_t</i>	New Kd gain

Return values

<i>None</i>

Definition at line 141 of file PID_PIRegulatorClass.c.

Referenced by UI_SetReg().

2.19.1.3 int16_t PID_GetKD (CPID_PI *this*)

It returns the Kd gain of the PID object passed.

Parameters

<i>this</i>	PID regulator object
-------------	----------------------

Return values

<i>int16_t</i>	Kd gain
----------------	---------

It returns the Kd gain of the PID object passed.

Parameters

<i>CPID_PI</i>	PI regulator object
----------------	---------------------

Return values

<i>int16_t</i>	Kd gain
----------------	---------

Definition at line 154 of file PID_PIRegulatorClass.c.

Referenced by UI_GetReg().

2.20 PWMnCurrFdbk class exported methods

Collaboration diagram for PWMnCurrFdbk class exported methods:



Functions

- uint16_t **PWMC_ExecRegularConv** (CPWMC this, uint8_t bChannel, uint8_t ADC_Unit)

Execute a regular conversion using ADC1. The function is not re-entrant (can't executed twice at the same time) It returns 0xFFFF in case of conversion error.
- uint16_t **PWMC_GetMultipleRegularConv** (CPWMC this, uint8_t Adc_Channel)

Get the user ADC_channel regular conversion.
- void **PWMC_ADC_SetSamplingTime** (CPWMC this, ADConv_t ADConv_struct)

It sets the specified sampling time for the specified ADC channel on ADC1. It must be called once for each channel utilized by user.

2.20.1 Function Documentation

2.20.1.1 uint16_t PWMC_ExecRegularConv (CPWMC this, uint8_t bChannel, uint8_t ADC_Unit)

Execute a regular conversion using ADC1. The function is not re-entrant (can't executed twice at the same time) It returns 0xFFFF in case of conversion error.

Parameters

<i>this</i>	related object of class CPWMC, ADC channel to be converted
<i>bChannel</i>	ADC channel used for the regular conversion

Return values

<i>It</i>	returns converted value or 0xFFFF for conversion error
-----------	--

Execute a regular conversion using ADC1. The function is not re-entrant (can't executed twice at the same time) It returns 0xFFFF in case of conversion error.

Parameters

<i>this</i>	related object of class CPWMC, ADC channel to be converted
<i>bChannel</i>	ADC channel used for the regular conversion

Return values

<i>It</i>	returns converted value or 0xFFFF for conversion error
-----------	--

Definition at line 345 of file PWMnCurrFdbkClass.c.

Referenced by TSK_SafetyTask().

2.20.1.2 `uint16_t PWMC_GetMultipleRegularConv (CPWMC this, uint8_t Adc_Channel)`

Get the user ADC_channel regular conversion.

Parameters

<i>this</i>	related object of class CPWMC, ADC channel to be converted
<i>Adc_Channel</i>	ADC channel used for the regular conversion

Return values

<i>It</i>	returns converted value
-----------	-------------------------

Definition at line 360 of file PWMnCurrFdbkClass.c.

Referenced by MC_GetMultipleRegularConv().

2.20.1.3 `void PWMC_ADC_SetSamplingTime (CPWMC this, ADConv_t ADConv_struct)`

It sets the specified sampling time for the specified ADC channel on ADC1. It must be called once for each channel utilized by user.

Parameters

<i>this</i>	related object of class CPWMC
<i>ADConv_struct</i>	struct containing ADC channel and sampling time

Return values

<i>none</i>

Definition at line 374 of file PWMnCurrFdbkClass.c.

Referenced by MC_RequestRegularConv().

2.21 RevupCtrl class exported methods

Collaboration diagram for RevupCtrl class exported methods:



Functions

- void [RUC_SetPhaseDurationms](#) (CRUC this, uint8_t bPhase, uint16_t hDurationms)

It is used to modify the default value of duration of a specific rev up phase. Note: The module can be also compiled commenting the define RUC_ALLOWS_TUNING to optimize the flash memory occupation and the RAM usage if the tuning is not required in this case this function has no effect.

- void [RUC_SetPhaseFinalMecSpeed01Hz](#) (CRUC this, uint8_t bPhase, int16_t hFinalMecSpeed01Hz)

It is used to modify the default value of mechanical speed at the end of a specific rev up phase. Note: The module can be also compiled commenting the define RUC_ALLOWS_TUNING to optimize the flash memory occupation and the RAM usage if the tuning is not required in this case this function has no effect.

- void [RUC_SetPhaseFinalTorque](#) (CRUC this, uint8_t bPhase, int16_t hFinalTorque)

It is used to modify the default value of motor torque at the end of a specific rev up phase. Note: The module can be also compiled commenting the define RUC_ALLOWS_TUNING to optimize the flash memory occupation and the RAM usage if the tuning is not required in this case this function has no effect.

- uint16_t [RUC_GetPhaseDurationms](#) (CRUC this, uint8_t bPhase)

It is used to read the current value of duration of a specific rev up phase. Note: The module can be also compiled commenting the define RUC_ALLOWS_TUNING to optimize the flash memory occupation and the RAM usage if the tuning is not required in this case this function has no effect.

- int16_t [RUC_GetPhaseFinalMecSpeed01Hz](#) (CRUC this, uint8_t bPhase)

It is used to read the current value of mechanical speed at the end of a specific rev up phase. Note: The module can be also compiled commenting the define RUC_ALLOWS_TUNING to optimize the flash memory occupation and the RAM usage if the tuning is not required in this case this function has no effect.

- int16_t [RUC_GetPhaseFinalTorque](#) (CRUC this, uint8_t bPhase)

It is used to read the current value of motor torque at the end of a specific rev up phase. Note: The module can be also compiled commenting the define RUC_ALLOWS_TUNING to optimize the flash memory occupation and the RAM usage if the tuning is not required in this case this function has no effect.

- uint8_t [RUC_GetNumberOfPhases](#) (CRUC this)

It is used to get information about the number of phases relative to the programmed rev up. Note: The module can be also compiled commenting the define RUC_ALLOWS_TUNING to optimize the flash memory occupation and the RAM usage if the tuning is not required in this case this function has no effect.

2.21.1 Function Documentation

2.21.1.1 void [RUC_SetPhaseDurationms](#) (CRUC this, uint8.t bPhase, uint16.t hDurationms)

It is used to modify the default value of duration of a specific rev up phase. Note: The module can be also compiled commenting the define RUC_ALLOWS_TUNING to optimize the flash memory occupation and the RAM usage if the tuning is not required in this case this function has no effect.

Parameters

<i>this</i>	related object of class CRUC.
<i>bPhase</i>	is the rev up phase, zero based, to be modified.
<i>hDurationms</i>	is the new value of duration for that phase.

Return values

<i>none.</i>

Definition at line 233 of file RevupCtrlClass.c.

Referenced by UI_SetRevupData().

2.21.1.2 void RUC_SetPhaseFinalMecSpeed01Hz (CRUC *this*, uint8_t *bPhase*, int16_t *hFinalMecSpeed01Hz*)

It is used to modify the default value of mechanical speed at the end of a specific rev up phase. Note: The module can be also compiled commenting the define RUC_ALLOWS_TUNING to optimize the flash memory occupation and the RAM usage if the tuning is not required in this case this function has no effect.

Parameters

<i>this</i>	related object of class CRUC.
<i>bPhase</i>	is the rev up phase, zero based, to be modified.
<i>hFinalMecSpeed01Hz</i>	is the new value of mechanical speed at the end of that phase expressed in 0.1Hz.

Return values

<i>none.</i>

Definition at line 255 of file RevupCtrlClass.c.

Referenced by UI_SetRevupData().

2.21.1.3 void RUC_SetPhaseFinalTorque (CRUC *this*, uint8_t *bPhase*, int16_t *hFinalTorque*)

It is used to modify the default value of motor torque at the end of a specific rev up phase. Note: The module can be also compiled commenting the define RUC_ALLOWS_TUNING to optimize the flash memory occupation and the RAM usage if the tuning is not required in this case this function has no effect.

Parameters

<i>this</i>	related object of class CRUC.
<i>bPhase</i>	is the rev up phase, zero based, to be modified.
<i>hFinalTorque</i>	is the new value of motor torque at the end of that phase. This value represents actually the Iq current expressed in digit.

Return values

<i>none.</i>

Definition at line 279 of file RevupCtrlClass.c.

Referenced by UI_SetRevupData().

2.21.1.4 uint16_t RUC_GetPhaseDurationms (CRUC *this*, uint8_t *bPhase*)

It is used to read the current value of duration of a specific rev up phase. Note: The module can be also compiled commenting the define RUC_ALLOWS_TUNING to optimize the flash memory occupation and the RAM usage if the tuning is not required in this case this function has no effect.

Parameters

<i>this</i>	related object of class CRUC.
<i>bPhase</i>	is the rev up phase, zero based, to be read.

Return values

<i>uint16_t</i>	The current value of duration for that phase expressed in milliseconds.
-----------------	---

Definition at line 300 of file RevupCtrlClass.c.

Referenced by UI_GetRevupData().

2.21.1.5 int16_t RUC_GetPhaseFinalMecSpeed01Hz (CRUC *this*, uint8_t *bPhase*)

It is used to read the current value of mechanical speed at the end of a specific rev up phase. Note: The module can be also compiled commenting the define RUC_ALLOWS_TUNING to optimize the flash memory occupation and the RAM usage if the tuning is not required in this case this function has no effect.

Parameters

<i>this</i>	related object of class CRUC.
<i>bPhase</i>	is the rev up phase, zero based, to be read.

Return values

<i>int16_t</i>	The current value of mechanical speed at the end of that phase expressed in 0.1Hz.
----------------	--

Definition at line 323 of file RevupCtrlClass.c.

Referenced by UI_GetRevupData().

2.21.1.6 int16_t RUC_GetPhaseFinalTorque (CRUC *this*, uint8_t *bPhase*)

It is used to read the current value of motor torque at the end of a specific rev up phase. Note: The module can be also compiled commenting the define RUC_ALLOWS_TUNING to optimize the flash memory occupation and the RAM usage if the tuning is not required in this case this function has no effect.

Parameters

<i>this</i>	related object of class CRUC.
<i>bPhase</i>	is the rev up phase, zero based, to be read.

Return values

<i>int16_t</i>	The current value of motor torque at the end of that phase. This value represents actually the Iq current expressed in digit.
----------------	---

Definition at line 346 of file RevupCtrlClass.c.

Referenced by UI_GetRevupData().

2.21.1.7 uint8_t RUC_GetNumberOfPhases (CRUC *this*)

It is used to get information about the number of phases relative to the programmed rev up. Note: The module can be also compiled commenting the define RUC_ALLOWS_TUNING to optimize the flash memory occupation and the RAM usage if the tuning is not required in this case this function has no effect.

Parameters

<i>this</i>	related object of class CRUC.
-------------	-------------------------------

Return values

<i>uint8_t</i>	The number of phases relative to the programmed rev up.
----------------	---

Definition at line 367 of file RevupCtrlClass.c.

Referenced by UI_GetReg().

2.22 SpeednPosFdbk class exported methods

Collaboration diagram for SpeednPosFdbk class exported methods:



Functions

- int16_t SPD_GetElAngle (**CSPD** this)

It returns the last computed rotor electrical angle, expressed in s16degrees. 1 s16degree = 360/65536.

- int16_t SPD_GetMecAngle (**CSPD** this)

It returns the last computed rotor mechanical angle, expressed in s16degrees. Mechanical angle frame is based on parameter bElToMecRatio and, if occasionally provided - through function SPD_SetMecAngle - of a measured mechanical angle, on information computed thereof.

- int16_t SPD_GetAvrgMecSpeed01Hz (**CSPD** this)

It returns the last computed average mechanical speed, expressed in 01Hz (tenth of Hertz).

- int16_t SPD_GetElSpeedDpp (**CSPD** this)

It returns the last computed electrical speed, expressed in Dpp. 1 Dpp = 1 s16Degree/control Period. The control period is the period on which the rotor electrical angle is computed (through function SPD_CalcElectricalAngle).

- bool SPD_Check (**CSPD** this)

It returns the result of the last reliability check performed. Reliability is measured with reference to parameters hMaxReliableElSpeed01Hz, hMinReliableElSpeed01Hz, bMaximumSpeedErrorsNumber and/or specific parameters of the derived TRUE = sensor information is reliable FALSE = sensor information is not reliable.

- int16_t SPD_GetS16Speed (**CSPD** this)

This method returns the average meccanical rotor speed expressed in "S16Speed". It means that:

2.22.1 Function Documentation

2.22.1.1 int16_t SPD_GetElAngle (**CSPD** this)

It returns the last computed rotor electrical angle, expressed in s16degrees. 1 s16degree = 360/65536.

Parameters

<i>this</i>	related object of class CSPD
-------------	------------------------------

Return values

<i>int16_t</i>	rotor electrical angle (s16degrees)
----------------	-------------------------------------

It returns the last computed rotor electrical angle, expressed in s16degrees. 1 s16degree = 360/65536.

Parameters

in	<i>this</i>	related object of class CSPD
----	-------------	------------------------------

Returns

the rotor electrical angle (s16degrees)

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Definition at line 166 of file SpeednPosFdbkClass.c.

Referenced by TSK_HighFrequencyTask(), and UI_GetReg().

2.22.1.2 int16_t SPD_GetMecAngle (CSPD *this*)

It returns the last computed rotor mechanical angle, expressed in s16degrees. Mechanical angle frame is based on parameter bEIToMecRatio and, if occasionally provided - through function SPD_SetMecAngle - of a measured mechanical angle, on information computed thereof.

Parameters

<i>this</i>	related object of class CSPD
-------------	------------------------------

Return values

<i>int16_t</i>	rotor mechanical angle (s16degrees)
----------------	-------------------------------------

It returns the last computed rotor mechanical angle, expressed in s16degrees. Mechanical angle frame is based on parameter bEIToMecRatio and, if occasionally provided - through function SPD_SetMecAngle - of a measured mechanical angle, on information computed thereof.

Parameters

in	<i>this</i>	related object of class CSPD
----	-------------	------------------------------

Returns

the rotor mechanical angle (s16degrees)

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Definition at line 190 of file SpeednPosFdbkClass.c.

2.22.1.3 int16_t SPD_GetAvrgMecSpeed01Hz (CSPD *this*)

It returns the last computed average mechanical speed, expressed in 01Hz (tenth of Hertz).

Parameters

<i>this</i>	related object of class CSPD
-------------	------------------------------

Return values

<i>int16_t</i>	rotor average mechanical speed (01Hz)
----------------	---------------------------------------

Parameters

in	<i>this</i>	related object of class CSPD
----	-------------	------------------------------

Returns

the rotor average mechanical speed (01Hz)

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Definition at line 210 of file SpeednPosFdbkClass.c.

Referenced by MCI_GetAvrgMecSpeed01Hz(), OL_Calc(), STC_CalcTorqueReference(), and STC_ExecRamp().

2.22.1.4 int16_t SPD_GetEISpeedDpp (CSPD *this*)

It returns the last computed electrical speed, expressed in Dpp. 1 Dpp = 1 s16Degree/control Period. The control period is the period on which the rotor electrical angle is computed (through function SPD_CalcElectricalAngle).

Parameters

<i>this</i>	related object of class CSPD
-------------	------------------------------

Return values

<i>int16_t</i>	rotor electrical speed (Dpp)
----------------	------------------------------

Parameters

in	<i>this</i>	related object of class CSPD
----	-------------	------------------------------

Returns

the rotor electrical speed (Dpp)

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Definition at line 232 of file SpeednPosFdbkClass.c.

2.22.1.5 bool SPD_Check (CSPD *this*)

It returns the result of the last reliability check performed. Reliability is measured with reference to parameters h-MaxReliableEISpeed01Hz, hMinReliableEISpeed01Hz, bMaximumSpeedErrorsNumber and/or specific parameters of the derived TRUE = sensor information is reliable FALSE = sensor information is not reliable.

Parameters

<i>this</i>	related object of class CSPD
-------------	------------------------------

Return values

<i>bool</i>	sensor reliability state
-------------	--------------------------

It returns the result of the last reliability check performed. Reliability is measured with reference to parameters h-

MaxReliableEISpeed01Hz, hMinReliableEISpeed01Hz, bMaximumSpeedErrorsNumber and/or specific parameters of the derived TRUE = sensor information is reliable FALSE = sensor information is not reliable.

Parameters

in	<i>this</i>	related object of class CSPD
----	-------------	------------------------------

Returns

bool sensor reliability state

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Definition at line 301 of file SpeednPosFdbkClass.c.

Referenced by MCI_GetSpdSensorReliability().

2.22.1.6 int16_t SPD_GetS16Speed (CSPD *this*)

This method returns the average mechanical rotor speed expressed in "S16Speed". It means that:

- it is zero for zero speed,
- it become S16_MAX when the average mechanical speed is equal to hMaxReliableMecSpeed01Hz,
- it becomes -S16_MAX when the average mechanical speed is equal to -hMaxReliableMecSpeed01Hz.

Parameters

<i>this</i>	related object of class CSPD
-------------	------------------------------

Return values

int16_t	The average mechanical rotor speed expressed in "S16Speed".
---------	---

This method returns the average mechanical rotor speed expressed in "S16Speed". It means that:

- it is zero for zero speed,
- it become S16_MAX when the average mechanical speed is equal to hMaxReliableMecSpeed01Hz,
- it becomes -S16_MAX when the average mechanical speed is equal to -hMaxReliableMecSpeed01Hz.

Parameters

in	<i>this</i>	related object of class CSPD
----	-------------	------------------------------

Returns

The average mechanical rotor speed expressed in "S16Speed".

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Definition at line 479 of file SpeednPosFdbkClass.c.

Referenced by UI_GetReg().

2.23 STO class exported methods

Collaboration diagram for STO class exported methods:



Functions

- **Volt_Components STO_GetEstimatedBemf (CSTO_SPD this)**
It exports estimated Bemf alpha-beta in Volt_Components format.
- **Curr_Components STO_GetEstimatedCurrent (CSTO_SPD this)**
It exports the stator current alpha-beta as estimated by state observer.
- void **STO_GetObserverGains (CSTO_SPD this, int16_t *pC2, int16_t *pC4)**
It exports current observer gains through parameters hC2 and hC4.
- void **STO_SetObserverGains (CSTO_SPD this, int16_t hC1, int16_t hC2)**
It allows setting new values for observer gains.
- void **STO_GetPLLGains (CSTO_SPD this, int16_t *pPgain, int16_t *plgain)**
It exports current PLL gains through parameters pPgain and plgain.
- void **STO_SetPLLGains (CSTO_SPD this, int16_t hPgain, int16_t hlgain)**
It allows setting new values for PLL gains.
- void **STO_ResetPLL (CSTO_SPD this)**
It resets integral term of PLL.
- int32_t **STO_GetEstimatedBemfLevel (CSTO_SPD this)**
It exports estimated Bemf squared level.
- int32_t **STO_GetObservedBemfLevel (CSTO_SPD this)**
It exports observed Bemf squared level.
- void **STO_BemfConsistencyCheckSwitch (CSTO_SPD this, bool bSel)**
It enables/disables the bemf consistency check.
- bool **STO_IsBemfConsistent (CSTO_SPD this)**
It returns the result of the Bemf consistency check.

2.23.1 Function Documentation

2.23.1.1 Volt_Components STO_GetEstimatedBemf (CSTO_SPD this)

It exports estimated Bemf alpha-beta in Volt_Components format.

Parameters

<i>this</i>	related object of class CSTO_SPD
-------------	----------------------------------

Return values

<i>Volt_Components</i>	Bemf alpha-beta
------------------------	-----------------

Definition at line 782 of file STO_SpeednPosFdbkClass.c.

Referenced by UI_GetReg().

2.23.1.2 Curr_Components STO_GetEstimatedCurrent (CSTO_SPD *this*)

It exports the stator current alpha-beta as estimated by state observer.

Parameters

<i>this</i>	related object of class CSTO_SPD
-------------	----------------------------------

Return values

<i>Curr_Components</i>	State observer estimated stator current lalpha-beta
------------------------	---

Definition at line 797 of file STO_SpeednPosFdbkClass.c.

Referenced by UI_GetReg().

2.23.1.3 void STO_GetObserverGains (CSTO_SPD *this*, int16_t * *pC2*, int16_t * *pC4*)

It exports current observer gains through parameters hC2 and hC4.

Parameters

<i>this</i>	related object of class CSTO_SPD
<i>pC2</i>	pointer to int16_t used to return parameters hC2
<i>pC4</i>	pointer to int16_t used to return parameters hC4

Return values

<i>none</i>

Definition at line 817 of file STO_SpeednPosFdbkClass.c.

Referenced by UI_GetReg(), and UI_SetReg().

2.23.1.4 void STO_SetObserverGains (CSTO_SPD *this*, int16_t *hC1*, int16_t *hC2*)

It allows setting new values for observer gains.

Parameters

<i>this</i>	related object of class CSTO_SPD
<i>hC2</i>	new value for observer gain hC1
<i>hC4</i>	new value for observer gain hC2

Return values

<i>none</i>

Parameters

<i>this</i>	related object of class CSTO_SPD
<i>wK1</i>	new value for observer gain hC1
<i>wK2</i>	new value for observer gain hC2

Return values

<i>none</i>

Definition at line 832 of file STO_SpeednPosFdbkClass.c.

Referenced by UI_SetReg().

2.23.1.5 void STO_GetPLLGains (CSTO_SPD *this*, int16_t * *pPgain*, int16_t * *plgain*)

It exports current PLL gains through parameters pPgain and plgain.

Parameters

<i>this</i>	related object of class CSTO_SPD
<i>pPgain</i>	pointer to int16_t used to return PLL proportional gain
<i>plgain</i>	pointer to int16_t used to return PLL integral gain

Return values

<i>none</i>

Definition at line 846 of file STO_SpeednPosFdbkClass.c.

Referenced by UI_GetReg(), and UI_SetReg().

2.23.1.6 void STO_SetPLLGains (CSTO_SPD *this*, int16_t *hPgain*, int16_t *hlgain*)

It allows setting new values for PLL gains.

Parameters

<i>this</i>	related object of class CSTO_SPD
<i>hPgain</i>	new value for PLL proportional gain
<i>hlgain</i>	new value for PLL integral gain

Return values

<i>none</i>

Definition at line 861 of file STO_SpeednPosFdbkClass.c.

Referenced by UI_SetReg().

2.23.1.7 void STO_ResetPLL (CSTO_SPD *this*)

It resets integral term of PLL.

Parameters

<i>this</i>	related object of class CSTO_SPD
-------------	----------------------------------

Return values

<i>none</i>

Definition at line 894 of file STO_SpeednPosFdbkClass.c.

Referenced by TSK_HighFrequencyTask().

2.23.1.8 int32_t STO_GetEstimatedBemfLevel (CSTO_SPD *this*)

It exports estimated Bemf squared level.

Parameters

<i>this</i>	related object of class CSTO_SPD
-------------	----------------------------------

Return values

<i>int32_t</i>

Definition at line 916 of file STO_SpeednPosFdbkClass.c.

Referenced by UI_GetReg().

2.23.1.9 int32_t STO_GetObservedBemfLevel (CSTO_SPD *this*)

It exports observed Bemf squared level.

Parameters

<i>this</i>	related object of class CSTO_SPD
-------------	----------------------------------

Return values

<i>int32_t</i>

Definition at line 926 of file STO_SpeednPosFdbkClass.c.

Referenced by UI_GetReg().

2.23.1.10 void STO_BemfConsistencyCheckSwitch (CSTO_SPD *this*, bool *bSel*)

It enables/disables the bemf consistency check.

Parameters

<i>this</i>	related object of class CSTO_SPD
<i>bSel</i>	boolean; TRUE enables check; FALSE disables check

Return values

<i>int32_t</i>

Parameters

<i>this</i>	related object of class CSTO_SPD
<i>bSel</i>	true enables check; false disables check

Definition at line 936 of file STO_SpeednPosFdbkClass.c.

2.23.1.11 bool STO_IsBemfConsistent(CSTO_SPD *this*)

It returns the result of the Bemf consistency check.

Parameters

<i>this</i>	related object of class CSTO_SPD
-------------	----------------------------------

Return values

<i>bool</i>	Bemf consistency state
-------------	------------------------

Definition at line 946 of file STO_SpeednPosFdbkClass.c.

2.24 STO_CORDIC class exported methods

Collaboration diagram for STO_CORDIC class exported methods:



Functions

- **[Volt_Components STO_CR_GetEstimatedBemf \(CSTO_CR_SPD this\)](#)**
It exports estimated Bemf alpha-beta in Volt_Components format.
- **[Curr_Components STO_CR_GetEstimatedCurrent \(CSTO_CR_SPD this\)](#)**
It exports the stator current alpha-beta as estimated by state observer.
- **[void STO_CR_GetObserverGains \(CSTO_CR_SPD this, int16_t *pC2, int16_t *pC4\)](#)**
It exports current observer gains through parameters hC2 and hC4.
- **[void STO_CR_SetObserverGains \(CSTO_CR_SPD this, int16_t hC1, int16_t hC2\)](#)**
It allows setting new values for observer gains.
- **[int32_t STO_CR_GetEstimatedBemfLevel \(CSTO_CR_SPD this\)](#)**
It exports estimated Bemf squared level.
- **[int32_t STO_CR_GetObservedBemfLevel \(CSTO_CR_SPD this\)](#)**
It exports observed Bemf squared level.
- **[void STO_CR_BemfConsistencyCheckSwitch \(CSTO_CR_SPD this, bool bSel\)](#)**
It enables/disables the bemf consistency check.
- **[bool STO_CR_IsBemfConsistent \(CSTO_CR_SPD this\)](#)**
It returns the result of the Bemf consistency check.

2.24.1 Function Documentation

2.24.1.1 **Volt_Components STO_CR_GetEstimatedBemf (CSTO_CR_SPD this)**

It exports estimated Bemf alpha-beta in Volt_Components format.

Parameters

<i>this</i>	related object of class CSTO_CR_SPD
-------------	-------------------------------------

Return values

<i>Volt_Components</i>	Bemf alpha-beta
------------------------	-----------------

Referenced by UI_GetReg().

2.24.1.2 **Curr_Components STO_CR_GetEstimatedCurrent (CSTO_CR_SPD this)**

It exports the stator current alpha-beta as estimated by state observer.

Parameters

<i>this</i>	related object of class CSTO_CR_SPD
-------------	-------------------------------------

Return values

<i>Curr_Components</i>	State observer estimated stator current lalpha-beta
------------------------	---

Referenced by UI_GetReg().

2.24.1.3 void STO_CR_GetObserverGains (CSTO_CR_SPD *this*, int16_t * *pC2*, int16_t * *pC4*)

It exports current observer gains through parameters hC2 and hC4.

Parameters

<i>this</i>	related object of class CSTO_CR_SPD
<i>pC2</i>	pointer to int16_t used to return parameters hC2
<i>pC4</i>	pointer to int16_t used to return parameters hC4

Return values

<i>none</i>

Referenced by UI_GetReg(), and UI_SetReg().

2.24.1.4 void STO_CR_SetObserverGains (CSTO_CR_SPD *this*, int16_t *hC1*, int16_t *hC2*)

It allows setting new values for observer gains.

Parameters

<i>this</i>	related object of class CSTO_CR_SPD
<i>hC1</i>	new value for observer gain hC1
<i>hC2</i>	new value for observer gain hC2

Return values

<i>none</i>

Referenced by UI_SetReg().

2.24.1.5 int32_t STO_CR_GetEstimatedBemfLevel (CSTO_CR_SPD *this*)

It exports estimated Bemf squared level.

Parameters

<i>this</i>	related object of class CSTO_CR_SPD
-------------	-------------------------------------

Return values

<i>int32_t</i>

Referenced by UI_GetReg().

2.24.1.6 int32_t STO_CR_GetObservedBemfLevel (CSTO_CR_SPD *this*)

It exports observed Bemf squared level.

Parameters

<i>this</i>	related object of class CSTO_CR_SPD
-------------	-------------------------------------

Return values

<i>int32_t</i>

Referenced by UI_GetReg().

2.24.1.7 void STO_CR_BemfConsistencyCheckSwitch (CSTO_CR_SPD *this*, bool *bSel*)

It enables/disables the bemf consistency check.

Parameters

<i>this</i>	related object of class CSTO_CR_SPD
<i>bSel</i>	boolean; TRUE enables check; FALSE disables check

2.24.1.8 bool STO_CR_IsBemfConsistent (CSTO_CR_SPD *this*)

It returns the result of the Bemf consistency check.

Parameters

<i>this</i>	related object of class CSTO_CR_SPD
-------------	-------------------------------------

Return values

<i>bool</i>	Bemf consistency state
-------------	------------------------

2.25 VSS class exported methods

Collaboration diagram for VSS class exported methods:



Functions

- void **VSPD_SetMecAcceleration** (**CSPD** this, int16_t hFinalMecSpeed01Hz, uint16_t hDurationms)

Set the mechanical acceleration of virtual sensor. This acceleration is defined starting from current mechanical speed, final mechanical speed expressed in 0.1Hz and duration expressed in milliseconds.
- int16_t **VSPD_GetLastRampFinalSpeed** (**CSPD** this)

Get the final speed of last settled ramp of virtual sensor expressed in 0.1Hz.

2.25.1 Function Documentation

2.25.1.1 void **VSPD_SetMecAcceleration** (**CSPD** this, int16_t hFinalMecSpeed01Hz, uint16_t hDurationms)

Set the mechanical acceleration of virtual sensor. This acceleration is defined starting from current mechanical speed, final mechanical speed expressed in 0.1Hz and duration expressed in milliseconds.

Parameters

<i>this</i>	related object of class CSTC.
<i>hFinalMecSpeed01Hz</i>	mechanical speed expressed in 0.1Hz assumed by the virtual sensor at the end of the duration.
<i>hDurationms</i>	Duration expressed in ms. It can be 0 to apply instantaneous the final speed.

Return values

<i>none</i>

Definition at line 317 of file VirtualSpeedSensor_SpeednPosFdbkClass.c.

Referenced by EAC_StartAlignment(), RUC_Clear(), and RUC_Exec().

2.25.1.2 int16.t **VSPD_GetLastRampFinalSpeed** (**CSPD** this)

Get the final speed of last settled ramp of virtual sensor expressed in 0.1Hz.

Parameters

<i>this</i>	related object of class CSTC.
<i>hFinalMecSpeed01Hz</i>	mechanical speed expressed in 0.1Hz assumed by the virtual sensor at the end of the duration.
<i>hDurationms</i>	Duration expressed in ms. It can be 0 to apply instantaneous the final speed.

Return values

<i>none</i>

Definition at line 385 of file VirtualSpeedSensor_SpeednPosFdbkClass.c.

2.26 SpeednTorqCtrl class exported methods

Collaboration diagram for SpeednTorqCtrl class exported methods:



Functions

- **int16_t STC_GetMecSpeedRef01Hz (CSTC this)**
Get the current mechanical rotor speed reference expressed in tenths of HZ.
- **int16_t STC_GetTorqueRef (CSTC this)**
*Get the current motor torque reference. This value represents actually the Iq current reference expressed in digit. To convert current expressed in digit to current expressed in Amps is possible to use the formula: Current(Amp) = [Current(digit) * Vdd micro] / [65536 * Rshunt * Aop].*
- **STC_Modality_t STC_GetControlMode (CSTC this)**
Get the modality of the speed and torque controller.
- **uint16_t STC_GetMaxAppPositiveMecSpeed01Hz (CSTC this)**
Get the Application maximum positive value of rotor speed. It's expressed in tenth of mechanical Hertz.
- **int16_t STC_GetMinAppNegativeMecSpeed01Hz (CSTC this)**
Get the Application minimum negative value of rotor speed. It's expressed in tenth of mechanical Hertz.
- **Curr_Components STC_GetDefaultIqdref (CSTC this)**
It returns the default values of Iqdref.

2.26.1 Function Documentation

2.26.1.1 int16_t STC_GetMecSpeedRef01Hz (CSTC this)

Get the current mechanical rotor speed reference expressed in tenths of HZ.

Parameters

<i>this</i>	related object of class CSTC.
-------------	-------------------------------

Return values

<i>int16_t</i>	current mechanical rotor speed reference expressed in tenths of HZ.
----------------	---

Definition at line 147 of file SpeednTorqCtrlClass.c.

Referenced by MCI_GetMecSpeedRef01Hz().

2.26.1.2 int16_t STC_GetTorqueRef (CSTC this)

Get the current motor torque reference. This value represents actually the Iq current reference expressed in digit. To convert current expressed in digit to current expressed in Amps is possible to use the formula: Current(Amp) = [Current(digit) * Vdd micro] / [65536 * Rshunt * Aop].

Parameters

<i>this</i>	related object of class CSTC.
-------------	-------------------------------

Return values

<i>int16_t</i>	current motor torque reference. This value represents actually the Iq current expressed in digit.
----------------	---

Definition at line 162 of file SpeednTorqCtrlClass.c.

Referenced by STC_ExecRamp().

2.26.1.3 STC_Modality_t STC_GetControlMode (CSTC *this*)

Get the modality of the speed and torque controller.

Parameters

<i>this</i>	related object of class CSTC.
-------------	-------------------------------

Return values

<i>STC_Modality_t</i>	It returns the modality of STC. It can be one of these two values: STC_TORQUE_MODE or STC_SPEED_MODE.
-----------------------	---

Definition at line 197 of file SpeednTorqCtrlClass.c.

2.26.1.4 uint16_t STC_GetMaxAppPositiveMecSpeed01Hz (CSTC *this*)

Get the Application maximum positive value of rotor speed. It's expressed in tenth of mechanical Hertz.

Parameters

<i>this</i>	related object of class CSTC.
-------------	-------------------------------

Return values

<i>uint16_t</i>	It returns the application maximum positive value of rotor speed expressed in tenth of mechanical Hertz.
-----------------	--

Definition at line 426 of file SpeednTorqCtrlClass.c.

Referenced by UI_GetReg().

2.26.1.5 int16_t STC_GetMinAppNegativeMecSpeed01Hz (CSTC *this*)

Get the Application minimum negative value of rotor speed. It's expressed in tenth of mechanical Hertz.

Parameters

<i>this</i>	related object of class CSTC.
-------------	-------------------------------

Return values

<i>uint16_t</i>	It returns the application minimum negative value of rotor speed expressed in tenth of mechanical Hertz.
-----------------	--

Definition at line 440 of file SpeednTorqCtrlClass.c.

Referenced by UI_GetReg().

2.26.1.6 Curr_Components STC_GetDefaultIqdref (CSTC *this*)

It returns the default values of Iqdref.

Parameters

<i>this</i>	related object of class CSTC
-------------	------------------------------

Return values

<i>default</i>	values of Iqdref.
----------------	-------------------

Definition at line 468 of file SpeednTorqCtrlClass.c.

Referenced by MCI_Clear_Iqdref().

2.27 StateMachine class exported methods

Collaboration diagram for StateMachine class exported methods:



Functions

- **[State_t STM_GetState \(CSTM this \)](#)**

Returns the current state machine state.
- **[uint32_t STM_GetFaultState \(CSTM this \)](#)**

*It returns two 16 bit fields containing information about both faults currently present and faults historically occurred since the state machine has been moved into state
Returned error codes are listed here .*

2.27.1 Function Documentation

2.27.1.1 **[State_t STM_GetState \(CSTM this \)](#)**

Returns the current state machine state.

Parameters

<i>this</i>	object of class CSTM
-------------	----------------------

Return values

<i>State_t</i>	Current state machine state
----------------	-----------------------------

Definition at line 324 of file StateMachineClass.c.

Referenced by [MCI_GetSTMState\(\)](#), [MCI_RampCompleted\(\)](#), [TSK_HighFrequencyTask\(\)](#), and [UI_GetReg\(\)](#).

2.27.1.2 **[uint32_t STM_GetFaultState \(CSTM this \)](#)**

It returns two 16 bit fields containing information about both faults currently present and faults historically occurred since the state machine has been moved into state

[Returned error codes are listed here .](#)

Parameters

<i>this</i>	object of class CSTM.
-------------	-----------------------

Return values

<i>uint32_t</i>	Two 16 bit fields: in the most significant half are stored the information about currently present faults. In the least significant half are stored the information about the faults historically occurred since the state machine has been moved into FAULT_NOW state Returned error codes are listed here
-----------------	--

It returns two 16 bit fields containing information about both faults currently present and faults historically occurred since the state machine has been moved into state

[Returned error codes are listed here .](#)

Parameters

<i>this</i>	object of class CSTM.
-------------	-----------------------

Return values

<i>uint32_t</i>	Two 16 bit fields: in the most significant half are stored the information about currently present faults. In the least significant half are stored the information about the faults historically occurred since the state machine has been moved into FAULT_NOW state
-----------------	--

Definition at line 367 of file StateMachineClass.c.

Referenced by MCI_GetCurrentFaults(), MCI_GetOccurredFaults(), and UI_GetReg().

2.28 Temperature sensor class

Collaboration diagram for Temperature sensor class:



2.28.1 Detailed Description

exported methods

Functions

- `int16_t TSNS_GetAvTemp_C (CTSNS this)`
It returns latest averaged temperature measurement expressed in Celsius degrees.
- `uint16_t TSNS_CheckTemp (CTSNS this)`
It returns MC_OVER_TEMP or MC_NO_ERROR depending on temperature sensor output measurement and protection threshold values.

2.28.2 Function Documentation

2.28.2.1 `int16_t TSNS_GetAvTemp_C (CTSNS this)`

It returns latest averaged temperature measurement expressed in Celsius degrees.

Parameters

<code>this</code>	related object of class CTSNS
-------------------	-------------------------------

Return values

<code>int16_t</code>	Latest averaged temperature measurement in Celsius degrees
----------------------	--

Parameters

<code>this</code>	related object of class CTSNS
-------------------	-------------------------------

Return values

<code>uint16_t</code>	Latest averaged temperature measurement in Celsius degrees
-----------------------	--

Definition at line 124 of file TemperatureSensorClass.c.

Referenced by UI_GetReg().

2.28.2.2 uint16_t TSNS_CheckTemp (CTSNS *this*)

It returns MC_OVER_TEMP or MC_NO_ERROR depending on temperature sensor output measurement and protection threshold values.

Parameters

<i>this</i>	related object of class CTSNS
-------------	-------------------------------

Return values

<i>uint16_t</i>	Fault code error
-----------------	------------------

It returns MC_OVER_TEMP or MC_NO_ERROR depending on temperature sensor output measurement and protection threshold values.

Parameters

<i>this</i>	related object of class CTSNS
-------------	-------------------------------

Return values

<i>uint16_t</i>	Fault code error
-----------------	------------------

Definition at line 135 of file TemperatureSensorClass.c.

2.29 BusVoltageSensor class exported methods

Collaboration diagram for BusVoltageSensor class exported methods:



Functions

- `uint16_t VBS_GetAvBusVoltage_V (CVBS this)`
It returns latest averaged Vbus measurement expressed in Volts.
- `uint16_t VBS_CheckVbus (CVBS this)`
It returns MC_OVER_VOLT, MC_UNDER_VOLT or MC_NO_ERROR depending on bus voltage measurement and protection threshold values.

2.29.1 Function Documentation

2.29.1.1 `uint16_t VBS_GetAvBusVoltage_V (CVBS this)`

It returns latest averaged Vbus measurement expressed in Volts.

Parameters

<code>this</code>	related object of class CVBS
-------------------	------------------------------

Return values

<code>uint16_t</code>	Latest averaged Vbus measurement in Volts
-----------------------	---

It returns latest averaged Vbus measurement expressed in Volts.

Parameters

<code>this</code>	related object of class CVBS
-------------------	------------------------------

Return values

<code>uint16_t</code>	Latest averaged Vbus measurement in Volts
-----------------------	---

Definition at line 139 of file BusVoltageSensorClass.c.

Referenced by UI_GetReg().

2.29.1.2 `uint16_t VBS_CheckVbus (CVBS this)`

It returns MC_OVER_VOLT, MC_UNDER_VOLT or MC_NO_ERROR depending on bus voltage measurement and protection threshold values.

Parameters

<i>this</i>	related object of class CVBS
-------------	------------------------------

Return values

<i>uint16_t</i>	Fault code error
-----------------	------------------

It returns MC_OVER_VOLT, MC_UNDER_VOLT or MC_NO_ERROR depending on bus voltage measurement and protection threshold values.

Parameters

<i>this</i>	related object of class CVBS
-------------	------------------------------

Return values

<i>uint16_t</i>	Fault code error
-----------------	------------------

Definition at line 156 of file BusVoltageSensorClass.c.

2.30 DigitalOutput class exported methods

Collaboration diagram for DigitalOutput class exported methods:



Functions

- `DOutputState_t DOUT_GetOutputState (CDOOUT this)`

It returns the state of the digital output.

2.30.1 Function Documentation

2.30.1.1 `DOutputState_t DOUT_GetOutputState (CDOOUT this)`

It returns the state of the digital output.

Parameters

<code>this</code>	object of class DOUT
-------------------	----------------------

Return values

<code>OutputState_t</code>	Digital output state (ACTIVE or INACTIVE)
----------------------------	---

2.31 MotorPowerMeasurement class exported methods

Collaboration diagram for MotorPowerMeasurement class exported methods:



Functions

- `int16_t MPM_GetElMotorPowerW (CMPM this)`
This method is used to get the last measured motor power (instantaneous value) expressed in watt.
- `int16_t MPM_GetAvrgElMotorPowerW (CMPM this)`
This method is used to get the average measured motor power expressed in watt.

2.31.1 Function Documentation

2.31.1.1 `int16_t MPM_GetElMotorPowerW (CMPM this)`

This method is used to get the last measured motor power (instantaneous value) expressed in watt.

Parameters

<code>this</code>	related object of class CMPM.
-------------------	-------------------------------

Return values

<code>int16_t</code>	The last measured motor power (instantaneous value) expressed in watt.
----------------------	--

Definition at line 146 of file MotorPowerMeasurementClass.c.

2.31.1.2 `int16_t MPM_GetAvrgElMotorPowerW (CMPM this)`

This method is used to get the average measured motor power expressed in watt.

Parameters

<code>this</code>	related object of class CMPM.
-------------------	-------------------------------

Return values

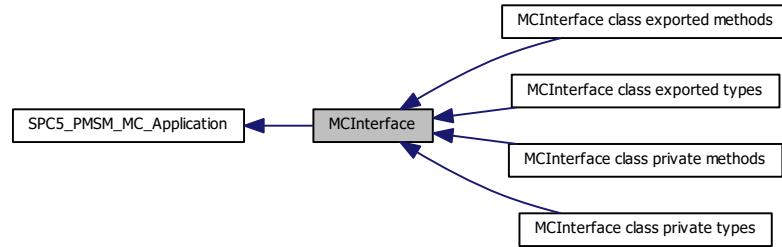
<code>int16_t</code>	The average measured motor power expressed in watt.
----------------------	---

Definition at line 158 of file MotorPowerMeasurementClass.c.

Referenced by `UI_GetReg()`.

2.32 MCInterface

Collaboration diagram for MCInterface:

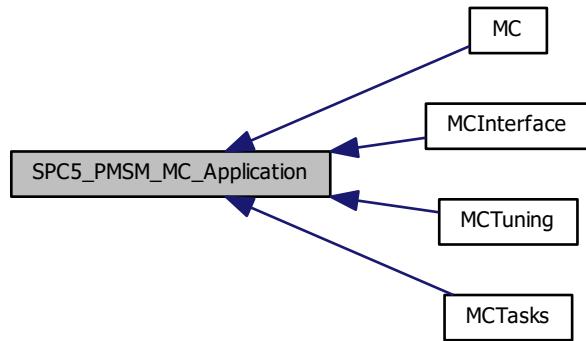


Modules

- `MCInterface class private methods`
- `MCInterface class private types`
- `MCInterface class exported types`
- `MCInterface class exported methods`

2.33 SPC5_PMSM_MC_Application

Collaboration diagram for SPC5_PMSM_MC_Application:

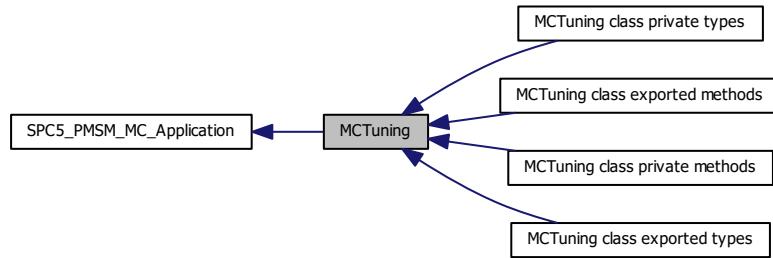


Modules

- MCInterface
- MCTuning
- MC
- MCTasks

2.34 MCTuning

Collaboration diagram for MCTuning:

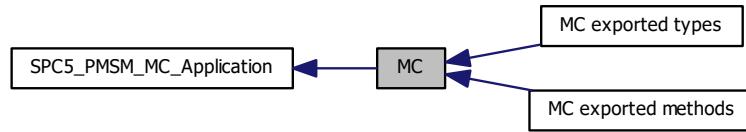


Modules

- `MCTuning class exported types`
- `MCTuning class private methods`
- `MCTuning class private types`
- `MCTuning class exported methods`

2.35 MC

Collaboration diagram for MC:

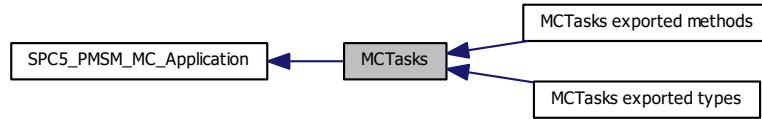


Modules

- MC exported types
- MC exported methods

2.36 MCTasks

Collaboration diagram for MCTasks:



Modules

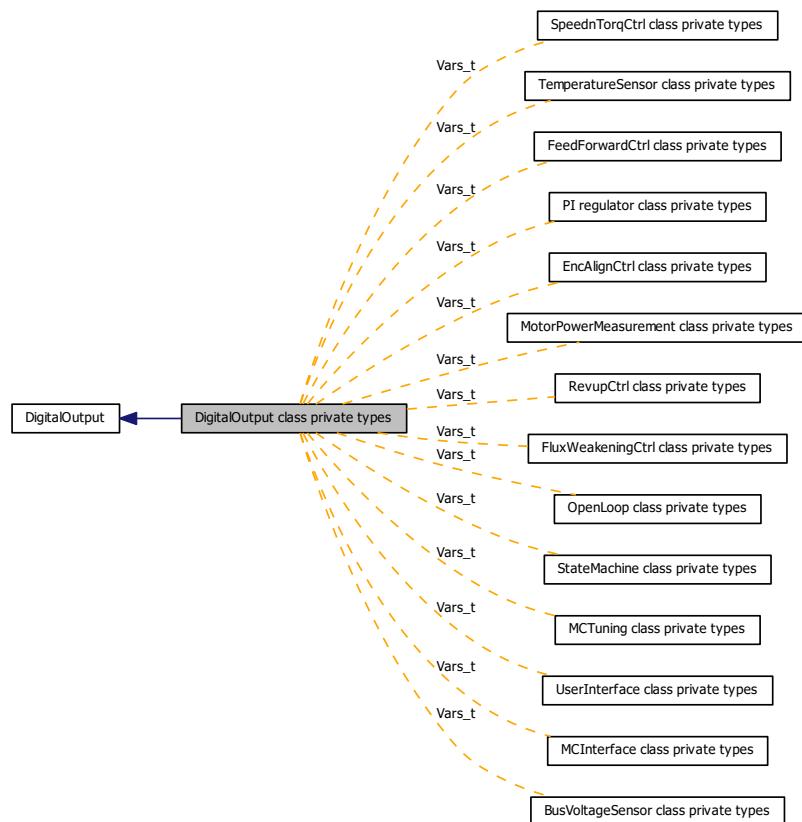
- MCTasks exported types
- MCTasks exported methods

Chapter 3

SPC5 PMSM MC Library

3.1 DigitalOutput class private types

Collaboration diagram for DigitalOutput class private types:



Data Structures

- struct [Vars_t](#)

MCInterface class members definition.

- struct [CDOOUT_t](#)
Private DigitalOutput class definition.

Typedefs

- [typedef DigitalOutputParams_t Params_t](#)
Redefinition of parameter structure.

3.1.1 Typedef Documentation

3.1.1.1 [typedef DigitalOutputParams_t Params_t](#)

Redefinition of parameter structure.

Definition at line 57 of file DigitalOutputPrivate.h.

3.2 DigitalOutput class exported constants

Collaboration diagram for DigitalOutput class exported constants:



3.3 DigitalOutput class exported types

Collaboration diagram for DigitalOutput class exported types:



Data Structures

- struct [DigitalOutputParams_t](#)
DigitalOutput class parameters definition.

Typedefs

- typedef struct CDOUT_t * [CDOUT](#)
Public DigitalOutput class definition.

3.3.1 Typedef Documentation

3.3.1.1 [typedef struct CDOUT_t*](#) [CDOUT](#)

Public DigitalOutput class definition.

Definition at line 77 of file DigitalOutputClass.h.

3.4 DigitalOutput class exported methods

Collaboration diagram for DigitalOutput class exported methods:



Functions

- **CDOUP DOUT_NewObject (pDigitalOutputParams_t pDigitalOutputParams)**
Creates an object of the class DigitalOutput.
- **void DOUT_Init (CDOUP this)**
Initializes object variables, port and pin. It must be called only after PWMnCurrFdbk object initialization and Digital-Output object creation.
- **void DOUT_SetOutputState (CDOUP this, DOutputState_t State)**
Accordingly with selected polarity, it sets to active or inactive the digital output.
- **DOutputState_t DOUT_GetOutputState (CDOUP this)**
It returns the state of the digital output.

3.4.1 Function Documentation

3.4.1.1 CDOUP DOUT_NewObject (pDigitalOutputParams_t pDigitalOutputParams)

Creates an object of the class DigitalOutput.

Parameters

<i>pDigitalOutput-Params</i>	pointer to an DigitalOutput parameters structure
------------------------------	--

Return values

<i>CDOUP</i>	new instance of DigitalOutput object
--------------	--------------------------------------

3.4.1.2 void DOUT_Init (CDOUP this)

Initializes object variables, port and pin. It must be called only after PWMnCurrFdbk object initialization and Digital-Output object creation.

Parameters

<i>this</i>	related object of class CDOUP.
-------------	--------------------------------

Return values

<i>none.</i>

3.4.1.3 void DOUT_SetOutputState (CDOUT *this*, DOOutputState_t *State*)

Accordingly with selected polarity, it sets to active or inactive the digital output.

Parameters

<i>this</i>	related object of class CDOUT.
<i>OutputState_t</i>	New requested state

Return values

<i>none</i>

3.4.1.4 DOOutputState_t DOUT_GetOutputState (CDOUT *this*)

It returns the state of the digital output.

Parameters

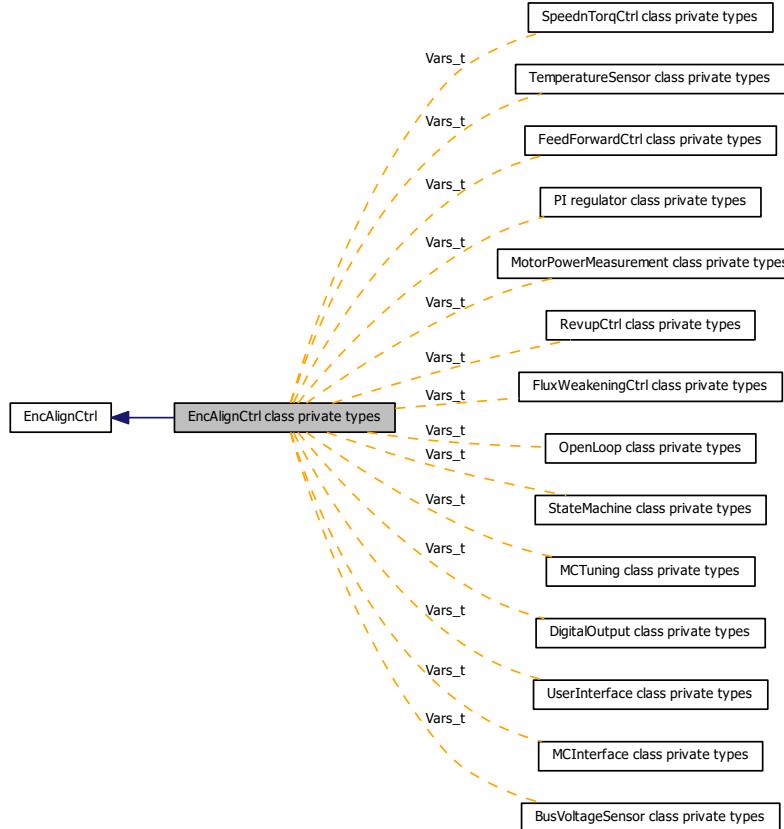
<i>this</i>	object of class DOUT
-------------	----------------------

Return values

<i>OutputState_t</i>	Digital output state (ACTIVE or INACTIVE)
----------------------	---

3.5 EncAlignCtrl class private types

Collaboration diagram for EncAlignCtrl class private types:



Data Structures

- struct `Vars_t`
MCInterface class members definition.
- struct `_CEAC_t`
Private EncAlignCtrl class definition.

TypeDefs

- typedef `EncAlignCtrlParams_t Params_t`
Redefinition of parameter structure.

3.5.1 Typedef Documentation

3.5.1.1 `typedef EncAlignCtrlParams_t Params_t`

Redefinition of parameter structure.

Definition at line 68 of file EncAlignCtrlPrivate.h.

3.6 EncAlignCtrl class exported types

Collaboration diagram for EncAlignCtrl class exported types:



Data Structures

- struct [EncAlignCtrlParams_t](#)
EncAlignCtrl class parameters definition.

Typedefs

- typedef struct CEAC_t * [CEAC](#)
Public EncAlignCtrl class definition.

3.6.1 Typedef Documentation

3.6.1.1 [typedef struct CEAC_t* CEAC](#)

Public EncAlignCtrl class definition.

Definition at line 55 of file EncAlignCtrlClass.h.

3.7 EncAlignCtrl class exported methods

Collaboration diagram for EncAlignCtrl class exported methods:



Functions

- **[CEAC EAC_NewObject \(pEncAlignCtrlParams_t pEncAlignCtrlParams\)](#)**
Creates an object of the class EncAlignCtrl.
- void **[EAC_Init \(CEAC this, CSTC oSTC, CVSS_SPD oVSS, CENC_SPD oENC\)](#)**
Initializes all the object variables, usually it has to be called once right after object creation. It is also used to assign the speed and torque controller, the virtual speed sensor objects and the encoder object to be used by encoder alignment controller.
- void **[EAC_StartAlignment \(CEAC this\)](#)**
It is called to start the encoder alignment procedure. It configures the VSS with the required angle and sets the STC to execute the required torque ramp.
- bool **[EAC_Exec \(CEAC this\)](#)**
It clocks the encoder alignment controller and must be called with a frequency equal to the one settled in the parameters hEACFrequencyHz. Calling this method the EAC is possible to verify if the alignment duration has been finished. At the end of alignment the encoder object is set to the defined electrical angle. Note: STC, VSS, ENC are not clocked by EAC_Exec.
- bool **[EAC_IsAligned \(CEAC this\)](#)**
This function returns TRUE if the encoder has been aligned at least one time, FALSE if hasn't been never aligned.
- void **[EAC_SetRestartState \(CEAC this, bool restart\)](#)**
This function is used to program a restart after an encoder alignment.
- bool **[EAC_GetRestartState \(CEAC this\)](#)**
This function is used to verify if a restart after an encoder alignment has been requested.

3.7.1 Function Documentation

3.7.1.1 CEAC EAC_NewObject (pEncAlignCtrlParams_t pEncAlignCtrlParams)

Creates an object of the class EncAlignCtrl.

Parameters

<i>pEncAlignCtrl- Params</i>	pointer to an EncAlignCtrl parameters structure
----------------------------------	---

Return values

<i>CEAC</i>	new instance of EncAlignCtrl object
-------------	-------------------------------------

Definition at line 51 of file EncAlignCtrlClass.c.

References MAX_EAC_NUM, MC_NULL, and _CEAC_t::pParams_str.

3.7.1.2 void EAC_Init (CEAC *this*, CSTC *oSTC*, CVSS_SPD *oVSS*, CENC_SPD *oENC*)

Initializes all the object variables, usually it has to be called once right after object creation. It is also used to assign the speed and torque controller, the virtual speed sensor objects and the encoder object to be used by encoder alignment controller.

Parameters

<i>this</i>	related object of class CEAC.
<i>oSTC</i>	the speed and torque controller used by the EAC.
<i>oVSS</i>	the virtual speed sensor used by the EAC.
<i>oENC</i>	the encoder object used by the EAC.

Return values

<i>none.</i>

Definition at line 84 of file EncAlignCtrlClass.c.

References Vars_t::EncAligned, Vars_t::EncRestart, Vars_t::oENC, Vars_t::oSTC, and Vars_t::oVSS.

3.7.1.3 void EAC_StartAlignment (CEAC *this*)

It is called to start the encoder alignment procedure. It configures the VSS with the required angle and sets the STC to execute the required torque ramp.

Parameters

<i>this</i>	related object of class CEAC.
-------------	-------------------------------

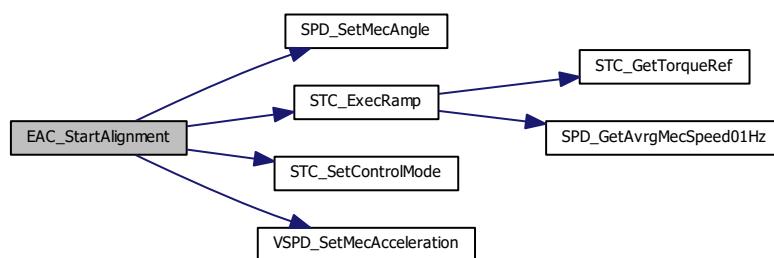
Return values

<i>none.</i>

Definition at line 101 of file EncAlignCtrlClass.c.

References Vars_t::hRemainingTicks, Vars_t::oSTC, Vars_t::oVSS, SPD_SetMecAngle(), STC_ExecRamp(), STC_SetControlMode(), STC_TORQUE_MODE, and VSPD_SetMecAcceleration().

Here is the call graph for this function:



3.7.1.4 bool EAC_Exec (CEAC *this*)

It clocks the encoder alignment controller and must be called with a frequency equal to the one settled in the parameters hEACFrequencyHz. Calling this method the EAC is possible to verify if the alignment duration has been finished. At the end of alignment the encoder object it is set to the defined electrical angle. Note: STC, VSS, ENC are not clocked by EAC_Exec.

Parameters

<i>this</i>	related object of class CEAC.
-------------	-------------------------------

Return values

<i>bool</i>	It returns TRUE when the programmed alignment has been completed.
-------------	---

It clocks the encoder alignment controller and must be called with a frequency equal to the one settled in the parameters hEACFrequencyHz. Calling this method the EAC is possible to verify if the alignment duration has been finished. At the end of alignment the encoder object it is set to the defined electrical angle. Note: STC, VSS, ENC are not clocked by EAC_Exec.

Parameters

<i>this</i>	related object of class CEAC.
-------------	-------------------------------

Return values

<i>bool</i>	It returns true when the programmed alignment has been completed.
-------------	---

Definition at line 141 of file EncAlignCtrlClass.c.

References Vars_t::EncAligned, Vars_t::hRemainingTicks, Vars_t::oENC, and SPD_SetMecAngle().

Here is the call graph for this function:



3.7.1.5 bool EAC_IsAligned (CEAC *this*)

This function returns TRUE if the encoder has been aligned at least one time, FALSE if hasn't been never aligned.

Parameters

<i>this</i>	related object of class CEAC.
-------------	-------------------------------

Return values

<i>bool</i>	It returns TRUE if the encoder has been aligned at least one time, FALSE if hasn't been never aligned.
-------------	--

This function returns TRUE if the encoder has been aligned at least one time, FALSE if hasn't been never aligned.

Parameters

<i>this</i>	related object of class CEAC.
-------------	-------------------------------

Return values

<i>bool</i>	It returns true if the encoder has been aligned at least one time, false if hasn't been never aligned.
-------------	--

Definition at line 175 of file EncAlignCtrlClass.c.

References Vars_t::EncAligned.

3.7.1.6 void EAC_SetRestartState (CEAC *this*, bool *restart*)

This function is used to program a restart after an encoder alignment.

Parameters

<i>this</i>	related object of class CEAC.
<i>restart</i>	Set to TRUE if a restart is programmed else FALSE

Return values

<i>none.</i>

Parameters

<i>this</i>	related object of class CEAC.
<i>restart</i>	Set to true if a restart is programmed else false

Return values

<i>none.</i>

Definition at line 188 of file EncAlignCtrlClass.c.

References Vars_t::EncRestart.

3.7.1.7 bool EAC_GetRestartState (CEAC *this*)

This function is used to verify if a restart after an encoder alignment has been requested.

Parameters

<i>this</i>	related object of class CEAC.
-------------	-------------------------------

Return values

<i>bool</i>	It is TRUE if a restart is programmed else FALSE.
-------------	---

Parameters

<i>this</i>	related object of class CEAC.
-------------	-------------------------------

Return values

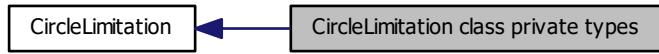
<i>bool</i>	It is true if a restart is programmed else false.
-------------	---

Definition at line 200 of file EncAlignCtrlClass.c.

References Vars_t::EncRestart.

3.8 CircleLimitation class private types

Collaboration diagram for CircleLimitation class private types:



Data Structures

- struct [_CCLM_t](#)
Private CircleLimitation class definition.

Typedefs

- [typedef CircleLimitationParams_t Params_t](#)
Redefinition of parameter structure.

3.8.1 Typedef Documentation

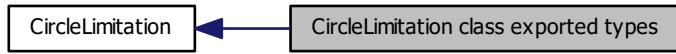
3.8.1.1 [typedef CircleLimitationParams_t Params_t](#)

Redefinition of parameter structure.

Definition at line 49 of file CircleLimitationPrivate.h.

3.9 CircleLimitation class exported types

Collaboration diagram for CircleLimitation class exported types:



Data Structures

- struct [CircleLimitationParams_t](#)
CircleLimitation class parameters definition.

Typedefs

- typedef struct CCLM_t * [CCLM](#)
Public CircleLimitation class definition.

3.9.1 Typedef Documentation

3.9.1.1 [typedef struct CCLM_t*](#) [CCLM](#)

Public CircleLimitation class definition.

Definition at line 52 of file CircleLimitationClass.h.

3.10 CircleLimitation class exported methods

Collaboration diagram for CircleLimitation class exported methods:



Functions

- [CCLM CLM_NewObject \(pCircleLimitationParams_t pCircleLimitationParams \)](#)
Creates an object of the class CircleLimitation.
- [Volt_Components Circle_Limitation \(CCLM this, Volt_Components Vqd \)](#)
Check whether $Vqd.qV_Component1^2 + Vqd.qV_Component2^2 \leq 32767^2$ and if not it applies a limitation keeping constant ratio $Vqd.qV_Component1 / Vqd.qV_Component2$.

3.10.1 Function Documentation

3.10.1.1 CCLM CLM_NewObject (pCircleLimitationParams_t pCircleLimitationParams)

Creates an object of the class CircleLimitation.

Parameters

<i>pCircle-Limitation-Params</i>	pointer to an CircleLimitation parameters structure
----------------------------------	---

Return values

<i>CCLM</i>	new instance of CircleLimitation object
-------------	---

Definition at line 51 of file CircleLimitationClass.c.

References MAX_CLM_NUM, MC_NULL, and *_CCLM_t::pParams_str*.

3.10.1.2 Volt_Components Circle_Limitation (CCLM this, Volt_Components Vqd)

Check whether $Vqd.qV_Component1^2 + Vqd.qV_Component2^2 \leq 32767^2$ and if not it applies a limitation keeping constant ratio $Vqd.qV_Component1 / Vqd.qV_Component2$.

Parameters

<i>this</i>	related object of class CCLM
<i>Vqd</i>	Voltage in qd reference frame

Return values

Volt_Components	Limited Vqd vector
---------------------------------	--------------------

Definition at line 81 of file CircleLimitationClass.c.

3.11 MCIRQ_Handler module exported definitions

Collaboration diagram for MCIRQ_Handler module exported definitions:



Defines

- `#define MC_IRQ_SPEEDNPOSFDBK_1 0u`
MC IRQ interrupts addresses inside MC vector table.
- `#define MC_IRQ_SPEEDNPOSFDBK_HS 1u`
- `#define MC_IRQ_ICS 2u`
- `#define MC_IRQ_R1 3u`

3.11.1 Define Documentation

3.11.1.1 `#define MC_IRQ_SPEEDNPOSFDBK_1 0u`

MC IRQ interrupts addresses inside MC vector table.

Reserved for SpeednPosFdbk first instance.

Definition at line 58 of file MCIRQHandlerClass.h.

3.11.1.2 `#define MC_IRQ_SPEEDNPOSFDBK_HS 1u`

Reserved for SpeednPosFdbk_HS first instance.

Definition at line 59 of file MCIRQHandlerClass.h.

3.11.1.3 `#define MC_IRQ_ICS 2u`

Reserved for ICS

Definition at line 60 of file MCIRQHandlerClass.h.

3.11.1.4 `#define MC_IRQ_R1 3u`

Reserved for Single Shunt

Definition at line 61 of file MCIRQHandlerClass.h.

3.12 MCIRQ Handler module exported functions

Collaboration diagram for MCIRQ Handler module exported functions:



Functions

- void * [Exec_IRQ_Handler](#) (uint8_t bIRQAddr, uint8_t flag)
Start the execution of the MC_IRQ at bIRQAddr position inside MC vector table.

3.12.1 Function Documentation

3.12.1.1 void* Exec_IRQ_Handler (uint8_t bIRQAddr, uint8_t flag)

Start the execution of the MC_IRQ at bIRQAddr position inside MC vector table.

Parameters

in	<i>bIRQAddr</i>	MC IRQ position inside MC vector table
in	<i>flag</i>	parameter passed to the MC interrupt, for instance to identify the event request the MC ISR execution

Returns

pvoid pointer to Drive object related to the interrupt

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Definition at line 95 of file MCIRQHandlerClass.c.

3.13 MotorPowerMeasurement class private defines

Collaboration diagram for MotorPowerMeasurement class private defines:



Defines

- #define MPM_BUFFER_LENGTH 128u

3.13.1 Define Documentation

3.13.1.1 #define MPM_BUFFER_LENGTH 128u

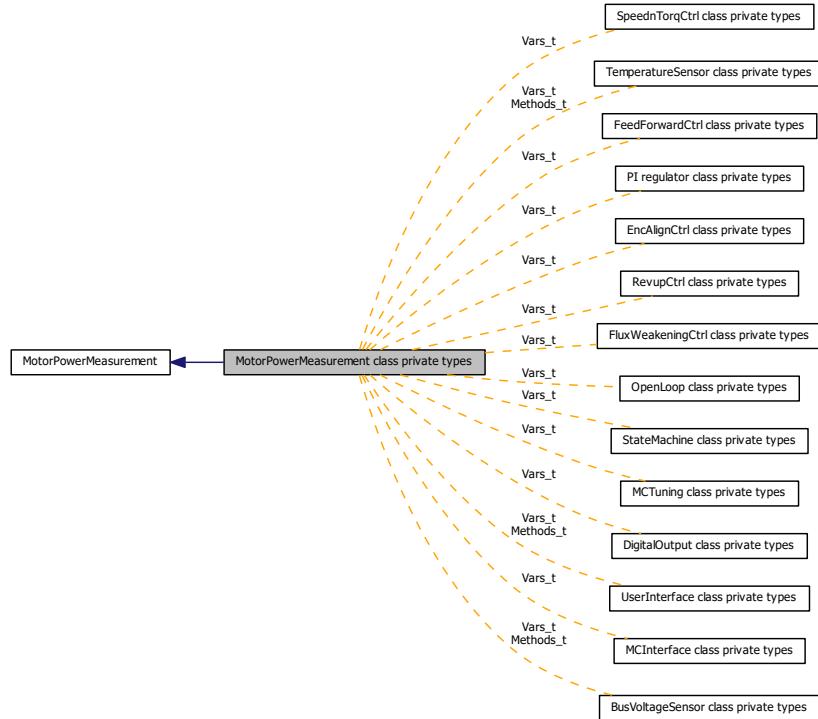
Length of buffer used to store the instantaneous measurements of motor power.

Definition at line 46 of file MotorPowerMeasurementPrivate.h.

Referenced by MPM_CalcEIMotorPower(), and MPM_Clear().

3.14 MotorPowerMeasurement class private types

Collaboration diagram for MotorPowerMeasurement class private types:

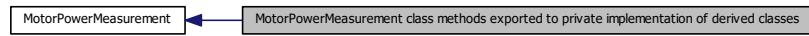


Data Structures

- struct [Vars_t](#)
MCInterface class members definition.
 - struct [Methods_t](#)
Virtual methods container.
 - struct [_CMPM_t](#)
Private MotorPowerMeasurement class definition.

3.15 MotorPowerMeasurement class methods exported to private implementation of derived classes

Collaboration diagram for MotorPowerMeasurement class methods exported to private implementation of derived classes:



Functions

- **CMPM MPM_NewObject (void)**
Creates an object of the class MotorPowerMeasurement.

3.15.1 Function Documentation

3.15.1.1 CMPM MPM_NewObject (void)

Creates an object of the class MotorPowerMeasurement.

Parameters

<i>pMotorPower- Measurement- Params</i>	pointer to an MotorPowerMeasurement parameters structure
---	--

Return values

<i>CMPM</i>	new instance of MotorPowerMeasurement object
-------------	--

Definition at line 52 of file MotorPowerMeasurementClass.c.

References MAX_MPM_NUM, and MC_NULL.

Referenced by PQD_NewObject().

3.16 MotorPowerMeasurement class exported types

Collaboration diagram for MotorPowerMeasurement class exported types:



TypeDefs

- **typedef struct CMPM_t * CMPM**
Public MotorPowerMeasurement class definition.
- **typedef void * pMPMInitStruct_t**
MotorPowerMeasurement class init struct definition.

3.16.1 TypeDef Documentation

3.16.1.1 **typedef struct CMPM_t* CMPM**

Public MotorPowerMeasurement class definition.

Definition at line 52 of file MotorPowerMeasurementClass.h.

3.16.1.2 **typedef void* pMPMInitStruct_t**

MotorPowerMeasurement class init struct definition.

Definition at line 57 of file MotorPowerMeasurementClass.h.

3.17 MotorPowerMeasurement class exported methods

Collaboration diagram for MotorPowerMeasurement class exported methods:



Functions

- void **MPM_Init** (**CMPM** this, **pMPMInitStruct_t** **pMPMInitStruct**)
Initializes all the object variables, usually it has to be called once right after object creation.
- void **MPM_Clear** (**CMPM** this)
It should be called before each motor restart. It clears the measurement buffer and initialize the index.
- int16_t **MPM_CalcEIMotorPower** (**CMPM** this)
*This method should be called with periodicity. It computes and returns the measured motor power expressed in watt.
It is also used to fill, with that measure, the buffer used to compute the average motor power.*
- int16_t **MPM_GetEIMotorPowerW** (**CMPM** this)
This method is used to get the last measured motor power (instantaneous value) expressed in watt.
- int16_t **MPM_GetAvgEIMotorPowerW** (**CMPM** this)
This method is used to get the average measured motor power expressed in watt.

3.17.1 Function Documentation

3.17.1.1 void **MPM_Init** (**CMPM** this, **pMPMInitStruct_t** **pMPMInitStruct**)

Initializes all the object variables, usually it has to be called once right after object creation.

Parameters

this	related object of class CMPM .
pMPMInitStruct	the pointer of the init structure, required by derived class, up-casted to pMPMInitStruct_t .

Return values

<i>none.</i>

Definition at line 80 of file **MotorPowerMeasurementClass.c**.

References **MPM_Clear()**.

Here is the call graph for this function:



3.17.1.2 void MPM_Clear (CMPM *this*)

It should be called before each motor restart. It clears the measurement buffer and initialize the index.

Parameters

<i>this</i>	related object of class CMPM.
-------------	-------------------------------

Return values

<i>none.</i>

Definition at line 92 of file MotorPowerMeasurementClass.c.

References Vars_t::hLastMeasBufferIndex, Vars_t::hMeasBuffer, Vars_t::hNextMeasBufferIndex, and MPM_BUFFER_LENGTH.

Referenced by MPM_Init().

3.17.1.3 int16_t MPM_CalcEIMotorPower (CMPM *this*)

This method should be called with periodicity. It computes and returns the measured motor power expressed in watt. It is also used to fill, with that measure, the buffer used to compute the average motor power.

Parameters

<i>this</i>	related object of class CMPM.
-------------	-------------------------------

Return values

<i>int16_t</i>	The measured motor power expressed in watt.
----------------	---

Definition at line 113 of file MotorPowerMeasurementClass.c.

References Vars_t::hAvrgEIMotorPowerW, Vars_t::hLastMeasBufferIndex, Vars_t::hMeasBuffer, Vars_t::hNextMeasBufferIndex, and MPM_BUFFER_LENGTH.

3.17.1.4 int16_t MPM_GetEIMotorPowerW (CMPM *this*)

This method is used to get the last measured motor power (instantaneous value) expressed in watt.

Parameters

<i>this</i>	related object of class CMPM.
-------------	-------------------------------

Return values

<i>int16_t</i>	The last measured motor power (instantaneous value) expressed in watt.
----------------	--

Definition at line 146 of file MotorPowerMeasurementClass.c.

References Vars_t::hLastMeasBufferIndex, and Vars_t::hMeasBuffer.

3.17.1.5 int16_t MPM_GetAvrgElMotorPowerW (CMPM *this*)

This method is used to get the average measured motor power expressed in watt.

Parameters

<i>this</i>	related object of class CMPM.
-------------	-------------------------------

Return values

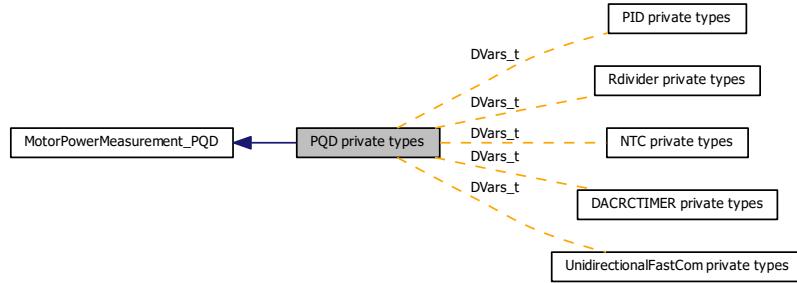
<i>int16_t</i>	The average measured motor power expressed in watt.
----------------	---

Definition at line 158 of file MotorPowerMeasurementClass.c.

References Vars_t::hAvrgElMotorPowerW.

3.18 PQD private types

Collaboration diagram for PQD private types:



Data Structures

- struct `DVars_t`
PQD class members definition.
- struct `_DCPQD_MPM_t`
Private PQD class definition.

Typedefs

- typedef `PQDParams_t DParams_t`
Redefinition of parameter structure.

3.18.1 Typedef Documentation

3.18.1.1 `typedef PQDParams_t DParams_t`

Redefinition of parameter structure.

Definition at line 58 of file `PQD_MotorPowerMeasurementPrivate.h`.

3.19 PQD class exported types

Collaboration diagram for PQD class exported types:



Data Structures

- struct [PQDParams_t](#)
PQD class parameters definition.
- struct [PQD_MPMinitStruct_t](#)
PQD class init struct definition.

Typedefs

- typedef struct CPQD_MPM_t * [CPQD_MPM](#)
Public PQD class definition.

3.19.1 Typedef Documentation

3.19.1.1 [typedef struct CPQD_MPM_t* CPQD_MPM](#)

Public PQD class definition.

Definition at line 51 of file [PQD_MotorPowerMeasurementClass.h](#).

3.20 PQD class exported methods

Collaboration diagram for PQD class exported methods:



Functions

- **CPQD_MPM PQD_NewObject (pPQDParams_t pPQDParams)**
Creates an object of the class PQD.

3.20.1 Function Documentation

3.20.1.1 CPQD_MPM PQD_NewObject (pPQDParams_t pPQDParams)

Creates an object of the class PQD.

Parameters

<i>pMotorPower-Measurement-Params</i>	pointer to an MotorPowerMeasurement parameters structure
<i>pPQDParams</i>	pointer to an PQD parameters structure

Return values

<i>CPQD_MPM</i>	new instance of PQD object
-----------------	----------------------------

Definition at line 78 of file PQD_MotorPowerMeasurementClass.c.

References *_CMPM_t::DerivedClass*, *MAX_PQD_MPM_NUM*, *MC_NULL*, *_CMPM_t::Methods_str*, *MPM_NewObject()*, and *_DCPQD_MPM_t::pDParams_str*.

Here is the call graph for this function:



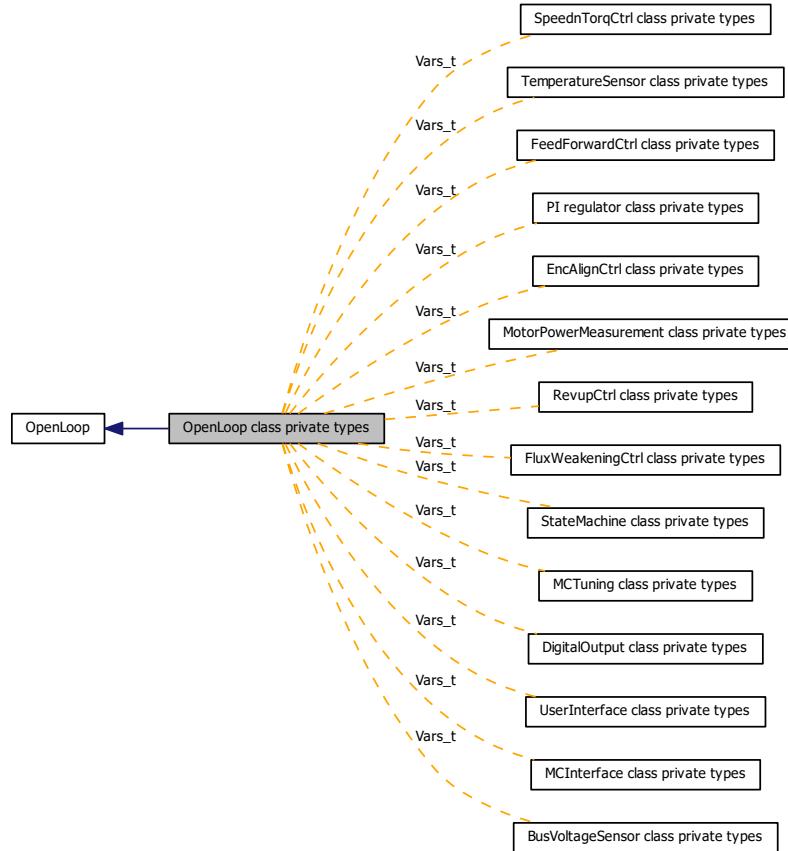
3.21 PQD class private methods

Collaboration diagram for PQD class private methods:



3.22 OpenLoop class private types

Collaboration diagram for OpenLoop class private types:



Data Structures

- struct [Vars_t](#)
MCInterface class members definition.
- struct [_COL_t](#)
Private OpenLoop class definition.

TypeDefs

- [typedef OpenLoopParams_t Params_t](#)
Redefinition of parameter structure.

3.22.1 Typedef Documentation

3.22.1.1 [typedef OpenLoopParams_t Params_t](#)

Redefinition of parameter structure.

Definition at line 59 of file OpenLoopPrivate.h.

3.23 OpenLoop class exported types

Collaboration diagram for OpenLoop class exported types:



Data Structures

- struct [OpenLoopParams_t](#)
OpenLoop class parameters definition.

Typedefs

- typedef struct COL_t * [COL](#)
Public OpenLoop class definition.

3.23.1 Typedef Documentation

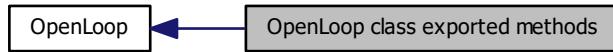
3.23.1.1 [typedef struct COL_t* COL](#)

Public OpenLoop class definition.

Definition at line 53 of file OpenLoopClass.h.

3.24 OpenLoop class exported methods

Collaboration diagram for OpenLoop class exported methods:



Functions

- **`COL OL_NewObject (pOpenLoopParams_t pOpenLoopParams)`**
Creates an object of the class OpenLoop.
- **`void OL_Init (COL this, CSPD oVSS)`**
Initializes all the object variables, usually it has to be called once right after object creation.
- **`Volt_Components OL_VqdConditioning (COL this)`**
It return the open loop Vqd.
- **`void OL_UpdateVoltage (COL this, int16_t hNewVoltage)`**
It allows changing applied open loop phase voltage.
- **`void OL_Calc (COL this)`**
It have to be called regularly to update the applied voltage in case V/F is enabled. If V/F is disabled nothing is done.
- **`void OL_VF (COL this, bool VFEnabling)`**
It is used to enable or disable the V/F mode.

3.24.1 Function Documentation

3.24.1.1 `COL OL_NewObject (pOpenLoopParams_t pOpenLoopParams)`

Creates an object of the class OpenLoop.

Parameters

<code>pOpenLoop- Params</code>	pointer to an OpenLoop parameters structure.
------------------------------------	--

Return values

<code>COL</code>	new instance of OpenLoop object.
------------------	----------------------------------

Definition at line 51 of file OpenLoopClass.c.

References MAX_OL_NUM, MC_NULL, and _COL_t::pParams_str.

3.24.1.2 `void OL_Init (COL this, CSPD oVSS)`

Initializes all the object variables, usually it has to be called once right after object creation.

Parameters

<i>this</i>	related object of class COL.
<i>oVSS</i>	Related VSS object used.

Return values

<i>none</i>

Definition at line 80 of file OpenLoopClass.c.

References Vars_t::hVoltage, Vars_t::oVSS, and Vars_t::VFMode.

3.24.1.3 Volt_Components OL_VqdConditioning (COL *this*)

It return the open loop Vqd.

Parameters

<i>this</i>	related object of class COL.
-------------	------------------------------

Return values

<i>Volt_Components</i>	Vqd conditioned values.
------------------------	-------------------------

Definition at line 94 of file OpenLoopClass.c.

References Vars_t::hVoltage.

3.24.1.4 void OL_UpdateVoltage (COL *this*, int16_t *hNewVoltage*)

It allows changing applied open loop phase voltage.

Parameters

<i>this</i>	related object of class COL
<i>hNewVoltage</i>	New voltage value to be applied by the open loop.

Return values

<i>None</i>

Definition at line 111 of file OpenLoopClass.c.

References Vars_t::hVoltage.

3.24.1.5 void OL_Calc (COL *this*)

It have to be called regularly to update the applied voltage in case V/F is enabled. If V/F is disabled nothing is done.

Parameters

<i>this</i>	related object of class COL
-------------	-----------------------------

Return values

<i>None</i>

Definition at line 123 of file OpenLoopClass.c.

References Vars_t::hVoltage, Vars_t::oVSS, SPD_GetAvrgMecSpeed01Hz(), and Vars_t::VFMode.

Here is the call graph for this function:



3.24.1.6 void OL_VF (COL *this*, bool *VFEnabling*)

It is used to enable or disable the V/F mode.

Parameters

<i>this</i>	related object of class COL
<i>VFEnabling</i>	TRUE to enable V/F mode, FALSE to disable.

Return values

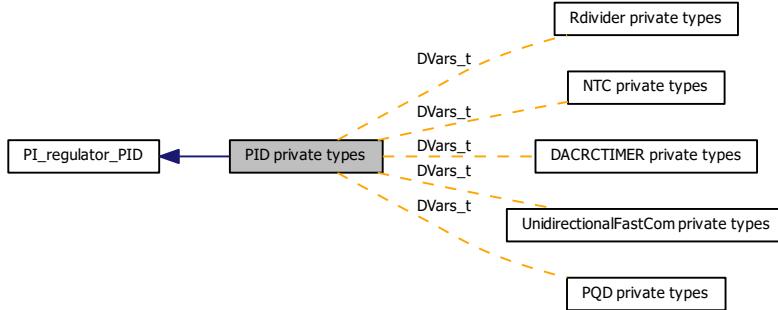
<i>None</i>

Definition at line 142 of file OpenLoopClass.c.

References Vars_t::VFMode.

3.25 PID private types

Collaboration diagram for PID private types:



Data Structures

- struct [DVars_t](#)
PQD class members definition.
- struct [_DCPID_t](#)
Private PID class definition.

Typedefs

- [typedef PIDParams_t DParams_t](#)
Redefinition of parameter structure.

3.25.1 Typedef Documentation

3.25.1.1 [typedef PIDParams_t DParams_t](#)

Redefinition of parameter structure.

Definition at line 60 of file PID_PIRegulatorPrivate.h.

3.26 PID regulator class exported types

Collaboration diagram for PID regulator class exported types:



Data Structures

- struct [PIDParams_t](#)
PID class parameters definition.

Typedefs

- typedef struct CPID_PI_t * [CPID_PI](#)
Public PID class definition.

3.26.1 Typedef Documentation

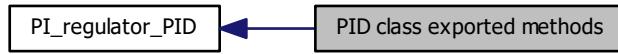
3.26.1.1 [typedef struct CPID_PI_t*](#) [CPID_PI](#)

Public PID class definition.

Definition at line 49 of file PID_PIRegulatorClass.h.

3.27 PID class exported methods

Collaboration diagram for PID class exported methods:



Functions

- [CPID_PI PID_NewObject \(pPIParams_t pPIParam, pPIDParams_t pPIDParam \)](#)
Creates an object of the derived class PID.
- [void PID_ObjectInit \(CPID_PI this\)](#)
It initializes all the object variables, usually it has to be called once right after object creation.
- [void PID_SetPrevError \(CPID_PI this, int32_t wPrevProcessVarError\)](#)
It set a new value into the PID Previous error variable required to compute derivative term.
- [void PID_SetKD \(CPID_PI this, int16_t hKdGain\)](#)
It updates the Kd gain.
- [int16_t PID_GetKD \(CPID_PI this\)](#)
It returns the Kd gain of the PID object passed.
- [uint16_t PID_GetKDDivisor \(CPID_PI this\)](#)
It returns the Kd gain divisor of the PID object passed.
- [int16_t PID_Controller \(CPID_PI this, int32_t wProcessVarError\)](#)
This function compute the output of a PID regulator sum of its proportional, integral and derivative terms.

3.27.1 Function Documentation

3.27.1.1 CPID_PI PID_NewObject (pPIParams_t pPIParam, pPIDParams_t pPIDParam)

Creates an object of the derived class PID.

Parameters

<code>pPIParams_t</code>	pointer to PI regulator class parameters structure
<code>pPIDParams_t</code>	pointer to a PID Derived class parameters structure

Return values

<code>CPID_PI</code>	new instance of derived class object
----------------------	--------------------------------------

Definition at line 62 of file PID_PIRegulatorClass.c.

References `_CPI_t::DerivedClass`, `MAX_PID_PI_NUM`, `MC_NULL`, and `PI_NewObject()`.

Here is the call graph for this function:



3.27.1.2 void PID_ObjectInit (*CPID_PI this*)

It initializes all the object variables, usually it has to be called once right after object creation.

Parameters

<i>CPID_PI</i>	PID regulator object
----------------	----------------------

Return values

<i>None</i>

Definition at line 94 of file PID_PIRegulatorClass.c.

3.27.1.3 void PID_SetPrevError (*CPID_PI this, int32_t wPrevProcessVarError*)

It set a new value into the PID Previous error variable required to compute derivative term.

Parameters

<i>CPID_PI</i>	PID regulator object
<i>int32_t</i>	New integral term value

Return values

<i>None</i>

Definition at line 127 of file PID_PIRegulatorClass.c.

3.27.1.4 void PID_SetKD (*CPID_PI this, int16_t hKdGain*)

It updates the Kd gain.

Parameters

<i>CPID_PI</i>	PID regulator object
<i>int16_t</i>	New Kd gain

Return values

<i>None</i>

Definition at line 141 of file PID_PIRegulatorClass.c.

References MC_NULL.

3.27.1.5 int16_t PID_GetKD (CPID_PI *this*)

It returns the Kd gain of the PID object passed.

Parameters

<i>CPID_PI</i>	PID regulator object
----------------	----------------------

Return values

<i>int16_t</i>	Kd gain
----------------	---------

It returns the Kd gain of the PID object passed.

Parameters

<i>CPID_PI</i>	PI regulator object
----------------	---------------------

Return values

<i>int16_t</i>	Kd gain
----------------	---------

Definition at line 154 of file PID_PIRegulatorClass.c.

References MC_NULL.

3.27.1.6 uint16_t PID_GetKDDivisor (CPID_PI *this*)

It returns the Kd gain divisor of the PID object passed.

Parameters

<i>CPID_PI</i>	PID regulator object
----------------	----------------------

Return values

<i>int16_t</i>	Kd gain
----------------	---------

Definition at line 174 of file PID_PIRegulatorClass.c.

References MC_NULL.

3.27.1.7 int16_t PID_Controller (CPID_PI *this*, int32_t *wProcessVarError*)

This function compute the output of a PID regulator sum of its proportional, integral and derivative terms.

Parameters

<i>CPID_PI</i>	PID regulator object
<i>Present</i>	process variable error, intended as the reference value minus the present process variable value

Return values

<i>int16_t</i>	PID output
----------------	------------

Parameters

<i>CPID_PI</i>	PID regulator object
<i>int32_t</i>	Present process variable error, intended as the reference value minus the present process variable value

Return values

<i>int16_t</i>	PI output
----------------	-----------

Definition at line 200 of file PID_PIRegulatorClass.c.

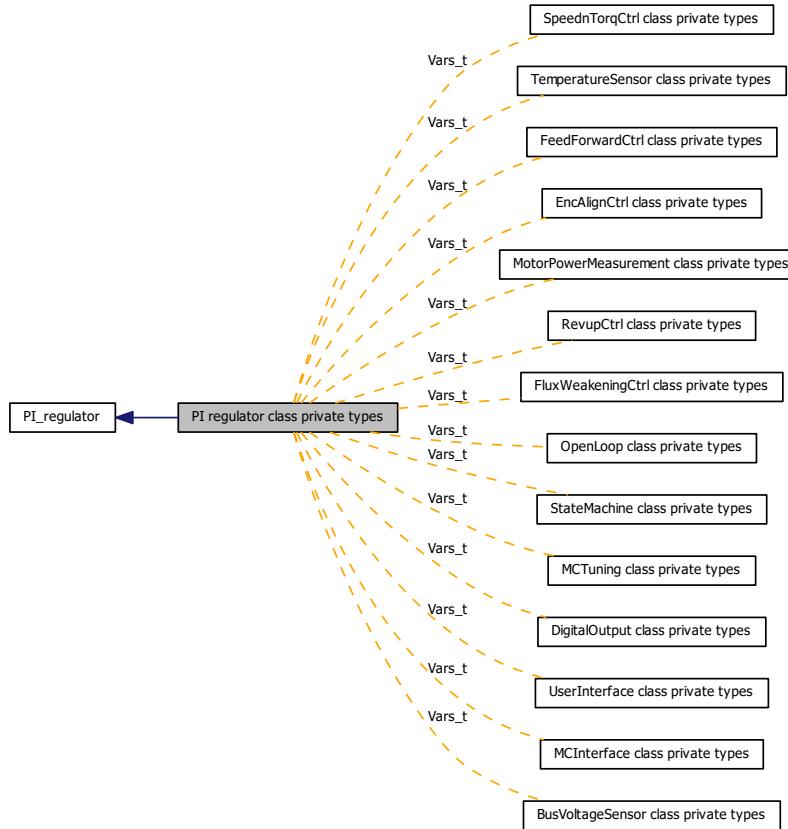
References PI_Controller().

Here is the call graph for this function:



3.28 PI regulator class private types

Collaboration diagram for PI regulator class private types:



Data Structures

- struct [Vars_t](#)
MCInterface class members definition.
- struct [_CPI_t](#)
Private PI regulator class definition.

TypeDefs

- typedef [PIParams_t Params_t](#)
Redefinition of parameter structure.

3.28.1 Typedef Documentation

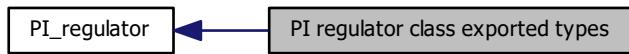
3.28.1.1 typedef PIParams_t Params_t

Redefinition of parameter structure.

Definition at line 63 of file PIRegulatorPrivate.h.

3.29 PI regulator class exported types

Collaboration diagram for PI regulator class exported types:



Data Structures

- struct [PIParams_t](#)
PI regulator class parameters structure definition.

Typedefs

- typedef struct CPI_t * [CPI](#)
Public PI regulator class definition.

3.29.1 Typedef Documentation

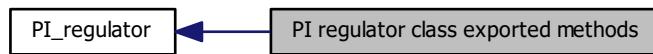
3.29.1.1 [typedef struct CPI_t* CPI](#)

Public PI regulator class definition.

Definition at line 52 of file PIRegulatorClass.h.

3.30 PI regulator class exported methods

Collaboration diagram for PI regulator class exported methods:



Functions

- **CPI PI_NewObject (pPIParams_t pPIParams)**
Creates an object of the class PI regulator.
- void **PI_ObjectInit (CPI this)**
Initialize all the object variables, usually it has to be called once right after object creation.
- int16_t **PI_Controller (CPI this, int32_t wProcessVarError)**
This function compute the output of a PI regulator sum of its proportional and integral terms.
- void **PI_SetKP (CPI this, int16_t hKpGain)**
It updates the Kp gain.
- void **PI_SetKI (CPI this, int16_t hKiGain)**
It updates the Ki gain.
- int16_t **PI_GetKP (CPI this)**
It returns the Kp gain of the passed PI object.
- int16_t **PI_GetKI (CPI this)**
It returns the Ki gain of the passed PI object.
- uint16_t **PI_GetKPDvisor (CPI this)**
It returns the Kp gain divisor of the passed PI object.
- uint16_t **PI_GetKIDvisor (CPI this)**
It returns the Ki gain divisor of the passed PI object.
- void **PI_SetIntegralTerm (CPI this, int32_t wIntegralTermValue)**
It set a new value into the PI integral term.
- void **PI_SetLowerIntegralTermLimit (CPI this, int32_t wLowerLimit)**
It set a new value for lower integral term limit.
- void **PI_SetUpperIntegralTermLimit (CPI this, int32_t wUpperLimit)**
It set a new value for upper integral term limit.
- void **PI_SetLowerOutputLimit (CPI this, int16_t hLowerLimit)**
It set a new value for lower output limit.
- void **PI_SetUpperOutputLimit (CPI this, int16_t hUpperLimit)**
It set a new value for upper output limit.
- int16_t **PI_GetDefaultKP (CPI this)**
It returns the Default Kp gain of the passed PI object.
- int16_t **PI_GetDefaultKI (CPI this)**
It returns the Default Ki gain of the passed PI object.

3.30.1 Function Documentation

3.30.1.1 CPI PI_NewObject (*pPIParams_t pPIParams*)

Creates an object of the class PI regulator.

Parameters

<i>pPIParams_t</i>	pointer to a PI parameters structure
--------------------	--------------------------------------

Return values

<i>CPI</i>	new instance of PI object
------------	---------------------------

Definition at line 49 of file PIRegulatorClass.c.

References *_CPI_t::DerivedClass*, *MAX_PI_NUM*, *MC_NULL*, and *_CPI_t::pParams_str*.

Referenced by *PID_NewObject()*, and *STO_NewObject()*.

3.30.1.2 void PI_ObjectInit (*CPI this*)

Initialize all the object variables, usually it has to be called once right after object creation.

Parameters

<i>CPI</i>	PI regulator object
------------	---------------------

Return values

<i>None</i>	
-------------	--

Definition at line 81 of file PIRegulatorClass.c.

3.30.1.3 int16_t PI_Controller (*CPI this, int32_t wProcessVarError*)

This function compute the output of a PI regulator sum of its proportional and integral terms.

Parameters

<i>CPI</i>	PI regulator object
<i>int32_t</i>	Present process variable error, intended as the reference value minus the present process variable value

Return values

<i>int16_t</i>	PI output
----------------	-----------

Definition at line 184 of file PIRegulatorClass.c.

Referenced by *FW_CalcCurrRef()*, *PID_Controller()*, and *STC_CalcTorqueReference()*.

3.30.1.4 void PI_SetKP (*CPI this, int16_t hKpGain*)

It updates the Kp gain.

Parameters

<i>CPI</i>	PI object
<i>int16_t</i>	New Kp gain

Return values

<i>None</i>

Definition at line 100 of file PIRegulatorClass.c.

3.30.1.5 void PI_SetKI (CPI *this*, int16_t *hKiGain*)

It updates the Ki gain.

Parameters

<i>CPI</i>	PI object
<i>int16_t</i>	New Ki gain

Return values

<i>None</i>

Definition at line 111 of file PIRegulatorClass.c.

3.30.1.6 int16.t PI_GetKP (CPI *this*)

It returns the Kp gain of the passed PI object.

Parameters

<i>CPI</i>	PI regulator object
------------	---------------------

Return values

<i>int16_t</i>	Kp gain
----------------	---------

Definition at line 121 of file PIRegulatorClass.c.

3.30.1.7 int16.t PI_GetKI (CPI *this*)

It returns the Ki gain of the passed PI object.

Parameters

<i>CPI</i>	PI regulator object
------------	---------------------

Return values

<i>int16_t</i>	Ki gain
----------------	---------

Definition at line 131 of file PIRegulatorClass.c.

3.30.1.8 uint16_t PI_GetKPDIVisor (CPI *this*)

It returns the Kp gain divisor of the passed PI object.

Parameters

<i>CPI</i>	PI regulator object
------------	---------------------

Return values

<i>int16_t</i>	Kp gain
----------------	---------

Definition at line 278 of file PIRegulatorClass.c.

3.30.1.9 uint16_t PI_GetKIDIVisor (CPI *this*)

It returns the Ki gain divisor of the passed PI object.

Parameters

<i>CPI</i>	PI regulator object
------------	---------------------

Return values

<i>int16_t</i>	Ki gain
----------------	---------

Definition at line 288 of file PIRegulatorClass.c.

Referenced by FW_CalcCurrRef(), and STO_SetPLL().

3.30.1.10 void PI_SetIntegralTerm (CPI *this*, int32_t *wIntegralTermValue*)

It set a new value into the PI integral term.

Parameters

<i>CPI</i>	PI regulator object
<i>int32_t</i>	New integral term value

Return values

<i>None</i>

Definition at line 162 of file PIRegulatorClass.c.

3.30.1.11 void PI_SetLowerIntegralTermLimit (CPI *this*, int32_t *wLowerLimit*)

It set a new value for lower integral term limit.

Parameters

<i>CPI</i>	PI regulator object
<i>int32_t</i>	New lower integral term limit value

Return values

<i>None</i>

Definition at line 299 of file PIRegulatorClass.c.

Referenced by FW_CalcCurrRef().

3.30.1.12 void PI_SetUpperIntegralTermLimit (CPI *this*, int32_t *wUpperLimit*)

It set a new value for upper integral term limit.

Parameters

<i>CPI</i>	PI regulator object
<i>int32_t</i>	New upper integral term limit value

Return values

<i>None</i>

Definition at line 310 of file PIRegulatorClass.c.

Referenced by FW_CalcCurrRef().

3.30.1.13 void PI_SetLowerOutputLimit (CPI *this*, int16_t *hLowerLimit*)

It set a new value for lower output limit.

Parameters

<i>CPI</i>	PI regulator object
<i>int32_t</i>	New lower output limit value

Return values

<i>None</i>

Definition at line 321 of file PIRegulatorClass.c.

3.30.1.14 void PI_SetUpperOutputLimit (CPI *this*, int16_t *hUpperLimit*)

It set a new value for upper output limit.

Parameters

<i>CPI</i>	PI regulator object
<i>int32_t</i>	New upper output limit value

Return values

<i>None</i>

Definition at line 332 of file PIRegulatorClass.c.

3.30.1.15 int16_t PI_GetDefaultKP (CPI *this*)

It returns the Default Kp gain of the passed PI object.

Parameters

CPI	PI regulator object
-----	---------------------

Return values

int16_t	Kp gain
---------	---------

Definition at line 141 of file PIRegulatorClass.c.

3.30.1.16 int16_t PI_GetDefaultKI (CPI *this*)

It returns the Default Ki gain of the passed PI object.

Parameters

CPI	PI regulator object
-----	---------------------

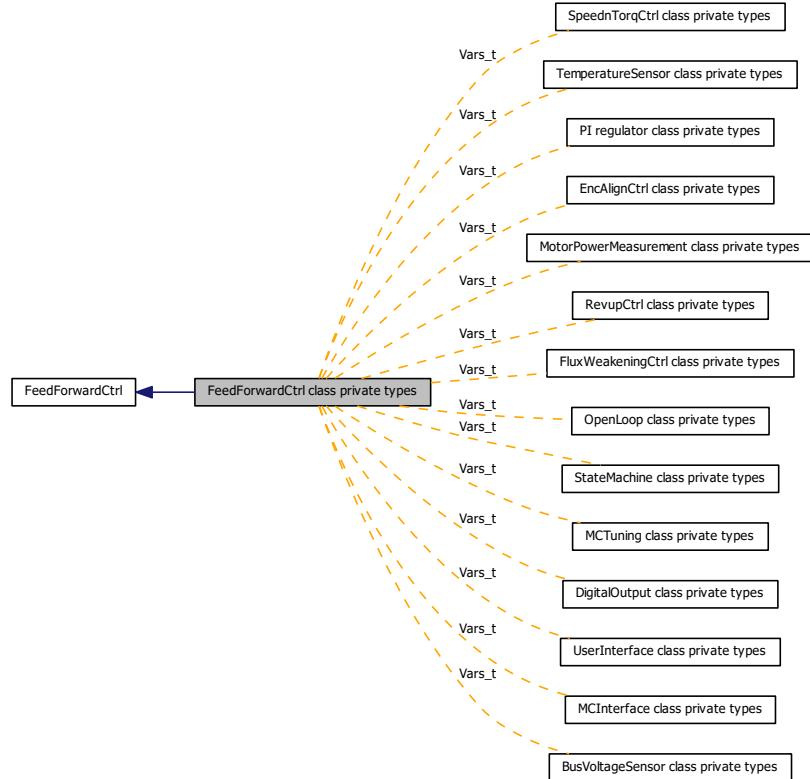
Return values

int16_t	Ki gain
---------	---------

Definition at line 151 of file PIRegulatorClass.c.

3.31 FeedForwardCtrl class private types

Collaboration diagram for FeedForwardCtrl class private types:



Data Structures

- struct [Vars_t](#)
MCInterface class members definition.
- struct [_CFF_t](#)
Private FeedForwardCtrl class definition.

TypeDefs

- [typedef FeedForwardCtrlParams_t Params_t](#)
Redefinition of parameter structure.

3.31.1 Typedef Documentation

3.31.1.1 [typedef FeedForwardCtrlParams_t Params_t](#)

Redefinition of parameter structure.

Definition at line 65 of file FeedForwardCtrlPrivate.h.

3.32 FeedForwardCtrl class exported types

Collaboration diagram for FeedForwardCtrl class exported types:



Data Structures

- struct [FFInit_t](#)
FeedForwardCtrl class init structure type definition.
- struct [FeedForwardCtrlParams_t](#)
FeedForwardCtrl class parameters definition.

TypeDefs

- typedef struct CFF_t * [CFF](#)
Public FeedForwardCtrl class definition.

3.32.1 Typedef Documentation

3.32.1.1 typedef struct CFF_t* CFF

Public FeedForwardCtrl class definition.

Definition at line 64 of file FeedForwardCtrlClass.h.

3.33 FeedForwardCtrl class exported methods

Collaboration diagram for FeedForwardCtrl class exported methods:



Functions

- **CFF FF_NewObject (pFeedForwardCtrlParams_t pFeedForwardCtrlParams)**
Creates an object of the class FeedForwardCtrl.
- **void FF_Init (CFF this, pFFInit_t pFFInitStruct)**
Initializes all the object variables, usually it has to be called once right after object creation.
- **void FF_Clear (CFF this)**
It should be called before each motor restart and clears the Flux weakening internal variables.
- **void FF_VqdffComputation (CFF this, CSPD oSPD, Curr_Components lqdref)**
It implements feed-forward controller by computing new Vqdff value. This will be then summed up to PI output in IMFF_VqdConditioning method.
- **Volt_Components FF_VqdConditioning (CFF this, Volt_Components Vqd)**
It return the Vqd components fed in input plus the feed forward action and store the last Vqd values in the internal variable.
- **void FF_DataProcess (CFF this)**
It low-pass filters the Vqd voltage coming from the speed PI. Filter bandwidth depends on hVqdLowPassFilterBW parameter.
- **void FF_InitFOCAdditionalMethods (CFF this)**
Use this method to initialize FF vars in START_TO_RUN state.
- **void FF_SetFFConstants (CFF this, FF_TuningStruct_t sNewConstants)**
Use this method to set new values for the constants utilized by feed-forward algorithm.
- **FF_TuningStruct_t FF_GetFFConstants (CFF this)**
Use this method to get present values for the constants utilized by feed-forward algorithm.
- **Volt_Components FF_GetVqdff (CFF this)**
Use this method to get present values for the Vqd feed-forward components.
- **Volt_Components FF_GetVqdAvPlout (CFF this)**
Use this method to get values of the averaged output of qd axes currents PI regulators.

3.33.1 Function Documentation

3.33.1.1 CFF FF_NewObject (pFeedForwardCtrlParams_t pFeedForwardCtrlParams)

Creates an object of the class FeedForwardCtrl.

Parameters

<i>pFeedForwardCtrlParams</i>	pointer to an FeedForwardCtrl parameters structure.
-------------------------------	---

Return values

<i>CFF</i>	new instance of FeedForwardCtrl object.
------------	---

3.33.1.2 void FF_Init (CFF *this*, pFFInit_t *pFFInitStruct*)

Initializes all the object variables, usually it has to be called once right after object creation.

Parameters

<i>this</i>	related object of class CFF.
<i>pFFInitStr</i>	Feed forward init strucuture.

Return values

<i>none</i>

3.33.1.3 void FF_Clear (CFF *this*)

It should be called before each motor restart and clears the Flux weakening internal variables.

Parameters

<i>this</i>	related object of class CFF.
-------------	------------------------------

Return values

<i>none</i>

3.33.1.4 void FF_VqdffComputation (CFF *this*, CSPD *oSPD*, Curr_Components *lqdref*)

It implements feed-forward controller by computing new Vqdff value. This will be then summed up to PI output in IMFF_VqdConditioning method.

Parameters

<i>this</i>	related object of class CFF.
<i>oSPD</i>	related SPD object.
<i>lqdref</i>	lqd reference componets used to calcupate the feed forward action.

Return values

<i>none</i>

3.33.1.5 Volt_Components FF_VqdConditioning (CFF *this*, Volt_Components *Vqd*)

It return the Vqd componets fed in input plus the feed forward action and store the last Vqd values in the internal variable.

Parameters

<i>this</i>	related object of class CFF.
<i>Vqd</i>	Initial value of Vqd to be manipulated by FF object.

Return values

<i>none</i>

3.33.1.6 void FF_DataProcess (CFF *this*)

It low-pass filters the Vqd voltage coming from the speed PI. Filter bandwidth depends on hVqdLowPassFilterBW parameter.

Parameters

<i>this</i>	related object of class CFF.
-------------	------------------------------

Return values

<i>none</i>

3.33.1.7 void FF_InitFOCAdditionalMethods (CFF *this*)

Use this method to initialize FF vars in START_TO_RUN state.

Parameters

<i>this</i>	related object of class CFF.
-------------	------------------------------

Return values

<i>none</i>

3.33.1.8 void FF_SetFFConstants (CFF *this*, FF_TuningStruct_t *sNewConstants*)

Use this method to set new values for the constants utilized by feed-forward algorithm.

Parameters

<i>this</i>	related object of class CFF.
-------------	------------------------------

<i>sNewConstants</i>	The FF_TuningStruct_t containing constants utilized by feed-forward algorithm.
----------------------	--

Return values

<i>none</i>

3.33.1.9 FF_TuningStruct_t FF_GetFFConstants (CFF *this*)

Use this method to get present values for the constants utilized by feed-forward algorithm.

Parameters

<i>this</i>	related object of class CFF.
-------------	------------------------------

Return values

FF_TuningStruct_t	Values of the constants utilized by feed-forward algorithm.
-------------------	---

3.33.1.10 Volt_Components FF_GetVqdff (CFF *this*)

Use this method to get present values for the Vqd feed-forward components.

Parameters

<i>this</i>	related object of class CFF.
-------------	------------------------------

Return values

Volt_Components	Vqd feed-forward components.
---------------------------------	------------------------------

3.33.1.11 Volt_Components FF_GetVqdAvPlout (CFF *this*)

Use this method to get values of the averaged output of qd axes currents PI regulators.

Parameters

<i>this</i>	related object of class CFF.
-------------	------------------------------

Return values

Volt_Components	Averaged output of qd axes currents PI regulators.
---------------------------------	--

3.34 MTPACtrl class private types

Collaboration diagram for MTPACtrl class private types:



Data Structures

- struct [_CMTPA_t](#)
Private MTPACtrl class definition.

Typedefs

- typedef [MTPACtrlParams_t Params_t](#)
Redefinition of parameter structure.

3.34.1 Typedef Documentation

3.34.1.1 [typedef MTPACtrlParams_t Params_t](#)

Redefinition of parameter structure.

Definition at line 47 of file MTPACtrlPrivate.h.

3.35 MTPACtrl class exported types

Collaboration diagram for MTPACtrl class exported types:



Data Structures

- struct [MTPACtrlParams_t](#)
MTPACtrl class parameters definition.

Typedefs

- typedef struct CMTPA_t * [CMTPA](#)
Public MTPACtrl class definition.

3.35.1 Typedef Documentation

3.35.1.1 [typedef struct CMTPA_t*](#) [CMTPA](#)

Public MTPACtrl class definition.

Definition at line 49 of file MTPACtrlClass.h.

3.36 MTPACtrl class exported methods

Collaboration diagram for MTPACtrl class exported methods:



Functions

- **CMTPA MTPA_NewObject (pMTPACtrlParams_t pMTPACtrlParams)**
Creates an object of the class MTPACtrl.
- **Curr_Components MTPA_CalcCurrRef (CMTPA this, Curr_Components lqdref)**
It compute new lqdref starting from the input lqdref. lqref is keep unchanged and is used to compute Idref according the MTPA algorithm.

3.36.1 Function Documentation

3.36.1.1 CMTPA MTPA_NewObject (pMTPACtrlParams_t pMTPACtrlParams)

Creates an object of the class MTPACtrl.

Parameters

<i>pMTPACtrl- Params</i>	pointer to an MTPACtrl parameters structure.
------------------------------	--

Return values

<i>CMTPA</i>	new instance of MTPACtrl object.
--------------	----------------------------------

3.36.1.2 Curr_Components MTPA_CalcCurrRef (CMTPA this, Curr_Components lqdref)

It compute new lqdref starting from the input lqdref. lqref is keep unchanged and is used to compute Idref according the MTPA algorithm.

Parameters

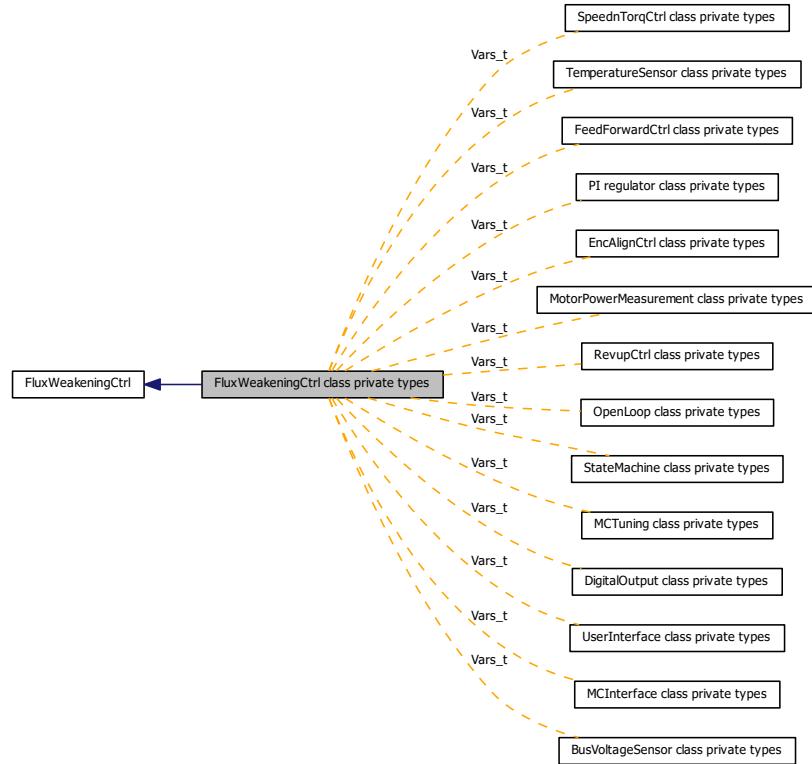
<i>this</i>	related object of class CMTPA
<i>lqdref</i>	The starting current components that have to be manipulated by the flux weakening algorithm.

Return values

<i>none</i>

3.37 FluxWeakeningCtrl class private types

Collaboration diagram for FluxWeakeningCtrl class private types:



Data Structures

- struct `Vars_t`
MCInterface class members definition.
- struct `_CFW_t`
Private FluxWeakeningCtrl class definition.

Typedefs

- `typedef FluxWeakeningCtrlParams_t Params_t`
Redefinition of parameter structure.

3.37.1 Typedef Documentation

3.37.1.1 `typedef FluxWeakeningCtrlParams_t Params_t`

Redefinition of parameter structure.

Definition at line 64 of file FluxWeakeningCtrlPrivate.h.

3.38 FluxWeakeningCtrl class exported types

Collaboration diagram for FluxWeakeningCtrl class exported types:



Data Structures

- struct [FWInit_t](#)
FluxWeakeningCtrl class init structure type definition.
- struct [FluxWeakeningCtrlParams_t](#)
FluxWeakeningCtrl class parameters definition.

Typedefs

- typedef struct CFW_t * [CFW](#)
Public FluxWeakeningCtrl class definition.

3.38.1 Typedef Documentation

3.38.1.1 typedef struct CFW_t* CFW

Public FluxWeakeningCtrl class definition.

Definition at line 62 of file FluxWeakeningCtrlClass.h.

3.39 FluxWeakeningCtrl class exported methods

Collaboration diagram for FluxWeakeningCtrl class exported methods:



Functions

- **CFW_FW_NewObject (pFluxWeakeningCtrlParams_t pFluxWeakeningCtrlParams)**
Creates an object of the class FeedForwardCtrl.
- **void FW_Init (CFW this, pFWInit_t pFWInitStr)**
Initializes all the object variables, usually it has to be called once right after object creation.
- **void FW_Clear (CFW this)**
It should be called before each motor restart and clears the Flux weakening internal variables with the exception of the target voltage (hFW_V_Ref).
- **Curr_Components FW_CalcCurrRef (CFW this, Curr_Components lqref)**
It computes lqref according the flux weakening algorithm. Inputs are the starting lqref components. As soon as the speed increases beyond the nominal one, fluxweakening algorithm take place and handles Idref value. Finally, accordingly with new Idref, a new lqref saturation value is also computed and put into speed PI.
- **void FW_DataProcess (CFW this, Volt_Components Vqd)**
It low-pass filters both the Vqd voltage components. Filter bandwidth depends on hVqdLowPassFilterBW parameter.
- **void FW_SetVref (CFW this, uint16_t hNewVref)**
Use this method to set a new value for the voltage reference used by flux weakening algorithm.
- **uint16_t FW_GetVref (CFW this)**
It returns the present value of target voltage used by flux weakening algorihtm.
- **int16_t FW_GetAvVAmplitude (CFW this)**
It returns the present value of voltage actually used by flux weakening algorihtm.
- **uint16_t FW_GetAvVPercentage (CFW this)**
It returns the measure of present voltage actually used by flux weakening algorihtm as percentage of available voltage.

3.39.1 Function Documentation

3.39.1.1 CFW_FW_NewObject (pFluxWeakeningCtrlParams_t pFluxWeakeningCtrlParams)

Creates an object of the class FeedForwardCtrl.

Parameters

<i>pFeedForward-CtrlParams</i>	pointer to an FeedForwardCtrl parameters structure.
--------------------------------	---

Return values

<i>CFF</i>	new instance of FeedForwardCtrl object.
------------	---

Creates an object of the class FeedForwardCtrl.

Parameters

<i>pFlux-WeakeningCtrl-Params</i>	pointer to an FluxWeakeningCtrl parameters structure
-----------------------------------	--

Return values

<i>CFW</i>	new instance of FluxWeakeningCtrl object
------------	--

Definition at line 52 of file FluxWeakeningCtrlClass.c.

References MAX_FW_NUM, MC_NULL, and _CFW_t::pParams_str.

3.39.1.2 void FW_Init (CFW *this*, pFWInit_t *pFWInitStr*)

Initializes all the object variables, usually it has to be called once right after object creation.

Parameters

<i>this</i>	related object of class CFW.
<i>pFWInitStr</i>	Flux weakening init structure.

Return values

<i>none.</i>	
--------------	--

Definition at line 81 of file FluxWeakeningCtrlClass.c.

References Vars_t::hFW_V_Ref, Vars_t::oFluxWeakeningPI, and Vars_t::oSpeedPI.

3.39.1.3 void FW_Clear (CFW *this*)

It should be called before each motor restart and clears the Flux weakening internal variables with the exception of the target voltage (hFW_V_Ref).

Parameters

<i>this</i>	related object of class CFW.
-------------	------------------------------

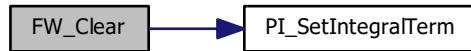
Return values

<i>none</i>	
-------------	--

Definition at line 100 of file FluxWeakeningCtrlClass.c.

References Vars_t::AvVolt_qd, Vars_t::AvVoltAmpl, Vars_t::hldRefOffset, Vars_t::oFluxWeakeningPI, and PI_SetIntegralTerm().

Here is the call graph for this function:



3.39.1.4 Curr_Components FW_CalcCurrRef (CFW this, Curr_Components lqdref)

It computes lqdref according the flux weakening algorithm. Inputs are the starting lqref components. As soon as the speed increases beyond the nominal one, fluxweakening algorithm take place and handles Idref value. Finally, accordingly with new Idref, a new lqref saturation value is also computed and put into speed PI.

Parameters

<i>this</i>	related object of class CFW
<i>lqdref</i>	The starting current components that have to be manipulated by the flux weakening algorithm.

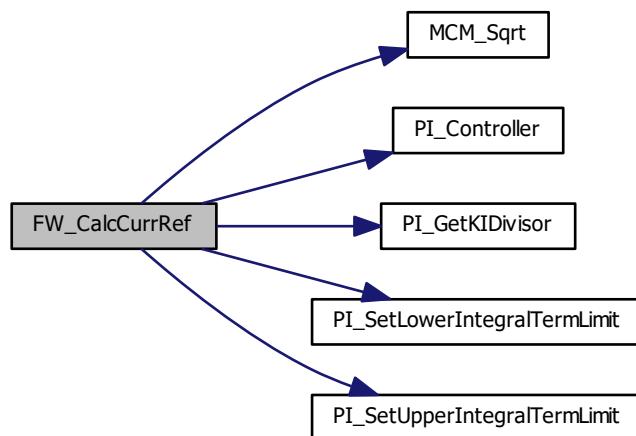
Return values

<i>Curr_Components</i>	Computed lqdref.
------------------------	------------------

Definition at line 123 of file FluxWeakeningCtrlClass.c.

References Vars_t::AvVolt_qd, Vars_t::AvVoltAmpl, Vars_t::hFW_V_Ref, Vars_t::hldRefOffset, MCM_Sqrt(), Vars_t::oFluxWeakeningPI, Vars_t::oSPEEDPI, PI_Controller(), PI_GetKIDivisor(), PI_SetLowerIntegralTermLimit(), and PI_SetUpperIntegralTermLimit().

Here is the call graph for this function:



3.39.1.5 void FW_DataProcess (CFW *this*, Volt_Components *Vqd*)

It low-pass filters both the Vqd voltage components. Filter bandwidth depends on hVqdLowPassFilterBW parameter.

Parameters

<i>this</i>	related object of class CFW.
<i>Vqd</i>	Voltage components to be averaged.

Return values

<i>none</i>

Definition at line 216 of file FluxWeakeningCtrlClass.c.

References Vars_t::AvVolt_qd.

3.39.1.6 void FW_SetVref (CFW *this*, uint16_t *hNewVref*)

Use this method to set a new value for the voltage reference used by flux weakening algorithm.

Parameters

<i>this</i>	related object of class CFW.
<i>uint16_t</i>	New target voltage value, expressend in tenth of percentage points of available voltage.

Return values

<i>none</i>

Definition at line 264 of file FluxWeakeningCtrlClass.c.

References Vars_t::hFW_V_Ref.

3.39.1.7 uint16_t FW_GetVref (CFW *this*)

It returns the present value of target voltage used by flux weakening algorihtm.

Parameters

<i>this</i>	related object of class CFW.
-------------	------------------------------

Return values

<i>int16_t</i>	Present target voltage value expressed in tenth of percentage points of available voltage.
----------------	--

Definition at line 277 of file FluxWeakeningCtrlClass.c.

References Vars_t::hFW_V_Ref.

3.39.1.8 int16_t FW_GetAvVAmplitude (CFW *this*)

It returns the present value of voltage actually used by flux weakening algorihtm.

Parameters

<i>this</i>	related object of class CFW.
-------------	------------------------------

Return values

<i>int16_t</i>	Present averaged phase stator voltage value, expressed in s16V (0-to-peak), where PhaseVoltage(V) = [PhaseVoltage(s16A) * Vbus(V)] /[sqrt(3) *32767].
----------------	---

Definition at line 291 of file FluxWeakeningCtrlClass.c.

References Vars_t::AvVoltAmpl.

3.39.1.9 uint16_t FW_GetAvVPercentage (CFW this)

It returns the measure of present voltage actually used by flux weakening algorihtm as percentage of available voltage.

Parameters

<i>this</i>	related object of class CFW.
-------------	------------------------------

Return values

<i>uint16_t</i>	Present averaged phase stator voltage value, expressed in tenth of percentage points of available voltage.
-----------------	--

Definition at line 304 of file FluxWeakeningCtrlClass.c.

References Vars_t::AvVoltAmpl.

3.40 PWMnCurrFdbk class exported types

Collaboration diagram for PWMnCurrFdbk class exported types:



Data Structures

- struct [PWMnCurrFdbkParams_t](#)
PWMnCurrFdbk class parameters definition.

Typedefs

- typedef struct CPWMC_t * [CPWMC](#)
Public PWMnCurrFdbk class definition.

Enumerations

- enum [LowSideOutputsFunction_t](#) { **LS_DISABLED** = 0, **LS_PWM_TIMER** = !0, **ES_GPIO** = 2 }
Low side or enabling signal definition.
- enum [CRCAction_t](#) { **CRC_START**, **CRC_EXEC** }
Low side or enabling signal definition.

3.40.1 Typedef Documentation

3.40.1.1 [typedef struct CPWMC_t* CPWMC](#)

Public PWMnCurrFdbk class definition.

Definition at line 63 of file PWMnCurrFdbkClass.h.

3.40.2 Enumeration Type Documentation

3.40.2.1 [enum LowSideOutputsFunction_t](#)

Low side or enabling signal definition.

Enumerator:

LS_DISABLED Low side signals and enabling signals always off. It is equivalent to DISABLED.

LS_PWM_TIMER Low side PWM signals are generated by timer. It is equivalent to ENABLED.

ES_GPIO Enabling signals are managed by GPIO.

Definition at line 84 of file PWMnCurrFdbkClass.h.

3.40.2.2 enum CRCAction_t

Low side or enabling signal definition.

Enumerator:

CRC_START Initialize the current reading calibration.

CRC_EXEC Execute the current reading calibration.

Definition at line 100 of file PWMnCurrFdbkClass.h.

3.41 PWMnCurrFdbk class exported methods

Collaboration diagram for PWMnCurrFdbk class exported methods:



Functions

- void [PWMC_Init \(CPWMC this\)](#)
Initialize all the object variables and MCU peripherals, usually it has to be called once right after object creation. Note: All the GPIOx port peripherals clocks are here enabled. Note: If the derived class is IHD2, R1HD2 or R3HD2 it is required to call the specific xxx_StartTimers method after the PWMC_Init call.
- void [PWMC_GetPhaseCurrents \(CPWMC this, Curr_Components *pStator_Currents\)](#)
It is used to get the motor phase current in Curr_Components format as read by AD converter.
- uint16_t [PWMC_SetPhaseVoltage \(CPWMC this, Volt_Components Valfa_beta\)](#)
It converts input voltage components Valfa, beta into duty cycles and feed it to the inverter.
- void [PWMC_SwitchOffPWM \(CPWMC this\)](#)
It switch off the PWM generation, setting to inactive the outputs.
- void [PWMC_SwitchOnPWM \(CPWMC this\)](#)
It switch on the PWM generation.
- bool [PWMC_CurrentReadingCalibr \(CPWMC this, CRCAction_t action\)](#)
It calibrates ADC current conversions by reading the offset voltage present on ADC pins when no motor current is flowing. It's suggested to call this function before each motor start-up.
- void [PWMC_TurnOnLowSides \(CPWMC this\)](#)
It turns on low sides. This function is intended to be used for charging boot capacitors of driving section. It has to be called each motor start-up when using high voltage drivers.
- uint16_t [PWMC_ExecRegularConv \(CPWMC this, uint8_t bChannel, uint8_t ADC_Unit\)](#)
Execute a regular conversion using ADC1. The function is not re-entrant (can't be executed twice at the same time) It returns 0xFFFF in case of conversion error.
- void [PWMC_ADC_SetSamplingTime \(CPWMC this, ADCConv_t ADConv_struct\)](#)
It sets the specified sampling time for the specified ADC channel. It must be called once for each channel utilized by user.
- uint16_t [PWMC_CheckOverCurrent \(CPWMC this\)](#)
It is used to check if an overcurrent occurred since last call.
- bool [PWMC_GetTurnOnLowSidesAction \(CPWMC this\)](#)
It is used to retrieve the status of TurnOnLowSides action.

3.41.1 Function Documentation

3.41.1.1 void PWMC_Init (CPWMC this)

Initialize all the object variables and MCU peripherals, usually it has to be called once right after object creation. Note: All the GPIOx port peripherals clocks are here enabled. Note: If the derived class is IHD2, R1HD2 or R3HD2 it is required to call the specific xxx_StartTimers method after the PWMC_Init call.

Parameters

<i>this</i>	PWM 'n Current feedback object
-------------	--------------------------------

Return values

<i>none</i>

Definition at line 97 of file PWMnCurrFdbkClass.c.

References Vars_t::bTurnOnLowSidesAction, and Vars_t::hT_Sqrt3.

3.41.1.2 void PWMC_GetPhaseCurrents (CPWMC *this*, Curr_Components * *pStator_Currents*)

It is used to get the motor phase current in [Curr_Components](#) format as read by AD converter.

Parameters

<i>this,:</i>	PWM 'n Current feedback object
<i>pStator_Currents</i>	Pointer to the struct that will receive motor current of phase A and B in Curr_Components format.

Return values

<i>none.</i>

Definition at line 125 of file PWMnCurrFdbkClass.c.

3.41.1.3 uint16_t PWMC_SetPhaseVoltage (CPWMC *this*, Volt_Components *Valfa_beta*)

It converts input voltage components *Valfa*, *beta* into duty cycles and feed it to the inverter.

Parameters

<i>this,:</i>	PWM 'n Current feedback object
<i>Valfa_beta,:</i>	Voltage Components in alfa beta reference frame

Return values

<i>It</i>	returns the code error 'MC_FOC_DURATION' if any, 'MC_NO_ERROR' otherwise. These error codes are defined in MC_type.h
-----------	---

Definition at line 147 of file PWMnCurrFdbkClass.c.

References Vars_t::hSector, and Vars_t::hT_Sqrt3.

3.41.1.4 void PWMC_SwitchOffPWM (CPWMC *this*)

It switch off the PWM generation, setting to inactive the outputs.

Parameters

<i>this,:</i>	PWM 'n Current feedback object
---------------	--------------------------------

Return values

<i>none</i>

Definition at line 254 of file PWMnCurrFdbkClass.c.

References Vars_t::bTurnOnLowSidesAction.

Referenced by PWMC_CurrentReadingCalibr(), and TSK_HardwareFaultTask().

3.41.1.5 void PWMC_SwitchOnPWM (CPWMC *this*)

It switch on the PWM generation.

Parameters

<i>this,:</i>	PWM 'n Current feedback object
---------------	--------------------------------

Return values

<i>None</i>

Definition at line 267 of file PWMnCurrFdbkClass.c.

References Vars_t::bTurnOnLowSidesAction.

3.41.1.6 bool PWMC_CurrentReadingCalibr (CPWMC *this*, CRCAction_t *action*)

It calibrates ADC current conversions by reading the offset voltage present on ADC pins when no motor current is flowing. It's suggested to call this function before each motor start-up.

Parameters

<i>this,:</i>	PWM 'n Current feedback object
<i>action,:</i>	it can be CRC_START to initialize the offset calibration or CRC_EXEC to execute the offset calibration.

Return values

<i>bool</i>	It returns TRUE if the current calibration has been completed otherwise if is ongoing it returns FALSE.
-------------	---

Parameters

<i>this,:</i>	PWM 'n Current feedback object
<i>action,:</i>	it can be CRC_START to initialize the offset calibration or CRC_EXEC to execute the offset calibration.

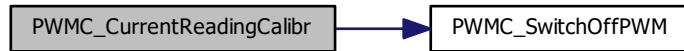
Return values

<i>bool</i>	It returns true if the current calibration has been completed otherwise if is ongoing it returns false.
-------------	---

Definition at line 284 of file PWMnCurrFdbkClass.c.

References CRC_EXEC, CRC_START, Vars_t::hOffCalibrWaitTimeCounter, and PWMC_SwitchOffPWM().

Here is the call graph for this function:



3.41.1.7 void PWMC_TurnOnLowSides (CPWMC *this*)

It turns on low sides. This function is intended to be used for charging boot capacitors of driving section. It has to be called each motor start-up when using high voltage drivers.

Parameters

<i>this,:</i>	PWM 'n Current feedback object
---------------	--------------------------------

Return values

<i>None</i>	It turns on low sides. This function is intended to be used for charging boot capacitors of driving section. It has to be called each motor start-up when using high voltage drivers.
-------------	---

Parameters

<i>this,:</i>	PWM 'n Current feedback object
---------------	--------------------------------

Return values

<i>None</i>

Definition at line 328 of file PWMnCurrFdbkClass.c.

References Vars_t::bTurnOnLowSidesAction.

3.41.1.8 uint16_t PWMC_ExecRegularConv (CPWMC *this*, uint8_t *bChannel*, uint8_t *ADC_Unit*)

Execute a regular conversion using ADC1. The function is not re-entrant (can't executed twice at the same time) It returns 0xFFFF in case of conversion error.

Parameters

<i>this</i>	related object of class CPWMC, ADC channel to be converted
<i>bChannel</i>	ADC channel used for the regular conversion

Return values

<i>It</i>	returns converted value or 0xFFFF for conversion error
-----------	--

Execute a regular conversion using ADC1. The function is not re-entrant (can't executed twice at the same time) It returns 0xFFFF in case of conversion error.

Parameters

<i>this</i>	related object of class CPWMC, ADC channel to be converted
<i>bChannel</i>	ADC channel used for the regular conversion

Return values

<i>l</i>	returns converted value or 0xFFFF for conversion error
----------	--

Definition at line 345 of file PWMnCurrFdbkClass.c.

3.41.1.9 void PWMC_ADC_SetSamplingTime (CPWMC *this*, ADConv_t *ADConv_struct*)

It sets the specified sampling time for the specified ADC channel. It must be called once for each channel utilized by user.

Parameters

<i>this</i>	related object of class CPWMC
<i>ADConv_struct</i>	struct containing ADC channel and sampling time

Return values

<i>none</i>	It sets the specified sampling time for the specified ADC channel. It must be called once for each channel utilized by user.
-------------	--

Parameters

<i>this</i>	related object of class CPWMC
<i>ADConv_struct</i>	struct containing ADC channel and sampling time

Return values

<i>none</i>	
-------------	--

Definition at line 374 of file PWMnCurrFdbkClass.c.

3.41.1.10 uint16_t PWMC_CheckOverCurrent (CPWMC *this*)

It is used to check if an overcurrent occurred since last call.

Parameters

<i>this</i>	related object of class CPWMC
-------------	-------------------------------

Return values

<i>uint16_t</i>	It returns MC_BREAK_IN whether an overcurrent has been detected since last method call, MC_NO_FAULTS otherwise.
-----------------	---

Definition at line 385 of file PWMnCurrFdbkClass.c.

3.41.1.11 bool PWMC_GetTurnOnLowSidesAction (CPWMC *this*)

It is used to retrieve the status of TurnOnLowSides action.

Parameters

<i>this</i>	related object of class CPWMC
-------------	-------------------------------

Return values

<i>bool</i>	It returns the state of TurnOnLowSides action: TRUE if TurnOnLowSides action is active, FALSE otherwise.
-------------	--

Parameters

<i>this</i>	related object of class CPWMC
-------------	-------------------------------

Return values

<i>bool</i>	It returns the state of TurnOnLowSides action: true if TurnOnLowSides action is active, false otherwise.
-------------	--

Definition at line 412 of file PWMnCurrFdbkClass.c.

References Vars_t::bTurnOnLowSidesAction.

3.42 ICS class Description

Collaboration diagram for ICS class Description:



ICS PWMnCurrFdbk class implementation. ICS current sensing carried out through isolated current sensors or on shunt resistors. Also the shunt resistors positioning must be configured: on phases or on legs of inverter. The sampling of the currents can be performed every where in the PWM period. It has been chosen to sample simultaneously the current flows when the bottom transistor of the respective inverter leg is switched on.

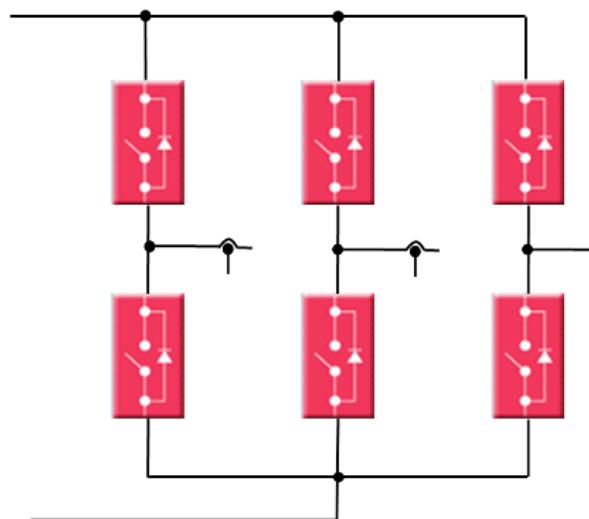


Figure 3.1: Current Sensing on phases

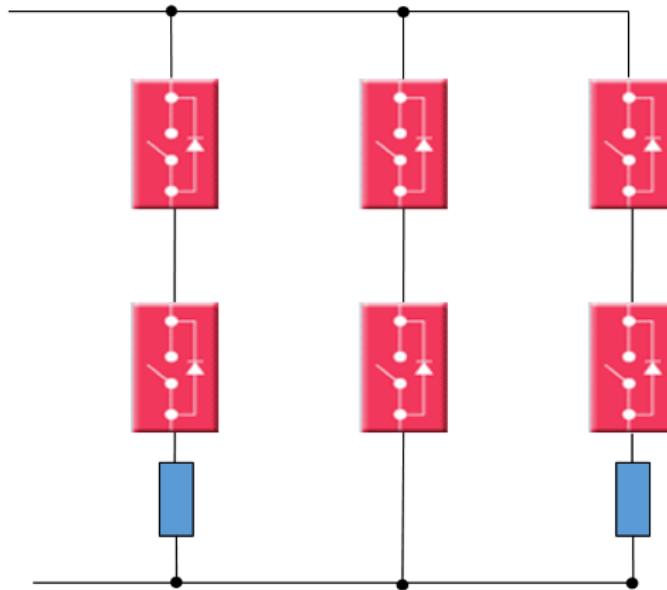


Figure 3.2: Current Sensing on legs

In standard motor operation, where the supplied voltage is generated using the space vector modulation, the sampling instant of phase current takes place in the middle of the PWM period in which all bottom transistors are switched on.

The three currents I_1 , I_2 , and I_3 flowing through a three-phase system follow the mathematical relationship:

$$I_1 + I_2 + I_3 = 0$$

Therefore, to reconstruct the currents flowing through a generic three-phase load, it is sufficient to sample only two out of the three currents while the third one can be computed by using the above relationship.

For correct calculation of the FOC algorithm, the phase current measurements must be executed at the same time. The flexibility of the ADC converter makes it possible the sampling of the phases at the same time using Dual Conversion Mode. Moreover, also the SINGLE ADC conversion can be used when only one ADC module is available. The sequence of these two measurements is handled by CTU triggers T0CR. The result of each conversion is stored in one of the four available FIFOs of CTU module. DMA transfer is used to access to CTU FIFO.

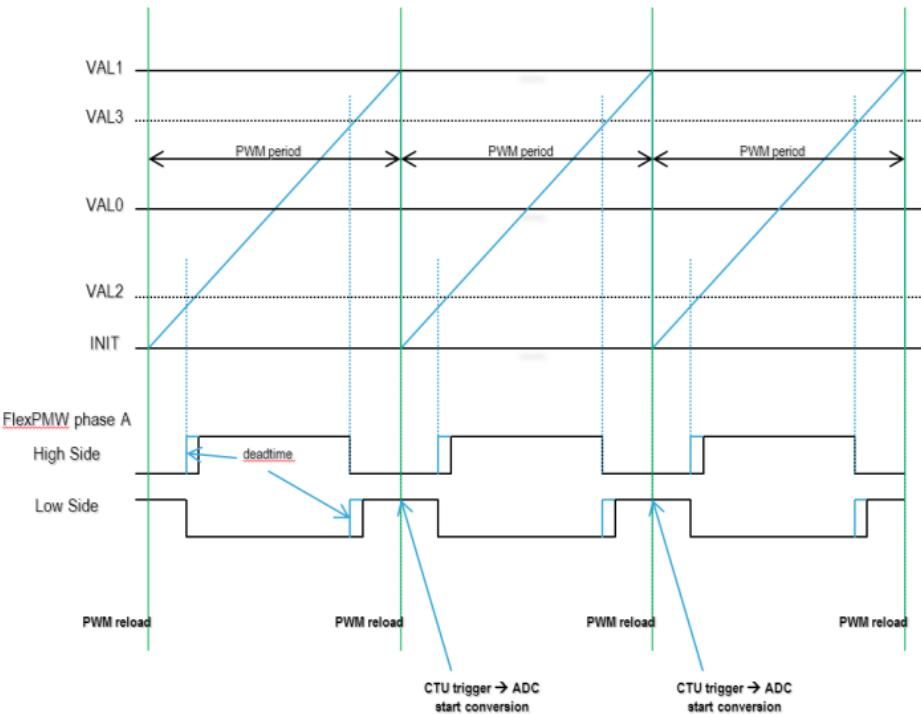


Figure 3.3: Current Sensing

This class generate duty cycles phase A, phase B, and phase C of different PWM periods. These phase voltage waveforms correspond to a center-aligned PWM.

Moreover, the class can be used to execute a regular conversion for other purpose (for example, for Bus voltage, temperature sensor, etc).

The CICS_PWMC derived class uses the following hardware peripherals:

FLEXPWM

- The class uses six PWM channels, each of which is configured to control a single half-bridge power stage
- PWM is configured in complementary mode to drive a 3-phase DC/AC inverter
- FlexPWM module is configured in PWM Center-Aligned mode and in signed-mode: INIT register is configured with negative value (2's complement) of VAL1
- FlexPWM modules are synchronized using the Master Reload Signal (MRS). The FlexPWM_0 submodule #0 is configured to run as a master and to generate MRS and counter synchronization signal (master sync) for other submodules.
- The MRS signal is generated every opportunity (configurable) of submodule #0, VAL1 compare, that is, full cycle reload. All double buffered registers, including compare registers VAL2, VAL3 are updated on the occurrence of MRS, therefore new PWM duty cycles are updated each PWM periods (configurable).

CTU

- The MRS signal generated from the FlexPWM module is internally routed to the Cross Triggering Unit (CTU) module
- The MRS signal is generated for each PWM periods (configurable). The CTU counter can count up to value equal to PWM period
- T0CR trigger compare registers is used to generate ADC command event: The value of T0CR is configured to zero. In this way, when a T0CR trigger events occurs, it generates the ADC start of conversion request at the beginning of each PWM reload cycle. So, the phase current is measured when the bottom transistor are swithed on.

ADC

- ADC channels are configured in Injection mode to perform Current Reading Calibration
- ADC channels are configured in Dual Conversion mode to sampling of the phases at the same time
- ADC can be configured in Single Conversion mode to execute User regular conversion.
- ADC can be configured in Single Conversion mode to execute Voltage Bus conversion.

DMA

- DMA module is used to transfer the phases converted values from CTU FIFO.

3.43 ICS_class exported types

Collaboration diagram for ICS_class exported types:



Data Structures

- struct [ICS_Params_t](#)
ICS class parameters definition.

TypeDefs

- typedef struct CICS_PWMC_t * [CICS_PWMC](#)
Public ICS class definition.

Enumerations

- enum [ICS_ShuntPosition_t](#) { [ICS_SHUNT_PHASES](#) = 0U, [ICS_SHUNT_LEGS](#) = 1U }
Shunt Position (PHASE/LEGS)
- enum [ICS_SensingSelection_t](#) { [ICS_SENSING_ON_UV_PHASES](#) = 0U, [ICS_SENSING_ON_UW_PHASES](#) = 1U, [ICS_SENSING_ON_VW_PHASES](#) = 2U }
Shunt Position (PHASE/LEGS)

3.43.1 TypeDef Documentation

3.43.1.1 typedef struct CICS_PWMC_t* CICS_PWMC

Public ICS class definition.

Definition at line 89 of file ICS_PWMnCurrFdbkClass.h.

3.43.2 Enumeration Type Documentation

3.43.2.1 enum ICS_ShuntPosition_t

Shunt Position (PHASE/LEGS)

Enumerator:

[ICS_SHUNT_PHASES](#) Shunt position on Phases.

[ICS_SHUNT_LEGS](#) Shunt position on Legs.

Definition at line 63 of file ICS_PWMnCurrFdbkClass.h.

3.43.2.2 enum ICS_SensingSelection_t

Shunt Position (PHASE/LEGS)

Enumerator:

ICS_SENSING_ON_UV_PHASES Sensing on U and V Phases.

ICS_SENSING_ON_UW_PHASES Sensing on U and W Phases.

ICS_SENSING_ON_VW_PHASES Sensing on V and W Phases.

Definition at line 75 of file ICS_PWMInCurrFdbkClass.h.

3.44 ICS class exported

Collaboration diagram for ICS class exported:



3.44.1 Detailed Description

methods

Functions

- **CICS_PWMC ICS_NewObject (pPWMnCurrFdbkParams_t pPWMnCurrFdbkParams, pICS_Params_t pICS_Params)**

Creates an object of the class ICS.

3.44.2 Function Documentation

3.44.2.1 CICS_PWMC ICS_NewObject (pPWMnCurrFdbkParams_t pPWMnCurrFdbkParams, pICS_Params_t pICS_Params)

Creates an object of the class ICS.

Parameters

in	<i>pPWMnCurrFdbkParams</i>	pointer to an PWMnCurrFdbk parameters structure
in	<i>pICS_Params</i>	pointer to an ICS parameters structure

Returns

CICS_PWMC new instance of ICS object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Definition at line 318 of file ICS_PWMnCurrFdbkClass.c.

References _CPWMC_t::DerivedClass, ICS_Params_t::IRQnb, MAX_DRV_PWMC_NUM, MC_NULL, _CPWMC_t::Methods_str, and _DCICS_PWMC_t::pDParams_str.

3.45 R1 class Description

Collaboration diagram for R1 class Description:



R1 PWMnCurrFdbk class implementation. The single-shunt current measurement is a low cost solution that measures the current and recreates each of the three-phase currents of the motor.

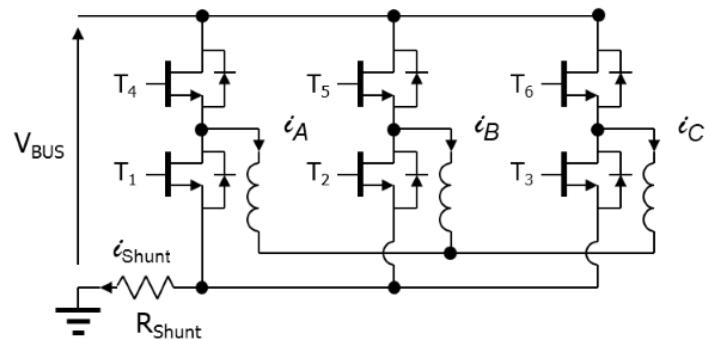


Figure 3.4: Single Shunt topology

For each configuration of the low side switches the current that flowing in the shunt resistor is indicated in the Table 1.

V	T ₁	T ₂	T ₃	i_{Shunt}
V ₀	0	0	0	0
V ₁	0	1	1	i_A
V ₂	0	0	1	$-i_C$
V ₃	1	0	1	i_B
V ₄	1	0	0	$-i_A$
V ₅	1	1	0	i_C
V ₆	0	1	0	$-i_B$
V ₇	1	1	1	0

Figure 3.5: Table 1: Current through the shunt resistor

T4, T5 and T6 assume the complementary value of T1, T2 and T3. In the Table 1 the value "0" means that the switch is open while the value "1" means that the switch is closed.

Using centered aligned pattern, each PWM period is subdivided in 7 sub periods. In three sub periods (I,IV,VII) the current flowing in the shunt resistor is null. In the remaining sub periods the current flowing in the shunt resistor is symmetrical respects the center of PWM.

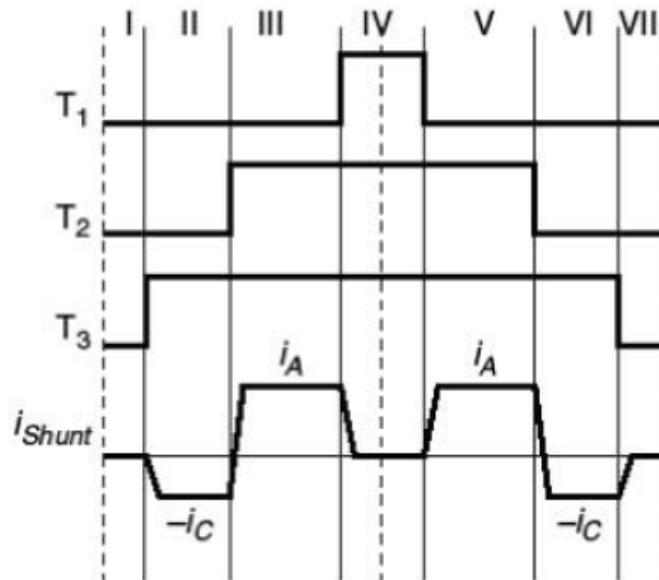


Figure 3.6: Single-shunt current reading

For the condition showed in figure above there are two pairs:

- sub periods II and VI in which the iShunt is equal to $-iC$
- sub periods III and V in which the iShunt is equal to iA

So in this condition it is possible to reconstruct the three phases motor current from the sampled values:

- iA is iShunt read in sub period III or V
- iC is -iShunt read in sub period II or VI
- $iB = -iA - iC$

The algorithm to work properly needs the following parameters:

- The Rising Time (TRISE): is the time between the turn on of the switches to a stable value in the ADC input. It measure both commutation noise and op-amp slew rate.
- Dead time
- Sampling Time: ADC conversion time

The minimum time required to perform the sampling will be referred as TMIN:

$$TMIN = TRISE + \text{Sampling Time} + \text{Dead Time}$$

When is not possible to sample one of the currents, an active vector is inserted inside the null sub periods V0 or V7 taking care of keeping unchanged the three duty cycles (and so the voltage applied to the motor phases). The duration of the active vector insertion is fixed and in order to reduce the harmonic content of the produced phase current must be kept as lower as possible (equal to Tmin).

The R1_PWM derived class uses the following hardware peripherals:

FLEXPWM

- The class uses six PWM channels, each of which is configured to control a single half-bridge power stage
- PWM is configured in complementary mode to drive a 3-phase DC/AC inverter
- FlexPWM module is configured in PWM Center-Aligned mode and in signed-mode: INIT register is configured with negative value (2's complement) of VAL1
- FlexPWM modules are synchronized using the Master Reload Signal (MRS). The FlexPWM_0 submodule #0 is configured to run as a master and to generate MRS and counter synchronization signal (master sync) for other submodules.
- The MRS signal is generated every HALF opportunity of submodule #0, VAL0 compare. All double buffered registers, including compare registers VAL2, VAL3 are updated on the occurrence of MRS.
- Double switching PWM output feature is used to generate the fixed time active vector insertion

CTU

- The MRS signal generated from the FlexPWM module is internally routed to the Cross Triggering Unit (CTU) module
- The MRS signal is generated for each PWM periods. The CTU counter can count up to value equal to half PWM period
- T0CR trigger compare registers is used to generate first ADC command event
- T4CR trigger compare registers is used to generate second ADC command event

ADC

- ADC channels are configured in Injection mode to perform Current Reading Calibration
- ADC channels are configured in Single Conversion/CTU mode
- ADC can be configured in Single Conversion mode to execute User regular conversion.

- ADC can be configured in Single Conversion mode to execute Voltage Bus conversion

DMA

- DMA module is used to transfer the phases converted values from CTU FIFO.

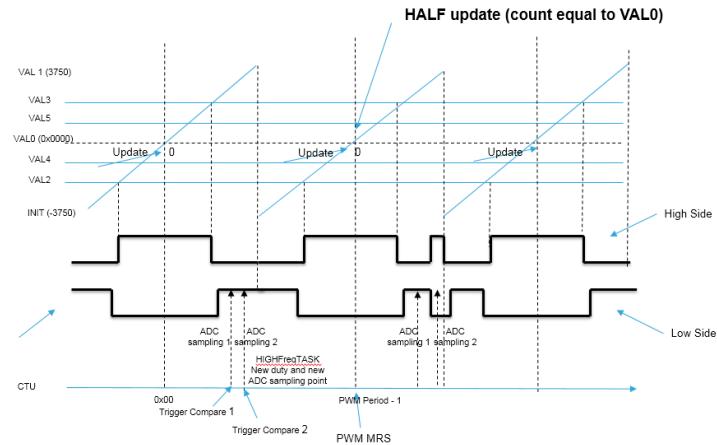


Figure 3.7: Single Shunt FlexPWM, CTU and ADC

3.46 R1_class exported types

Collaboration diagram for R1_class exported types:



Data Structures

- struct [R1_Params_t](#)
R1 class parameters definition.

Typedefs

- typedef struct CR1_PWM_C_t * [CR1_PWM_C](#)
Public R1 class definition.

3.46.1 Typedef Documentation

3.46.1.1 [typedef struct CR1_PWM_C_t* CR1_PWM_C](#)

Public R1 class definition.

Definition at line 60 of file R1_PWMnCurrFdbkClass.h.

3.47 R1 class exported methods

Collaboration diagram for R1 class exported methods:



Functions

- [CR1_PWM C R1_NewObject \(pPWMnCurrFdbkParams_t pPWMnCurrFdbkParams, pR1_Params_t pR1_Params \)](#)

Creates an object of the class R1.

3.47.1 Function Documentation

3.47.1.1 CR1_PWM C R1_NewObject (pPWMnCurrFdbkParams_t pPWMnCurrFdbkParams, pR1_Params_t pR1_Params)

Creates an object of the class R1.

Parameters

in	<i>pPWMnCurrFdbkParams</i>	pointer to an PWMnCurrFdbk parameters structure
in	<i>pR1_Params</i>	pointer to an R1 parameters structure

Returns

CR1_PWM C new instance of R1 object

Function Class:

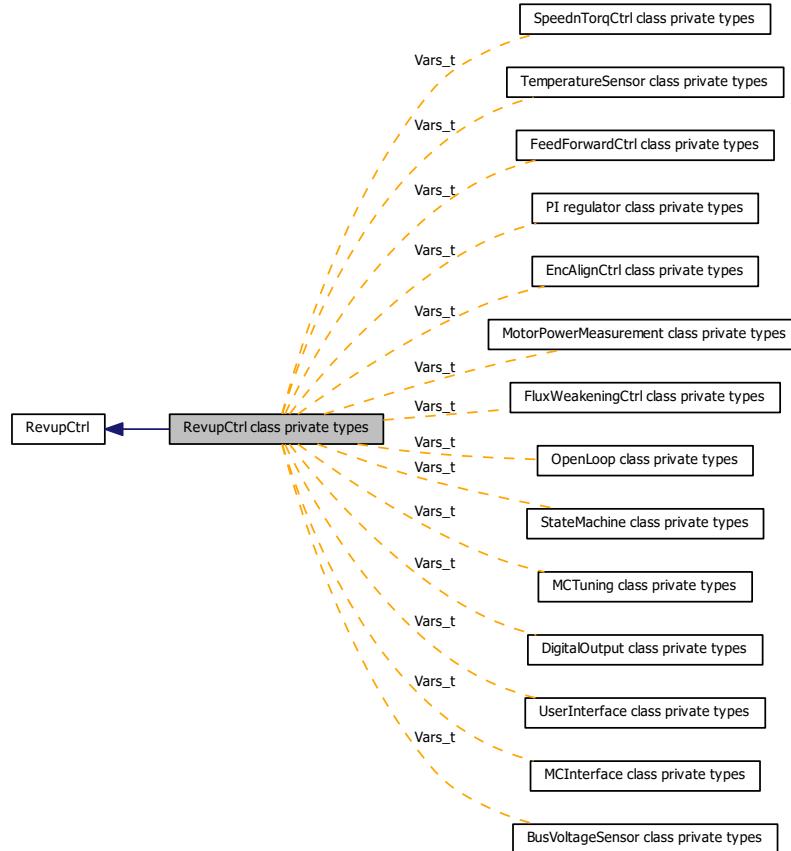
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Definition at line 302 of file R1_PWMnCurrFdbkClass.c.

References _CPWMC_t::DerivedClass, R1_Params_t::IRQnb, MAX_DRV_PWM_NUM, MC_NULL, _CPWMC_t::Methods_str, and _DCR1_PWM_C_t::pDParams_str.

3.48 RevupCtrl class private types

Collaboration diagram for RevupCtrl class private types:



Data Structures

- struct `Vars_t`
MCInterface class members definition.
- struct `_CRUC_t`
Private RevupCtrl class definition.

Typedefs

- typedef `RevupCtrlParams_t Params_t`
Redefinition of parameter structure.

3.48.1 Typedef Documentation

3.48.1.1 `typedef RevupCtrlParams_t Params_t`

Redefinition of parameter structure.

Definition at line 105 of file RevupCtrlPrivate.h.

3.49 RevupCtrl class exported types

Collaboration diagram for RevupCtrl class exported types:



Data Structures

- struct [RUCPhasesParams_t](#)
Parameters related to each rev up phase.
- struct [RevupCtrlParams_t](#)
RevupCtrl class parameters definition.

TypeDefs

- typedef struct CRUC_t * [CRUC](#)
Public RevupCtrl class definition.

3.49.1 Typedef Documentation

3.49.1.1 typedef struct CRUC_t* CRUC

Public RevupCtrl class definition.

Definition at line 70 of file RevupCtrlClass.h.

3.50 RevupCtrl class exported methods

Collaboration diagram for RevupCtrl class exported methods:



Functions

- **CRUC RUC_NewObject (pRevupCtrlParams_t pRevupCtrlParams)**
Creates an object of the class RevupCtrl.
- **void RUC_Init (CRUC this, CSTC oSTC, CVSS_SPD oVSS)**
Initializes all the object variables, usually it has to be called once right after object creation. It is also used to assign the speed and torque controller and the virtual speed sensor objects to be used by rev up controller.
- **void RUC_Clear (CRUC this, int16_t hMotorDirection)**
It should be called before each motor restart. It initialize internal state of RUC sets first commands to VSS and STC and calls the Clear method of VSS. It also sets the Speed and Torque controller in TORQUE mode.
- **bool RUC_Exec (CRUC this)**
It clocks the rev up controller and must be called with a frequency equal to the one set in the parameters hRUC-FrequencyHz. Calling this method the rev up controller perform the programmed sequence. Note: STC and VSS aren't clocked by RUC_Exec.
- **void RUC_SetPhaseDurationms (CRUC this, uint8_t bPhase, uint16_t hDurationms)**
It is used to modify the default value of duration of a specific rev up phase. Note: The module can be also compiled commenting the define RUC_ALLOWS_TUNING to optimize the flash memory occupation and the RAM usage if the tuning is not required in this case this function has no effect.
- **void RUC_SetPhaseFinalMecSpeed01Hz (CRUC this, uint8_t bPhase, int16_t hFinalMecSpeed01Hz)**
It is used to modify the default value of mechanical speed at the end of a specific rev up phase. Note: The module can be also compiled commenting the define RUC_ALLOWS_TUNING to optimize the flash memory occupation and the RAM usage if the tuning is not required in this case this function has no effect.
- **void RUC_SetPhaseFinalTorque (CRUC this, uint8_t bPhase, int16_t hFinalTorque)**
It is used to modify the default value of motor torque at the end of a specific rev up phase. Note: The module can be also compiled commenting the define RUC_ALLOWS_TUNING to optimize the flash memory occupation and the RAM usage if the tuning is not required in this case this function has no effect.
- **uint16_t RUC_GetPhaseDurationms (CRUC this, uint8_t bPhase)**
It is used to read the current value of duration of a specific rev up phase. Note: The module can be also compiled commenting the define RUC_ALLOWS_TUNING to optimize the flash memory occupation and the RAM usage if the tuning is not required in this case this function has no effect.
- **int16_t RUC_GetPhaseFinalMecSpeed01Hz (CRUC this, uint8_t bPhase)**
It is used to read the current value of mechanical speed at the end of a specific rev up phase. Note: The module can be also compiled commenting the define RUC_ALLOWS_TUNING to optimize the flash memory occupation and the RAM usage if the tuning is not required in this case this function has no effect.
- **int16_t RUC_GetPhaseFinalTorque (CRUC this, uint8_t bPhase)**
It is used to read the current value of motor torque at the end of a specific rev up phase. Note: The module can be also compiled commenting the define RUC_ALLOWS_TUNING to optimize the flash memory occupation and the RAM usage if the tuning is not required in this case this function has no effect.
- **uint8_t RUC_GetNumberOfPhases (CRUC this)**
It is used to get information about the number of phases relative to the programmed rev up. Note: The module can be also compiled commenting the define RUC_ALLOWS_TUNING to optimize the flash memory occupation and the RAM usage if the tuning is not required in this case this function has no effect.

- bool **RUC_FirstAccelerationStageReached (CRUC this)**

It is used to know if the programmed "first acceleration stage" has been reached inside the rev up sequence. Is intended that the stages previous to this are reserved for alignments. When is reached the first stage of acceleration the observer should be cleared once. NOTE: The rev up stage are zero-based indexed so that the fist stage is the number zero.

3.50.1 Function Documentation

3.50.1.1 CRUC RUC_NewObject (pRevupCtrlParams_t pRevupCtrlParams)

Creates an object of the class RevupCtrl.

Parameters

<i>pRevupCtrl- Params</i>	pointer to an RevupCtrl parameters structure
-------------------------------	--

Return values

<i>CRUC</i>	new instance of RevupCtrl object
-------------	----------------------------------

Definition at line 51 of file RevupCtrlClass.c.

References MAX_RUC_NUM, MC_NULL, and _CRUC_t::pParams_str.

3.50.1.2 void RUC_Init (CRUC this, CSTC oSTC, CVSS_SPD oVSS)

Initializes all the object variables, usually it has to be called once right after object creation. It is also used to assign the speed and torque controller and the virtual speed sensor objects to be used by rev up controller.

Parameters

<i>this</i>	related object of class CSTC.
<i>oSTC</i>	the speed and torque controller used by the RUC
<i>oVSS</i>	the virtual speed sensor used by the RUC

Return values

<i>none.</i>

Definition at line 83 of file RevupCtrlClass.c.

References Vars_t::bPhaseNbr, RUCPhasesParams_t::hDurationms, RUCPhasesParams_t::hFinalMecSpeed01-Hz, RUCPhasesParams_t::hFinalTorque, MC_NULL, Vars_t::oSTC, Vars_t::oVSS, Vars_t::ParamsData, and RU-CPhasesParams_t::pNext.

3.50.1.3 void RUC_Clear (CRUC this, int16_t hMotorDirection)

It should be called before each motor restart. It initialize internal state of RUC sets first commands to VSS and STC and calls the Clear method of VSS. It also sets the Speed and Torque controller in TORQUE mode.

Parameters

<i>this</i>	related object of class CSTC.
<i>hMotorDirection</i>	If it is "1" the programmed revup sequence is performed. If it is "-1" the revup sequence is performed with opposite values of targets (speed, torque).

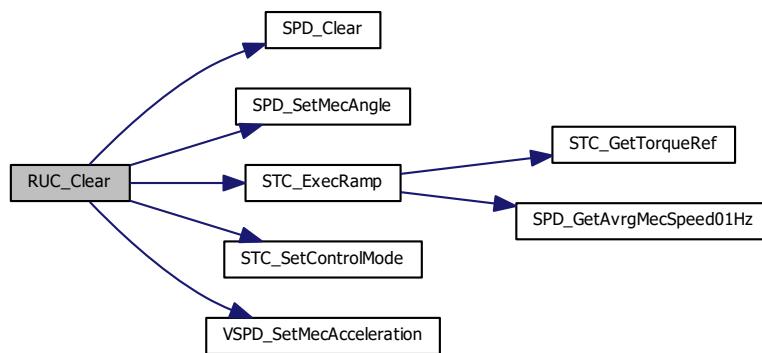
Return values

<i>none.</i>

Definition at line 122 of file RevupCtrlClass.c.

References `Vars_t::bStageCnt`, `Vars_t::hDirection`, `Vars_t::hPhaseRemainingTicks`, `Vars_t::oSTC`, `Vars_t::oVSS`, `Vars_t::ParamsData`, `RUCPhasesParams_t::pNext`, `Vars_t::pPhaseParams`, `SPD_Clear()`, `SPD_SetMecAngle()`, `STC_ExecRamp()`, `STC_SetControlMode()`, `STC_TORQUE_MODE`, and `VSPD_SetMecAcceleration()`.

Here is the call graph for this function:

**3.50.1.4 bool RUC_Exec (CRUC *this*)**

It clocks the rev up controller and must be called with a frequency equal to the one set in the parameters `hRUCFrequencyHz`. Calling this method the rev up controller perform the programmed sequence. Note: STC and VSS arent clocked by `RUC_Exec`.

Parameters

<i>this</i>	related object of class CRUC.
-------------	-------------------------------

Return values

<i>bool</i>	It returns FALSE when the programmed rev up has been completed.
-------------	---

Parameters

<i>this</i>	related object of class CRUC.
-------------	-------------------------------

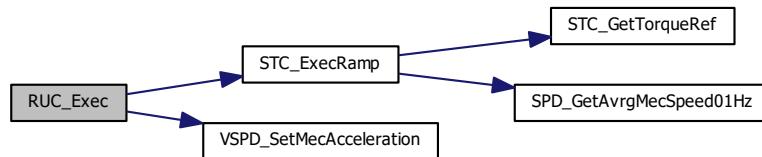
Return values

<i>bool</i>	It returns false when the programmed rev up has been completed.
-------------	---

Definition at line 176 of file RevupCtrlClass.c.

References `Vars_t::bStageCnt`, `Vars_t::hDirection`, `RUCPhasesParams_t::hDurationms`, `RUCPhasesParams_t::hFinalMecSpeed01Hz`, `RUCPhasesParams_t::hFinalTorque`, `Vars_t::hPhaseRemainingTicks`, `MC_NULL`, `Vars_t::oSTC`, `Vars_t::oVSS`, `RUCPhasesParams_t::pNext`, `Vars_t::pPhaseParams`, `STC_ExecRamp()`, and `VSPD_SetMecAcceleration()`.

Here is the call graph for this function:



3.50.1.5 void RUC_SetPhaseDurationms (CRUC *this*, uint8_t *bPhase*, uint16_t *hDurationms*)

It is used to modify the default value of duration of a specific rev up phase. Note: The module can be also compiled commenting the define RUC_ALLOWS_TUNING to optimize the flash memory occupation and the RAM usage if the tuning is not required in this case this function has no effect.

Parameters

<i>this</i>	related object of class CRUC.
<i>bPhase</i>	is the rev up phase, zero based, to be modified.
<i>hDurationms</i>	is the new value of duration for that phase.

Return values

<i>none.</i>

Definition at line 233 of file RevupCtrlClass.c.

References Vars_t::ParamsData.

3.50.1.6 void RUC_SetPhaseFinalMecSpeed01Hz (CRUC *this*, uint8_t *bPhase*, int16_t *hFinalMecSpeed01Hz*)

It is used to modify the default value of mechanical speed at the end of a specific rev up phase. Note: The module can be also compiled commenting the define RUC_ALLOWS_TUNING to optimize the flash memory occupation and the RAM usage if the tuning is not required in this case this function has no effect.

Parameters

<i>this</i>	related object of class CRUC.
<i>bPhase</i>	is the rev up phase, zero based, to be modified.
<i>hFinalMecSpeed01Hz</i>	is the new value of mechanical speed at the end of that phase expressed in 0.1Hz.

Return values

<i>none.</i>

Definition at line 255 of file RevupCtrlClass.c.

References Vars_t::ParamsData.

3.50.1.7 void RUC_SetPhaseFinalTorque (CRUC *this*, uint8_t *bPhase*, int16_t *hFinalTorque*)

It is used to modify the default value of motor torque at the end of a specific rev up phase. Note: The module can be also compiled commenting the define RUC_ALLOWS_TUNING to optimize the flash memory occupation and the RAM usage if the tuning is not required in this case this function has no effect.

Parameters

<i>this</i>	related object of class CRUC.
<i>bPhase</i>	is the rev up phase, zero based, to be modified.
<i>hFinalTorque</i>	is the new value of motor torque at the end of that phase. This value represents actually the Iq current expressed in digit.

Return values

<i>none.</i>

Definition at line 279 of file RevupCtrlClass.c.

References Vars_t::ParamsData.

3.50.1.8 uint16_t RUC_GetPhaseDurationms (CRUC *this*, uint8_t *bPhase*)

It is used to read the current value of duration of a specific rev up phase. Note: The module can be also compiled commenting the define RUC_ALLOWS_TUNING to optimize the flash memory occupation and the RAM usage if the tuning is not required in this case this function has no effect.

Parameters

<i>this</i>	related object of class CRUC.
<i>bPhase</i>	is the rev up phase, zero based, to be read.

Return values

<i>uint16_t</i>	The current value of duration for that phase expressed in milliseconds.
-----------------	---

Definition at line 300 of file RevupCtrlClass.c.

References Vars_t::ParamsData.

3.50.1.9 int16_t RUC_GetPhaseFinalMecSpeed01Hz (CRUC *this*, uint8_t *bPhase*)

It is used to read the current value of mechanical speed at the end of a specific rev up phase. Note: The module can be also compiled commenting the define RUC_ALLOWS_TUNING to optimize the flash memory occupation and the RAM usage if the tuning is not required in this case this function has no effect.

Parameters

<i>this</i>	related object of class CRUC.
<i>bPhase</i>	is the rev up phase, zero based, to be read.

Return values

<i>int16_t</i>	The current value of mechanical speed at the end of that phase expressed in 0.1Hz.
----------------	--

Definition at line 323 of file RevupCtrlClass.c.

References Vars_t::ParamsData.

3.50.1.10 int16_t RUC_GetPhaseFinalTorque (CRUC *this*, uint8_t *bPhase*)

It is used to read the current value of motor torque at the end of a specific rev up phase. Note: The module can be also compiled commenting the define RUC_ALLOWS_TUNING to optimize the flash memory occupation and the RAM usage if the tuning is not required in this case this function has no effect.

Parameters

<i>this</i>	related object of class CRUC.
<i>bPhase</i>	is the rev up phase, zero based, to be read.

Return values

<i>int16_t</i>	The current value of motor torque at the end of that phase. This value represents actually the Iq current expressed in digit.
----------------	---

Definition at line 346 of file RevupCtrlClass.c.

References Vars_t::ParamsData.

3.50.1.11 uint8_t RUC_GetNumberOfPhases (CRUC *this*)

It is used to get information about the number of phases relative to the programmed rev up. Note: The module can be also compiled commenting the define RUC_ALLOWS_TUNING to optimize the flash memory occupation and the RAM usage if the tuning is not required in this case this function has no effect.

Parameters

<i>this</i>	related object of class CRUC.
-------------	-------------------------------

Return values

<i>uint8_t</i>	The number of phases relative to the programmed rev up.
----------------	---

Definition at line 367 of file RevupCtrlClass.c.

References Vars_t::bPhaseNbr.

3.50.1.12 bool RUC_FirstAccelerationStageReached (CRUC *this*)

It is used to know if the programmed "first acceleration stage" has been reached inside the rev up sequence. Is intended that the stages previous to this are reserved for alignments. When is reached the first stage of acceleration the observer should be cleared once. NOTE: The rev up stage are zero-based indexed so that the fist stage is the number zero.

Parameters

<i>this</i>	related object of class CRUC.
-------------	-------------------------------

Return values

<i>bool</i>	It returns TRUE if the first acceleration stage has been reached and FALSE otherwise.
-------------	---

Parameters

<i>this</i>	related object of class CRUC.
-------------	-------------------------------

Return values

<i>bool</i>	It returns true if the first acceleration stage has been reached and false otherwise.
-------------	---

Definition at line 396 of file RevupCtrlClass.c.

References Vars_t::bStageCnt.

Referenced by TSK_HighFrequencyTask().

3.51 Encoder Description

Collaboration diagram for Encoder Description:



Encoder implementation using eTimer. An incremental encoder provides a specified amount of pulses in one rotation of the encoder. The output is composed by two lines of pulses (an A and B channel) that are offset in order to determine rotation. This phasing between the two signals is called quadrature. An index channel is used to generate a supplementary pulse per rotation. The index signal is used to check and adjust the absolute reference.

Quadrature incremental encoders are widely used to read the rotor position of electric machines. As the name implies, incremental encoders actually read angular displacements with respect to an initial position: if that position is known, then the rotor absolute angle is known too. For this reason, it is always necessary, when processing the encoder feedback, to perform a rotor prepositioning before the first startup.

The quadrature encoder is a relative position sensor, but the absolute informations is required for performing field-oriented control. To obtain absolute reference, the alignment phase is needed. This task is performed by means of Encoder Alignment Controller (CEAC) class, and shall be carried out at the first motor startup.

The CENC_SPD derived class use the eTimer hardware peripheral:

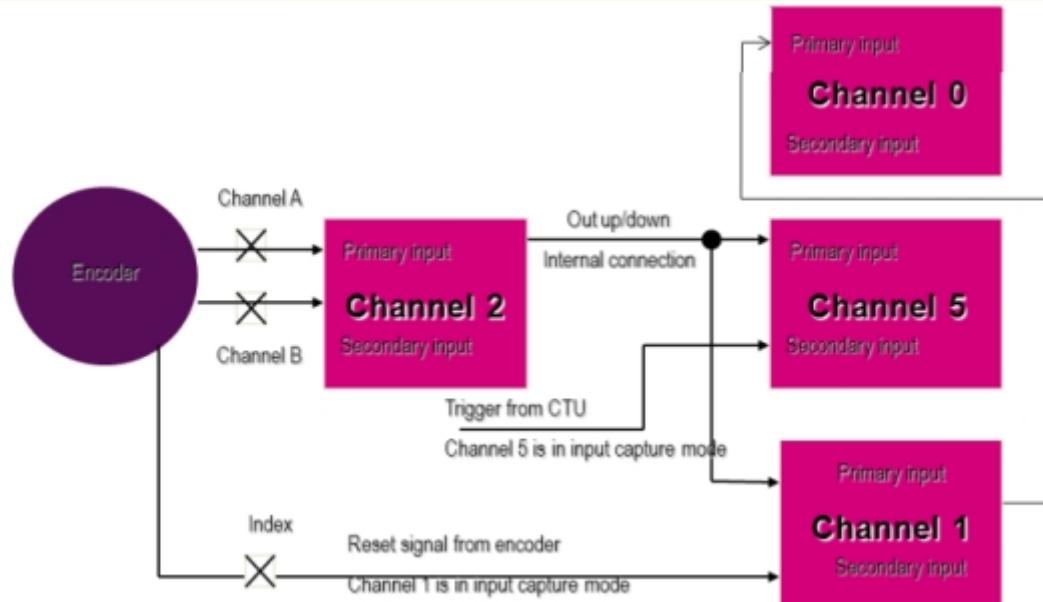


Figure 3.8: ENCODER - eTimer module

- The two square wave signals of the incremental encoder, channel A and channel B, are connected to primary and secondary input of channel 2 of ETIMER module
- Channel 2 is in QUADRATURE-COUNT Mode and it provides the decoding of the incremental encoder signals. Channel 2 does not count, but it generates counting pulses (up or down according to the incremental encoder direction) for channel 1 and channel 5.

- Channel 1 and Channel 5 are in CASCADE-COUNT Mode which really count
- If position_acq_pwm is configured as TRUE, the position acquisition is synchronized with PWM period, so, the secondary input of channel 5 is connected to CTU trigger. On each rising edge of CTU trigger, current value of channel 5 is latched. If position_acq_pwm is configured as FALSE, the angle is obtained as the instantaneous value of the timer counter of channel 5 value.
- Channel 0 (configured in Count rising and falling edges of primary source mode) is used to manage the overflow of the encoder reference counter. It is incremented at each overflow of channel 1 to take in account the number of overflow during speed calculation.
- When the index signal is available and enabled, (and after the alignment phase), the class stores the first value of the index counter (first index rotor angle counter); this value is compared with the reference index counter (current index rotor angle counter) which happens once every turn in order to verify the correct operation of the encoder. Moreover, the function Enc_CalcAngle and able to adjust the value of the angle considering the following formula: rotor angle counter = current rotor angle counter + (first index rotor angle counter - current index rotor angle counter).

3.52 ENCODER class exported types

Collaboration diagram for ENCODER class exported types:



Data Structures

- struct [ENCODERParams_t](#)
ENCODER class parameters definition.

Typedefs

- typedef struct CENC_SPD_t * [CENC_SPD](#)
Public ENCODER class definition.

3.52.1 Typedef Documentation

3.52.1.1 [typedef struct CENC_SPD_t* CENC_SPD](#)

Public ENCODER class definition.

Definition at line 62 of file ENCODER_SpeednPosFdbkClass.h.

3.53 ENCODER class exported methods

Collaboration diagram for ENCODER class exported methods:



Functions

- [CENC_SPD Enc_NewObject \(pSpeednPosFdbkParams_t pSpeednPosFdbkParams, pENCODERParams_t pENCODERParams \)](#)

Creates an object of the class ENCODER.

3.53.1 Function Documentation

3.53.1.1 CENC_SPD Enc_NewObject (pSpeednPosFdbkParams_t pSpeednPosFdbkParams, pENCODERParams_t pENCODERParams)

Creates an object of the class ENCODER.

Parameters

in	<i>pSpeednPos-FdbkParams</i>	pointer to an SpeednPosFdbk parameters structure
in	<i>pENCODER-Params</i>	pointer to an ENCODER parameters structure

Returns

CENC_SPD new instance of ENCODER object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Definition at line 191 of file ENCODER_SpeednPosFdbkClass.c.

References _CSPD_t::DerivedClass, ENCODERParams_t::IRQnb, MAX_ENC_SPD_NUM, MC_NULL, _CSPD_t::Methods_str, _DCENC_SPD_t::pDParams_str, and SPD_NewObject().

Here is the call graph for this function:



3.54 HALL Sensor Description

Collaboration diagram for HALL Sensor Description:



HALL sensor implementation using eTimer. A Hall effect sensor is a transducer that varies its output voltage in response to changes in magnetic field density, based on the Hall effect.

A lot of electric motors are equipped with Hall sensors to sense the rotor position. This is a choice to reduce the whole cost with a low resolution sensor. These sensors are mounted on the motor stator sensing the rotor magnetic field and produce a voltage signal proportional to field intensity that decrease when the magnetic poles go away from the sensor. The analogical signal produced by sensor drives a detector of threshold so to obtain a digital signal used to calculate the rotor position.

The recurrence of the voltage signal generated by the Hall sensor in a mechanical cycle of the rotor is the same of the motor pole pairs number and so the mechanical resolution of the Hall sensor improves using motors with more pole pairs while the electrical resolution does not change for each sensor.

Rolling direction identification

The rolling direction is performed by acquiring the three hall sensor outputs. As shown in figure below, it is possible to associate any of Hall sensor output combinations with a state whose number is obtainable by considering H3-H2-H1 as a three-digit binary number (H3 is the most significant bit).

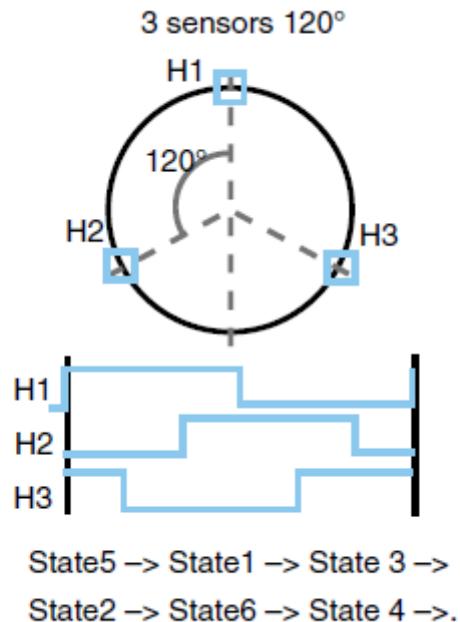


Figure 3.9: Three Hall sensors, output-state correspondence

Consequently, it is possible to reconstruct the rolling direction of the rotor by comparing the present state with

the previous one. In the presence of a positive speed, the sequence must be as illustrated in the previous figure.

Electrical angle extrapolation implementation

The use of FOC algorithm implies the need of a good and constant rotor position accuracy, between two consecutive edges of the hall sensors (which occurs each 60 electrical degrees with a 3 hall sensor spaced 120 degree). So it is necessary to interpolate rotor electrical angle information. For this purpose, the latest available speed measurement is used to predict the angle variation respect to the last edge of the hall sensor signal.

The motor rotor position at time t^* , $e(t^*)$, can be obtained as the difference between the actual time t^* and the instant t when the last edge was captured multiplied by the speed of the motor rotor on a mechanical angle of 360 degree. The following expression is used for the calculation of the motor rotor position at time $e(t^*) = ((t^* - t) / t_0) * 360 + 0$

- e : Electric angle
- t^* : channel 3 counter value in correspondence of CTU trigger
- t : channel x counter value in correspondence of the last edge of one of the 3 hall sensors
- t_0 : is the time to run a mechanical angle of 360
- 0 : is the angle in correspondence of the edge and can be 60-120-180-240-360

Hardware CHALL_SPD peripherals

The CHALL_SPD derived class uses eTimer module hardware peripheral.

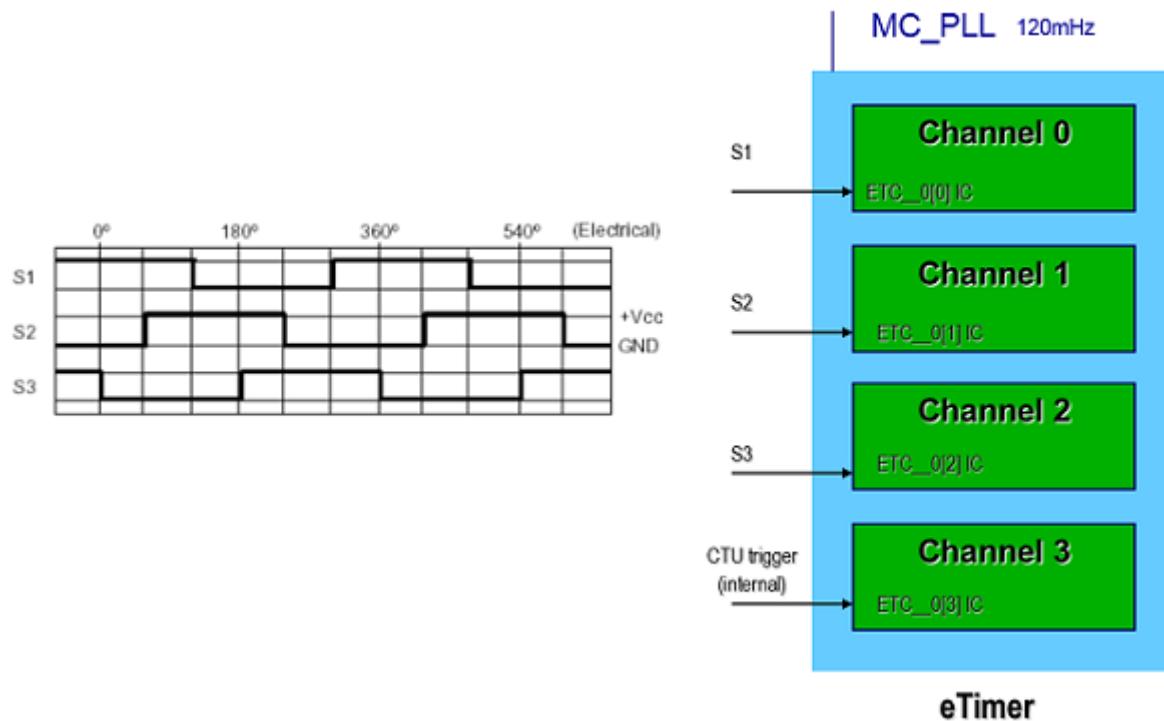


Figure 3.10: eTimer module channels connection to hall sensors

- the four channels are in free running
- the three Hall sensor signals S1, S2, S3 are connected to Channel 0, Channel 1, Channel 2 of eTimer module
- Channel 3 is connected internally to CTU trigger.

Setting up the system when using Hall-effect sensors

Typically, in a three-phase PM motor, three Hall-effect sensors are used to feed back the rotor position information. They are usually mechanically displaced by 120°. The presented firmware library was designed to support 120 degree displacement. As shown in the following figure, the typical waveforms can be visualized at the sensor outputs. More particularly the figure refers to an electrical period (that is, one mechanical revolution, in case of one pole pair motor).

Because the rotor position information they provide is absolute, there is no need for any initial rotor prepositioning. Particular attention must be paid, however, when connecting the sensors to the proper microcontroller inputs. This software library assumes that the positive rolling direction is the rolling direction of a machine that is fed with a three-phase system of positive sequence. In this case, to work correctly, the software library expects the Hall sensor signal transitions to be in the sequence shown in the following figure.

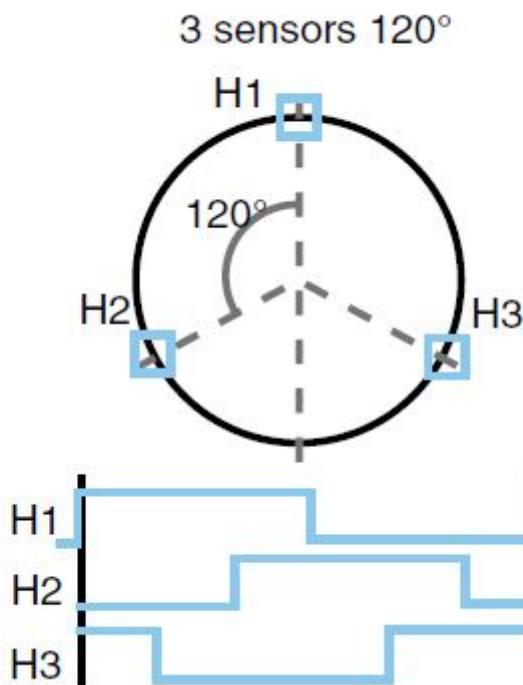


Figure 3.11: Determination of Hall electrical phase shift

For these reasons, it is suggested to follow the instructions given below when connecting a Hall-sensor equipped PM motor to your board:

- Turn the rotor by hand in the direction assumed to be positive and look at the B-emf induced on the three motor phases. If the real neutral point is not available, it can be reconstructed by means of three resistors, for instance.
- Connect the motor phases to the hardware respecting the positive sequence. Let "phase A", "phase B" and "phase C" be the motor phases driven by Flex_PWM1 Flex_PWM2 Flex_PWM3, respectively.
- Turn the rotor by hand in the direction assumed to be positive, look at the three Hall sensor outputs (H1, H2 and H3) and connect them to the selected timer on channels 1, 2 and 3, respectively, making sure that the sequence shown in previous figure is respected.
- Measure the delay in electrical degrees between the maximum of the B-emf induced on phase A and the first rising edge of signal H1.
- Enter two parameters displacement and delay found in the ST MC Workbench GUI, inside the window related to motor speed and position sensor. An example with delay equal to 270 degree is illustrated in figure below

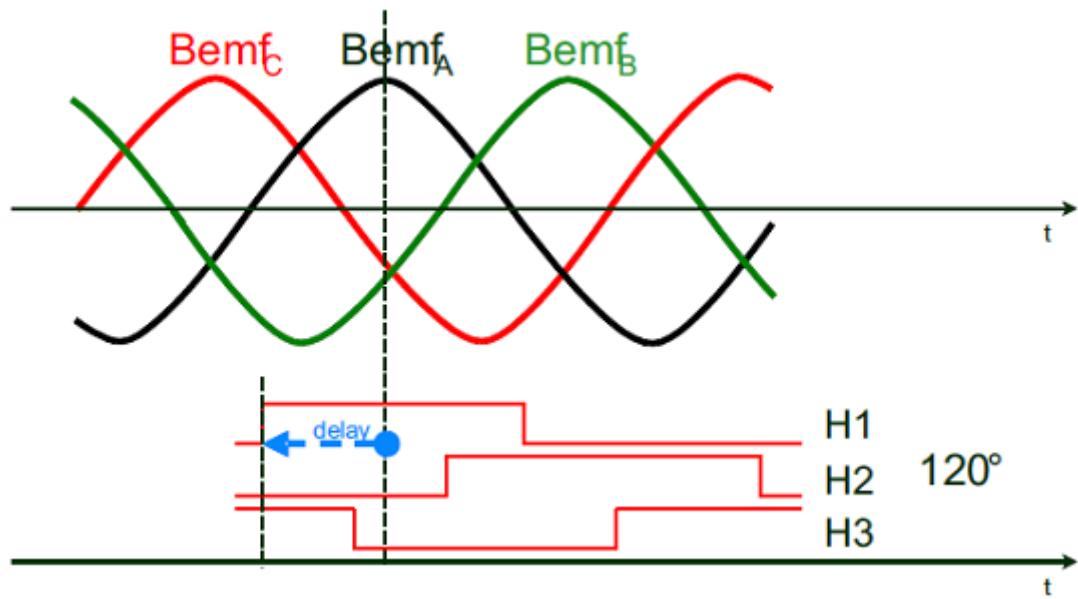


Figure 3.12: Determination of Hall electrical phase shift

3.55 HALL class exported types

Collaboration diagram for HALL class exported types:



Data Structures

- struct [HALLParams_t](#)
HALL class parameters definition.

Typedefs

- typedef struct CHALL_SPD_t * [CHALL_SPD](#)
Public HALL class definition.

3.55.1 Typedef Documentation

3.55.1.1 [typedef struct CHALL_SPD_t* CHALL_SPD](#)

Public HALL class definition.

Definition at line 76 of file HALL_SpeednPosFdbkClass.h.

3.56 HALL class exported methods

Collaboration diagram for HALL class exported methods:



Functions

- **CHALL_SPD HALL_NewObject (pSpeednPosFdbkParams_t pSpeednPosFdbkParams, pHALLParams_t pHALLParams)**

Creates an object of the class HALL.

3.56.1 Function Documentation

3.56.1.1 CHALL_SPD HALL_NewObject (pSpeednPosFdbkParams_t pSpeednPosFdbkParams, pHALLParams_t pHALLParams)

Creates an object of the class HALL.

Parameters

in	<i>pSpeednPosFdbkParams</i>	pointer to an SpeednPosFdbk parameters structure
in	<i>pHALLParams</i>	pointer to an HALL parameters structure

Returns

CHALL_SPD new instance of HALL object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Definition at line 282 of file HALL_SpeednPosFdbkClass.c.

References _CSPD_t::DerivedClass, HALLParams_t::IRQnb, MAX_HALL_SPD_NUM, MC_NULL, _CSPD_t::Methods_str, _DCHALL_SPD_t::pDParams_str, and SPD_NewObject().

Here is the call graph for this function:



3.57 RESOLVER Sensor Description

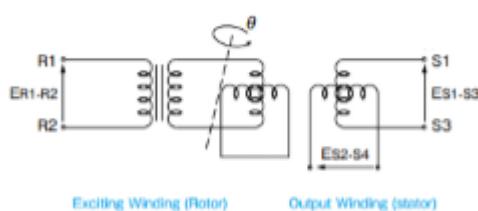
Collaboration diagram for RESOLVER Sensor Description:



RESOLVER sensor implementation. A resolver is an absolute mechanical position sensor used, for example, in Electric Power Steering (EPS). Resolver hardware can be viewed as two inductive position sensors, which, upon a supplied sinusoidal-shaped signal on input, generate two sinusoidal signals on output. The sinusoidal input reference is amplitude modulated with the sine and cosine of the mechanical angle, respectively. These two output signals have to be decoded to obtain the angular position.

The input/output relations of the resolver are expressed by the following equations:

$$\begin{aligned} ER1-R2 &= A \cdot \sin \omega t \\ ES1-S3 &= K \cdot ER1-R2 \cdot \cos \theta \\ ES2-S4 &= K \cdot ER1-R2 \cdot \sin \theta \end{aligned}$$



where K is the transformation ratio, $\omega = 2\pi f$, where f is the frequency [Hz], $ER1-R2$ is the input sinusoidal voltage exciting the resolver, $ES1-S3$ and $ES2-S4$ are the output amplitude-modulated (AM) voltage output of the resolver.

The exciting signal, $ER1-R2$, is generated by a 10 kHz PWM signal, from microcontroller eTimer pin, processed by a RC low pass filter.

The two resolver output signals are sampled by the ADC close to the maximum of the exciting signal in this way obtaining $\sin \theta$ and $\cos \theta$. The user should measure the delay between the PWM signal and the maximum of $\sin \omega t$ from low pass filter. The ADC sampling will be synchronized with the PWM exciting signal plus the delay.

Figure 3.13: Resolver block diagram

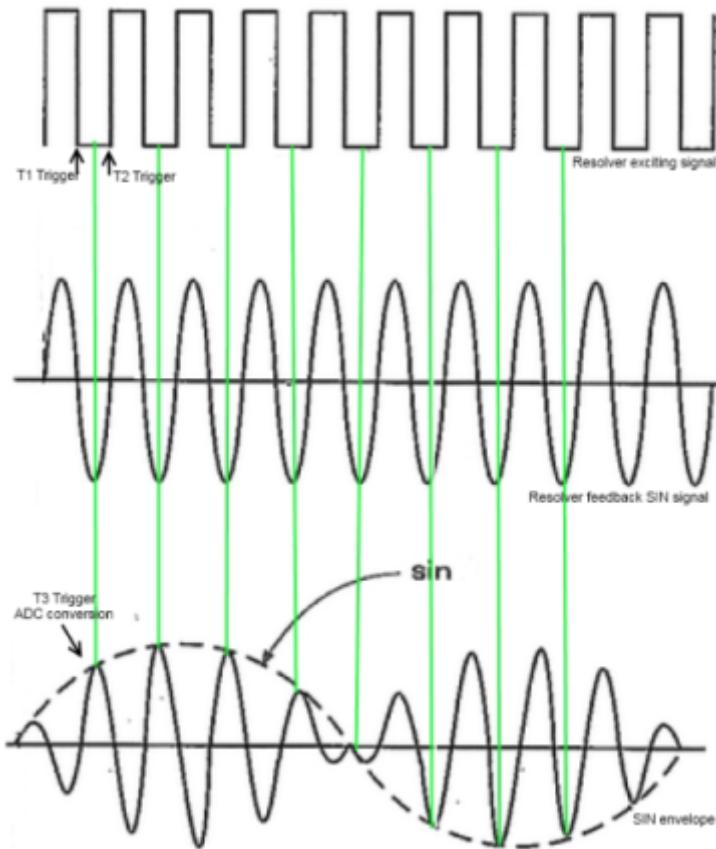


Figure 3.14: Resolver feedback signals

This task is performed by means of Resolver (CRES_SPD) class, and shall be carried out at the first motor startup. The CRES_SPD derived class use the eTimer, ADC and CTU hardware peripherals.

- The Resolver has been implemented inside a system where the PWM duty cycles are updated every two PWM periods. The MRS signal generated from the FlexPWM module is internally routed to the Cross Triggering Unit (CTU) module. The trigger events T1CR and T2CR are used to generate a square wave signal (exciting signal for resolver), synchronized with MRS.
- The eTimer module is used for generation of an exciting signal of resolver. The purpose is to generate a square wave signal with frequency 10kHz (configurable) which is then processed by a hardware low pass filter. Because of the low pass filter, the resulting harmonic signal is phase shifted.
- The CTU Scheduler subUnit (SU) generates the following trigger events:
 1. eTimer channel 1 output:T1CR generates eTimer1 event output, which toggles its output to generate rising edge of resolver exciting signal. T1CR is phase shifted to account for the delay caused by resolver HW circuitry.
 2. eTimer channel 2 output:T2CR generates eTimer1 event output, which toggles its output to the generate falling edge of the resolver exciting signal.
 3. ADC command output: T3CR generates ADC command event for ADC: when a T3CR trigger events occurs, it generates the ADC start of conversion request to sample both resolver signals at point of their maximal amplitude variation.
- For correct calculation, the Resolver measurements of channel A (SIN) and channel B (COS) must be executed at the same time. To do this, the CTU ADC command register is configured in DUAL CONVERSION.

- This sampling/conversion is triggered by the CTU trigger T3CR, which is delayed in time to sample both resolver signals at the point of their maximal amplitude variation, see below figure.

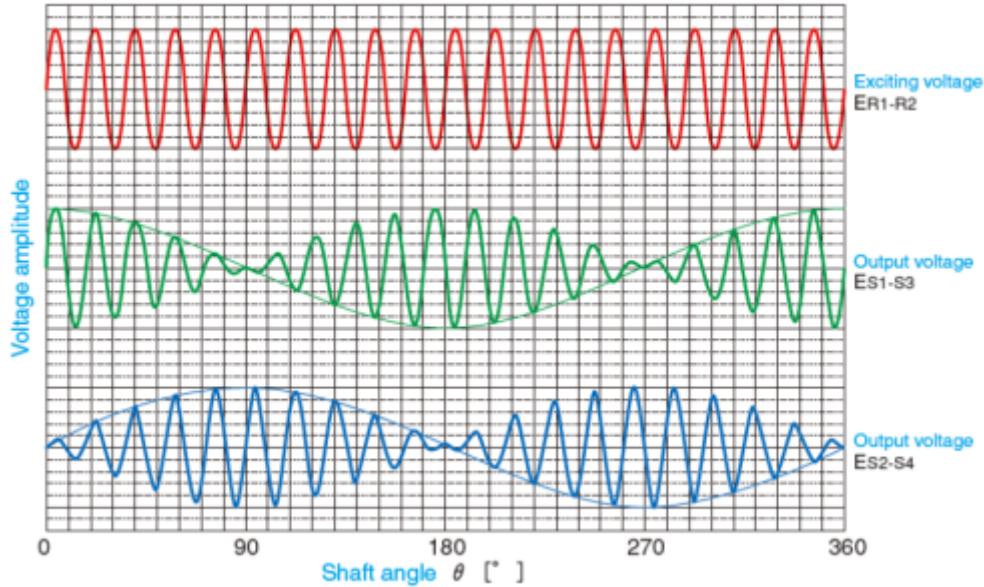


Figure 3.15: Resolver trigger

CTU triggers T1CR and T2CR generate the resolver exciting signal phase shifted to fulfill the following requirements:

- The point where amplitude of both resulting resolver signals varies the most must be as close as possible before phase currents measurement to ensure position and current samples correspond to each other.
- Keep 50% duty-cycle ratio of the resolver exciting signal.

The angular position can be obtain using the following two methods:

- Lookup table
- Arctan

3.58 RESOLVER class exported types

Collaboration diagram for RESOLVER class exported types:



Data Structures

- struct [RESOLVERParams_t](#)
RESOLVER class parameters definition.

Typedefs

- typedef struct CRES_SPD_t * [CRES_SPD](#)
Public RESOLVER class definition.

3.58.1 Typedef Documentation

3.58.1.1 [typedef struct CRES_SPD_t* CRES_SPD](#)

Public RESOLVER class definition.

Definition at line 67 of file RESOLVER_SpeednPosFdbkClass.h.

3.59 RESOLVER class exported methods

Collaboration diagram for RESOLVER class exported methods:



Functions

- **CRES_SPD Res_NewObject (pSpeednPosFdbkParams_t pSpeednPosFdbkParams, pRESOLVERParams_t pRESOLVERParams)**

Creates an object of the class RESOLVER.

3.59.1 Function Documentation

3.59.1.1 **CRES_SPD Res_NewObject (pSpeednPosFdbkParams_t pSpeednPosFdbkParams, pRESOLVERParams_t pRESOLVERParams)**

Creates an object of the class RESOLVER.

Parameters

in	<i>pSpeednPos-FdbkParams</i>	pointer to an SpeednPosFdbk parameters structure
in	<i>pRESOLVER-Params</i>	pointer to an RESOLVER parameters structure

Returns

CENC_SPD new instance of RESOLVER object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Definition at line 245 of file RESOLVER_SpeednPosFdbkClass.c.

References *_CSPD_t::DerivedClass*, *MAX_RES_SPD_NUM*, *MC_NULL*, *_CSPD_t::Methods_str*, *DVars_t::pole_pair_ranges*, and *SPD_NewObject()*.

Here is the call graph for this function:



3.60 SpeednPosFdbk class exported types

Collaboration diagram for SpeednPosFdbk class exported types:



Data Structures

- struct [SpeednPosFdbkParams_t](#)
SpeednPosFdbk class parameters definition.

Typedefs

- typedef struct CSPD_t * [CSPD](#)
Public SpeednPosFdbk class definition.

3.60.1 Typedef Documentation

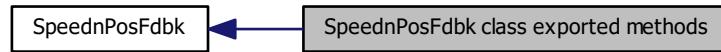
3.60.1.1 [typedef struct CSPD_t* CSPD](#)

Public SpeednPosFdbk class definition.

Definition at line 73 of file SpeednPosFdbkClass.h.

3.61 SpeednPosFdbk class exported methods

Collaboration diagram for SpeednPosFdbk class exported methods:



Functions

- **CSPD SPD_NewObject (pSpeednPosFdbkParams_t pSpeednPosFdbkParams)**
Creates an object of the class SpeednPosFdbk.
- **void SPD_Init (CSPD this)**
Initializes all the object variables and MCU peripherals, usually it has to be called once right after object creation.
- **int16_t SPD_GetElAngle (CSPD this)**
It returns the last computed rotor electrical angle, expressed in s16degrees. 1 s16degree = 360 %65536.
- **int16_t SPD_GetMecAngle (CSPD this)**
It returns the last computed rotor mechanical angle, expressed in s16degrees. Mechanical angle frame is based on parameter bElToMecRatio and, if occasionally provided - through function SPD_SetMecAngle - of a measured mechanical angle, on information computed thereof. Note: both Hall sensor and Sensor-less do not implement either mechanical angle computation or acceleration computation.
- **int16_t SPD_GetAvrgMecSpeed01Hz (CSPD this)**
It returns the last computed average mechanical speed, expressed in 01Hz (tenth of Hertz).
- **int16_t SPD_GetElSpeedDpp (CSPD this)**
It returns the last computed electrical speed, expressed in Dpp. 1 Dpp = 1 s16Degree/control Period. The control period is the period on which the rotor electrical angle is computed (through function SPD_CalcElectricalAngle).
- **void SPD_SetMecAngle (CSPD this, int16_t hMecAngle)**
It could be used to set instantaneous information on rotor mechanical angle. As a consequence, the offset relationship between electrical angle and mechanical angle is computed. Note: both Hall sensor and Sensor-less do not implement either mechanical angle computation or acceleration computation.
- **void SPD_Clear (CSPD this)**
It restore all the functional variables to initial values.
- **bool SPD_Check (CSPD this)**
It returns the result of the last reliability check performed. Reliability is measured with reference to parameters hMaxReliableElSpeed01Hz, hMinReliableElSpeed01Hz, bMaximumSpeedErrorsNumber and/or specific parameters of the derived TRUE = sensor information is reliable FALSE = sensor information is not reliable.
- **int16_t SPD_CalcAngle (CSPD this, const void *pInputVars_str)**
This method must be called with the same periodicity on which FOC is executed. It computes and returns the rotor electrical angle, expressed in s16Degrees. 1 s16Degree = 360 %65536. It computes also the rotor mechanical angle.
- **bool SPD_CalcAvrgMecSpeed01Hz (CSPD this, int16_t *pMecSpeed01Hz)**
This method must be called - at least - with the same periodicity on which speed control is executed. It computes and returns - through parameter pMecSpeed01Hz - the rotor average mechanical speed, expressed in 01Hz. It computes and returns the reliability state of the sensor; reliability is measured with reference to parameters hMaxReliableElSpeed01Hz, hMinReliableElSpeed01Hz, bMaximumSpeedErrorsNumber and/or specific parameters of the derived TRUE = sensor information is reliable FALSE = sensor information is not reliable.
- **int16_t SPD_GetS16Speed (CSPD this)**
This method returns the average mechanical rotor speed expressed in "S16Speed". It means that:

3.61.1 Function Documentation

3.61.1.1 **CSPD SPD_NewObject (pSpeednPosFdbkParams_t pSpeednPosFdbkParams)**

Creates an object of the class SpeednPosFdbk.

Parameters

<i>pSpeednPos-FdbkParams</i>	pointer to an SpeednPosFdbk parameters structure
------------------------------	--

Return values

<i>CSPD</i>	new instance of SpeednPosFdbk object
-------------	--------------------------------------

Parameters

in	<i>pSpeednPos-FdbkParams</i>	pointer to an SpeednPosFdbk parameters structure
----	------------------------------	--

Returns

CSPD new instance of SpeednPosFdbk object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Definition at line 108 of file SpeednPosFdbkClass.c.

References MAX_SPD_NUM, MC_NULL, and _CSPD_t::pParams_str.

Referenced by Enc_NewObject(), HALL_NewObject(), Res_NewObject(), STO_NewObject(), and VSS_NewObject().

3.61.1.2 **void SPD_Init (CSPD this)**

Initializes all the object variables and MCU peripherals, usually it has to be called once right after object creation.

Parameters

<i>this</i>	related object of class CSPD
-------------	------------------------------

Return values

<i>none</i>

Parameters

in	<i>this</i>	related object of class CSPD
----	-------------	------------------------------

Returns

void

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Definition at line 148 of file SpeednPosFdbkClass.c.

3.61.1.3 int16_t SPD_GetElAngle (CSPD *this*)

It returns the last computed rotor electrical angle, expressed in s16degrees. 1 s16degree = 360°/65536.

Parameters

<i>this</i>	related object of class CSPD
-------------	------------------------------

Return values

<i>int16_t</i>	rotor electrical angle (s16degrees)
----------------	-------------------------------------

It returns the last computed rotor electrical angle, expressed in s16degrees. 1 s16degree = 360°/65536.

It returns the last computed rotor electrical angle, expressed in s16degrees. 1 s16degree = 360/65536.

Parameters

in	<i>this</i>	related object of class CSPD
----	-------------	------------------------------

Returns

the rotor electrical angle (s16degrees)

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Definition at line 166 of file SpeednPosFdbkClass.c.

3.61.1.4 int16_t SPD_GetMecAngle (CSPD *this*)

It returns the last computed rotor mechanical angle, expressed in s16degrees. Mechanical angle frame is based on parameter bElToMecRatio and, if occasionally provided - through function SPD_SetMecAngle - of a measured mechanical angle, on information computed thereof. Note: both Hall sensor and Sensor-less do not implement either mechanical angle computation or acceleration computation.

Parameters

<i>this</i>	related object of class CSPD
-------------	------------------------------

Return values

<i>int16_t</i>	rotor mechanical angle (s16degrees)
----------------	-------------------------------------

It returns the last computed rotor mechanical angle, expressed in s16degrees. Mechanical angle frame is based on parameter bElToMecRatio and, if occasionally provided - through function SPD_SetMecAngle - of a measured mechanical angle, on information computed thereof.

Parameters

in	<i>this</i>	related object of class CSPD
----	-------------	------------------------------

Returns

the rotor mechanical angle (s16degrees)

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Definition at line 190 of file SpeednPosFdbkClass.c.

3.61.1.5 int16_t SPD_GetAvrgMecSpeed01Hz (CSPD *this*)

It returns the last computed average mechanical speed, expressed in 01Hz (tenth of Hertz).

Parameters

<i>this</i>	related object of class CSPD
-------------	------------------------------

Return values

<i>int16_t</i>	rotor average mechanical speed (01Hz)
----------------	---------------------------------------

Parameters

in	<i>this</i>	related object of class CSPD
----	-------------	------------------------------

Returns

the rotor average mechanical speed (01Hz)

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Definition at line 210 of file SpeednPosFdbkClass.c.

3.61.1.6 int16_t SPD_GetElSpeedDpp (CSPD *this*)

It returns the last computed electrical speed, expressed in Dpp. 1 Dpp = 1 s16Degree/control Period. The control period is the period on which the rotor electrical angle is computed (through function SPD_CalcElectricalAngle).

Parameters

<i>this</i>	related object of class CSPD
-------------	------------------------------

Return values

<i>int16_t</i>	rotor electrical speed (Dpp)
----------------	------------------------------

Parameters

in	<i>this</i>	related object of class CSPD
----	-------------	------------------------------

Returns

the rotor electrical speed (Dpp)

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Definition at line 232 of file SpeednPosFdbkClass.c.

3.61.1.7 void SPD_SetMecAngle (*CSPD this*, int16_t *hMecAngle*)

It could be used to set instantaneous information on rotor mechanical angle. As a consequence, the offset relationship between electrical angle and mechanical angle is computed. Note: both Hall sensor and Sensor-less do not implement either mechanical angle computation or acceleration computation.

Parameters

<i>this</i>	related object of class CSPD
<i>hMecAngle</i>	instantaneous measure of rotor mechanical angle (s16degrees)

Return values

<i>none</i>

Parameters

in	<i>this</i>	related object of class CSPD
in	<i>hMecAngle</i>	instantaneous measure of rotor mechanical angle (s16degrees)

Returns

void

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Definition at line 256 of file SpeednPosFdbkClass.c.

Referenced by EAC_Exec(), EAC_StartAlignment(), and RUC_Clear().

3.61.1.8 void SPD_Clear (*CSPD this*)

It restore all the functional variables to initial values.

Parameters

<i>this</i>	related object of class CSPD
-------------	------------------------------

Return values

<i>none</i>

Parameters

in	<i>this</i>	related object of class CSPD
----	-------------	------------------------------

Returns

void

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Definition at line 274 of file SpeednPosFdbkClass.c.

Referenced by RUC_Clear().

3.61.1.9 bool SPD_Check (CSPD *this*)

It returns the result of the last reliability check performed. Reliability is measured with reference to parameters h-MaxReliableEISpeed01Hz, hMinReliableEISpeed01Hz, bMaximumSpeedErrorsNumber and/or specific parameters of the derived TRUE = sensor information is reliable FALSE = sensor information is not reliable.

Parameters

<i>this</i>	related object of class CSPD
-------------	------------------------------

Return values

<i>bool</i>	sensor reliability state
-------------	--------------------------

It returns the result of the last reliability check performed. Reliability is measured with reference to parameters h-MaxReliableEISpeed01Hz, hMinReliableEISpeed01Hz, bMaximumSpeedErrorsNumber and/or specific parameters of the derived TRUE = sensor information is reliable FALSE = sensor information is not reliable.

Parameters

in	<i>this</i>	related object of class CSPD
----	-------------	------------------------------

Returns

bool sensor reliability state

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Definition at line 301 of file SpeednPosFdbkClass.c.

3.61.1.10 int16_t SPD_CalcAngle (CSPD *this*, const void * *pInputVars_str*)

This method must be called with the same periodicity on which FOC is executed. It computes and returns the rotor electrical angle, expressed in s16Degrees. 1 s16Degree = 360/65536. It computes also the rotor mechanical angle.

Parameters

<i>this</i>	related object of class CSPD
<i>pInputVars_str</i>	pointer to input structure. For derived class STO input structure type is Observer_Inputs_t , for HALL and ENCODER this parameter will not be used (thus it can be equal to MC_NULL).

Return values

<i>int16_t</i>	rotor electrical angle (s16Degrees)
----------------	-------------------------------------

This method must be called with the same periodicity on which FOC is executed. It computes and returns the rotor electrical angle, expressed in s16Degrees. 1 s16Degree = 360/65536. It computes also the rotor mechanical angle.

Parameters

in	<i>this</i>	related object of class CSPD
in	<i>pInputVars_str</i>	pointer to input structure. For derived class STO input structure type is Observer_Inputs_t , for HALL and ENCODER this parameter will not be used (thus it can be equal to MC_NULL).

Returns

the rotor electrical angle (s16Degrees)

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Definition at line 334 of file SpeednPosFdbkClass.c.

Referenced by TSK_HighFrequencyTask().

3.61.1.11 bool SPD_CalcAvrgMecSpeed01Hz (**CSPD this**, int16_t * *pMecSpeed01Hz*)

This method must be called - at least - with the same periodicity on which speed control is executed. It computes and returns - through parameter *pMecSpeed01Hz* - the rotor average mechanical speed, expressed in 01Hz. It computes and returns the reliability state of the sensor; reliability is measured with reference to parameters *hMaxReliableElSpeed01Hz*, *hMinReliableElSpeed01Hz*, *bMaximumSpeedErrorsNumber* and/or specific parameters of the derived TRUE = sensor information is reliable FALSE = sensor information is not reliable.

Parameters

<i>this</i>	related object of class CSPD
<i>pMecSpeed01-Hz</i>	pointer to int16_t, used to return the rotor average mechanical speed (01Hz)

Return values

<i>none</i>	This method must be called - at least - with the same periodicity on which speed control is executed. It computes and returns - through parameter <i>pMecSpeed01Hz</i> - the rotor average mechanical speed, expressed in 01Hz. It computes and returns the reliability state of the sensor; reliability is measured with reference to parameters <i>hMaxReliableElSpeed01Hz</i> , <i>hMinReliableElSpeed01Hz</i> , <i>bMaximumSpeedErrorsNumber</i> and/or specific parameters of the derived TRUE = sensor information is reliable FALSE = sensor information is not reliable.
-------------	--

Parameters

in	<i>this</i>	related object of class CSPD
in	<i>pMecSpeed01-Hz</i>	pointer to int16_t, used to return the rotor average mechanical speed (01Hz)

Returns

bool sensor information is reliable

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Definition at line 361 of file SpeednPosFdbkClass.c.

3.61.1.12 int16_t SPD_GetS16Speed (**CSPD this**)

This method returns the average mechanical rotor speed expressed in "S16Speed". It means that:

- it is zero for zero speed,\n

- it become S16_MAX when the average mechanical speed is equal to hMaxReliableMecSpeed01Hz,\n
- it becomes -S16_MAX when the average mechanical speed is equal to -hMaxReliableMecSpeed01Hz.

Parameters

<i>this</i>	related object of class CSPD
-------------	------------------------------

Return values

<i>int16_t</i>	The average mechanical rotor speed expressed in "S16Speed".
----------------	---

This method returns the average mechanical rotor speed expressed in "S16Speed". It means that:

.

This method returns the average meccanical rotor speed expressed in "S16Speed". It means that:

.

- it is zero for zero speed,
- it become S16_MAX when the average mechanical speed is equal to hMaxReliableMecSpeed01Hz,
- it becomes -S16_MAX when the average mechanical speed is equal to -hMaxReliableMecSpeed01Hz.

Parameters

in	<i>this</i>	related object of class CSPD
----	-------------	------------------------------

Returns

The average mechanical rotor speed expressed in "S16Speed".

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Definition at line 479 of file SpeednPosFdbkClass.c.

3.62 STO class exported types

Collaboration diagram for STO class exported types:



Data Structures

- struct [Observer_Inputs_t](#)
STO_SPD derived class input structure type definition for SPD_CalcAngle.
- struct [STOParams_t](#)
STO class parameters definition.

Typedefs

- [typedef struct CSTO_SPD_t * CSTO_SPD](#)
Public STO class definition.

3.62.1 Typedef Documentation

3.62.1.1 [typedef struct CSTO_SPD_t * CSTO_SPD](#)

Public STO class definition.

Definition at line 59 of file [STO_SpeednPosFdbkClass.h](#).

3.63 STO class exported methods

Collaboration diagram for STO class exported methods:



Functions

- **CSTO_SPD STO_NewObject (pSpeednPosFdbkParams_t pSpeednPosFdbkParams, pSTOParams_t pSTOParams)**
Creates an object of the class STO.
- **bool STO_IsObserverConverged (CSTO_SPD this, int16_t hForcedMecSpeed01Hz)**
It internally performs a set of checks necessary to state whether the state observer algorithm converged. To be periodically called during motor open-loop ramp-up (e.g. at the same frequency of SPD_CalcElAngle), it returns TRUE if the estimated angle and speed can be considered reliable, FALSE otherwise.
- **Volt_Components STO_GetEstimatedBemf (CSTO_SPD this)**
It exports estimated Bemf alpha-beta in Volt_Components format.
- **Curr_Components STO_GetEstimatedCurrent (CSTO_SPD this)**
It exports the stator current alpha-beta as estimated by state observer.
- **void STO_GetObserverGains (CSTO_SPD this, int16_t *pC2, int16_t *pC4)**
It exports current observer gains through parameters hC2 and hC4.
- **void STO_SetObserverGains (CSTO_SPD this, int16_t hC1, int16_t hC2)**
It allows setting new values for observer gains.
- **void STO_GetPLLGains (CSTO_SPD this, int16_t *pPgain, int16_t *plgain)**
It exports current PLL gains through parameters pPgain and plgain.
- **void STO_SetPLLGains (CSTO_SPD this, int16_t hPgain, int16_t hlgain)**
It allows setting new values for PLL gains.
- **void STO_ResetPLL (CSTO_SPD this)**
It resets integral term of PLL.
- **void STO_SetPLL (CSTO_SPD this, int16_t hElSpeedDpp, int16_t hElAngle)**
It sends locking info for PLL.
- **void STO_CalcAvgElSpeedDpp (CSTO_SPD this)**
This method must be called - at least - with the same periodicity on which speed control is executed. It computes and update the estimated average electrical speed - expressed in dpp - used for instance in observer equations. Average is computed considering a FIFO depth equal to bSpeedBufferSizedpp.
- **int32_t STO_GetEstimatedBemfLevel (CSTO_SPD this)**
It exports estimated Bemf squared level.
- **int32_t STO_GetObservedBemfLevel (CSTO_SPD this)**
It exports observed Bemf squared level.
- **void STO_BemfConsistencyCheckSwitch (CSTO_SPD this, bool bSel)**
It enables/disables the bemf consistency check.
- **bool STO_IsBemfConsistent (CSTO_SPD this)**
It returns the result of the Bemf consistency check.

3.63.1 Function Documentation

3.63.1.1 **CSTO_SPD STO_NewObject (pSpeednPosFdbkParams_t pSpeednPosFdbkParams, pSTOParams_t pSTOParams)**

Creates an object of the class STO.

Parameters

<i>pSpeednPos-FdbkParams</i>	pointer to an SpeednPosFdbk parameters structure
<i>pSTOParams</i>	pointer to an STO parameters structure

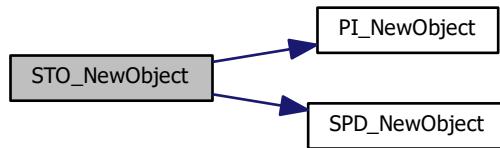
Return values

<i>CSTO_SPD</i>	new instance of STO object
-----------------	----------------------------

Definition at line 95 of file STO_SpeednPosFdbkClass.c.

References *_CSPD_t::DerivedClass*, *MAX_STO_SPD_NUM*, *MC_NULL*, *_CSPD_t::Methods_str*, *_DCSTO_SPD_t::pDParams_str*, *PI_NewObject()*, and *SPD_NewObject()*.

Here is the call graph for this function:



3.63.1.2 **bool STO_IsObserverConverged (CSTO_SPD this, int16_t hForcedMecSpeed01Hz)**

It internally performs a set of checks necessary to state whether the state observer algorithm converged. To be periodically called during motor open-loop ramp-up (e.g. at the same frequency of *SPD_CalcElAngle*), it returns TRUE if the estimated angle and speed can be considered reliable, FALSE otherwise.

Parameters

<i>this</i>	related object of class CSTO_SPD
<i>hForcedMec-Speed01Hz</i>	Mechanical speed in 0.1Hz unit as forced by VSS

Return values

<i>bool</i>	sensor reliability state
-------------	--------------------------

It internally performs a set of checks necessary to state whether the state observer algorithm converged. To be periodically called during motor open-loop ramp-up (e.g. at the same frequency of *SPD_CalcElAngle*), it returns TRUE if the estimated angle and speed can be considered reliable, FALSE otherwise.

Parameters

<i>this</i>	related object of class CSTO_SPD
<i>hForcedMec-Speed01Hz</i>	Mechanical speed in 0.1Hz unit as forced by VSS

Return values

<i>bool</i>	sensor reliability state
-------------	--------------------------

Definition at line 700 of file STO_SpeednPosFdbkClass.c.

References DVars_t::bConsistencyCounter, DVars_t::blsAlgorithmConverged, and DVars_t::blsSpeedReliable.

3.63.1.3 Volt_Components STO_GetEstimatedBemf (CSTO_SPD *this*)

It exports estimated Bemf alpha-beta in [Volt_Components](#) format.

Parameters

<i>this</i>	related object of class CSTO_SPD
-------------	----------------------------------

Return values

Volt_Components	Bemf alpha-beta
---------------------------------	-----------------

Definition at line 782 of file STO_SpeednPosFdbkClass.c.

3.63.1.4 Curr_Components STO_GetEstimatedCurrent (CSTO_SPD *this*)

It exports the stator current alpha-beta as estimated by state observer.

Parameters

<i>this</i>	related object of class CSTO_SPD
-------------	----------------------------------

Return values

Curr_Components	State observer estimated stator current lalpha-beta
---------------------------------	---

Definition at line 797 of file STO_SpeednPosFdbkClass.c.

References DVars_t::wlalpha_est, and DVars_t::wlbeta_est.

3.63.1.5 void STO_GetObserverGains (CSTO_SPD *this*, int16_t * *pC2*, int16_t * *pC4*)

It exports current observer gains through parameters hC2 and hC4.

Parameters

<i>this</i>	related object of class CSTO_SPD
<i>pC2</i>	pointer to int16_t used to return parameters hC2
<i>pC4</i>	pointer to int16_t used to return parameters hC4

Return values

<i>none</i>	
-------------	--

Definition at line 817 of file STO_SpeednPosFdbkClass.c.

References DVars_t::hC2, and DVars_t::hC4.

3.63.1.6 void STO_SetObserverGains (CSTO_SPD *this*, int16_t *hC1*, int16_t *hC2*)

It allows setting new values for observer gains.

Parameters

<i>this</i>	related object of class CSTO_SPD
<i>hC1</i>	new value for observer gain hC1
<i>hC2</i>	new value for observer gain hC2

Return values

<i>none</i>

Parameters

<i>this</i>	related object of class CSTO_SPD
<i>wK1</i>	new value for observer gain hC1
<i>wK2</i>	new value for observer gain hC2

Return values

<i>none</i>

Definition at line 832 of file STO_SpeednPosFdbkClass.c.

References DVars_t::hC2, and DVars_t::hC4.

3.63.1.7 void STO_GetPLLGains (CSTO_SPD *this*, int16_t * *pPgain*, int16_t * *plgain*)

It exports current PLL gains through parameters pPgain and plgain.

Parameters

<i>this</i>	related object of class CSTO_SPD
<i>pPgain</i>	pointer to int16_t used to return PLL proportional gain
<i>plgain</i>	pointer to int16_t used to return PLL integral gain

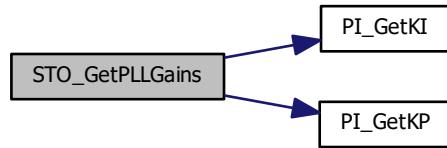
Return values

<i>none</i>

Definition at line 846 of file STO_SpeednPosFdbkClass.c.

References DVars_t::oPIRegulator, PI_GetKI(), and PI_GetKP().

Here is the call graph for this function:



3.63.1.8 void STO_SetPLLGains (CSTO_SPD this, int16_t hPgain, int16_t hIgain)

It allows setting new values for PLL gains.

Parameters

<i>this</i>	related object of class CSTO_SPD
<i>hPgain</i>	new value for PLL proportional gain
<i>hIgain</i>	new value for PLL integral gain

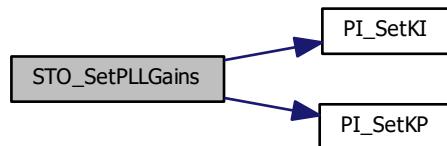
Return values

<i>none</i>

Definition at line 861 of file STO_SpeednPosFdbkClass.c.

References DVars_t::oPIRegulator, PI_SetKI(), and PI_SetKP().

Here is the call graph for this function:



3.63.1.9 void STO_ResetPLL (CSTO_SPD this)

It resets integral term of PLL.

Parameters

<i>this</i>	related object of class CSTO_SPD
-------------	----------------------------------

Return values

<i>none</i>

Definition at line 894 of file STO_SpeednPosFdbkClass.c.

References PI_SetIntegralTerm().

Here is the call graph for this function:



3.63.1.10 void STO_SetPLL (CSTO_SPD this, int16_t hEISpeedDpp, int16_t hEIAngle)

It sends locking info for PLL.

Parameters

<i>this</i>	related object of class CSTO_SPD
-------------	----------------------------------

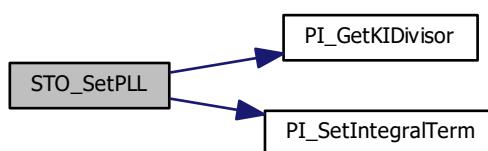
Return values

<i>none</i>

Definition at line 904 of file STO_SpeednPosFdbkClass.c.

References PI_GetKIDivisor(), and PI_SetIntegralTerm().

Here is the call graph for this function:



3.63.1.11 void STO_CalcAvrgEISpeedDpp (CSTO_SPD this)

This method must be called - at least - with the same periodicity on which speed control is executed. It computes and update the estimated average electrical speed - expressed in dpp - used for instance in observer equations. Average is computed considering a FIFO depth equal to bSpeedBufferSizedpp.

Parameters

<i>this</i>	related object of class CSPD
-------------	------------------------------

Return values

<i>none</i>	This method must be called - at least - with the same periodicity on which speed control is executed. It computes and update the estimated average electrical speed - expressed in dpp - used for instance in observer equations. Average is computed considering a FIFO depth equal to bSpeedBufferSizedpp.
-------------	--

Parameters

<i>this</i>	related object of class CSPD
-------------	------------------------------

Return values

<i>none</i>	
-------------	--

Definition at line 545 of file STO_SpeednPosFdbkClass.c.

References DVars_t::bSpeed_Buffer_Index, DVars_t::hSpeed_Buffer, DVars_t::hSpeedBufferOldestEl, and DVars_t::wDppBufferSum.

Referenced by TSK_HighFrequencyTask().

3.63.1.12 int32_t STO_GetEstimatedBemfLevel (CSTO_SPD *this*)

It exports estimated Bemf squared level.

Parameters

<i>this</i>	related object of class CSTO_SPD
-------------	----------------------------------

Return values

<i>int32_t</i>	
----------------	--

Definition at line 916 of file STO_SpeednPosFdbkClass.c.

3.63.1.13 int32_t STO_GetObservedBemfLevel (CSTO_SPD *this*)

It exports observed Bemf squared level.

Parameters

<i>this</i>	related object of class CSTO_SPD
-------------	----------------------------------

Return values

<i>int32_t</i>	
----------------	--

Definition at line 926 of file STO_SpeednPosFdbkClass.c.

3.63.1.14 void STO_BemfConsistencyCheckSwitch (CSTO_SPD *this*, bool *bSel*)

It enables/disables the bemf consistency check.

Parameters

<i>this</i>	related object of class CSTO_SPD
<i>bSel</i>	boolean; TRUE enables check; FALSE disables check

Return values

<i>int32_t</i>

Parameters

<i>this</i>	related object of class CSTO_SPD
<i>bSel</i>	boolean; true enables check; false disables check

Definition at line 936 of file STO_SpeednPosFdbkClass.c.

3.63.1.15 bool STO_IsBemfConsistent(CSTO_SPD *this*)

It returns the result of the Bemf consistency check.

Parameters

<i>this</i>	related object of class CSTO_SPD
-------------	----------------------------------

Return values

<i>bool</i>	Bemf consistency state
-------------	------------------------

Definition at line 946 of file STO_SpeednPosFdbkClass.c.

3.64 VirtualSpeedSensor class exported types

Collaboration diagram for VirtualSpeedSensor class exported types:



Data Structures

- struct [VirtualSpeedSensorParams_t](#)
VirtualSpeedSensor class parameters definition.

TypeDefs

- typedef struct CVSS_SPD_t * [CVSS_SPD](#)
Public VirtualSpeedSensor class definition.

3.64.1 TypeDef Documentation

3.64.1.1 [typedef struct CVSS_SPD_t* CVSS_SPD](#)

Public VirtualSpeedSensor class definition.

Definition at line 49 of file VirtualSpeedSensor_SpeednPosFdbkClass.h.

3.65 VirtualSpeedSensor class exported methods

Collaboration diagram for VirtualSpeedSensor class exported methods:



Functions

- **[CVSS_SPD VSS_NewObject](#)** (`pSpeednPosFdbkParams_t pSpeednPosFdbkParams, pVirtualSpeedSensorParams_t pVirtualSpeedSensorParams`)

Creates an object of the class VirtualSpeedSensor.
- **[VSPD_SetMecAcceleration](#)** (`CSPD this, int16_t hFinalMecSpeed01Hz, uint16_t hDurationms`)

Set the mechanical acceleration of virtual sensor. This acceleration is defined starting from current mechanical speed, final mechanical speed expressed in 0.1Hz and duration expressed in milliseconds.
- **[int16_t VSPD_GetLastRampFinalSpeed](#)** (`CSPD this`)

Get the final speed of last settled ramp of virtual sensor expressed in 0.1Hz.
- **[bool VSPD_SetStartTransition](#)** (`CSPD this, bool bCommand`)

Set the command to Start the transition phase from CVSS_SPD to other CSPD. Transition is to be considered ended when Sensor information is declared 'Reliable' or if function returned value is FALSE.

3.65.1 Function Documentation

3.65.1.1 **[CVSS_SPD VSS_NewObject](#)** (`pSpeednPosFdbkParams_t pSpeednPosFdbkParams, pVirtualSpeedSensorParams_t pVirtualSpeedSensorParams`)

Creates an object of the class VirtualSpeedSensor.

Parameters

<code>pSpeednPos-FdbkParams</code>	pointer to an SpeednPosFdbk parameters structure
<code>pVirtualSpeed-SensorParams</code>	pointer to an VirtualSpeedSensor parameters structure

Return values

<code>CVSS_SPD</code>	new instance of VirtualSpeedSensor object
-----------------------	---

Definition at line 63 of file VirtualSpeedSensor_SpeednPosFdbkClass.c.

References `_CSPD_t::DerivedClass`, `MAX_VSS_SPD_NUM`, `MC_NULL`, `_CSPD_t::Methods_str`, `_DCVSS_SPD_t::pDParams_str`, and `SPD_NewObject()`.

Here is the call graph for this function:



3.65.1.2 void VSPD_SetMecAcceleration (CSPD this, int16_t hFinalMecSpeed01Hz, uint16_t hDurationms)

Set the mechanical acceleration of virtual sensor. This acceleration is defined starting from current mechanical speed, final mechanical speed expressed in 0.1Hz and duration expressed in milliseconds.

Parameters

<i>this</i>	related object of class CSTC.
<i>hFinalMecSpeed01Hz</i>	mechanical speed expressed in 0.1Hz assumed by the virtual sensor at the end of the duration.
<i>hDurationms</i>	Duration expressed in ms. It can be 0 to apply instantaneous the final speed.

Return values

<i>none</i>

Definition at line 317 of file VirtualSpeedSensor_SpeednPosFdbkClass.c.

References DVars_t::bTransitionStarted, DVars_t::hFinalMecSpeed01Hz, DVars_t::hRemainingStep, DVars_t::wElAccDppP32, and DVars_t::wElSpeedDpp32.

3.65.1.3 int16.t VSPD_GetLastRampFinalSpeed (CSPD this)

Get the final speed of last settled ramp of virtual sensor expressed in 0.1Hz.

Parameters

<i>this</i>	related object of class CSTC.
<i>hFinalMecSpeed01Hz</i>	mechanical speed expressed in 0.1Hz assumed by the virtual sensor at the end of the duration.
<i>hDurationms</i>	Duration expressed in ms. It can be 0 to apply instantaneous the final speed.

Return values

<i>none</i>

Definition at line 385 of file VirtualSpeedSensor_SpeednPosFdbkClass.c.

References DVars_t::hFinalMecSpeed01Hz.

3.65.1.4 bool VSPD_SetStartTransition (CSPD this, bool bCommand)

Set the command to Start the transition phase from CVSS_SPD to other CSPD. Transition is to be considered ended when Sensor information is declared 'Reliable' or if function returned value is FALSE.

Parameters

<i>this</i>	related object of class CSPD.
<i>bool</i>	TRUE to Start the transition phase, FALSE has no effect

Return values

<i>bool</i>	TRUE if Transition phase is enabled (started or not), FALSE if transition has been triggered but it's actually disabled (parameter hTransitionSteps = 0)
-------------	--

Set the command to Start the transition phase from CVSS_SPD to other CSPD. Transition is to be considered ended when Sensor information is declared 'Reliable' or if function returned value is FALSE.

Parameters

<i>this</i>	related object of class CSPD.
<i>bool</i>	true to Start the transition phase, false has no effect

Return values

<i>bool</i>	true if Transition phase is enabled (started or not), false if transition has been triggered but it's actually disabled (parameter hTransitionSteps = 0)
-------------	--

Definition at line 401 of file VirtualSpeedSensor_SpeednPosFdbkClass.c.

3.66 VirtualSpeedSensor class private methods

Collaboration diagram for VirtualSpeedSensor class private methods:



Functions

- void [VSPD_SetMecAcceleration \(CSPD this, int16_t hFinalMecSpeed01Hz, uint16_t hDurationms\)](#)

Set the mechanical acceleration of virtual sensor. This acceleration is defined starting from current mechanical speed, final mechanical speed expressed in 0.1Hz and duration expressed in milliseconds.
- int16_t [VSPD_GetLastRampFinalSpeed \(CSPD this\)](#)

Get the final speed of last settled ramp of virtual sensor expressed in 0.1Hz.
- bool [VSPD_SetStartTransition \(CSPD this, bool bCommand\)](#)

Set the command to Start the transition phase from CVSS_SPD to other CSPD. Transition is to be considered ended when Sensor information is declared 'Reliable' or if function returned value is false.

3.66.1 Function Documentation

3.66.1.1 void VSPD_SetMecAcceleration (CSPD this, int16_t hFinalMecSpeed01Hz, uint16_t hDurationms)

Set the mechanical acceleration of virtual sensor. This acceleration is defined starting from current mechanical speed, final mechanical speed expressed in 0.1Hz and duration expressed in milliseconds.

Parameters

<i>this</i>	related object of class CSTC.
<i>hFinalMecSpeed01Hz</i>	mechanical speed expressed in 0.1Hz assumed by the virtual sensor at the end of the duration.
<i>hDurationms</i>	Duration expressed in ms. It can be 0 to apply instantaneous the final speed.

Return values

<i>none</i>

Definition at line 317 of file VirtualSpeedSensor_SpeednPosFdbkClass.c.

References DVars_t::bTransitionStarted, DVars_t::hFinalMecSpeed01Hz, DVars_t::hRemainingStep, DVars_t::wElAccDppP32, and DVars_t::wElSpeedDpp32.

Referenced by EAC_StartAlignment(), RUC_Clear(), and RUC_Exec().

3.66.1.2 int16_t VSPD_GetLastRampFinalSpeed (CSPD this)

Get the final speed of last settled ramp of virtual sensor expressed in 0.1Hz.

Parameters

<i>this</i>	related object of class CSTC.
-------------	-------------------------------

<i>hFinalMec-Speed01Hz</i>	mechanical speed expressed in 0.1Hz assumed by the virtual sensor at the end of the duration.
<i>hDurationms</i>	Duration expressed in ms. It can be 0 to apply instantaneous the final speed.

Return values

<i>none</i>

Definition at line 385 of file VirtualSpeedSensor_SpeednPosFdbkClass.c.

References DVars_t::hFinalMecSpeed01Hz.

3.66.1.3 bool VSPD_SetStartTransition (CSPD *this*, bool *bCommand*)

Set the command to Start the transition phase from CVSS_SPD to other CSPD. Transition is to be considered ended when Sensor information is declared 'Reliable' or if function returned value is false.

Set the command to Start the transition phase from CVSS_SPD to other CSPD. Transition is to be considered ended when Sensor information is declared 'Reliable' or if function returned value is FALSE.

Parameters

<i>this</i>	related object of class CSPD.
<i>bool</i>	true to Start the transition phase, false has no effect

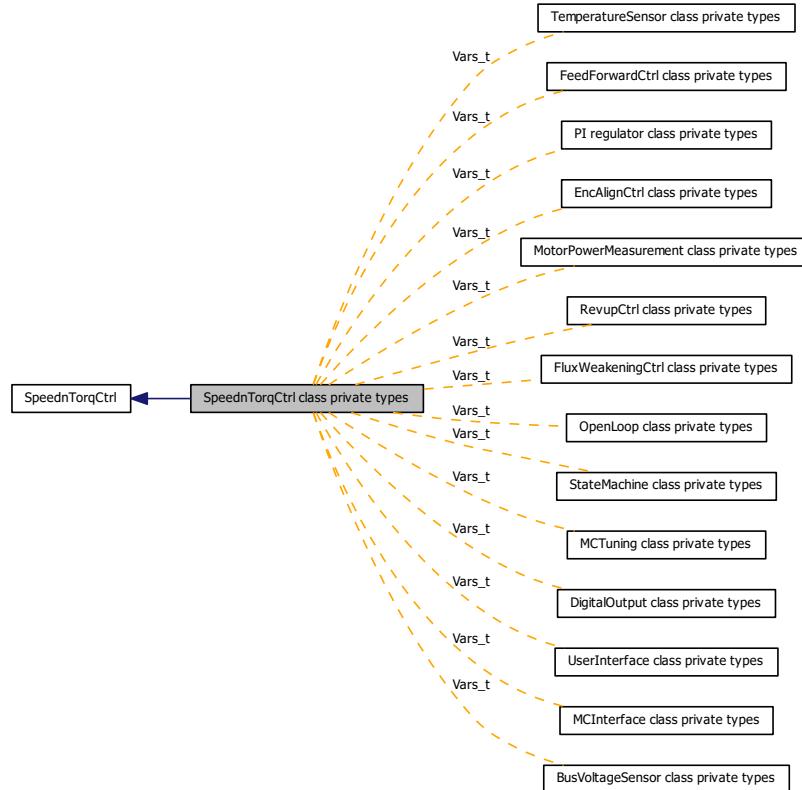
Return values

<i>bool</i>	true if Transition phase is enabled (started or not), false if transition has been triggered but it's actually disabled (parameter hTransitionSteps = 0)
-------------	--

Definition at line 401 of file VirtualSpeedSensor_SpeednPosFdbkClass.c.

3.67 SpeednTorqCtrl class private types

Collaboration diagram for SpeednTorqCtrl class private types:



Data Structures

- struct [Vars_t](#)
MCInterface class members definition.
- struct [_CSTC_t](#)
Private SpeednTorqCtrl class definition.

Typedefs

- [typedef SpeednTorqCtrlParams_t Params_t](#)
Redefinition of parameter structure.

3.67.1 Typedef Documentation

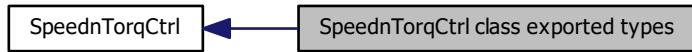
3.67.1.1 [typedef SpeednTorqCtrlParams_t Params_t](#)

Redefinition of parameter structure.

Definition at line 77 of file SpeednTorqCtrlPrivate.h.

3.68 SpeednTorqCtrl class exported types

Collaboration diagram for SpeednTorqCtrl class exported types:



Data Structures

- struct [SpeednTorqCtrlParams_t](#)
SpeednTorqCtrl class parameters definition.

Typedefs

- typedef struct CSTC_t * [CSTC](#)
Public SpeednTorqCtrl class definition.

3.68.1 Typedef Documentation

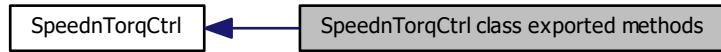
3.68.1.1 [typedef struct CSTC_t* CSTC](#)

Public SpeednTorqCtrl class definition.

Definition at line 54 of file SpeednTorqCtrlClass.h.

3.69 SpeednTorqCtrl class exported methods

Collaboration diagram for SpeednTorqCtrl class exported methods:



Functions

- **CSTC STC_NewObject (pSpeednTorqCtrlParams_t pSpeednTorqCtrlParams)**
Creates an object of the class SpeednTorqCtrl.
- void **STC_Init (CSTC this, CPI oPI, CSPD oSPD)**
Initializes all the object variables, usually it has to be called once right after object creation.
- void **STC_Clear (CSTC this)**
It should be called before each motor restart. If STC is set in speed mode, this method resets the integral term of speed regulator.
- int16_t **STC_GetMecSpeedRef01Hz (CSTC this)**
Get the current mechanical rotor speed reference expressed in tenths of HZ.
- int16_t **STC_GetTorqueRef (CSTC this)**
*Get the current motor torque reference. This value represents actually the Iq current reference expressed in digit. To convert current expressed in digit to current expressed in Amps is possible to use the formula: Current(Amp) = [Current(digit) * Vdd micro] / [65536 * Rshunt * Aop].*
- void **STC_SetControlMode (CSTC this, STC_Modality_t bMode)**
Set the modality of the speed and torque controller. Two modality are available Torque mode and Speed mode. In Torque mode is possible to set directly the motor torque reference or execute a motor torque ramp. This value represents actually the Iq current reference expressed in digit. In Speed mode is possible to set the mechanical rotor speed reference or execute a speed ramp. The required motor torque is automatically calculated by the STC. This command interrupts the execution of any previous ramp command maintaining the last value of Iq.
- **STC_Modality_t STC_GetControlMode (CSTC this)**
Get the modality of the speed and torque controller.
- bool **STC_ExecRamp (CSTC this, int16_t hTargetFinal, uint16_t hDurationms)**
Starts the execution of a ramp using new target and duration. This command interrupts the execution of any previous ramp command. The generated ramp will be in the modality previously set by STC_SetControlMode method.
- void **STC_StopRamp (CSTC this)**
This command interrupts the execution of any previous ramp command. If STC has been set in Torque mode the last value of Iq is maintained. If STC has been set in Speed mode the last value of mechanical rotor speed reference is maintained.
- int16_t **STC_CalcTorqueReference (CSTC this)**
It is used to compute the new value of motor torque reference. It must be called at fixed time equal to hSTCFrequency-Hz. It is called passing as parameter the speed sensor used to perform the speed regulation.
- int16_t **STC_GetMecSpeedRef01HzDefault (CSTC this)**
Get the Default mechanical rotor speed reference expressed in tenths of HZ.
- uint16_t **STC_GetMaxAppPositiveMecSpeed01Hz (CSTC this)**
Get the Application maximum positive value of rotor speed. It's expressed in tenth of mechanical Hertz.
- int16_t **STC_GetMinAppNegativeMecSpeed01Hz (CSTC this)**
Get the Application minimum negative value of rotor speed. It's expressed in tenth of mechanical Hertz.
- bool **STC_RampCompleted (CSTC this)**

Check if the settled speed or torque ramp has been completed.

- void **STC_SetSpeedSensor** (**CSTC** this, **CSPD** oSPD)

It sets in real time the speed sensor utilized by the FOC.
- **CSPD STC_GetSpeedSensor** (**CSTC** this)

It returns the speed sensor utilized by the FOC.
- **Curr_Components STC_GetDefaultIqdref** (**CSTC** this)

It returns the default values of Iqdref.

3.69.1 Function Documentation

3.69.1.1 **CSTC STC_NewObject** (**pSpeednTorqCtrlParams_t pSpeednTorqCtrlParams**)

Creates an object of the class SpeednTorqCtrl.

Parameters

pSpeednTorqCtrlParams	pointer to an SpeednTorqCtrl parameters structure
------------------------------	---

Return values

CSTC	new instance of SpeednTorqCtrl object
-------------	---------------------------------------

Definition at line 53 of file SpeednTorqCtrlClass.c.

References MAX_STC_NUM, MC_NULL, and **_CSTC_t::pParams_str**.

3.69.1.2 void **STC_Init** (**CSTC this, CPI oPI, CSPD oSPD**)

Initializes all the object variables, usually it has to be called once right after object creation.

Parameters

this	related object of class CSTC.
oPI	the PI object used as controller for the speed regulation. It can be equal to MC_NULL if the STC is initialized in torque mode and it will never be configured in speed mode.
oSPD	the speed sensor used to perform the speed regulation. It can be equal to MC_NULL if the STC is used only in torque mode.

Return values

<i>none.</i>

Definition at line 87 of file SpeednTorqCtrlClass.c.

References **Vars_t::bMode**, **Vars_t::hRampRemainingStep**, **Vars_t::hTargetFinal**, **Vars_t::oPISpeed**, **Vars_t::oSPD**, **Vars_t::wIncDecAmount**, **Vars_t::wSpeedRef01HzExt**, and **Vars_t::wTorqueRef**.

3.69.1.3 void **STC_Clear** (**CSTC this**)

It should be called before each motor restart. If STC is set in speed mode, this method resets the integral term of speed regulator.

Parameters

<i>this</i>	related object of class CSTC.
-------------	-------------------------------

Return values

<i>none.</i>

Definition at line 131 of file SpeednTorqCtrlClass.c.

References Vars_t::bMode, PI_SetIntegralTerm(), and STC_SPEED_MODE.

Here is the call graph for this function:

**3.69.1.4 int16_t STC_GetMecSpeedRef01Hz (CSTC *this*)**

Get the current mechanical rotor speed reference expressed in tenths of HZ.

Parameters

<i>this</i>	related object of class CSTC.
-------------	-------------------------------

Return values

<i>int16_t</i>	current mechanical rotor speed reference expressed in tenths of HZ.
----------------	---

Definition at line 147 of file SpeednTorqCtrlClass.c.

3.69.1.5 int16_t STC_GetTorqueRef (CSTC *this*)

Get the current motor torque reference. This value represents actually the Iq current reference expressed in digit. To convert current expressed in digit to current expressed in Amps is possible to use the formula: Current(Amp) = [Current(digit) * Vdd micro] / [65536 * Rshunt * Aop].

Parameters

<i>this</i>	related object of class CSTC.
-------------	-------------------------------

Return values

<i>int16_t</i>	current motor torque reference. This value represents actually the Iq current expressed in digit.
----------------	---

Definition at line 162 of file SpeednTorqCtrlClass.c.

3.69.1.6 void STC_SetControlMode (CSTC *this*, STC_Modality_t *bMode*)

Set the modality of the speed and torque controller. Two modality are available Torque mode and Speed mode. In Torque mode is possible to set directly the motor torque reference or execute a motor torque ramp. This value represents actually the Iq current reference expressed in digit. In Speed mode is possible to set the mechanical rotor speed reference or execute a speed ramp. The required motor torque is automatically calculated by the STC. This command interrupts the execution of any previous ramp command maintaining the last value of Iq.

Parameters

<i>this</i>	related object of class CSTC.
<i>bMode</i>	modality of STC. It can be one of these two settings: STC_TORQUE_MODE to enable the Torque mode or STC_SPEED_MODE to enable the Speed mode.

Return values

<i>none</i>	
-------------	--

Definition at line 184 of file SpeednTorqCtrlClass.c.

References Vars_t::bMode, and Vars_t::hRampRemainingStep.

Referenced by EAC_StartAlignment(), MCI_ExecBufferedCommands(), and RUC_Clear().

3.69.1.7 STC_Modality_t STC_GetControlMode (CSTC *this*)

Get the modality of the speed and torque controller.

Parameters

<i>this</i>	related object of class CSTC.
-------------	-------------------------------

Return values

<i>STC_Modality_t</i>	It returns the modality of STC. It can be one of these two values: STC_TORQUE_MODE or STC_SPEED_MODE.
-----------------------	---

Definition at line 197 of file SpeednTorqCtrlClass.c.

References Vars_t::bMode.

3.69.1.8 bool STC_ExecRamp (CSTC *this*, int16_t *hTargetFinal*, uint16_t *hDurationms*)

Starts the execution of a ramp using new target and duration. This command interrupts the execution of any previous ramp command. The generated ramp will be in the modality previously set by STC_SetControlMode method.

Parameters

<i>this</i>	related object of class CSTC
<i>hTargetFinal</i>	final value of command. This is different accordingly the STC modality. If STC is in Torque mode hTargetFinal is the value of motor torque reference at the end of the ramp. This value represents actually the Iq current expressed in digit. To convert current expressed in Amps to current expressed in digit is possible to use the formula: Current(digit) = [Current(Amp) * 65536 * Rshunt * Aop] / Vdd micro If STC is in Speed mode hTargetFinal is the value of mechanical rotor speed reference at the end of the ramp expressed in tenths of HZ.
<i>hDurationms</i>	the duration of the ramp expressed in milliseconds. It is possible to set 0 to perform an instantaneous change in the value.

Return values

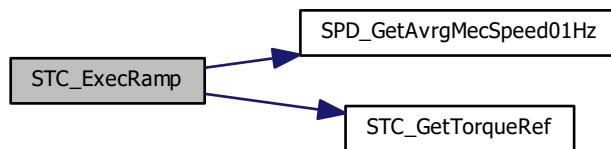
<i>bool</i>	It return false if the absolute value of hTargetFinal is out of the boundary of the application (Above max application speed or max application torque or below min application speed depending on current modality of TSC) in this case the command is ignored and the previous ramp is not interrupted, otherwise it returns true.
-------------	--

Definition at line 228 of file SpeednTorqCtrlClass.c.

References Vars_t::bMode, Vars_t::hRampRemainingStep, Vars_t::hTargetFinal, Vars_t::oSPD, SPD_GetAvrgMecSpeed01Hz(), STC_GetTorqueRef(), STC_SPEED_MODE, STC_TORQUE_MODE, Vars_t::wIncDecAmount, Vars_t::wSpeedRef01HzExt, and Vars_t::wTorqueRef.

Referenced by EAC_StartAlignment(), MCI_ExecBufferedCommands(), RUC_Clear(), and RUC_Exec().

Here is the call graph for this function:

**3.69.1.9 void STC_StopRamp (CSTC *this*)**

This command interrupts the execution of any previous ramp command. If STC has been set in Torque mode the last value of Iq is maintained. If STC has been set in Speed mode the last value of mechanical rotor speed reference is maintained.

Parameters

<i>this</i>	related object of class CSTC.
-------------	-------------------------------

Return values

<i>none</i>	
-------------	--

Definition at line 325 of file SpeednTorqCtrlClass.c.

References Vars_t::hRampRemainingStep, and Vars_t::wIncDecAmount.

3.69.1.10 int16_t STC_CalcTorqueReference (CSTC *this*)

It is used to compute the new value of motor torque reference. It must be called at fixed time equal to hSTC-FrequencyHz. It is called passing as parameter the speed sensor used to perform the speed regulation.

Parameters

<i>this</i>	related object of class CSTC.
-------------	-------------------------------

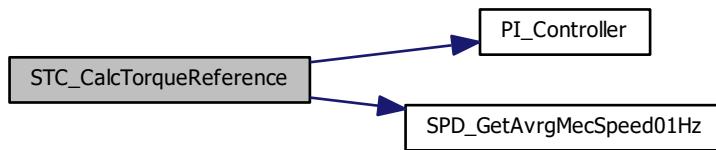
Return values

<i>int16_t</i>	motor torque reference. This value represents actually the Iq current expressed in digit. To convert current expressed in Amps to current expressed in digit is possible to use the formula: Current(digit) = [Current(Amp) * 65536 * Rshunt * Aop] / Vdd micro
----------------	---

Definition at line 344 of file SpeednTorqCtrlClass.c.

References Vars_t::bMode, Vars_t::hRampRemainingStep, Vars_t::hTargetFinal, Vars_t::oPISpeed, Vars_t::oSPD, PI_Controller(), SPD_GetAvrgMecSpeed01Hz(), STC_SPEED_MODE, STC_TORQUE_MODE, Vars_t::wIncDecAmount, Vars_t::wSpeedRef01HzExt, and Vars_t::wTorqueRef.

Here is the call graph for this function:

**3.69.1.11 int16_t STC_GetMecSpeedRef01HzDefault (CSTC *this*)**

Get the Default mechanical rotor speed reference expressed in tenths of HZ.

Parameters

<i>this</i>	related object of class CSTC.
-------------	-------------------------------

Return values

<i>int16_t</i>	It returns the Default mechanical rotor speed. reference expressed in tenths of HZ.
----------------	---

Definition at line 412 of file SpeednTorqCtrlClass.c.

3.69.1.12 uint16_t STC_GetMaxAppPositiveMecSpeed01Hz (CSTC *this*)

Get the Application maximum positive value of rotor speed. It's expressed in tenth of mechanical Hertz.

Parameters

<i>this</i>	related object of class CSTC.
-------------	-------------------------------

Return values

<i>uint16_t</i>	It returns the application maximum positive value of rotor speed expressed in tenth of mechanical Hertz.
-----------------	--

Definition at line 426 of file SpeednTorqCtrlClass.c.

3.69.1.13 int16_t STC_GetMinAppNegativeMecSpeed01Hz(CSTC *this*)

Get the Application minimum negative value of rotor speed. It's expressed in tenth of mechanical Hertz.

Parameters

<i>this</i>	related object of class CSTC.
-------------	-------------------------------

Return values

<i>uint16_t</i>	It returns the application minimum negative value of rotor speed expressed in tenth of mechanical Hertz.
-----------------	--

Definition at line 440 of file SpeednTorqCtrlClass.c.

3.69.1.14 bool STC_RampCompleted(CSTC *this*)

Check if the settled speed or torque ramp has been completed.

Parameters

<i>this</i>	related object of class CSTC.
-------------	-------------------------------

Return values

<i>bool</i>	It returns TRUE if the ramp is completed, FALSE otherwise.
-------------	--

Parameters

<i>this</i>	related object of class CSTC.
-------------	-------------------------------

Return values

<i>bool</i>	It returns true if the ramp is completed, false otherwise.
-------------	--

Definition at line 452 of file SpeednTorqCtrlClass.c.

References Vars_t::hRampRemainingStep.

Referenced by MCI_RampCompleted().

3.69.1.15 void STC_SetSpeedSensor(CSTC *this*, CSPD *oSPD*)

It sets in real time the speed sensor utilized by the FOC.

Parameters

<i>this</i>	related object of class CSTC
<i>oSPD</i>	Speed sensor object to be set.

Return values

<i>none</i>	It sets in real time the speed sensor utilized by the FOC.
-------------	--

Parameters

<i>this</i>	related object of class CSTC
<i>oSPD</i>	Speed sensor object to be set.

Return values

<i>none</i>

Definition at line 108 of file SpeednTorqCtrlClass.c.

References Vars_t::oSPD.

3.69.1.16 CSPD STC_GetSpeedSensor(CSTC *this*)

It returns the speed sensor utilized by the FOC.

Parameters

<i>this</i>	related object of class CSTC
-------------	------------------------------

Return values

<i>CSPD</i>	speed sensor utilized by the FOC.
-------------	-----------------------------------

Definition at line 119 of file SpeednTorqCtrlClass.c.

References Vars_t::oSPD.

Referenced by MCI_GetAvrgMecSpeed01Hz(), and MCI_GetSpdSensorReliability().

3.69.1.17 Curr_Components STC_GetDefaultIqdref(CSTC *this*)

It returns the default values of Iqdref.

Parameters

<i>this</i>	related object of class CSTC
-------------	------------------------------

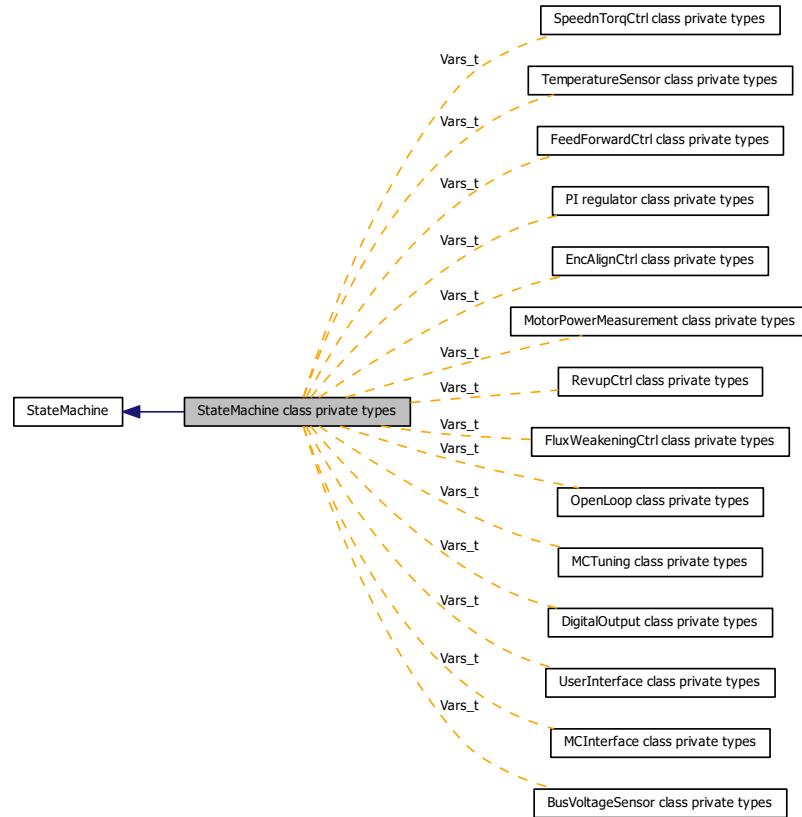
Return values

<i>default</i>	values of Iqdref.
----------------	-------------------

Definition at line 468 of file SpeednTorqCtrlClass.c.

3.70 StateMachine class private types

Collaboration diagram for StateMachine class private types:



Data Structures

- struct `Vars_t`
MCInterface class members definition.
- struct `_CSTM_t`
Private StateMachine class definition.

3.71 StateMachine class exported constants

Collaboration diagram for StateMachine class exported constants:



3.72 StateMachine class exported types

Collaboration diagram for StateMachine class exported types:



TypeDefs

- `typedef struct CSTM_t * CSTM`
Public StateMachine class definition.

3.72.1 Typedef Documentation

3.72.1.1 `typedef struct CSTM_t* CSTM`

Public StateMachine class definition.

Definition at line 62 of file StateMachineClass.h.

3.73 StateMachine class exported methods

Collaboration diagram for StateMachine class exported methods:



Functions

- [CSTM STM_NewObject \(void\)](#)

Creates an object of the class StateMachine.

- void [STM_Init \(CSTM this\)](#)

Initializes all the object variables, usually it has to be called once right after object creation.

- bool [STM_NextState \(CSTM this, State_t bState\)](#)

It submits the request for moving the state machine into the state specified by bState (FAULT_NOW and FAUL_OVER are not handled by this method). Accordingly with the current state, the command is really executed (state machine set to bState) or discarded (no state changes)

- [State_t STM_FaultProcessing \(CSTM this, uint16_t hSetErrors, uint16_t hResetErrors\)](#)

It clocks both HW and SW faults processing and update the state machine accordingly with hSetErrors, hResetErrors and present state. Refer to State_t description for more information about fault states.

- [State_t STM_GetState \(CSTM this\)](#)

Returns the current state machine state.

- bool [STM_FaultAcknowledged \(CSTM this\)](#)

It reports to the state machine that the fault state has been acknowledged by the user. If the state machine is in FAULT_OVER state then it is moved into STOP_IDLE and the bit field variable containing information about the faults historically occurred is cleared. The method call is discarded if the state machine is not in FAULT_OVER.

- uint32_t [STM_GetFaultState \(CSTM this\)](#)

It returns two 16 bit fields containing information about both faults currently present and faults historically occurred since the state machine has been moved into state

Returned error codes are listed here .

3.73.1 Function Documentation

3.73.1.1 CSTM STM_NewObject (void)

Creates an object of the class StateMachine.

Parameters

<i>pStateMachine-Params</i>	pointer to a StateMachine parameters structure.
-----------------------------	---

Return values

<i>CSTM</i>	new instance of StateMachine object.
-------------	--------------------------------------

Parameters

<i>pStateMachine-Params</i>	pointer to an StateMachine parameters structure
-----------------------------	---

Return values

<i>CSTM</i>	new instance of StateMachine object
-------------	-------------------------------------

Definition at line 52 of file StateMachineClass.c.

References MAX_STM_NUM, and MC_NULL.

3.73.1.2 void STM_Init (CSTM *this*)

Initializes all the object variables, usually it has to be called once right after object creation.

Parameters

<i>this</i>	related object of class CSTM.
-------------	-------------------------------

Return values

<i>none.</i>	
--------------	--

Definition at line 79 of file StateMachineClass.c.

References Vars_t::bState, Vars_t::hFaultNow, Vars_t::hFaultOccurred, IDLE, and MC_NOFAULTS.

3.73.1.3 bool STM_NextState (CSTM *this*, State_t *bState*)

It submits the request for moving the state machine into the state specified by bState (FAULT_NOW and FAULT_OVER are not handled by this method). Accordingly with the current state, the command is really executed (state machine set to bState) or discarded (no state changes)

Parameters

<i>this</i>	related object of class CSTM.
<i>bState</i>	New requested state

Return values

<i>bool</i>	It returns TRUE if the state has been really set equal to bState, FALSE if the request has been discarded
-------------	---

It submits the request for moving the state machine into the state specified by bState (FAULT_NOW and FAULT_OVER are not handled by this method). Accordingly with the current state, the command is really executed (state machine set to bState) or discarded (no state changes)

Parameters

<i>this</i>	related object of class CSTM.
<i>bState</i>	New requested state

Return values

<i>bool</i>	It returns TRUE if the state has been really set equal to bState, FALSE if the requested state can't be reached
-------------	---

Definition at line 104 of file StateMachineClass.c.

References ALIGN_CHARGE_BOOT_CAP, ALIGN_CLEAR, ALIGN_OFFSET_CALIB, ALIGNMENT, ANY_STOP, Vars_t::bState, CHARGE_BOOT_CAP, CLEAR, ICLWAIT, IDLE, IDLE_ALIGNMENT, IDLE_START, MC_SW_ERROR, OFFSET_CALIB, RUN, START, START_RUN, STM_FaultProcessing(), STOP, and STOP_IDLE.

Referenced by MCI_EncoderAlign(), MCI_StartMotor(), and MCI_StopMotor().

Here is the call graph for this function:



3.73.1.4 State_t STM_FaultProcessing (CSTM this, uint16_t hSetErrors, uint16_t hResetErrors)

It clocks both HW and SW faults processing and update the state machine accordingly with hSetErrors, hResetErrors and present state. Refer to State_t description for more information about fault states.

Parameters

<i>this</i>	object of class CSTM
<i>hSetErrors</i>	Bit field reporting faults currently present
<i>hResetErrors</i>	Bit field reporting faults to be cleared

Return values

<i>State_t</i>	New state machine state after fault processing
----------------	--

Definition at line 280 of file StateMachineClass.c.

References Vars_t::bState, FAULT_NOW, FAULT_OVER, Vars_t::hFaultNow, Vars_t::hFaultOccurred, and MC_NOFAULTS.

Referenced by STM_NextState(), TSK_HardwareFaultTask(), and TSK_HighFrequencyTask().

3.73.1.5 State_t STM_GetState (CSTM this)

Returns the current state machine state.

Parameters

<i>this</i>	object of class CSTM
-------------	----------------------

Return values

<i>State_t</i>	Current state machine state
----------------	-----------------------------

Definition at line 324 of file StateMachineClass.c.

3.73.1.6 bool STM_FaultAcknowledged (CSTM *this*)

It reports to the state machine that the fault state has been acknowledged by the user. If the state machine is in FAULT_OVER state then it is moved into STOP_IDLE and the bit field variable containing information about the faults historically occurred is cleared. The method call is discarded if the state machine is not in FAULT_OVER.

Parameters

<i>this</i>	object of class CSTM
-------------	----------------------

Return values

<i>bool</i>	TRUE if the state machine has been moved to IDLE, FALSE if the method call had no effects
-------------	---

Definition at line 340 of file StateMachineClass.c.

References Vars_t::bState, FAULT_OVER, Vars_t::hFaultOccurred, MC_NOFAULTS, and STOP_IDLE.

Referenced by MCI_FaultAcknowledged().

3.73.1.7 uint32_t STM_GetFaultState (CSTM *this*)

It returns two 16 bit fields containing information about both faults currently present and faults historically occurred since the state machine has been moved into state

[Returned error codes are listed here](#) .

Parameters

<i>this</i>	object of class CSTM.
-------------	-----------------------

Return values

<i>uint32_t</i>	Two 16 bit fields: in the most significant half are stored the information about currently present faults. In the least significant half are stored the information about the faults historically occurred since the state machine has been moved into FAULT_NOW state Returned error codes are listed here
-----------------	--

It returns two 16 bit fields containing information about both faults currently present and faults historically occurred since the state machine has been moved into state

[Returned error codes are listed here](#) .

Parameters

<i>this</i>	object of class CSTM.
-------------	-----------------------

Return values

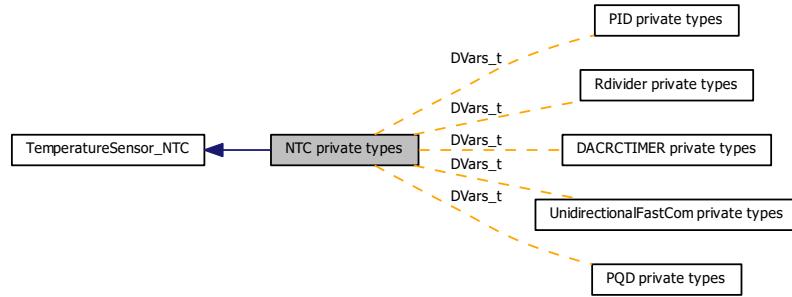
<i>uint32_t</i>	Two 16 bit fields: in the most significant half are stored the information about currently present faults. In the least significant half are stored the information about the faults historically occurred since the state machine has been moved into FAULT_NOW state
-----------------	--

Definition at line 367 of file StateMachineClass.c.

References Vars_t::hFaultNow, and Vars_t::hFaultOccurred.

3.74 NTC private types

Collaboration diagram for NTC private types:



Data Structures

- struct `DVars_t`
PQD class members definition.
- struct `_CNTC_TSNS_t`
Private NTC class definition.

Typedefs

- `typedef NTCParams_t DParams_t`
Redefinition of parameter structure.

3.74.1 Typedef Documentation

3.74.1.1 `typedef NTCParams_t DParams_t`

Redefinition of parameter structure.

Definition at line 60 of file `NTC_TemperatureSensorPrivate.h`.

3.75 NTC class exported types

Collaboration diagram for NTC class exported types:



Data Structures

- struct [NTCParams_t](#)
NTC class parameters definition.

Typedefs

- typedef struct CNTC_TSNS_t * [CNTC_TSNS](#)
Public NTC class definition.

3.75.1 Typedef Documentation

3.75.1.1 [typedef struct CNTC_TSNS_t* CNTC_TSNS](#)

Public NTC class definition.

Definition at line 49 of file `NTC_TemperatureSensorClass.h`.

3.76 NTC class exported methods

Collaboration diagram for NTC class exported methods:



Functions

- **CNTC_TSNS NTC_NewObject (pTempSensorParams_t pTempSensorParams, pNTCParams_t pNTCParams)**

Creates an object of the class NTC.

3.76.1 Function Documentation

3.76.1.1 CNTC_TSNS NTC_NewObject (pTempSensorParams_t pTempSensorParams, pNTCParams_t pNTCParams)

Creates an object of the class NTC.

Parameters

<i>pTempSensor- Params</i>	pointer to an TemperatureSensor parameters structure
<i>pNTCParams</i>	pointer to an NTC parameters structure

Return values

<i>CNTC_TSNS</i>	new instance of NTC object
------------------	----------------------------

Parameters

<i>pTempSensor- Params</i>	pointer to an TempSensor parameters structure
<i>pNTCParams</i>	pointer to an NTC parameters structure

Return values

<i>CNTC_TSNS</i>	new instance of NTC object
------------------	----------------------------

Definition at line 68 of file NTC_TemperatureSensorClass.c.

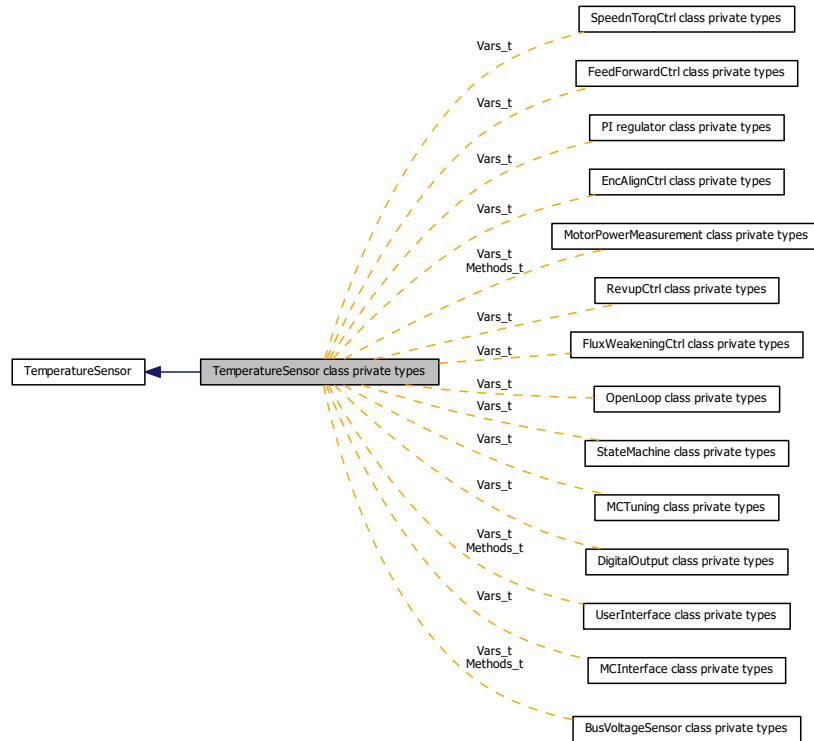
References _CTSNS_t::DerivedClass, MAX_NTC_TSNS_NUM, MC_NULL, _CTSNS_t::Methods_str, _CNTC_TSNS_t::pDParams_str, and TSNS_NewObject().

Here is the call graph for this function:



3.77 TemperatureSensor class private types

Collaboration diagram for TemperatureSensor class private types:



Data Structures

- struct [Vars_t](#)
MCInterface class members definition.
 - struct [Methods_t](#)
Virtual methods container.
 - struct [_CTSNS_t](#)
Private TemperatureSensor class definition.

TypeDefs

- `typedef TempSensorParams_t Params_t`
Redefinition of parameter structure

3.77.1 Typedef Documentation

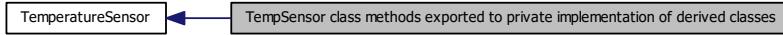
3.77.1.1 **typedef TempSensorParams t Params**

Redefinition of parameter structure.

Definition at line 60 of file TemperatureSensorPrivate.h.

3.78 TempSensor class methods exported to private implementation of derived classes

Collaboration diagram for TempSensor class methods exported to private implementation of derived classes:



Functions

- [CTSNS_TSNS_NewObject \(pTempSensorParams_t pTempSensorParams \)](#)
Creates an object of the class temperature sensor.

3.78.1 Function Documentation

3.78.1.1 CTSNS_TSNS_NewObject (pTempSensorParams_t pTempSensorParams)

Creates an object of the class temperature sensor.

Parameters

<code>pTempSensor- Params</code>	pointer to a temperature sensor parameters structure
--------------------------------------	--

Return values

<code>CTSNS</code>	new instance of temperature sensor object
--------------------	---

Creates an object of the class temperature sensor.

Parameters

<code>pTempSensor- Params</code>	pointer to an TempSensor parameters structure
--------------------------------------	---

Return values

<code>CTSNS</code>	new instance of TempSensor object
--------------------	-----------------------------------

Definition at line 51 of file TemperatureSensorClass.c.

References MAX_TSNS_NUM, MC_NULL, and _CTSNS_t::pParams_str.

Referenced by NTC_NewObject(), and VTS_NewObject().

3.79 Temperature Sensor class exported types

Collaboration diagram for Temperature Sensor class exported types:



Data Structures

- struct [TempSensorParams_t](#)
Temperature sensor class parameters definition.

Typedefs

- typedef struct CTSNS_t * [CTSNS](#)
Public temperature sensor class definition.

3.79.1 Typedef Documentation

3.79.1.1 [typedef struct CTSNS_t*](#) [CTSNS](#)

Public temperature sensor class definition.

Definition at line 53 of file TemperatureSensorClass.h.

3.80 Temperature sensor class exported methods

Collaboration diagram for Temperature sensor class exported methods:



Functions

- void [TSNS_Init \(CTSNS this, CPWMC pPWMnCurrentSensor\)](#)
It initializes temperature sensing conversions. It must be called only after current sensor initialization (PWMC_Init)
- void [TSNS_Clear \(CTSNS this\)](#)
It clears FW variable containing average temperature measurement value.
- uint16_t [TSNS_CalcAvTemp \(CTSNS this\)](#)
It clock the temperature sensing. It performs ADC conversion and updates the average. It returns MC_OVER_TEMP or MC_NO_ERROR depending on temperature average value measurement and protection threshold values.
- uint16_t [TSNS_GetAvTemp_d \(CTSNS this\)](#)
It return latest averaged temperature measurement expressed in u16Celsius.
- int16_t [TSNS_GetAvTemp_C \(CTSNS this\)](#)
It returns latest averaged temperature measurement expressed in Celsius degrees.
- uint16_t [TSNS_CheckTemp \(CTSNS this\)](#)
It returns MC_OVER_TEMP or MC_NO_ERROR depending on temperature measurement and protection threshold values.

3.80.1 Function Documentation

3.80.1.1 void [TSNS_Init \(CTSNS this, CPWMC pPWMnCurrentSensor \)](#)

It initializes temperature sensing conversions. It must be called only after current sensor initialization (PWMC_Init)

Parameters

this	related object of class CTSNS
----------------------	-------------------------------

Return values

none

Definition at line 79 of file TemperatureSensorClass.c.

3.80.1.2 void [TSNS_Clear \(CTSNS this \)](#)

It clears FW variable containing average temperature measurement value.

Parameters

this	related object of class CTSNS
----------------------	-------------------------------

Return values

<i>none</i>

Definition at line 91 of file TemperatureSensorClass.c.

3.80.1.3 uint16_t TSNS_CalcAvTemp (CTSNS *this*)

It clock the temperature sensing. It performs ADC conversion and updates the average. It returns MC_OVER_T-EMP or MC_NO_ERROR depending on temperature average value measurement and protection threshold values.

Parameters

<i>this</i>	related object of class CTSNS
-------------	-------------------------------

Return values

<i>uint16_t</i>	Fault code error
-----------------	------------------

It clock the temperature sensing. It performs ADC conversion and updates the average. It returns MC_OVER_T-EMP or MC_NO_ERROR depending on temperature average value measurement and protection threshold values.

Parameters

<i>this</i>	related object of class CTSNS
-------------	-------------------------------

Return values

<i>uint16_t</i>	Fault code error
-----------------	------------------

Definition at line 102 of file TemperatureSensorClass.c.

3.80.1.4 uint16_t TSNS_GetAvTemp_d (CTSNS *this*)

It return latest averaged temperature measurement expressed in u16Celsius.

Parameters

<i>this</i>	related object of class CTSNS
-------------	-------------------------------

Return values

<i>uint16_t</i>	Latest averaged temperature measurement in u16Celsius
-----------------	---

Definition at line 113 of file TemperatureSensorClass.c.

3.80.1.5 int16_t TSNS_GetAvTemp_C (CTSNS *this*)

It returns latest averaged temperature measurement expressed in Celsius degrees.

Parameters

<i>this</i>	related object of class CTSNS
-------------	-------------------------------

Return values

<i>int16_t</i>	Latest averaged temperature measurement in Celsius degrees
----------------	--

Parameters

<i>this</i>	related object of class CTSNS
-------------	-------------------------------

Return values

<i>uint16_t</i>	Latest averaged temperature measurement in Celsius degrees
-----------------	--

Definition at line 124 of file TemperatureSensorClass.c.

3.80.1.6 *uint16_t TSNS_CheckTemp(CTSNS this)*

It returns MC_OVER_TEMP or MC_NO_ERROR depending on temperature measurement and protection threshold values.

Parameters

<i>this</i>	related object of class CTSNS
-------------	-------------------------------

Return values

<i>uint16_t</i>	Fault code error
-----------------	------------------

It returns MC_OVER_TEMP or MC_NO_ERROR depending on temperature measurement and protection threshold values.

It returns MC_OVER_TEMP or MC_NO_ERROR depending on temperature sensor output measurement and protection threshold values.

Parameters

<i>this</i>	related object of class CTSNS
-------------	-------------------------------

Return values

<i>uint16_t</i>	Fault code error
-----------------	------------------

Definition at line 135 of file TemperatureSensorClass.c.

3.81 Virtual temperature sensor private types

Collaboration diagram for Virtual temperature sensor private types:



Data Structures

- struct [_CVTS_TSNS_t](#)
Private Virtual temperature sensor class definition.

Typedefs

- [typedef VirtualTPParams_t DParams_t](#)
VTS_TSNS class members definition.

3.81.1 Typedef Documentation

3.81.1.1 [typedef VirtualTPParams_t DParams_t](#)

VTS_TSNS class members definition.

Redefinition of parameter structure

Definition at line 52 of file `Virtual_TemperatureSensorPrivate.h`.

3.82 Virtual Temperature sensor class exported types

Collaboration diagram for Virtual Temperature sensor class exported types:



Data Structures

- struct [VirtualTPParams_t](#)
Virtual temperature sensor class parameters definition.

Typedefs

- typedef struct CVTS_TSNS_t * [CVTS_TSNS](#)
Public Virtual Temperature sensor class definition.

3.82.1 Typedef Documentation

3.82.1.1 [typedef struct CVTS_TSNS_t* CVTS_TSNS](#)

Public Virtual Temperature sensor class definition.

Definition at line 49 of file `Virtual_TemperatureSensorClass.h`.

3.83 Virtual Temperature sensor class exported methods

Collaboration diagram for Virtual Temperature sensor class exported methods:



Functions

- **`CVTS_TSNS VTS_NewObject`** (`pTempSensorParams_t pTempSensorParams, pVirtualTPParams_t pVirtualParams`)
Creates an object of the class Virtual temperature sensor.

3.83.1 Function Documentation

3.83.1.1 `CVTS_TSNS VTS_NewObject` (`pTempSensorParams_t pTempSensorParams, pVirtualTPParams_t pVirtualParams`)

Creates an object of the class Virtual temperature sensor.

Parameters

<code>pTempSensorParams</code>	pointer to an TempSensor parameters structure
<code>pVirtualParams</code>	pointer to a virtual temperature sensor parameters structure

Return values

<code>CVTS_TSNS</code>	new instance of virtual temperature sensor object
------------------------	---

Creates an object of the class Virtual temperature sensor.

Parameters

<code>pTempSensorParams</code>	pointer to an TemperatureSensor parameters structure
<code>pVirtualParams</code>	pointer to an Virtual Temperature sensor parameters structure

Return values

<code>CVTS_TSNS</code>	new instance of Virtual Temperature sensor object
------------------------	---

Definition at line 61 of file `Virtual_TemperatureSensorClass.c`.

References `_CTSNS_t::DerivedClass`, `MAX_VTS_TSNS_NUM`, `MC_NULL`, `_CTSNS_t::Methods_str`, `_CVTS_TSNS_t::pDParams_str`, and `TSNS_NewObject()`.

Here is the call graph for this function:



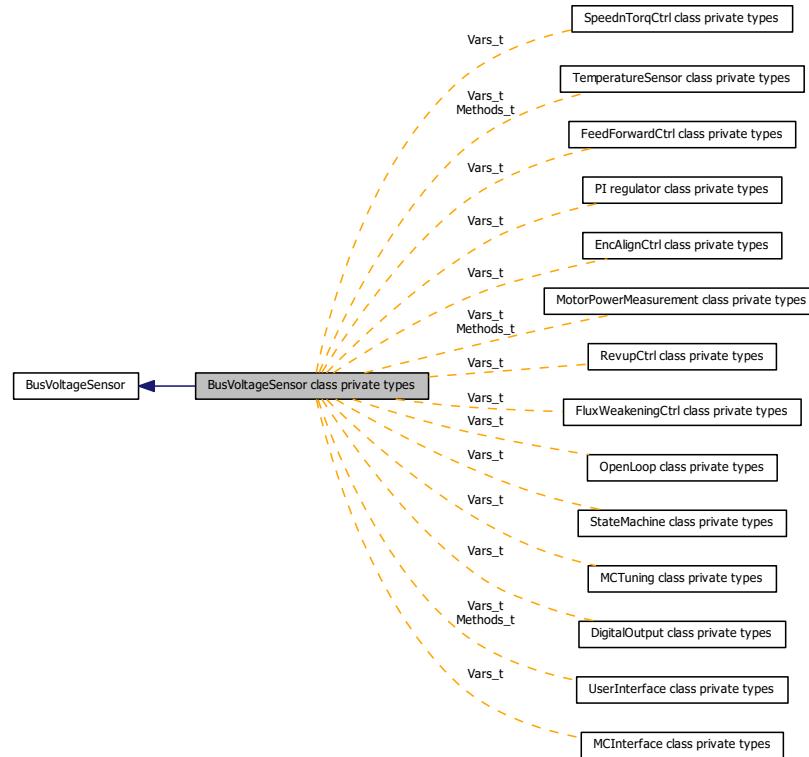
3.84 Virtual Temperature sensor class private methods

Collaboration diagram for Virtual Temperature sensor class private methods:



3.85 BusVoltageSensor class private types

Collaboration diagram for BusVoltageSensor class private types:



Data Structures

- struct [Vars_t](#)
MCInterface class members definition.
 - struct [Methods_t](#)
Virtual methods container.
 - struct [_CVBS_t](#)
Private BusVoltageSensor class definition.

TypeDefs

- `typedef BusVoltageSensorParams_t Params_t`
Redefinition of parameter structure.

3.85.1 Typedef Documentation

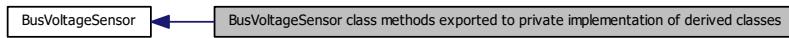
3.85.1.1 **typedef BusVoltageSensorParams_t Params_t**

Redefinition of parameter structure.

Definition at line 62 of file BusVoltageSensorPrivate.h.

3.86 BusVoltageSensor class methods exported to private implementation of derived classes

Collaboration diagram for BusVoltageSensor class methods exported to private implementation of derived classes:



Functions

- [CVBS_VBS_NewObject \(pBusVoltageSensorParams_t pBusVoltageSensorParams\)](#)
Creates an object of the class BusVoltageSensor.

3.86.1 Function Documentation

3.86.1.1 CVBS_VBS_NewObject (pBusVoltageSensorParams_t pBusVoltageSensorParams)

Creates an object of the class BusVoltageSensor.

Parameters

<code>pBusVoltage-SensorParams</code>	pointer to a BusVoltageSensor parameters structure
---------------------------------------	--

Return values

<code>CVBS</code>	new instance of BusVoltageSensor object
-------------------	---

Parameters

<code>pBusVoltage-SensorParams</code>	pointer to an BusVoltageSensor parameters structure
---------------------------------------	---

Return values

<code>CVBS</code>	new instance of BusVoltageSensor object
-------------------	---

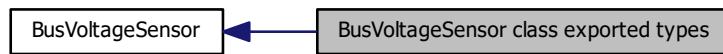
Definition at line 51 of file BusVoltageSensorClass.c.

References MAX_VBS_NUM, MC_NULL, and _CVBS_t::pParams_str.

Referenced by RVBS_NewObject(), and VVBS_NewObject().

3.87 BusVoltageSensor class exported types

Collaboration diagram for BusVoltageSensor class exported types:



Data Structures

- struct [BusVoltageSensorParams_t](#)
BusVoltageSensor class parameters definition.

Typedefs

- typedef struct CVBS_t * [CVBS](#)
Public BusVoltageSensor class definition.

3.87.1 Typedef Documentation

3.87.1.1 [typedef struct CVBS_t* CVBS](#)

Public BusVoltageSensor class definition.

Definition at line 53 of file BusVoltageSensorClass.h.

3.88 BusVoltageSensor class exported methods

Collaboration diagram for BusVoltageSensor class exported methods:



Functions

- void **VBS_Init** (**CVBS** this, **CPWMC** oPWMnCurrentSensor)

It initializes bus voltage conversion. It must be called only after current sensor initialization (PWMC_Init)
- void **VBS_Clear** (**CVBS** this)

It clears bus voltage FW variable containing average bus voltage value.
- uint16_t **VBS_CalcAvVbus** (**CVBS** this)

It clocks the bus voltage reading, performs Vbus conversion and updates the average.
- uint16_t **VBS_GetBusVoltage_d** (**CVBS** this)

It returns latest Vbus conversion result expressed in u16Volt.
- uint16_t **VBS_GetAvBusVoltage_d** (**CVBS** this)

It returns latest averaged Vbus measurement expressed in u16Volt.
- uint16_t **VBS_GetAvBusVoltage_V** (**CVBS** this)

It returns latest averaged Vbus measurement expressed in Volts.
- uint16_t **VBS_CheckVbus** (**CVBS** this)

It returns MC_OVER_VOLT, MC_UNDER_VOLT or MC_NO_ERROR depending on bus voltage measurement and protection threshold values.

3.88.1 Function Documentation

3.88.1.1 void VBS_Init (CVBS this, CPWMC oPWMnCurrentSensor)

It initializes bus voltage conversion. It must be called only after current sensor initialization (PWMC_Init)

Parameters

this	related object of class CVBS
oPWMnCurrent-Sensor	CPWMC object to be used for regular conversions

Return values

none

Parameters

this	related object of class CVBS
-------------	------------------------------

Return values

<i>none</i>

Definition at line 79 of file BusVoltageSensorClass.c.

3.88.1.2 void VBS_Clear (CVBS *this*)

It clears bus voltage FW variable containing average bus voltage value.

Parameters

<i>this</i>	related object of class CVBS
-------------	------------------------------

Return values

<i>none</i>

Definition at line 91 of file BusVoltageSensorClass.c.

3.88.1.3 uint16_t VBS_CalcAvVbus (CVBS *this*)

It clocks the bus voltage reading, performs Vbus conversion and updates the average.

Parameters

<i>this</i>	related object of class CVBS
-------------	------------------------------

Return values

<i>uint16_t</i>	Fault code error
-----------------	------------------

Definition at line 102 of file BusVoltageSensorClass.c.

3.88.1.4 uint16_t VBS_GetBusVoltage_d (CVBS *this*)

It returns latest Vbus conversion result expressed in u16Volt.

Parameters

<i>this</i>	related object of class CVBS
-------------	------------------------------

Return values

<i>uint16_t</i>	Latest Vbus conversion result
-----------------	-------------------------------

It returns latest Vbus conversion result expressed in u16Volt.

Parameters

<i>this</i>	related object of class CVBS
-------------	------------------------------

Return values

<i>uint16_t</i>	Latest Vbus conversion result in digit
-----------------	--

Definition at line 112 of file BusVoltageSensorClass.c.

3.88.1.5 uint16_t VBS_GetAvBusVoltage_d (CVBS *this*)

It returns latest averaged Vbus measurement expressed in u16Volt.

Parameters

<i>this</i>	related object of class CVBS
-------------	------------------------------

Return values

<i>uint16_t</i>	Latest averaged Vbus measurement in digital value
-----------------	---

It returns latest averaged Vbus measurement expressed in u16Volt.

Parameters

<i>this</i>	related object of class CVBS
-------------	------------------------------

Return values

<i>uint16_t</i>	Latest averaged Vbus measurement in digit
-----------------	---

Definition at line 129 of file BusVoltageSensorClass.c.

Referenced by TSK_HighFrequencyTask().

3.88.1.6 uint16_t VBS_GetAvBusVoltage_V (CVBS *this*)

It returns latest averaged Vbus measurement expressed in Volts.

Parameters

<i>this</i>	related object of class CVBS
-------------	------------------------------

Return values

<i>uint16_t</i>	Latest averaged Vbus measurement in Volts
-----------------	---

It returns latest averaged Vbus measurement expressed in Volts.

Parameters

<i>this</i>	related object of class CVBS
-------------	------------------------------

Return values

<i>uint16_t</i>	Latest averaged Vbus measurement in Volts
-----------------	---

Definition at line 139 of file BusVoltageSensorClass.c.

3.88.1.7 uint16_t VBS_CheckVbus (CVBS *this*)

It returns MC_OVER_VOLT, MC_UNDER_VOLT or MC_NO_ERROR depending on bus voltage measurement and protection threshold values.

Parameters

<i>this</i>	related object of class CVBS
-------------	------------------------------

Return values

<i>uint16_t</i>	Fault code error
-----------------	------------------

It returns MC_OVER_VOLT, MC_UNDER_VOLT or MC_NO_ERROR depending on bus voltage measurement and protection threshold values.

Parameters

<i>this</i>	related object of class CVBS
-------------	------------------------------

Return values

<i>uint16_t</i>	Fault code error
-----------------	------------------

Definition at line 156 of file BusVoltageSensorClass.c.

3.89 Virtual Vbus sensor private types

Collaboration diagram for Virtual Vbus sensor private types:



Data Structures

- struct [_CVVBS_VBS_t](#)
Private Virtual Vbus sensor class definition.

Typedefs

- typedef [VirtualParams_t DParams_t](#)
Redefinition of parameter structure.

3.89.1 Typedef Documentation

3.89.1.1 [typedef VirtualParams_t DParams_t](#)

Redefinition of parameter structure.

Definition at line 49 of file `Virtual_BusVoltageSensorPrivate.h`.

3.90 Virtual Vbus sensor class exported types

Collaboration diagram for Virtual Vbus sensor class exported types:



Data Structures

- struct [VirtualParams_t](#)
Virtual Vbus sensor class parameters definition.

Typedefs

- typedef struct CVVBS_VBS_t * [CVVBS_VBS](#)
Public Virtual Vbus sensor class definition.

3.90.1 Typedef Documentation

3.90.1.1 [typedef struct CVVBS_VBS_t* CVVBS_VBS](#)

Public Virtual Vbus sensor class definition.

Definition at line 49 of file `Virtual_BusVoltageSensorClass.h`.

3.91 Virtual Vbus sensor class exported methods

Collaboration diagram for Virtual Vbus sensor class exported methods:



Functions

- [CVVBS_VBS VVBS_NewObject \(pBusVoltageSensorParams_t pBusVoltageSensorParams, pVirtualParams_t pVirtualParams\)](#)

Creates an object of the class Virtual Vbus sensor.

3.91.1 Function Documentation

3.91.1.1 **CVVBS_VBS VVBS_NewObject (pBusVoltageSensorParams_t pBusVoltageSensorParams, pVirtualParams_t pVirtualParams)**

Creates an object of the class Virtual Vbus sensor.

Parameters

<i>pBusVoltage-SensorParams</i>	pointer to an BusVoltageSensor parameters structure
<i>pVirtualParams</i>	pointer to an Virtual Vbus sensor parameters structure

Return values

<i>CVVBS_VBS</i>	new instance of Virtual Vbus sensor object
------------------	--

Definition at line 60 of file Virtual_BusVoltageSensorClass.c.

References *_CVBS_t*::DerivedClass, MAX_VVBS_VBS_NUM, MC_NULL, *_CVBS_t*::Methods_str, *_CVVBS_VBS_t*::pDParams_str, and *VBS_NewObject()*.

Here is the call graph for this function:



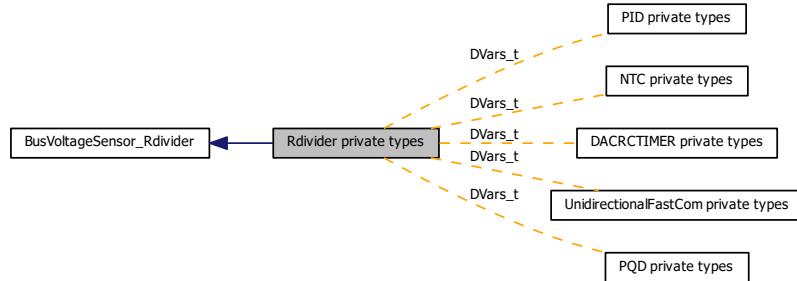
3.92 Virtual Vbus sensor class private methods

Collaboration diagram for Virtual Vbus sensor class private methods:



3.93 Rdivider private types

Collaboration diagram for Rdivider private types:



Data Structures

- struct `DVars_t`
PQD class members definition.
- struct `_CRVBS_VBS_t`
Private Rdivider class definition.

Typedefs

- `typedef RdividerParams_t DParams_t`
Redefinition of parameter structure.

3.93.1 Typedef Documentation

3.93.1.1 `typedef RdividerParams_t DParams_t`

Redefinition of parameter structure.

Definition at line 58 of file `Rdivider_BusVoltageSensorPrivate.h`.

3.94 Rdivider class exported types

Collaboration diagram for Rdivider class exported types:



Data Structures

- struct [RdividerParams_t](#)
Rdivider class parameters definition.

TypeDefs

- typedef struct CRVBS_VBS_t * [CRVBS_VBS](#)
Public Rdivider class definition.

3.94.1 TypeDef Documentation

3.94.1.1 [typedef struct CRVBS_VBS_t* CRVBS_VBS](#)

Public Rdivider class definition.

Definition at line 49 of file Rdivider_BusVoltageSensorClass.h.

3.95 Rdivider class exported methods

Collaboration diagram for Rdivider class exported methods:



Functions

- [CRVBS_VBS RVBS_NewObject \(pBusVoltageSensorParams_t pBusVoltageSensorParams, pRdividerParams_t pRdividerParams \)](#)

Creates an object of the class Rdivider.

3.95.1 Function Documentation

3.95.1.1 CRVBS_VBS RVBS_NewObject (pBusVoltageSensorParams_t pBusVoltageSensorParams, pRdividerParams_t pRdividerParams)

Creates an object of the class Rdivider.

Parameters

<code>pBusVoltage-SensorParams</code>	pointer to an BusVoltageSensor parameters structure
<code>pRdivider-Params</code>	pointer to an Rdivider parameters structure

Return values

<code>CRVBS_VBS</code>	new instance of Rdivider object
------------------------	---------------------------------

Definition at line 63 of file Rdivider_BusVoltageSensorClass.c.

References `_CVBS_t::DerivedClass`, `MAX_RVBS_VBS_NUM`, `MC_NULL`, `_CVBS_t::Methods_str`, `_CRVBS_VBS_t::pDParams_str`, and `VBS_NewObject()`.

Here is the call graph for this function:



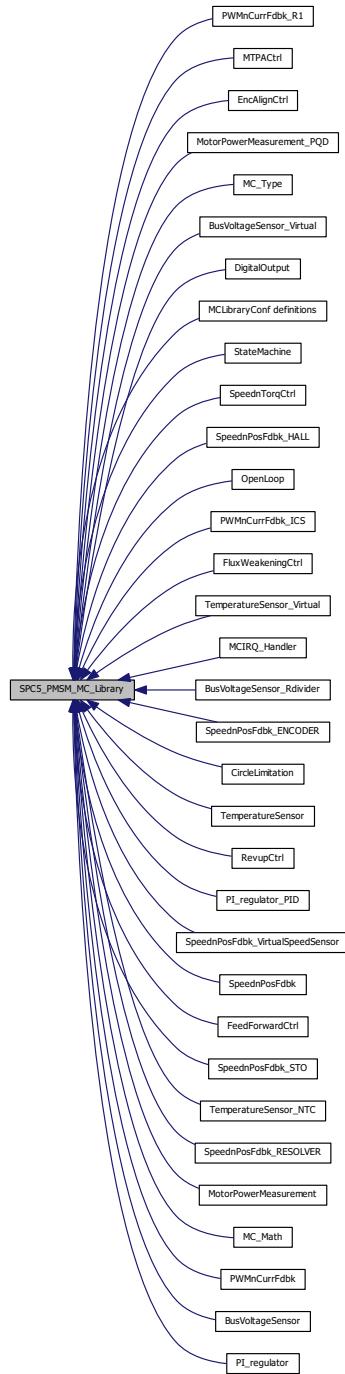
3.96 Rdivider class private methods

Collaboration diagram for Rdivider class private methods:



3.97 SPC5_PMSM_MC_Library

Collaboration diagram for SPC5_PMSM_MC_Library:



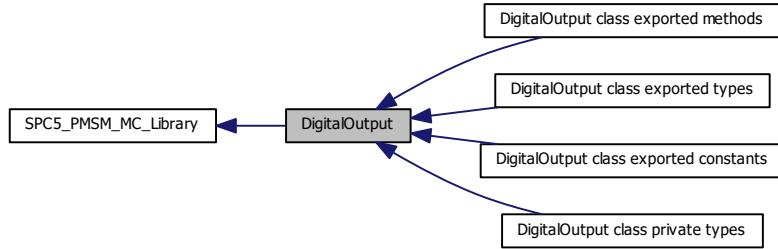
Modules

- **MCLibraryConf definitions**
- **DigitalOutput**
- **EncAlignCtrl**

- CircleLimitation
- MCIRQ_Handler
- MotorPowerMeasurement
- MotorPowerMeasurement_PQD
- OpenLoop
- PI_regulator_PID
- PI_regulator
- FeedForwardCtrl
- FluxWeakeningCtrl
- MTPACtrl
- PWMnCurrFdbk_ICS
- PWMnCurrFdbk
- PWMnCurrFdbk_R1
- RevupCtrl
- SpeednPosFdbk_ENCODER
- SpeednPosFdbk_HALL
- SpeednPosFdbk_RESOLVER
- SpeednPosFdbk
- SpeednPosFdbk_STO
- SpeednPosFdbk_VirtualSpeedSensor
- SpeednTorqCtrl
- StateMachine
- TemperatureSensor_NTC
- TemperatureSensor
- TemperatureSensor_Virtual
- BusVoltageSensor
- BusVoltageSensor_Rdivider
- BusVoltageSensor_Virtual
- MC_Math
- MC_Type

3.98 DigitalOutput

Collaboration diagram for DigitalOutput:

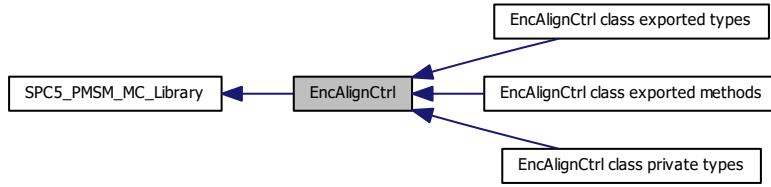


Modules

- `DigitalOutput class private types`
- `DigitalOutput class exported constants`
- `DigitalOutput class exported types`
- `DigitalOutput class exported methods`

3.99 EncAlignCtrl

Collaboration diagram for EncAlignCtrl:

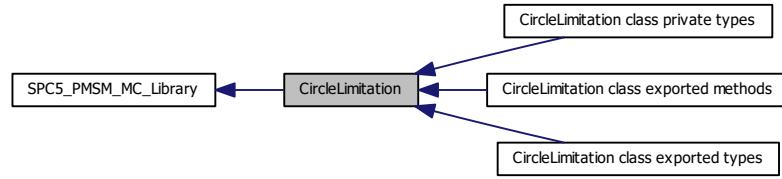


Modules

- `EncAlignCtrl class private types`
- `EncAlignCtrl class exported types`
- `EncAlignCtrl class exported methods`

3.100 CircleLimitation

Collaboration diagram for CircleLimitation:

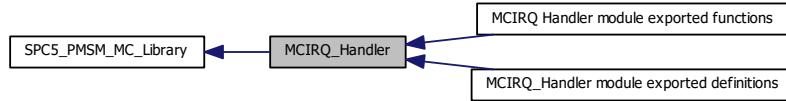


Modules

- CircleLimitation class private types
- CircleLimitation class exported types
- CircleLimitation class exported methods

3.101 MCIRQ_Handler

Collaboration diagram for MCIRQ_Handler:

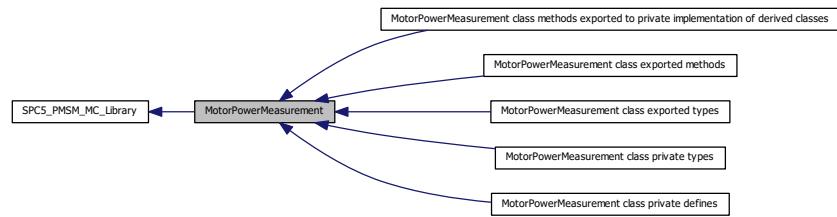


Modules

- **MCIRQ_Handler module exported definitions**
- **MCIRQ Handler module exported functions**

3.102 MotorPowerMeasurement

Collaboration diagram for MotorPowerMeasurement:

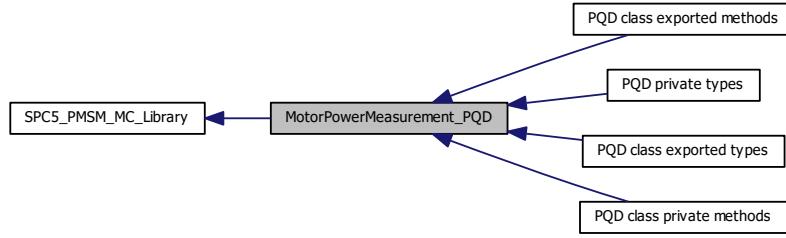


Modules

- `MotorPowerMeasurement class private defines`
- `MotorPowerMeasurement class private types`
- `MotorPowerMeasurement class methods exported to private implementation of derived classes`
- `MotorPowerMeasurement class exported types`
- `MotorPowerMeasurement class exported methods`

3.103 MotorPowerMeasurement_PQD

Collaboration diagram for MotorPowerMeasurement_PQD:

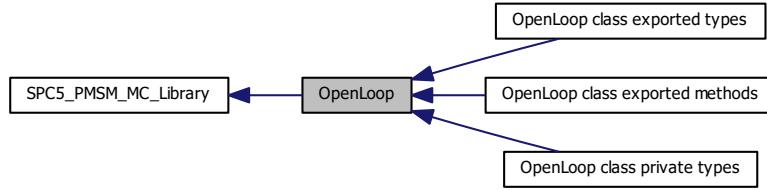


Modules

- PQD private types
- PQD class exported types
- PQD class exported methods
- PQD class private methods

3.104 OpenLoop

Collaboration diagram for OpenLoop:

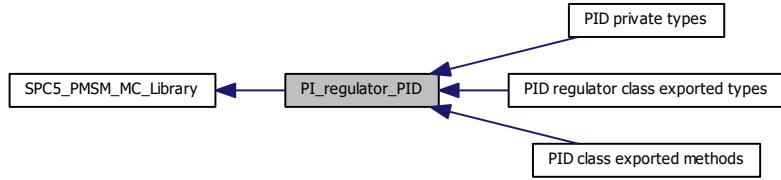


Modules

- `OpenLoop class private types`
- `OpenLoop class exported types`
- `OpenLoop class exported methods`

3.105 PI_regulator_PID

Collaboration diagram for PI_regulator_PID:

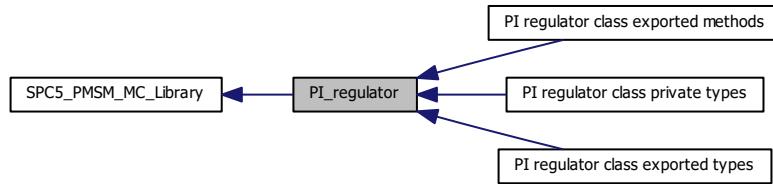


Modules

- PID private types
- PID regulator class exported types
- PID class exported methods

3.106 PI_regulator

Collaboration diagram for PI_regulator:

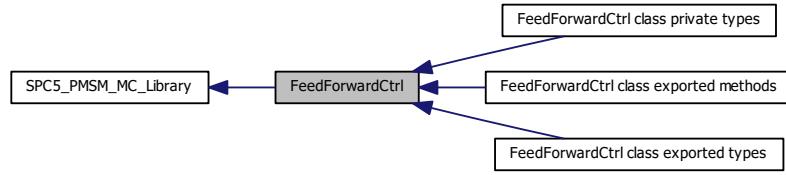


Modules

- PI regulator class private types
- PI regulator class exported types
- PI regulator class exported methods

3.107 FeedForwardCtrl

Collaboration diagram for FeedForwardCtrl:

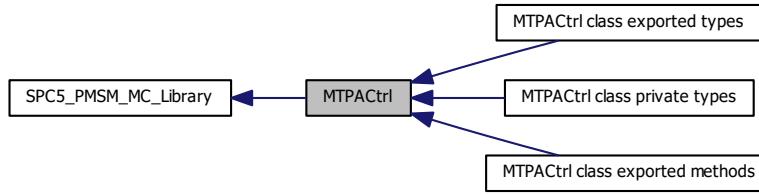


Modules

- `FeedForwardCtrl class private types`
- `FeedForwardCtrl class exported types`
- `FeedForwardCtrl class exported methods`

3.108 MTPACtrl

Collaboration diagram for MTPACtrl:

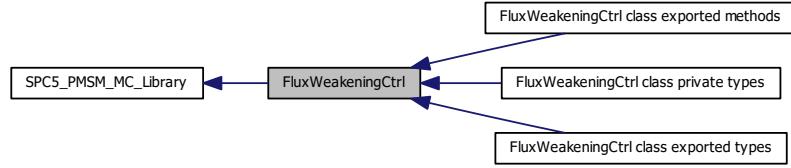


Modules

- `MTPACtrl class private types`
- `MTPACtrl class exported types`
- `MTPACtrl class exported methods`

3.109 FluxWeakeningCtrl

Collaboration diagram for FluxWeakeningCtrl:

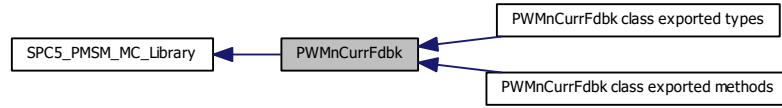


Modules

- `FluxWeakeningCtrl class private types`
- `FluxWeakeningCtrl class exported types`
- `FluxWeakeningCtrl class exported methods`

3.110 PWMnCurrFdbk

Collaboration diagram for PWMnCurrFdbk:

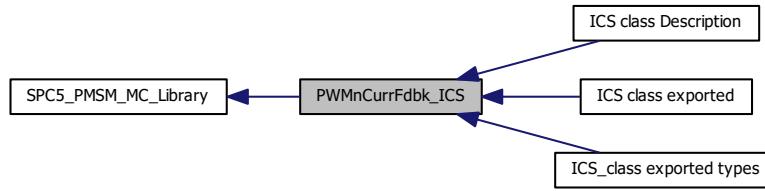


Modules

- [PWMnCurrFdbk class exported types](#)
- [PWMnCurrFdbk class exported methods](#)

3.111 PWMnCurrFdbk_ICS

Collaboration diagram for PWMnCurrFdbk_ICS:



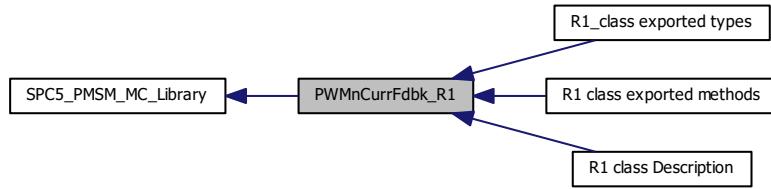
Modules

- [ICS_class exported types](#)
- [ICS class exported](#)
- [ICS class Description](#)

ICS PWMnCurrFdbk class implementation.

3.112 PWMnCurrFdbk_R1

Collaboration diagram for PWMnCurrFdbk_R1:



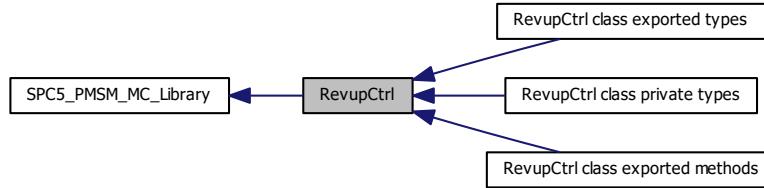
Modules

- [R1_class exported types](#)
- [R1 class exported methods](#)
- [R1 class Description](#)

R1 PWMnCurrFdbk class implementation.

3.113 RevupCtrl

Collaboration diagram for RevupCtrl:

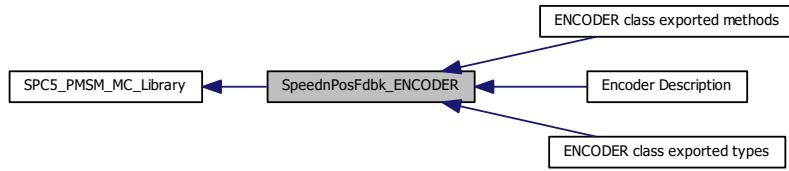


Modules

- `RevupCtrl class private types`
- `RevupCtrl class exported types`
- `RevupCtrl class exported methods`

3.114 SpeednPosFdbk_ENCODER

Collaboration diagram for SpeednPosFdbk_ENCODER:



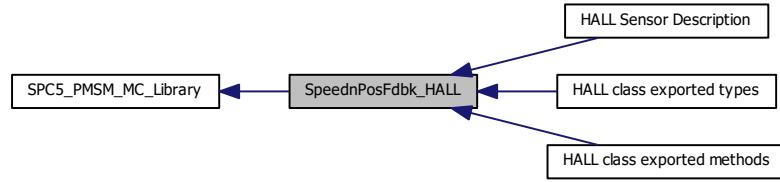
Modules

- `ENCODER class exported types`
- `ENCODER class exported methods`
- `Encoder Description`

Encoder implementation using eTimer.

3.115 SpeednPosFdbk_HALL

Collaboration diagram for SpeednPosFdbk_HALL:



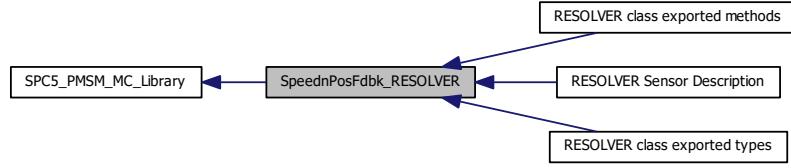
Modules

- [HALL class exported types](#)
- [HALL class exported methods](#)
- [HALL Sensor Description](#)

HALL sensor implementation using eTimer.

3.116 SpeednPosFdbk_RESOLVER

Collaboration diagram for SpeednPosFdbk_RESOLVER:



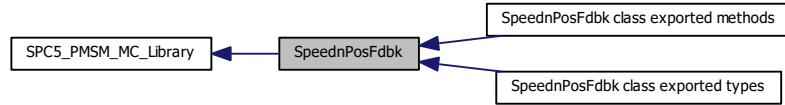
Modules

- RESOLVER class exported types
- RESOLVER class exported methods
- RESOLVER Sensor Description

RESOLVER sensor implementation.

3.117 SpeednPosFdbk

Collaboration diagram for SpeednPosFdbk:



Modules

- `SpeednPosFdbk class exported types`
- `SpeednPosFdbk class exported methods`

3.118 SpeednPosFdbk_STO

Collaboration diagram for SpeednPosFdbk_STO:

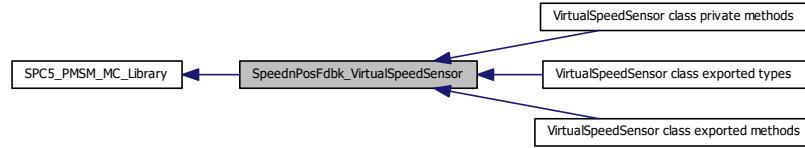


Modules

- [STO class exported types](#)
- [STO class exported methods](#)

3.119 SpeednPosFdbk_VirtualSpeedSensor

Collaboration diagram for SpeednPosFdbk_VirtualSpeedSensor:

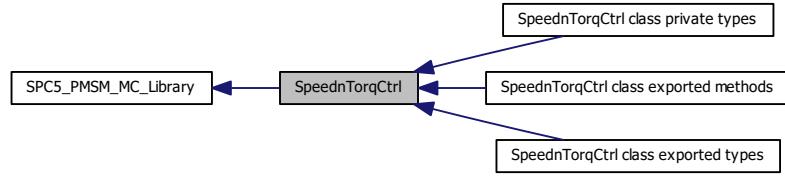


Modules

- [VirtualSpeedSensor class exported types](#)
- [VirtualSpeedSensor class exported methods](#)
- [VirtualSpeedSensor class private methods](#)

3.120 SpeednTorqCtrl

Collaboration diagram for SpeednTorqCtrl:

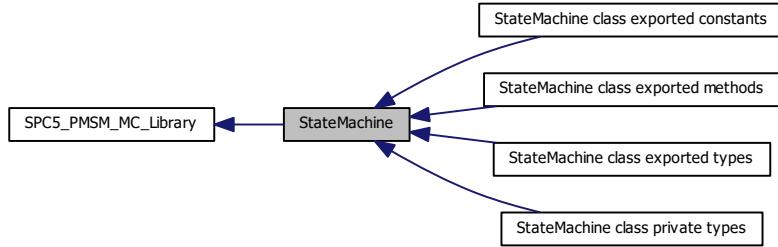


Modules

- `SpeednTorqCtrl` class private types
- `SpeednTorqCtrl` class exported types
- `SpeednTorqCtrl` class exported methods

3.121 StateMachine

Collaboration diagram for StateMachine:

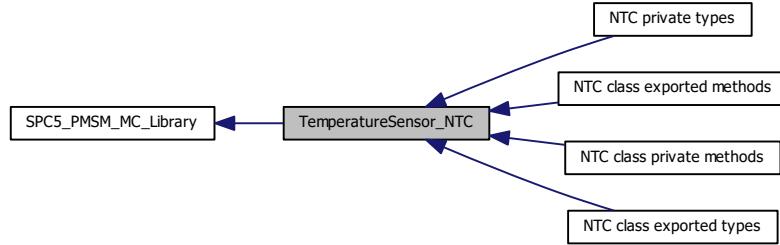


Modules

- `StateMachine` class private types
- `StateMachine` class exported constants
- `StateMachine` class exported types
- `StateMachine` class exported methods

3.122 TemperatureSensor_NTC

Collaboration diagram for TemperatureSensor_NTC:

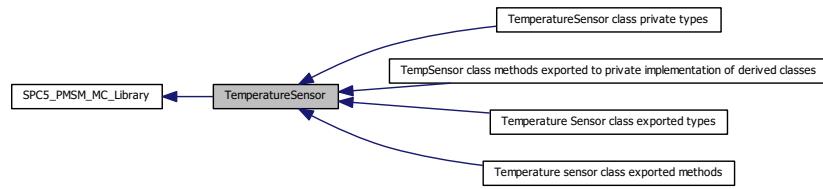


Modules

- `NTC private types`
- `NTC class exported types`
- `NTC class exported methods`
- `NTC class private methods`

3.123 TemperatureSensor

Collaboration diagram for TemperatureSensor:

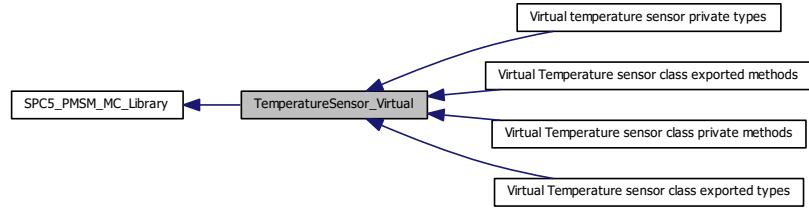


Modules

- TemperatureSensor class private types
- TempSensor class methods exported to private implementation of derived classes
- Temperature Sensor class exported types
- Temperature sensor class exported methods

3.124 TemperatureSensor_Virtual

Collaboration diagram for TemperatureSensor_Virtual:

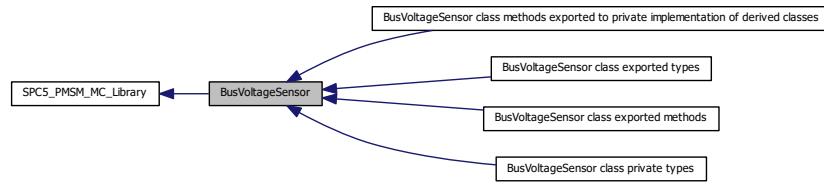


Modules

- Virtual temperature sensor private types
- Virtual Temperature sensor class exported types
- Virtual Temperature sensor class exported methods
- Virtual Temperature sensor class private methods

3.125 BusVoltageSensor

Collaboration diagram for BusVoltageSensor:

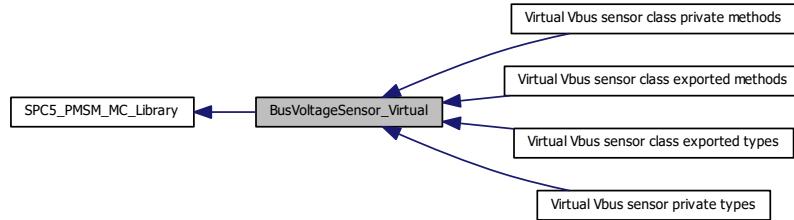


Modules

- `BusVoltageSensor` class private types
- `BusVoltageSensor` class methods exported to private implementation of derived classes
- `BusVoltageSensor` class exported types
- `BusVoltageSensor` class exported methods

3.126 BusVoltageSensor_Virtual

Collaboration diagram for BusVoltageSensor_Virtual:

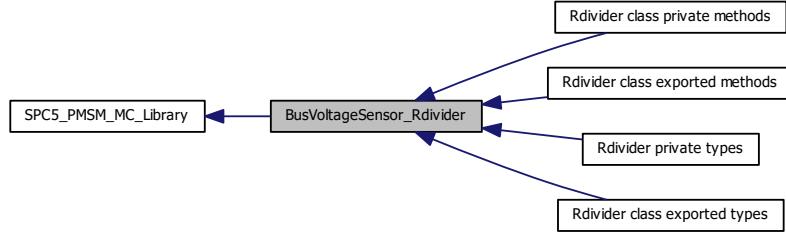


Modules

- Virtual Vbus sensor private types
- Virtual Vbus sensor class exported types
- Virtual Vbus sensor class exported methods
- Virtual Vbus sensor class private methods

3.127 BusVoltageSensor_Rdivider

Collaboration diagram for BusVoltageSensor_Rdivider:



Modules

- `Rdivider private types`
- `Rdivider class exported types`
- `Rdivider class exported methods`
- `Rdivider class private methods`

Chapter 4

SPC5 PMSM UI Library

4.1 UserInterface class exported types

Collaboration diagram for UserInterface class exported types:



Modules

- [UserInterface sensor code](#)
- [UserInterface configuration option](#)

Defines

- `#define UI_IRQ_USART 0u`
- `#define UI_IRQ_CAN 0u`

TypeDefs

- `typedef struct CUI_t * CUI`
Public UserInterface class definition.
- `typedef const void UserInterfaceParams_t`
UserInterface class parameters definition.

4.1.1 Define Documentation

4.1.1.1 `#define UI_IRQ_USART 0u`

Reserved for UIClass serial communication.

Definition at line 48 of file UIIRQHandlerClass.h.

4.1.1.2 #define UI_IRQ_CAN 0u

Reserved for UIClass serial communication.

Definition at line 50 of file UIIRQHandlerClass.h.

Referenced by CAN IRQHandlerRX().

4.1.2 Typedef Documentation

4.1.2.1 typedef struct CUI_t* CUI

Public UserInterface class definition.

Definition at line 135 of file UserInterfaceClass.h.

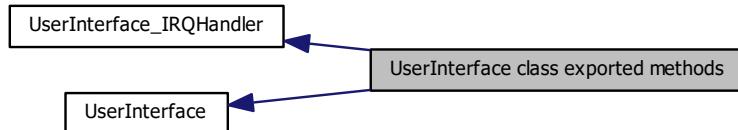
4.1.2.2 typedef const void UserInterfaceParams_t

UserInterface class parameters definition.

Definition at line 140 of file UserInterfaceClass.h.

4.2 UserInterface class exported methods

Collaboration diagram for UserInterface class exported methods:



Functions

- **CUI UI_NewObject** (pUserInterfaceParams_t pUserInterfaceParams)

Creates an object of the class UserInterface.

- **void UI_Init** (CUI this, uint8_t bMCNum, CMCI *pMC1, CMCT *pMCT, uint32_t *pUICfg)

Initialization of UI object. It perform the link between the UI object and the MC interface and MC tuning objects. It must be called before the derived class initialization.

- **bool UI_SelectMC** (CUI this, uint8_t bSelectMC)

It is used to select the MC on which UI operates.

- **uint8_t UI_GetSelectedMC** (CUI this)

It is used to retrieve the MC on which UI currently operates.

- **uint32_t UI_GetSelectedMCConfig** (CUI this)

It is used to retrieve the configuration of the MC on which UI currently operates.

- **bool UI_SetReg** (CUI this, MC_Protocol_REG_t bRegID, int32_t wValue)

It is used to execute a SetReg command coming from the user.

- **int32_t UI_GetReg** (CUI this, MC_Protocol_REG_t bRegID)

It is used to execute a GetReg command coming from the user.

- **CMCT UI_GetCurrentMCT** (CUI this)

It is used to retrieve the current selected MC tuning object.

- **bool UI_ExecCmd** (CUI this, uint8_t bCmdID)

It is used to execute a command coming from the user.

- **bool UI_ExecSpeedRamp** (CUI this, int32_t wFinalMecSpeedRPM, uint16_t hDurationms)

It is used to execute a speed ramp command coming from the user.

- **bool UI_ExecTorqueRamp** (CUI this, int16_t hTargetFinal, uint16_t hDurationms)

It is used to execute a torque ramp command coming from the user.

- **bool UI_GetRevupData** (CUI this, uint8_t bStage, uint16_t *pDurationms, int16_t *pFinalMecSpeed01Hz, int16_t *pFinalTorque)

It is used to execute a get Revup data command coming from the user.

- **bool UI_SetRevupData** (CUI this, uint8_t bStage, uint16_t hDurationms, int16_t hFinalMecSpeed01Hz, int16_t hFinalTorque)

It is used to execute a set Revup data command coming from the user.

- **void UI_SetCurrentReferences** (CUI this, int16_t hlqRef, int16_t hldRef)

It is used to execute a set current reference command coming from the user.

- **void UI_DACInit** (CUI this)

Hardware and software initialization of the DAC object. This is a virtual function and is implemented by related object.

- **void UI_DACExec** (CUI this)

This method is used to update the DAC outputs. The selected variables will be provided in the related output channels. This is a virtual function and is implemented by related object.

- void **UI_SetDAC** (CUI this, DAC_Channel_t bChannel, MC_Protocol_REG_t bVariable)

This method is used to set up the DAC outputs. The selected variables will be provided in the related output channels after next DACEexec. This is a virtual function and is implemented by related object.

- MC_Protocol_REG_t **UI_GetDAC** (CUI this, DAC_Channel_t bChannel)

This method is used to get the current DAC channel selected output.

- void **UI_SetUserDAC** (CUI this, DAC_UserChannel_t bUserChNumber, int16_t hValue)

This method is used to set the value of the "User DAC channel".

- void **UI_LCDInit** (CUI this, CUI oDAC, const char *s_fwVer)

Initialization of LCD object. It must be called after the UI_Init.

- void **UI_LCDEexec** (CUI this)

Execute the LCD execution and refreshing. It must be called periodically.

- void **UI_LCDUpdateAll** (CUI this)

It is used to force a refresh of all LCD values.

- void **UI_LCDUpdateMeasured** (CUI this)

It is used to force a refresh of only measured LCD values.

4.2.1 Function Documentation

4.2.1.1 CUI UI_NewObject (pUserInterfaceParams_t pUserInterfaceParams)

Creates an object of the class UserInterface.

Parameters

<i>pUserInterface- Params</i>	pointer to an UserInterface parameters structure
-----------------------------------	--

Return values

<i>CUI</i>	new instance of UserInterface object
------------	--------------------------------------

Definition at line 71 of file UserInterfaceClass.c.

References MC_NULL, and _CUI_t::pParams_str.

Referenced by DACT_NewObject(), MCP_NewObject(), and UFC_NewObject().

4.2.1.2 void UI_Init (CUI this, uint8_t bMCNum, CMCI * pMCi, CMCT * pMCT, uint32_t * pUICfg)

Initialization of UI object. It perform the link between the UI object and the MC interface and MC tuning objects. It must be called before the derived class initialization.

Parameters

<i>this</i>	related object of class CUI.
<i>bMCNum</i>	Is the total number of MC object present in the list.
<i>pMCi</i>	is the pointer of the list of MC interface objects to be linked with the UI.
<i>pMCT</i>	is the pointer of the list of MC tuning objects to be linked with the UI.
<i>pUICfg</i>	is the pointer of the user interface configuration list. Each element of the list must be a bit field containing one (or more) of the exported configuration option UI_CFGOPT_xxx (eventually OR-ed).

Return values

<i>none.</i>

Definition at line 108 of file UserInterfaceClass.c.

References Vars_t::bDriveNum, Vars_t::bSelectedDrive, Vars_t::pMCI, Vars_t::pMCT, and Vars_t::pUICfg.

4.2.1.3 bool UI_SelectMC (CUI *this*, uint8_t *bSelectMC*)

It is used to select the MC on which UI operates.

Parameters

<i>this</i>	related object of class CUI.
<i>bSelectMC</i>	The new selected MC, zero based, on which UI operates.

Return values

<i>bool</i>	It return true if the bSelectMC is valid oterwise return false.
-------------	---

Definition at line 124 of file UserInterfaceClass.c.

References Vars_t::bDriveNum, and Vars_t::bSelectedDrive.

Referenced by UI_SetReg().

4.2.1.4 uint8_t UI_GetSelectedMC (CUI *this*)

It is used to retrive the MC on which UI currently operates.

Parameters

<i>this</i>	related object of class CUI.
-------------	------------------------------

Return values

<i>uint8_t</i>	It returns the currently selected MC, zero based, on which UI operates.
----------------	---

Definition at line 145 of file UserInterfaceClass.c.

Referenced by UI_GetReg().

4.2.1.5 uint32_t UI_GetSelectedMCConfig (CUI *this*)

It is used to retrive the configuration of the MC on which UI currently operates.

Parameters

<i>this</i>	related object of class CUI.
-------------	------------------------------

Return values

<i>uint32_t</i>	It returns the currently configuration of selected MC on which UI operates. It represents a bit field containing one (or more) of the exported configuration option UI_CFGOPT_xxx (eventually OR-ed).
-----------------	---

Parameters

<i>this</i>	related object of class CUI.
-------------	------------------------------

Return values

<i>uint16_t</i>	It returns the currently configuration of selected MC on which UI operates. It represents a bit field containing one (or more) of the exported configuration option UI_CFGOPT_xxx (eventually OR-ed).
-----------------	---

Definition at line 159 of file UserInterfaceClass.c.

References Vars_t::bSelectedDrive, and Vars_t::pUICfg.

4.2.1.6 bool UI_SetReg (CUI *this*, MC_Protocol_REG_t *bRegID*, int32_t *wValue*)

It is used to execute a SetReg command coming from the user.

Parameters

<i>this</i>	related object of class CUI.
<i>bRegID</i>	Code of register to be updated. Valid code is one of the MC_PROTOCOL_REG_xxx values exported by UserInterfaceClass.
<i>wValue</i>	is the new value to be set.

Return values

<i>bool</i>	It returns true if the SetReg command has been performed successfully otherwise returns false.
-------------	--

Parameters

<i>this</i>	related object of class CUI.
<i>bRegID</i>	Code of register to be updated. Valid code is one of the MC_PROTOCOL_REG_xxx values exported by UserInterfaceClass.
<i>wValue</i>	is the new value to be set.

Return values

<i>uint8_t</i>	It returns the currently selected MC, zero based, on which UI operates.
----------------	---

Definition at line 187 of file UserInterfaceClass.c.

References Vars_t::bSelectedDrive, FF_GetFFConstants(), FF_SetFFConstants(), FW_SetVref(), HFI_FP_GetPI-Track(), HFI_FP_SetMinSaturationDifference(), MC_NULL, MCI_ExecSpeedRamp(), MCI_ExecTorqueRamp(), MCI_GetIqdref(), MCI_GetMecSpeedRef01Hz(), MCI_GetTeref(), MCI_SetCurrentReferences(), MCI_SetIdref(), MC-T_GetFeedForwardCtrl(), MCT_GetFluxWeakeningCtrl(), MCT_GetFluxWeakeningLoopPID(), MCT_GetHFIctrl(), MCT_GetIdLoopPID(), MCT_GetIqdLoopPID(), MCT_GetSpeedLoopPID(), MCT_SetSpeednPosSensorAuxiliary(), MCT_SetSpeednPosSensorMain(), PI_SetKI(), PI_SetKP(), PID_SetKD(), Vars_t::pMCI, Vars_t::pMCT, Vars_t::pUICfg, STC_SPEED_MODE, STC_TORQUE_MODE, STO_CR_GetObserverGains(), STO_CR_SetObserverGains(), STO_GetObserverGains(), STO_SetPLLGains(), STO_SetObserverGains(), STO_SetPLLGains(), UI_CFGOPT_PFC_I_KD, UI_CFGOPT_PFC_V_KD, UI_SCODE_HFINJ, UI_SCODE_STO_CR, UI_SCODE_STO_PLL, and UI_SelectMC().

4.2.1.7 int32_t UI_GetReg (CUI *this*, MC_Protocol_REG_t *bRegID*)

It is used to execute a GetReg command coming from the user.

Parameters

<i>this</i>	related object of class CUI.
<i>bRegID</i>	Code of register to be updated. Valid code is one of the MC_PROTOCOL_REG_xxx values exported by UserInterfaceClass.

Return values

<i>int32_t</i>	is the current value of register bRegID.
----------------	--

Definition at line 615 of file UserInterfaceClass.c.

References Vars_t::bSelectedDrive, FF_GetFFConstants(), FF_GetVqdAvPlout(), FF_GetVqdff(), FW_GetAvVPercentage(), FW_GetVref(), HFI_FP_GetCurrent(), HFI_FP_GetPITrack(), HFI_FP_GetRotorAngleLock(), HFI_F_P_GetSaturationDifference(), MC_NULL, MCI_GetAvrgMecSpeed01Hz(), MCI_GetControlMode(), MCI_Getlab(), MCI_Getlalphabet(), MCI_Getlqd(), MCI_Getlqdref(), MCI_GetLastRampFinalSpeed(), MCI_GetMecSpeedRef01Hz(), MCI_GetValphabet(), MCI_GetVqd(), MCT_GetBusVoltageSensor(), MCT_GetFeedForwardCtrl(), MCT_GetFluxWeakeningCtrl(), MCT_GetFluxWeakeningLoopPID(), MCT_GetHFIctrl(), MCT_GetIdLoopPID(), MCT_GetIqLoopPID(), MCT_GetMotorPowerMeasurement(), MCT_GetRevupCtrl(), MCT_GetSpeedLoopPID(), MCT_GetSpeednPosSensorAuxiliary(), MCT_GetSpeednPosSensorMain(), MCT_GetSpeednTorqueController(), MCT_GetStateMachine(), MCT_GetTemperatureSensor(), MPM_GetAvrgEIMotorPowerW(), PI_GetKI(), PI_GetKP(), PID_GetKD(), Vars_t::pMCI, Vars_t::pMCT, Vars_t::pUICfg, RUC_GetNumberOfPhases(), SPD_GetEIAngle(), SPD_GetS16Speed(), STC_GetMaxAppPositiveMecSpeed01Hz(), STC_GetMinAppNegativeMecSpeed01Hz(), STC_SPEED_MODE, STM_GetFaultState(), STM_GetState(), STO_CR_GetEstimatedBemf(), STO_CR_GetEstimatedBemfLevel(), STO_CR_GetEstimatedCurrent(), STO_CR_GetObservedBemfLevel(), STO_CR_GetObserverGains(), STO_GetEstimatedBemf(), STO_GetEstimatedBemfLevel(), STO_GetEstimatedCurrent(), STO_GetObservedBemfLevel(), STO_GetObserverGains(), STO_GetPLLGains(), TSNS_GetAvTemp_C(), UI_CFGOPT_PFC_I_KD, UI_CFGOPT_PFC_V_KD, UI_GetDAC(), UI_GetSelectedMC(), UI_SCODE_ENC, UI_SCODE_H_ALL, UI_SCODE_HFINJ, UI_SCODE_RES, UI_SCODE_STO_CR, UI_SCODE_STO_PLL, and VBS_GetAvBus-Voltage_V().

4.2.1.8 CMCT UI_CurrentMCT (CUI *this*)

It is used to retrieve the current selected MC tuning object.

Parameters

<i>this</i>	related object of class CUI.
-------------	------------------------------

Return values

<i>CMCT</i>	It returns the currently selected MC tuning object on which UI operates.
-------------	--

Definition at line 172 of file UserInterfaceClass.c.

References Vars_t::bSelectedDrive, and Vars_t::pMCT.

4.2.1.9 bool UI_ExecCmd (CUI *this*, uint8_t *bCmdID*)

It is used to execute a command coming from the user.

Parameters

<i>this</i>	related object of class CUI.
<i>bCmdID</i>	Code of register to be updated. Valid code is one of the MC_PROTOCOL_CMD_xxx define exported by UserInterfaceClass.

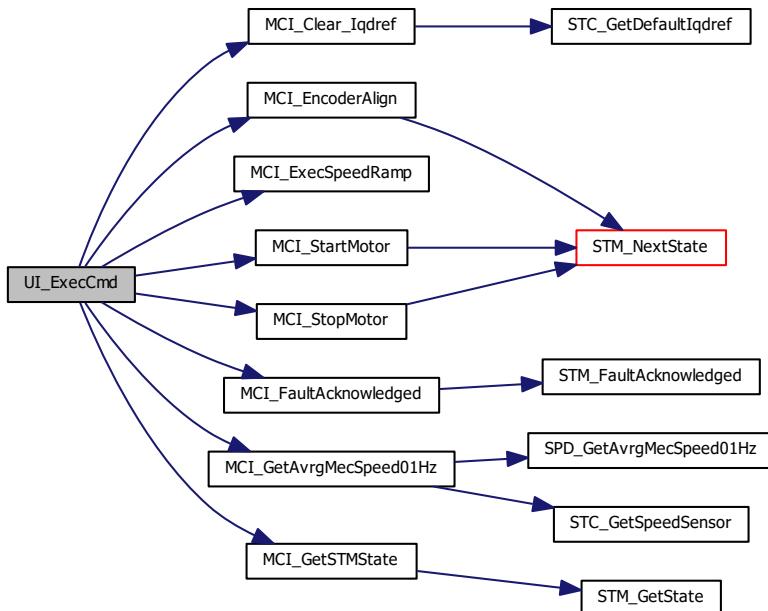
Return values

<i>bool</i>	It returns true if the command has been performed successfully otherwise returns false.
-------------	---

Definition at line 1641 of file UserInterfaceClass.c.

References Vars_t::bSelectedDrive, IDLE, MCI_Clear_Iqdref(), MCI_EncoderAlign(), MCI_ExecSpeedRamp(), MCI_FaultAcknowledged(), MCI_GetAvrgMecSpeed01Hz(), MCI_GetSTMState(), MCI_StartMotor(), MCI_StopMotor(), Vars_t::pMCI, Vars_t::pUICfg, RUN, and UI_CFGOPT_PFC.

Here is the call graph for this function:



4.2.1.10 bool UI_ExecSpeedRamp (CUI *this*, int32_t *wFinalMecSpeedRPM*, uint16_t *hDurationms*)

It is used to execute a speed ramp command coming from the user.

Parameters

<i>this</i>	related object of class CUI.
<i>wFinalMecSpeedRPM</i>	final speed value expressed in RPM.
<i>hDurationms</i>	the duration of the ramp expressed in milliseconds. It is possible to set 0 to perform an instantaneous change in the value.

Return values

<i>bool</i>	It returns true if the command has been performed successfully otherwise returns false.
-------------	---

Definition at line 1764 of file UserInterfaceClass.c.

References Vars_t::bSelectedDrive, MCI_ExecSpeedRamp(), and Vars_t::pMCI.

Here is the call graph for this function:



4.2.1.11 bool UI_ExecTorqueRamp (CUI this, int16_t hTargetFinal, uint16_t hDurationms)

It is used to execute a torque ramp command coming from the user.

Parameters

<i>this</i>	related object of class CUI.
<i>hTargetFinal</i>	final torque value. See MCI interface for more details.
<i>hDurationms</i>	the duration of the ramp expressed in milliseconds. It is possible to set 0 to perform an instantaneous change in the value.

Return values

<i>bool</i>	It returns true if the command has been performed successfully otherwise returns false.
-------------	---

Definition at line 1784 of file UserInterfaceClass.c.

References Vars_t::bSelectedDrive, MCI_ExecTorqueRamp(), and Vars_t::pMCI.

Here is the call graph for this function:



4.2.1.12 bool UI_GetRevupData (CUI this, uint8_t bStage, uint16_t * pDurationms, int16_t * pFinalMecSpeed01Hz, int16_t * pFinalTorque)

It is used to execute a get Revup data command coming from the user.

Parameters

<i>this</i>	related object of class CUI.
<i>bStage</i>	is the rev up phase, zero based, to be read.
<i>pDurationms</i>	is the pointer to an uint16_t variable used to retrieve the duration of the Revup stage.
<i>pFinalMecSpeed01Hz</i>	is the pointer to an int16_t variable used to retrieve the mechanical speed at the end of that stage expressed in 0.1Hz.

<i>pFinalTorque</i>	is the pointer to an int16_t variable used to retrieve the value of motor torque at the end of that stage. This value represents actually the Iq current expressed in digit.
---------------------	--

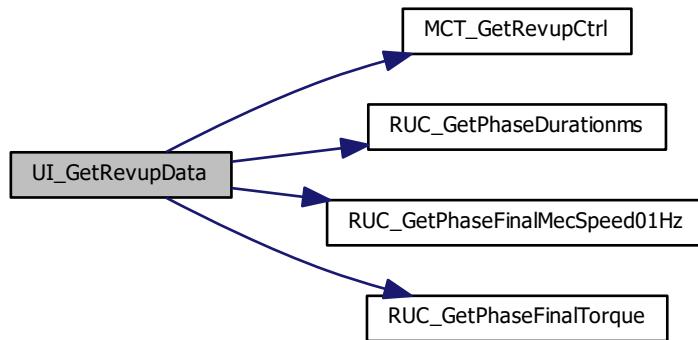
Return values

<i>bool</i>	It returns true if the command has been performed successfully otherwise returns false.
-------------	---

Definition at line 1810 of file UserInterfaceClass.c.

References Vars_t::bSelectedDrive, MCT_GetRevupCtrl(), Vars_t::pMCT, RUC_SetPhaseDurationms(), RUC_SetPhaseFinalMecSpeed01Hz(), and RUC_SetPhaseFinalTorque().

Here is the call graph for this function:



4.2.1.13 bool UI_SetRevupData (CUI *this*, uint8_t *bStage*, uint16_t *hDurationms*, int16_t *hFinalMecSpeed01Hz*, int16_t *hFinalTorque*)

It is used to execute a set Revup data command coming from the user.

Parameters

<i>this</i>	related object of class CUI.
<i>bStage</i>	is the rev up phase, zero based, to be modified.
<i>hDurationms</i>	is the new duration of the Revup stage.
<i>hFinalMec-Speed01Hz</i>	is the new mechanical speed at the end of that stage expressed in 0.1Hz.
<i>hFinalTorque</i>	is the new value of motor torque at the end of that stage. This value represents actually the Iq current expressed in digit.

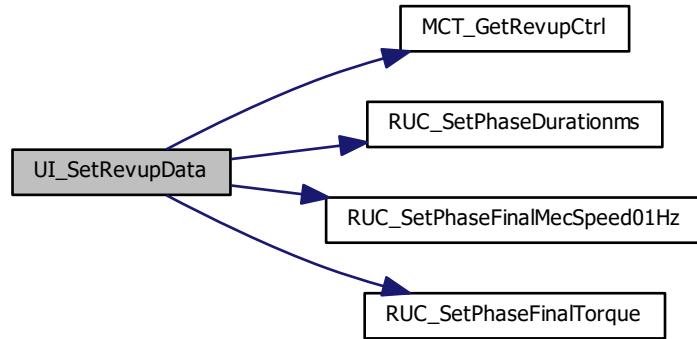
Return values

<i>bool</i>	It returns true if the command has been performed successfully otherwise returns false.
-------------	---

Definition at line 1843 of file UserInterfaceClass.c.

References Vars_t::bSelectedDrive, MCT_SetRevupCtrl(), Vars_t::pMCT, RUC_SetPhaseDurationms(), RUC_SetPhaseFinalMecSpeed01Hz(), and RUC_SetPhaseFinalTorque().

Here is the call graph for this function:



4.2.1.14 void UI_SetCurrentReferences (CUI this, int16_t hIqRef, int16_t hIdRef)

It is used to execute a set current reference command coming from the user.

Parameters

<i>this</i>	related object of class CUI.
<i>hIqRef</i>	is the current Iq reference on qd reference frame. This value is expressed in digit. To convert current expressed in digit to current expressed in Amps is possible to use the formula: Current(Amp) = [Current(digit) * Vdd micro] / [65536 * Rshunt * Aop]
<i>hIdRef</i>	is the current Id reference on qd reference frame. This value is expressed in digit. See <i>hIqRef</i> param description.

Return values

none.

Definition at line 1867 of file UserInterfaceClass.c.

References Vars_t::bSelectedDrive, MCI_SetCurrentReferences(), and Vars_t::pMCI.

Here is the call graph for this function:



4.2.1.15 void UI_DACInit (CUI *this*)

Hardware and software initialization of the DAC object. This is a virtual function and is implemented by related object.

Parameters

<i>this</i>	related object of class UI. It must be a DACx_UI object casted to CUI otherwise the DACInit method will have no effect.
-------------	---

Return values

<i>none.</i>

Definition at line 1884 of file UserInterfaceClass.c.

4.2.1.16 void UI_DACExec (CUI *this*)

This method is used to update the DAC outputs. The selected variables will be provided in the related output channels. This is a virtual function and is implemented by related object.

Parameters

<i>this</i>	related object of class UI. It must be a DACx_UI object casted to CUI otherwise the DACInit method will have no effect.
-------------	---

Return values

<i>none.</i>

Definition at line 1900 of file UserInterfaceClass.c.

4.2.1.17 void UI_SetDAC (CUI *this*, DAC_Channel_t *bChannel*, MC_Protocol_REG_t *bVariable*)

This method is used to set up the DAC outputs. The selected variables will be provided in the related output channels after next DACExec. This is a virtual function and is implemented by related object.

Parameters

<i>this</i>	related object of class UI. It must be a DACx_UI object casted to CUI otherwise the DACInit method will have no effect.
<i>bChannel</i>	the DAC channel to be programmed. It must be one of the exported channels Ex. DAC_CH0.
<i>bVariable</i>	the variables to be provided in out through the selected channel. It must be one of the exported UI register Ex. MC_PROTOCOL_REG_I_A.

Return values

<i>none.</i>	This method is used to set up the DAC outputs. The selected variables will be provided in the related output channels after next DACExec. This is a virtual function and is implemented by related object.
--------------	--

Parameters

<i>this</i>	related object of class UI. It must be a DACx_UI object casted to CUI otherwise the method will have no effect.
<i>bChannel</i>	the DAC channel to be programmed. It must be one of the exported channels Ex. DAC_CH0.

<i>bVariable</i>	the variables to be provided in out through the selected channel. It must be one of the exported UI register Ex. MC_PROTOCOL_REG_I_A.
------------------	---

Return values

<i>none.</i>

Definition at line 1921 of file UserInterfaceClass.c.

Referenced by MC_SetDAC().

4.2.1.18 MC_Protocol_REG_t UI_GetDAC (CUI *this*, DAC_Channel_t *bChannel*)

This method is used to get the current DAC channel selected output.

Parameters

<i>this</i>	related object of class UI. It must be a DACx_UI object casted to CUI otherwise the method will have no effect.
<i>bChannel</i>	the inspected DAC channel. It must be one of the exported channels (Ex. DAC_CH0).

Return values

<i>MC_Protocol_REG_t</i>	The variables provided in out through the inspected channel. It will be one of the exported UI register (Ex. MC_PROTOCOL_REG_I_A).
--------------------------	--

Definition at line 1940 of file UserInterfaceClass.c.

Referenced by UI_GetReg().

4.2.1.19 void UI_SetUserDAC (CUI *this*, DAC_UserChannel_t *bUserChNumber*, int16_t *hValue*)

This method is used to set the value of the "User DAC channel".

Parameters

<i>this</i>	related object of class UI. It must be a DACx_UI object casted to CUI otherwise the DACInit method will have no effect.
<i>bUserChNumber</i>	the "User DAC channel" to be programmed.
<i>hValue</i>	the value to be put in output.

Return values

<i>none.</i>

Parameters

<i>this</i>	related object of class UI. It must be a DACx_UI object casted to CUI otherwise the method will have no effect.
<i>bUserChNumber</i>	the "User DAC channel" to be programmed.
<i>hValue</i>	the value to be put in output.

Return values

<i>none.</i>

Definition at line 1958 of file UserInterfaceClass.c.

Referenced by MC_SetUserDAC().

4.2.1.20 void UI_LCDInit (CUI *this*, CUI *oDAC*, const char * *s_fwVer*)

Initialization of LCD object. It must be called after the UI_Init.

Parameters

<i>this</i>	related object of class CUI. It must be a LCDx_UI object casted to CUI otherwise the method will have no effect.
<i>oDAC</i>	related DAC object upcasted to CUI. It can be MC_NULL.
<i>s_fwVer</i>	String containing firmware version.

Return values

<i>none.</i>

Definition at line 1975 of file UserInterfaceClass.c.

4.2.1.21 void UI_LCDExec (CUI *this*)

Execute the LCD execution and refreshing. It must be called periodically.

Parameters

<i>this</i>	related object of class CUI. It must be a LCDx_UI object casted to CUI otherwise the method will have no effect.
-------------	--

Return values

<i>none.</i>

Definition at line 1990 of file UserInterfaceClass.c.

4.2.1.22 void UI_LCDUpdateAll (CUI *this*)

It is used to force a refresh of all LCD values.

Parameters

<i>this</i>	related object of class CUI. It must be a LCDx_UI object casted to CUI otherwise the method will have no effect.
-------------	--

Return values

<i>none.</i>

Definition at line 2004 of file UserInterfaceClass.c.

4.2.1.23 void UI_LCDUpdateMeasured (CUI *this*)

It is used to force a refresh of only measured LCD values.

Parameters

<i>this</i>	related object of class CUI. It must be a LCDx_UI object casted to CUI otherwise the method will have no effect.
-------------	--

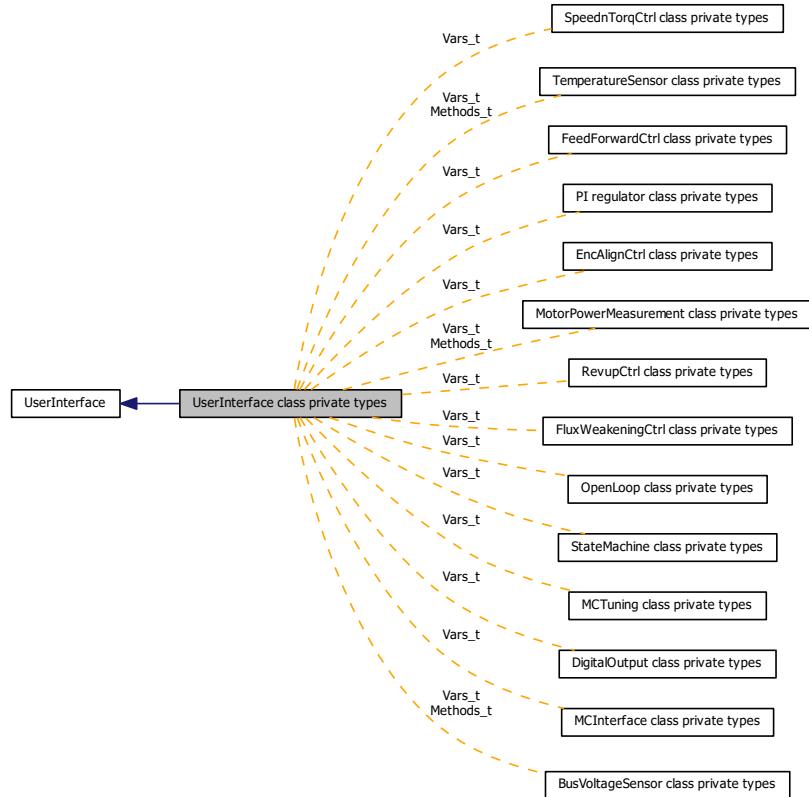
Return values

<i>none.</i>

Definition at line 2018 of file UserInterfaceClass.c.

4.3 UserInterface class private types

Collaboration diagram for UserInterface class private types:



Data Structures

- struct [Vars_t](#)
MCInterface class members definition.
- struct [Methods_t](#)
Virtual methods container.
- struct [_CUI_t](#)
Private UserInterface class definition.

TypeDefs

- typedef [UserInterfaceParams_t Params_t](#)
Redefinition of parameter structure.

4.3.1 Typedef Documentation

4.3.1.1 typedef UserInterfaceParams_t Params_t

Redefinition of parameter structure.

Definition at line 62 of file UserInterfacePrivate.h.

4.4 UserInterface sensor code

Collaboration diagram for UserInterface sensor code:



Defines

- #define UI_SCODE_HALL 0x1u
- #define UI_SCODE_ENC 0x2u
- #define UI_SCODE_RES 0x3u
- #define UI_SCODE_STO_PLL 0x9u
- #define UI_SCODE_STO_CR 0xAu
- #define UI_SCODE_HFINJ 0xBu

4.4.1 Define Documentation

4.4.1.1 #define UI_SCODE_HALL 0x1u

This code identifies the Hall sensor.

Definition at line 61 of file UserInterfaceClass.h.

Referenced by UI_GetReg().

4.4.1.2 #define UI_SCODE_ENC 0x2u

This code identifies the Encoder sensor.

Definition at line 62 of file UserInterfaceClass.h.

Referenced by UI_GetReg().

4.4.1.3 #define UI_SCODE_RES 0x3u

This code identifies the Resolver sensor.

Definition at line 63 of file UserInterfaceClass.h.

Referenced by UI_GetReg().

4.4.1.4 #define UI_SCODE_STO_PLL 0x9u

This code identifies the State observer + PLL sensor.

Definition at line 64 of file UserInterfaceClass.h.

Referenced by UI_GetReg(), and UI_SetReg().

4.4.1.5 #define UI_SCODE_STO_CR 0xAu

This code identifies the State observer + CORDIC sensor.

Definition at line 65 of file UserInterfaceClass.h.

Referenced by UI_GetReg(), and UI_SetReg().

4.4.1.6 #define UI_SCODE_HFINJ 0xBu

This code identifies the HF injection sensor.

Definition at line 66 of file UserInterfaceClass.h.

Referenced by UI_GetReg(), and UI_SetReg().

4.5 UserInterface configuration option

Collaboration diagram for UserInterface configuration option:



Defines

- #define `UI_CFGOPT_NONE` 0x00000000u
- #define `UI_CFGOPT_FW` 0x00000001u
- #define `UI_CFGOPT_SPEED_KD` 0x00000002u
- #define `UI_CFGOPT_Iq_KD` 0x00000004u
- #define `UI_CFGOPT_Id_KD` 0x00000008u
- #define `UI_CFGOPT_DAC` 0x00000010u
- #define `UI_CFGOPT_SETIDINSPDMODE` 0x00000020u
- #define `UI_CFGOPT_PLLTUNING` 0x00000040u
- #define `UI_CFGOPT_PFC` 0x00000080u
- #define `UI_CFGOPT_PFC_I_KD` 0x00000100u
- #define `UI_CFGOPT_PFC_V_KD` 0x00000200u

4.5.1 Define Documentation

4.5.1.1 #define `UI_CFGOPT_NONE` 0x00000000u

Enable this option when no other option is selected.

Definition at line 75 of file UserInterfaceClass.h.

4.5.1.2 #define `UI_CFGOPT_FW` 0x00000001u

Enable this option when the flux weakening is enabled in the MC firmware.

Definition at line 77 of file UserInterfaceClass.h.

4.5.1.3 #define `UI_CFGOPT_SPEED_KD` 0x00000002u

Enable this option when the speed controller has derivative action.

Definition at line 80 of file UserInterfaceClass.h.

4.5.1.4 #define `UI_CFGOPT_Iq_KD` 0x00000004u

Enable this option when the Iq controller has derivative action.

Definition at line 83 of file UserInterfaceClass.h.

4.5.1.5 #define UI_CFGOPT_Id_KD 0x00000008u

Enable this option when the Id controller has derivative action.

Definition at line 86 of file UserInterfaceClass.h.

4.5.1.6 #define UI_CFGOPT_DAC 0x00000010u

Enable this option if a DAC object will be associated with the UI.

Definition at line 89 of file UserInterfaceClass.h.

4.5.1.7 #define UI_CFGOPT_SETIDINSPDMODE 0x00000020u

Enable this option to allow setting the Id reference when MC is in speed mode.

Definition at line 91 of file UserInterfaceClass.h.

4.5.1.8 #define UI_CFGOPT_PLLTUNING 0x00000040u

Enable this option to allow setting the PLL KP and KI.

Definition at line 94 of file UserInterfaceClass.h.

4.5.1.9 #define UI_CFGOPT_PFC 0x00000080u

Enable this option to allow PFC tuning.

Definition at line 96 of file UserInterfaceClass.h.

Referenced by UI_ExecCmd().

4.5.1.10 #define UI_CFGOPT_PFC_I_KD 0x00000100u

Enable this option when PFC current controller has derivative action.

Definition at line 98 of file UserInterfaceClass.h.

Referenced by UI_GetReg(), and UI_SetReg().

4.5.1.11 #define UI_CFGOPT_PFC_V_KD 0x00000200u

Enable this option when PFC voltage controller has derivative action.

Definition at line 100 of file UserInterfaceClass.h.

Referenced by UI_GetReg(), and UI_SetReg().

4.6 Motor Control Protocol class exported types

Collaboration diagram for Motor Control Protocol class exported types:



Data Structures

- struct [MCPParams_t](#)
MotorControlProtocol class parameters definition.

4.7 Motor Control Protocol class exported methods

Collaboration diagram for Motor Control Protocol class exported methods:



Functions

- CMCP_UI [MCP_NewObject](#) (pUserInterfaceParams_t pUserInterfaceParams, pMCPPParams_t pMCPPParams)

Creates an object of the class MotorControlProtocol.

4.7.1 Function Documentation

4.7.1.1 CMCP_UI MCP_NewObject (pUserInterfaceParams_t pUserInterfaceParams, pMCPPParams_t pMCPPParams)

Creates an object of the class MotorControlProtocol.

Parameters

<i>pMCPPParam,:</i>	MotorControlProtocol parameters
---------------------	---------------------------------

Return values

<i>oMCP_MCUI,::</i>	new MotorControlProtocol object
---------------------	---------------------------------

Definition at line 74 of file MotorControlProtocolClass.c.

References [_CUI_t::DerivedClass](#), [MC_NULL](#), and [UI_NewObject\(\)](#).

Here is the call graph for this function:



4.8 Frame Communication Protocol class exported types

Collaboration diagram for Frame Communication Protocol class exported types:

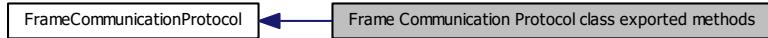


Data Structures

- struct [FCPPParams_t](#)
FrameCommunicationProtocol class parameters definition.

4.9 Frame Communication Protocol class exported methods

Collaboration diagram for Frame Communication Protocol class exported methods:



Functions

- CFCP [FCP_NewObject \(pFCPPParams_t pFCPPParam\)](#)
Creates an object of the class FrameCommunicationProtocol.

4.9.1 Function Documentation

4.9.1.1 CFCP FCP_NewObject (pFCPPParams_t pFCPPParam)

Creates an object of the class FrameCommunicationProtocol.

Parameters

<i>pSensorParam,:</i>	Physical Layer parameters
-----------------------	---------------------------

Return values

<i>oFCP,:</i>	new Physical Layer object
---------------	---------------------------

Definition at line 56 of file FrameCommunicationProtocolClass.c.

References MC_NULL.

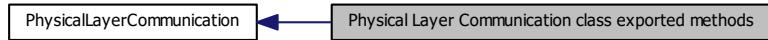
4.10 Physical Layer Communication class exported types

Collaboration diagram for Physical Layer Communication class exported types:



4.11 Physical Layer Communication class exported methods

Collaboration diagram for Physical Layer Communication class exported methods:



Functions

- CCOM COM_NewObject (void)
Creates an object of the class "Physical Layer Communication".
- void COM_StartReceive (CCOM this)
Start receive from the channel (IRQ enabling)
- void COM_StartTransmit (CCOM this)
Start transmit to the channel (IRQ enabling)

4.11.1 Function Documentation

4.11.1.1 CCOM COM_NewObject (void)

Creates an object of the class "Physical Layer Communication".

Parameters

<i>pSensorParam,:</i>	Physical Layer parameters
-----------------------	---------------------------

Return values

<i>oCOM,:</i>	new Physical Layer object
---------------	---------------------------

Definition at line 49 of file PhysicalLayerCommunication_Class.c.

References MC_NULL.

Referenced by CAN_NewObject(), and USART_NewObject().

4.11.1.2 void COM_StartReceive (CCOM this)

Start receive from the channel (IRQ enabling)

Parameters

<i>this,:</i>	COM object
---------------	------------

Return values

<i>None</i>

Definition at line 146 of file PhysicalLayerCommunication_Class.c.

4.11.1.3 void COM_StartTransmit (CCOM *this*)

Start transmit to the channel (IRQ enabling)

Parameters

<i>this</i> :	COM object
---------------	------------

Return values

<i>None</i>

Definition at line 156 of file PhysicalLayerCommunication_Class.c.

4.12 USART physical layer communication class exported types

Collaboration diagram for USART physical layer communication class exported types:



Data Structures

- struct [USART_TypeDef](#)
Universal Synchronous Asynchronous Receiver Transmitter.
- struct [USARTParams_t](#)
UserInterface class parameters definition.

Defines

- #define [__I volatile const](#)
- #define [__O volatile](#)
- #define [__IO volatile](#)
- #define [USART_CR1_RE](#) ((uint32_t)0x00000004)
- #define [USART_CR1_TE](#) ((uint32_t)0x00000008)

Typedefs

- typedef struct CUSART_COM_t * [CUSART_COM](#)
Public UserInterface class definition.

4.12.1 Define Documentation

4.12.1.1 #define __I volatile const

IO definitions

define access restrictions to peripheral registers defines 'read only' permissions

Definition at line 62 of file USART_PhysicalLayerCommunication_Class.h.

4.12.1.2 #define __O volatile

defines 'write only' permissions

Definition at line 64 of file USART_PhysicalLayerCommunication_Class.h.

4.12.1.3 #define __IO volatile

defines 'read / write' permissions

Definition at line 65 of file USART_PhysicalLayerCommunication_Class.h.

4.12.1.4 `#define USART_CR1_RE ((uint32_t)0x00000004)`

Receiver Enable

Definition at line 148 of file USART_PhysicalLayerCommunication_Class.h.

4.12.1.5 `#define USART_CR1_TE ((uint32_t)0x00000008)`

Transmitter Enable

Definition at line 149 of file USART_PhysicalLayerCommunication_Class.h.

4.12.2 Typedef Documentation

4.12.2.1 `typedef struct CUSART_COM_t* CUSART_COM`

Public UserInterface class definition.

Definition at line 155 of file USART_PhysicalLayerCommunication_Class.h.

4.13 USART physical layer communication class exported methods

Collaboration diagram for USART physical layer communication class exported methods:



Functions

- **CUSART_COM USART_NewObject (pUSARTParams_t pUSARTParams)**
Creates an object of the class CUSART_COM_t.
- void **USART_ITConfig (SerialDriver *USARTx, uint32_t USART_IT, FunctionalState NewState)**
Enables or disables the specified USART interrupts.
- void **USART_SendData (SerialDriver *USARTx, uint16_t Data)**
Transmits single data through the USARTx peripheral.

4.13.1 Function Documentation

4.13.1.1 CUSART_COM USART_NewObject (pUSARTParams_t pUSARTParams)

Creates an object of the class CUSART_COM_t.

Parameters

<i>pUSARTParams</i>	pointer to pUSARTParams_t structure
---------------------	-------------------------------------

Return values

<i>CUSART_COM</i>	new instance of CUSART_COM object
-------------------	-----------------------------------

Creates an object of the class CUSART_COM_t.

Parameters

<i>pSensorParam,:</i>	Physical Layer parameters
-----------------------	---------------------------

Return values

<i>oCOM,:</i>	new Physical Layer object
---------------	---------------------------

Definition at line 64 of file USART_PhysicalLayerCommunication_Class.c.

References COM_NewObject(), and MC_NULL.

Here is the call graph for this function:



4.13.1.2 void USART_ITConfig (SerialDriver * USARTx, uint32_t USART_IT, FunctionalState NewState)

Enables or disables the specified USART interrupts.

Parameters

<i>USARTx</i> ,	where x can be 1, 2, 3, 4, 5 or 6 to select the USART or UART peripheral.
<i>USART_IT</i> :	<p>specifies the USART interrupt sources to be enabled or disabled. This parameter can be one of the following values:</p> <ul style="list-style-type: none"> • USART_IT_CTS: CTS change interrupt • USART_IT_LBD: LIN Break detection interrupt • USART_IT_TXE: Transmit Data Register empty interrupt • USART_IT_TC: Transmission complete interrupt • USART_IT_RXNE: Receive Data register not empty interrupt • USART_IT_IDLE: Idle line detection interrupt • USART_IT_PE: Parity Error interrupt • USART_IT_ERR: Error interrupt(Frame error, noise error, overrun error)
<i>NewState</i> ,	new state of the specified USARTx interrupts. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>

Definition at line 205 of file USART_PhysicalLayerCommunication_Class.c.

Referenced by UFC_Init().

4.13.1.3 void USART_SendData (SerialDriver * USARTx, uint16_t Data)

Transmits single data through the USARTx peripheral.

Parameters

<i>USARTx</i> ,	Select the USART or the UART peripheral. This parameter can be one of the following values: USART1, USART2, USART3, USART4 or USART5.
<i>Data</i> ,	the data to transmit.

Return values

<i>None</i>

Definition at line 249 of file USART_PhysicalLayerCommunication_Class.c.

Referenced by UFC_StartCom().

4.14 Can Physical Layer Communication class Description

Collaboration diagram for Can Physical Layer Communication class Description:



CAN Physical Layer Communication class implementation. Applications on the market, that require an electrical motor to be driven, usually have the electronics split in two parts: application board and motor drive board.

To drive the system correctly, the application board requires a method to send a command to the motor drive board and get a feedback.

This is usually performed using a CAN communication.

See Figure 1: "Can communication in motor control application".



Figure 4.1: Figure 1: Can Communication in Motor Control application.

Motor Control library is designed using an object oriented methodology.

Interface is managed through a sequence of classes to keep separated the physical communication(uart, can,...) the transport protocol and the messages (motor control protocol)

CAN feature has been added in a coherency with this class structure by adding a specific CCAN_COM class to manage the can phy communication

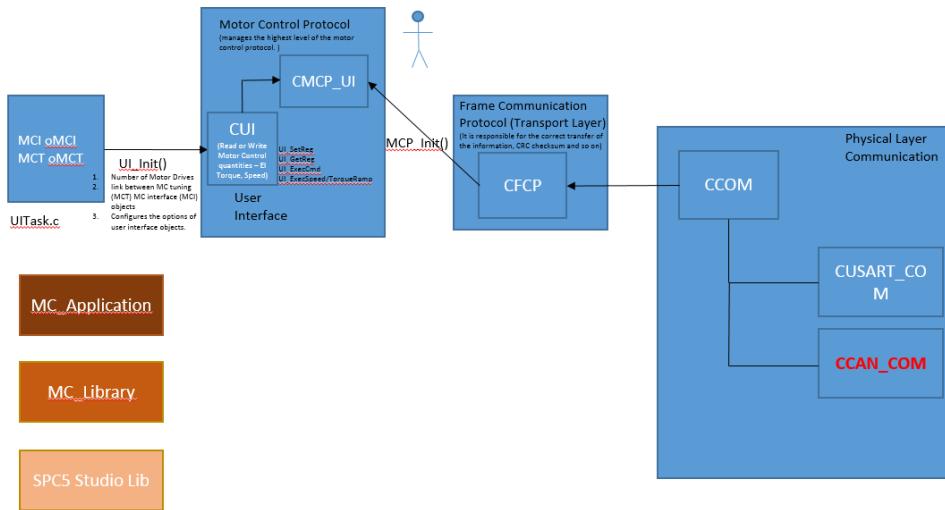


Figure 4.2: Figure 2: Can Communication Infrastructure.

To send a CAN command to the MCTK, it should be set as below:

- CAN ID = Frame ID,
- CAN Length = 8 or less
- Data containing following fields:
 - CAN Data[0] = PAYLOAD_LENGTH
 - CAN Data[1] = PAYLOAD_ID
 - ...
 - CAN Data[PAYLOAD_LENGTH] = PAYLOAD[n]
 - CAN Data[PAYLOAD_LENGTH+1] = CRC

It means that only command with $\text{PAYLOAD_LENGTH} \leq 6$ can be managed by MCTK (Set revup data not available yet)

Only 2 baud rate are configured in the MCTK:

- 250 kbps
- 500 kbps

They can be selected from:

Drive Management -> User Interface -> Can Details -> Baudrate!

Available frame ID are presented in table below, MCTK has already set the filters to receive those messages:

Command	Can ID STD	Can Data
RAMP FINAL SPEED (2000 rpm)	01	05 5B D0 07 00 00 39
BUS VOLTAGE	02	01 19 1C
START/STOP MOTOR	03	01 06 0A
GET BOARD INFO	06	01 00 07
EXEC RAMP(1000,2000)	07	06 E8 03 00 00 D0 07 D0
GET REV UP	08	01 Stage CRC
SET REF	0A	04 Iq_LB Iq_HB Id_LB Id_HB CRC

Table 4.73: Table 1: Can Frame Example

4.15 CAN physical layer communication class exported types

Collaboration diagram for CAN physical layer communication class exported types:



Data Structures

- struct [CANParams_t](#)
UserInterface class parameters definition.

Typedefs

- typedef struct CCAN_COM_t * [CCAN_COM](#)
Public UserInterface class definition.

4.15.1 Typedef Documentation

4.15.1.1 [typedef struct CCAN_COM_t* CCAN_COM](#)

Public UserInterface class definition.

Definition at line 88 of file CAN_PhysicalLayerCommunication_Class.h.

4.16 CAN physical layer communication class exported methods

Collaboration diagram for CAN physical layer communication class exported methods:



Functions

- **CCAN_COM CAN_NewObject (pCANParams_t pCANParams)**
Creates an object of the class CCAN_COM_t.
- void **CAN_ITConfig (CANDriver *CANx, uint32_t CAN_IT, FunctionalState NewState)**
Enables or disables the specified CAN interrupts.
- void **CAN_IRQHandlerRX (uint16_t Data)**
This function handles CAN1 interrupt request.

4.16.1 Function Documentation

4.16.1.1 CCAN_COM CAN_NewObject (pCANParams_t pCANParams)

Creates an object of the class CCAN_COM_t.

Parameters

<i>pCANParams</i>	pointer to pCANParams_t structure
-------------------	-----------------------------------

Return values

<i>CCAN_COM</i>	new instance of CAN_COM object
-----------------	--------------------------------

Definition at line 165 of file CAN_PhysicalLayerCommunication_Class.c.

References COM_NewObject(), and MC_NULL.

Here is the call graph for this function:



4.16.1.2 void CAN_ITConfig (CANDriver * CANx, uint32_t CAN_IT, FunctionalState NewState)

Enables or disables the specified CAN interrupts.

Parameters

<i>CANx,:</i>	where x can be 1, 2 to select the CAN peripheral.
<i>CAN_IT,:</i>	specifies the CAN interrupt sources to be enabled or disabled. This parameter can be one of the following values: <ul style="list-style-type: none"> • CAN_IT_TXE: Transmit Data Register empty interrupt • CAN_IT_RXNE: Receive Data register not empty interrupt
<i>NewState,:</i>	new state of the specified CANx interrupts. This parameter can be: ENABLE or DISABLE.

Return values

<i>None</i>

Definition at line 356 of file CAN_PhysicalLayerCommunication_Class.c.

4.16.1.3 void CAN_IRQHandlerRX (uint16_t *Data*)

This function handles CAN1 interrupt request.

Parameters

<i>None</i>

Return values

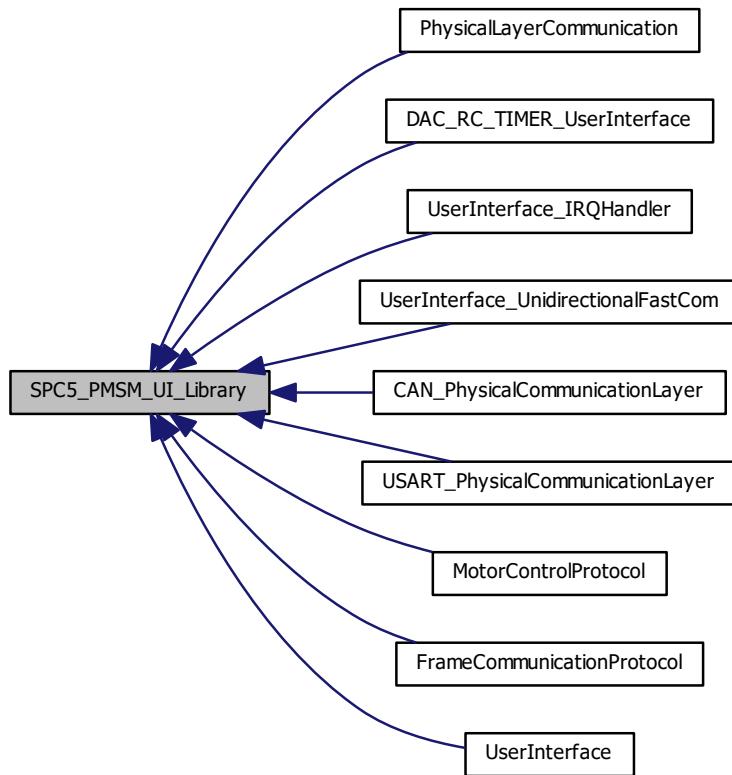
<i>None</i>

Definition at line 395 of file CAN_PhysicalLayerCommunication_Class.c.

References UI_IRQ_CAN.

4.17 SPC5_PMSM_UI_Library

Collaboration diagram for SPC5_PMSM_UI_Library:

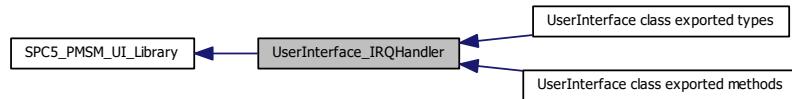


Modules

- **DAC_RC_TIMER_UserInterface**
- **UserInterface_UnidirectionalFastCom**
- **UserInterface**
- **CAN_PhysicalCommunicationLayer**
- **FrameCommunicationProtocol**
- **MotorControlProtocol**
- **PhysicalLayerCommunication**
- **UserInterface_IRQHandler**
- **USART_PhysicalCommunicationLayer**

4.18 UserInterface_IRQHandler

Collaboration diagram for UserInterface_IRQHandler:

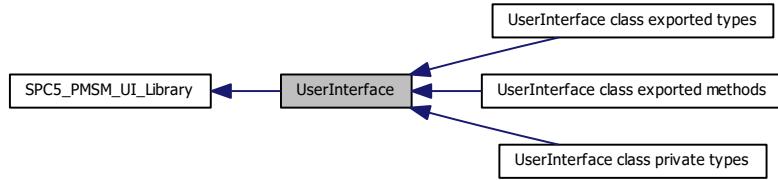


Modules

- UserInterface class exported types
- UserInterface class exported methods

4.19 UserInterface

Collaboration diagram for UserInterface:

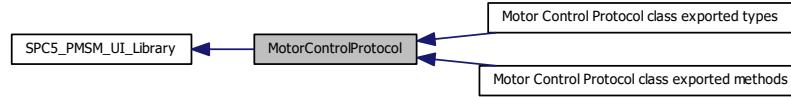


Modules

- `UserInterface class private types`
- `UserInterface class exported types`
- `UserInterface class exported methods`

4.20 MotorControlProtocol

Collaboration diagram for MotorControlProtocol:

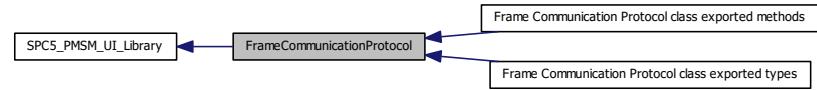


Modules

- Motor Control Protocol class exported types
- Motor Control Protocol class exported methods

4.21 FrameCommunicationProtocol

Collaboration diagram for FrameCommunicationProtocol:

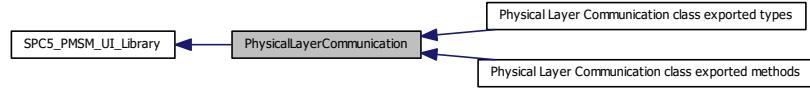


Modules

- Frame Communication Protocol class exported types
- Frame Communication Protocol class exported methods

4.22 PhysicalLayerCommunication

Collaboration diagram for PhysicalLayerCommunication:

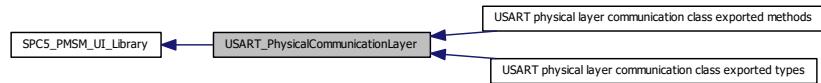


Modules

- Physical Layer Communication class exported types
- Physical Layer Communication class exported methods

4.23 USART_PhysicalCommunicationLayer

Collaboration diagram for USART_PhysicalCommunicationLayer:

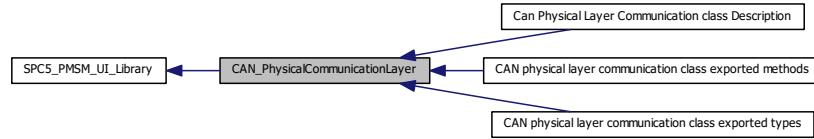


Modules

- USART physical layer communication class exported types
- USART physical layer communication class exported methods

4.24 CAN_PhysicalCommunicationLayer

Collaboration diagram for CAN_PhysicalCommunicationLayer:



Modules

- CAN physical layer communication class exported types
- CAN physical layer communication class exported methods
- Can Physical Layer Communication class Description

CAN Physical Layer Communication class implementation.

Chapter 5

Data Structure Documentation

5.1 _CMCI_t Struct Reference

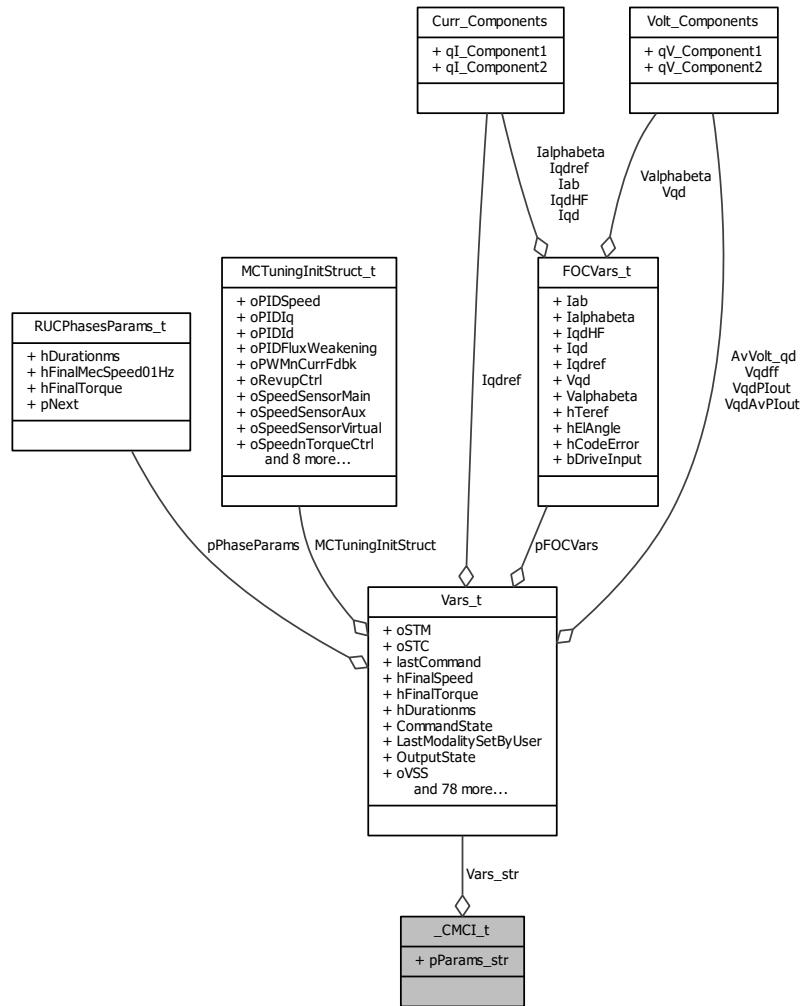
5.1.1 Detailed Description

Private MCInterface class definition.

Definition at line 78 of file MCInterfacePrivate.h.

```
#include <MCInterfacePrivate.h>
```

Collaboration diagram for `_CMCI_t`:



Data Fields

- `Vars_t Vars_str`
- `pParams_t pParams_str`

5.1.2 Field Documentation

5.1.2.1 `Vars_t _CMCI_t::Vars_str`

Class members container

Definition at line 80 of file MCInterfacePrivate.h.

5.1.2.2 `pParams_t _CMCI_t::pParams_str`

Class parameters container

Definition at line 81 of file MCInterfacePrivate.h.

Referenced by MCI_NewObject().

5.2 _CMCT_t Struct Reference

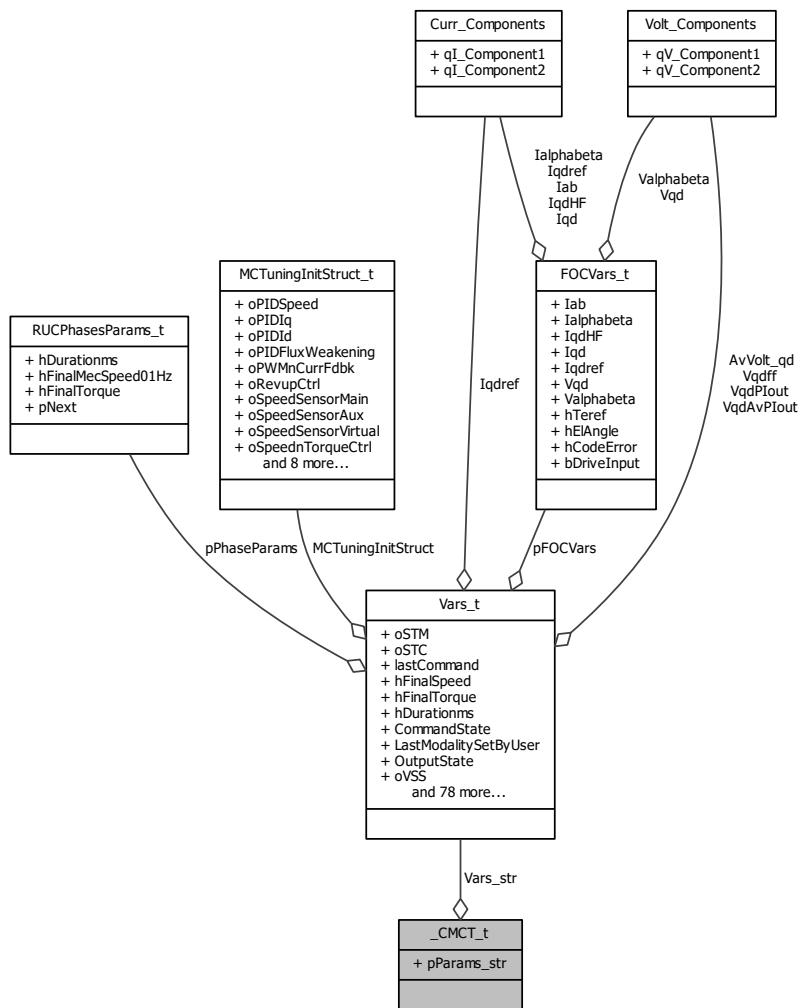
5.2.1 Detailed Description

Private MCTuning class definition.

Definition at line 63 of file MCTuningPrivate.h.

```
#include <MCTuningPrivate.h>
```

Collaboration diagram for _CMCT_t:



Data Fields

- `Vars_t Vars_str`
- `pParams_t pParams_str`

5.2.2 Field Documentation

5.2.2.1 Vars_t_CMCT_t::Vars_str

Class members container

Definition at line 65 of file MCTuningPrivate.h.

5.2.2.2 pParams_t_CMCT_t::pParams_str

Class parameters container

Definition at line 66 of file MCTuningPrivate.h.

Referenced by MCT_NewObject().

5.3 MCTuningInitStruct_t Struct Reference

5.3.1 Detailed Description

MC tuning internal objects initialization structure type:;

Definition at line 53 of file MCTuningClassPrivate.h.

```
#include <MCTuningClassPrivate.h>
```

5.4 Vars_t Struct Reference

5.4.1 Detailed Description

MCInterface class members definition.

UserInterface class members definition.

BusVoltageSensor class members definition.

TemperatureSensor class members definition.

StateMachine class members definition.

SpeednTorqCtrl class members definition.

SpeednPosFdbk class members definition.

RevupCtrl class members definition.

PWMnCurrFdbk class members definition.

FluxWeakeningCtrl class members definition.

FeedForwardCtrl class members definition.

PI regulator class members definition.

OpenLoop class members definition.

MotorPowerMeasurement class members definition.

EncAlignCtrl class members definition.

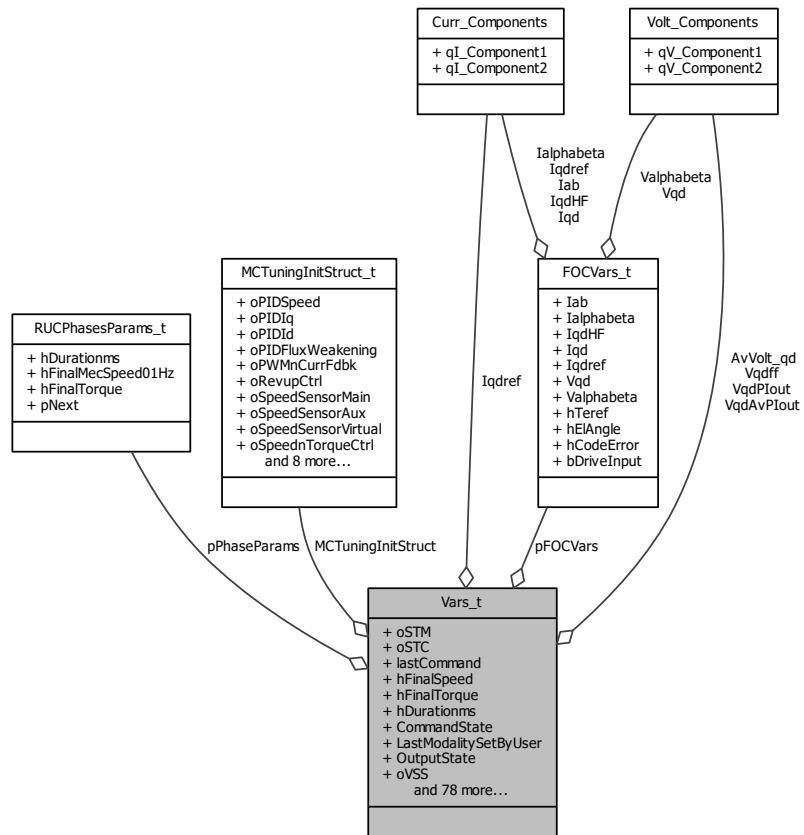
DigitalOutput class members definition.

MCTuning class members definition.

Definition at line 50 of file MCInterfacePrivate.h.

```
#include <MCInterfacePrivate.h>
```

Collaboration diagram for Vars_t:



Data Fields

- `CSTM oSTM`
- `CSTC oSTC`
- `pFOCVars_t pFOCVars`
- `UserCommands_t lastCommand`
- `int16_t hFinalSpeed`
- `int16_t hFinalTorque`
- `Curr_Components Iqdref`
- `uint16_t hDurationms`
- `CommandState_t CommandState`
- `STC_Modality_t LastModalitySetByUser`
- `CVSS_SPD oVSS`
- `CENC_SPD oENC`
- `uint16_t hRemainingTicks`
- `bool EncAligned`
- `bool EncRestart`
- `int16_t hMeasBuffer [MPM_BUFFER_LENGTH]`
- `uint16_t hNextMeasBufferIndex`
- `uint16_t hLastMeasBufferIndex`

- int16_t hAvrgElMotorPowerW
- int16_t hVoltage
- **CSPD** oVSS
- bool VFMode
- **Volt_Components** Vqdff
- **Volt_Components** VqdPlout
- **Volt_Components** VqdAvPlout
- int32_t wConstant_1D
- int32_t wConstant_1Q
- int32_t wConstant_2
- **CVBS** oVBS
- **CPI** oPI_q
- **CPI** oPI_d
- **CPI** oFluxWeakeningPI
- **CPI** oSpeedPI
- uint16_t hFW_V_Ref
- **Volt_Components** AvVolt_qd
- int16_t AvVoltAmpl
- int16_t hldRefOffset
- uint16_t hT_Sqrt3
- uint16_t hSector
- bool bTurnOnLowSidesAction
- uint16_t hOffCalibrWaitTimeCounter
- uint8_t bMotor
- uint16_t hPhaseRemainingTicks
- **pRUCPhasesParams_t** pPhaseParams
- int16_t hDirection
- uint8_t bStageCnt
- RUCPhasesData_t ParamsData [MAX_PHASE_NUMBER]
- uint8_t bPhaseNbr
- **STC_Modality_t** bMode
- int16_t hTargetFinal
- int32_t wSpeedRef01HzExt
- int32_t wTorqueRef
- uint16_t hRampRemainingStep
- **CPI** oPISpeed
- **CSPD** oSPD
- int32_t wIncDecAmount
- **State_t** bState
- uint16_t hFaultNow
- uint16_t hFaultOccurred
- uint16_t hFaultState
- uint16_t hLatestConv
- uint8_t bDriveNum
- **CMCI** * pMCI
- **CMCT** * pMCT
- uint32_t * pUICfg
- uint8_t bSelectedDrive

5.4.2 Field Documentation

5.4.2.1 CSTM Vars_t::oSTM

State machine object used by MCI.

Definition at line 52 of file MCInterfacePrivate.h.

Referenced by MCI_Init().

5.4.2.2 **CSTC_Vars_t::oSTC**

Speed and torque controller object used by MCI.

Speed and torque controller object used by EAC.

Speed and torque controller object used by RUC.

Definition at line 53 of file MCInterfacePrivate.h.

Referenced by EAC_Init(), EAC_StartAlignment(), MCI_Clear_Iqdref(), MCI_ExecBufferedCommands(), MCI_GetAvrgMecSpeed01Hz(), MCI_GetMecSpeedRef01Hz(), MCI_GetSpdSensorReliability(), MCI_Init(), MCI_RampCompleted(), RUC_Clear(), RUC_Exec(), and RUC_Init().

5.4.2.3 **pFOCVars_t Vars_t::pFOCVars**

Pointer to FOC vars used by MCI.

Definition at line 54 of file MCInterfacePrivate.h.

Referenced by MCI_Clear_Iqdref(), MCI_ExecBufferedCommands(), MCI_GetElAngledpp(), MCI_Getlab(), MCI_Getlalphabet(), MCI_Getlqd(), MCI_GetlqdHF(), MCI_Getlqdref(), MCI_GetPhaseCurrentAmplitude(), MCI_GetPhaseVoltageAmplitude(), MCI_GetTeref(), MCI_GetValphabet(), MCI_GetVqd(), MCI_Init(), and MCI_SetIdref().

5.4.2.4 **UserCommands_t Vars_t::lastCommand**

Last command coming from the user.

Definition at line 56 of file MCInterfacePrivate.h.

Referenced by MCI_ExecBufferedCommands(), MCI_ExecSpeedRamp(), MCI_ExecTorqueRamp(), MCI_GetImposedMotorDirection(), MCI_GetLastRampFinalSpeed(), MCI_Init(), and MCI_SetCurrentReferences().

5.4.2.5 **int16_t Vars_t::hFinalSpeed**

Final speed of last ExecSpeedRamp command.

Definition at line 57 of file MCInterfacePrivate.h.

Referenced by MCI_ExecBufferedCommands(), MCI_ExecSpeedRamp(), MCI_GetImposedMotorDirection(), MCI_GetLastRampFinalSpeed(), and MCI_Init().

5.4.2.6 **int16_t Vars_t::hFinalTorque**

Final torque of last ExecTorqueRamp command.

Definition at line 58 of file MCInterfacePrivate.h.

Referenced by MCI_ExecBufferedCommands(), MCI_ExecTorqueRamp(), MCI_GetImposedMotorDirection(), and MCI_Init().

5.4.2.7 **Curr_Components Vars_t::lqdref**

Current component of last SetCurrentReferences command.

Definition at line 60 of file MCInterfacePrivate.h.

Referenced by MCI_ExecBufferedCommands(), MCI_GetImposedMotorDirection(), and MCI_SetCurrentReferences().

5.4.2.8 `uint16_t Vars_t::hDurationms`

Duration in ms of last ExecSpeedRamp or ExecTorqueRamp command.

Definition at line 62 of file MCInterfacePrivate.h.

Referenced by MCI_ExecBufferedCommands(), MCI_ExecSpeedRamp(), MCI_ExecTorqueRamp(), and MCI_Init().

5.4.2.9 `CommandState_t Vars_t::CommandState`

The status of the buffered command.

Definition at line 65 of file MCInterfacePrivate.h.

Referenced by MCI_ExecBufferedCommands(), MCI_ExecSpeedRamp(), MCI_ExecTorqueRamp(), MCI_Init(), MCI_IsCommandAcknowledged(), MCI_SetCurrentReferences(), and MCI_StartMotor().

5.4.2.10 `STC_Modality_t Vars_t::LastModalitySetByUser`

The last STC_Modality_t set by the user.

Definition at line 66 of file MCInterfacePrivate.h.

Referenced by MCI_ExecSpeedRamp(), MCI_ExecTorqueRamp(), MCI_GetControlMode(), and MCI_SetCurrentReferences().

5.4.2.11 `CVSS_SPD Vars_t::oVSS`

Virtual speed sensor object used by EAC.

Virtual speed sensor object used by RUC.

Definition at line 53 of file EncAlignCtrlPrivate.h.

Referenced by EAC_Init(), EAC_StartAlignment(), OL_Calc(), OL_Init(), RUC_Clear(), RUC_Exec(), and RUC_Init().

5.4.2.12 `CENC_SPD Vars_t::oENC`

Encoder object used by EAC.

Definition at line 54 of file EncAlignCtrlPrivate.h.

Referenced by EAC_Exec(), and EAC_Init().

5.4.2.13 `uint16_t Vars_t::hRemainingTicks`

Number of clock events remaining to complete the alignment.

Definition at line 55 of file EncAlignCtrlPrivate.h.

Referenced by EAC_Exec(), and EAC_StartAlignment().

5.4.2.14 `bool Vars_t::EncAligned`

This flag is TRUE if the encoder has been aligned at least one time, FALSE if hasn't been never aligned.

Definition at line 57 of file EncAlignCtrlPrivate.h.

Referenced by EAC_Exec(), EAC_Init(), and EAC_IsAligned().

5.4.2.15 bool Vars_t::EncRestart

This flag is used to force a restart of the motor after the encoder alignment. It is TRUE if a restart is programmed else FALSE

Definition at line 60 of file EncAlignCtrlPrivate.h.

Referenced by EAC_GetRestartState(), EAC_Init(), and EAC_SetRestartState().

5.4.2.16 int16_t Vars_t::hMeasBuffer[MPM_BUFFER_LENGTH]

Buffer used by MPM object to store the instantaneous measurements of motor power.

Definition at line 62 of file MotorPowerMeasurementPrivate.h.

Referenced by MPM_CalcElMotorPower(), MPM_Clear(), and MPM_GetElMotorPowerW().

5.4.2.17 uint16_t Vars_t::hNextMeasBufferIndex

Index of the buffer that will contain the next motor power measurement.

Definition at line 65 of file MotorPowerMeasurementPrivate.h.

Referenced by MPM_CalcElMotorPower(), and MPM_Clear().

5.4.2.18 uint16_t Vars_t::hLastMeasBufferIndex

Index of the buffer that contains the last motor power measurement.

Definition at line 67 of file MotorPowerMeasurementPrivate.h.

Referenced by MPM_CalcElMotorPower(), MPM_Clear(), and MPM_GetElMotorPowerW().

5.4.2.19 int16_t Vars_t::hAvrgElMotorPowerW

The average measured motor power expressed in watt.

Definition at line 69 of file MotorPowerMeasurementPrivate.h.

Referenced by MPM_CalcElMotorPower(), and MPM_GetAvrgElMotorPowerW().

5.4.2.20 int16_t Vars_t::hVoltage

Open loop voltage to be applied

Definition at line 51 of file OpenLoopPrivate.h.

Referenced by OL_Calc(), OL_Init(), OL_UpdateVoltage(), and OL_VqdConditioning().

5.4.2.21 CSPD Vars_t::oVSS

Related VSS object used

Definition at line 52 of file OpenLoopPrivate.h.

5.4.2.22 bool Vars_t::VFMode

TRUE to enable V/F mode otherwise FALSE

Definition at line 53 of file OpenLoopPrivate.h.

Referenced by OL_Calc(), OL_Init(), and OL_VF().

5.4.2.23 Volt_Components Vars_t::Vqdff

Feed-forward Iqd controller contribution to Vqd

Definition at line 49 of file FeedForwardCtrlPrivate.h.

5.4.2.24 Volt_Components Vars_t::VqdPlout

Vqd as output by PI controller

Definition at line 51 of file FeedForwardCtrlPrivate.h.

5.4.2.25 Volt_Components Vars_t::VqdAvPlout

Averaged Vqd as output by PI controller

Definition at line 52 of file FeedForwardCtrlPrivate.h.

5.4.2.26 int32_t Vars_t::wConstant_1D

Feed forward default constant for d axes

Definition at line 53 of file FeedForwardCtrlPrivate.h.

5.4.2.27 int32_t Vars_t::wConstant_1Q

Feed forward default constant for q axes

Definition at line 54 of file FeedForwardCtrlPrivate.h.

5.4.2.28 int32_t Vars_t::wConstant_2

Default constant value used by Feed-Forward algorithm

Definition at line 55 of file FeedForwardCtrlPrivate.h.

5.4.2.29 CVBS Vars_t::oVBS

Related bus voltage sensor

Definition at line 57 of file FeedForwardCtrlPrivate.h.

5.4.2.30 CPI Vars_t::oPI_q

Related PI for Iq regulator

Definition at line 58 of file FeedForwardCtrlPrivate.h.

5.4.2.31 CPI Vars_t::oPI_d

Related PI for Id regulator

Definition at line 59 of file FeedForwardCtrlPrivate.h.

5.4.2.32 CPI Vars_t::oFluxWeakeningPI

PI object used for flux weakening

Definition at line 51 of file FluxWeakeningCtrlPrivate.h.

Referenced by FW_CalcCurrRef(), FW_Clear(), and FW_Init().

5.4.2.33 CPI Vars_t::oSPEEDPI

PI object used for speed control

Definition at line 52 of file FluxWeakeningCtrlPrivate.h.

Referenced by FW_CalcCurrRef(), and FW_Init().

5.4.2.34 uint16_t Vars_t::hFW_V_Ref

Voltage reference, tenth of percentage points

Definition at line 53 of file FluxWeakeningCtrlPrivate.h.

Referenced by FW_CalcCurrRef(), FW_GetVref(), FW_Init(), and FW_SetVref().

5.4.2.35 Volt_Components Vars_t::AvVolt_qd

Average stator voltage in qd reference frame

Definition at line 55 of file FluxWeakeningCtrlPrivate.h.

Referenced by FW_CalcCurrRef(), FW_Clear(), and FW_DataProcess().

5.4.2.36 int16_t Vars_t::AvVoltAmpl

Average stator voltage amplitude

Definition at line 57 of file FluxWeakeningCtrlPrivate.h.

Referenced by FW_CalcCurrRef(), FW_Clear(), FW_GetAvVAmplitude(), and FW_GetAvVPercentage().

5.4.2.37 int16_t Vars_t::hIdRefOffset

Id reference offset

Definition at line 58 of file FluxWeakeningCtrlPrivate.h.

Referenced by FW_CalcCurrRef(), and FW_Clear().

5.4.2.38 uint16_t Vars_t::hT_Sqrt3

Contains a constant utilized by SVPWM algorithm

Definition at line 64 of file PWMnCurrFdbkPrivate.h.

Referenced by PWMC_Init(), and PWMC_SetPhaseVoltage().

5.4.2.39 uint16_t Vars_t::hSector

Contains the space vector sector number

Definition at line 65 of file PWMnCurrFdbkPrivate.h.

Referenced by PWMC_SetPhaseVoltage().

5.4.2.40 bool Vars_t::bTurnOnLowSidesAction

TRUE if TurnOnLowSides action is active, FALSE otherwise.

Definition at line 70 of file PWMnCurrFdbkPrivate.h.

Referenced by PWMC_GetTurnOnLowSidesAction(), PWMC_Init(), PWMC_SwitchOffPWM(), PWMC_SwitchOnPWM(), and PWMC_TurnOnLowSides().

5.4.2.41 uint16_t Vars_t::hOffCalibrWaitTimeCounter

Counter to wait fixed time before motor current measurement offset calibration.

Definition at line 72 of file PWMnCurrFdbkPrivate.h.

Referenced by PWMC_CurrentReadingCalibr().

5.4.2.42 uint8_t Vars_t::bMotor

Motor reference number

Definition at line 75 of file PWMnCurrFdbkPrivate.h.

5.4.2.43 uint16_t Vars_t::hPhaseRemainingTicks

Number of clock events remaining to complete the phase.

Definition at line 77 of file RevupCtrlPrivate.h.

Referenced by RUC_Clear(), and RUC_Exec().

5.4.2.44 pRUCPhasesParams_t Vars_t::pPhaseParams

Pointer to parameter of the current executed rev up phase.

Definition at line 79 of file RevupCtrlPrivate.h.

Referenced by RUC_Clear(), and RUC_Exec().

5.4.2.45 int16_t Vars_t::hDirection

If it is "1" the programmed revup sequence is performed. If it is "-1" the revup sequence is performed with opposite values of targets (speed, torque).

Definition at line 81 of file RevupCtrlPrivate.h.

Referenced by RUC_Clear(), and RUC_Exec().

5.4.2.46 uint8_t Vars_t::bStageCnt

It counts the rev up stages that have been already started. NOTE: The rev up stage are zero-based indexed so that the fist stage is the number zero.

Definition at line 85 of file RevupCtrlPrivate.h.

Referenced by RUC_Clear(), RUC_Exec(), and RUC_FirstAccelerationStageReached().

5.4.2.47 RUCPhasesData_t Vars_t::ParamsData[MAX_PHASE_NUMBER]

Rev up phases data. It is initialized equal to rev up phases parameters but can be changed in run-time using tuning methods.

Definition at line 93 of file RevupCtrlPrivate.h.

Referenced by RUC_Clear(), RUC_GetPhaseDurationms(), RUC_GetPhaseFinalMecSpeed01Hz(), RUC_GetPhaseFinalTorque(), RUC_Init(), RUC_SetPhaseDurationms(), RUC_SetPhaseFinalMecSpeed01Hz(), and RUC_SetPhaseFinalTorque().

5.4.2.48 uint8_t Vars_t::bPhaseNbr

Number of phases relative to the programmed rev up sequence.

Definition at line 97 of file RevupCtrlPrivate.h.

Referenced by RUC_GetNumberOfPhases(), and RUC_Init().

5.4.2.49 STC_Modality_t Vars_t::bMode

Modality of STC. It can be one of these two settings: STC_TORQUE_MODE to enable the Torque mode or STC_SPEED_MODE to enable the Speed mode.

Definition at line 51 of file SpeednTorqCtrlPrivate.h.

Referenced by STC_CalcTorqueReference(), STC_Clear(), STC_ExecRamp(), STC_GetControlMode(), STC_Init(), and STC_SetControlMode().

5.4.2.50 int16_t Vars_t::hTargetFinal

Backup of hTargetFinal to be applied in the last step.

Definition at line 55 of file SpeednTorqCtrlPrivate.h.

Referenced by STC_CalcTorqueReference(), STC_ExecRamp(), and STC_Init().

5.4.2.51 int32_t Vars_t::wSpeedRef01HzExt

Current mechanical rotor speed reference expressed in tenths of HZ multiplied by

1.

Definition at line 57 of file SpeednTorqCtrlPrivate.h.

Referenced by STC_CalcTorqueReference(), STC_ExecRamp(), and STC_Init().

5.4.2.52 int32_t Vars_t::wTorqueRef

Current motor torque reference. This value represents actually the Iq current expressed in digit multiplied by 65536.

Definition at line 60 of file SpeednTorqCtrlPrivate.h.

Referenced by STC_CalcTorqueReference(), STC_ExecRamp(), and STC_Init().

5.4.2.53 uint16_t Vars_t::hRampRemainingStep

Number of steps remaining to complete the ramp.

Definition at line 63 of file SpeednTorqCtrlPrivate.h.

Referenced by STC_CalcTorqueReference(), STC_ExecRamp(), STC_Init(), STC_RampCompleted(), STC_SetControlMode(), and STC_StopRamp().

5.4.2.54 **CPI Vars_t::oPISpeed**

The regulator used to perform the speed control loop.

Definition at line 65 of file SpeednTorqCtrlPrivate.h.

Referenced by STC_CalcTorqueReference(), and STC_Init().

5.4.2.55 **CSPD Vars_t::oSPD**

The speed sensor used to perform the speed regulation.

Definition at line 67 of file SpeednTorqCtrlPrivate.h.

Referenced by STC_CalcTorqueReference(), STC_ExecRamp(), STC_GetSpeedSensor(), STC_Init(), and STC_SetSpeedSensor().

5.4.2.56 **int32_t Vars_t::wIncDecAmount**

Increment/decrement amount to be applied to the reference value at each CalcTorqueReference.

Definition at line 69 of file SpeednTorqCtrlPrivate.h.

Referenced by STC_CalcTorqueReference(), STC_ExecRamp(), STC_Init(), and STC_StopRamp().

5.4.2.57 **State_t Vars_t::bState**

Variable containing state machine current state

Definition at line 51 of file StateMachinePrivate.h.

Referenced by STM_FaultAcknowledged(), STM_FaultProcessing(), STM_Init(), and STM_NextState().

5.4.2.58 **uint16_t Vars_t::hFaultNow**

Bit fields variable containing faults currently present

Definition at line 53 of file StateMachinePrivate.h.

Referenced by STM_FaultProcessing(), STM_GetFaultState(), and STM_Init().

5.4.2.59 **uint16_t Vars_t::hFaultOccurred**

Bit fields variable containing faults historically occurred since the state machine has been moved to FAULT_NOW state

Definition at line 55 of file StateMachinePrivate.h.

Referenced by STM_FaultAcknowledged(), STM_FaultProcessing(), STM_GetFaultState(), and STM_Init().

5.4.2.60 **uint16_t Vars_t::hFaultState**

It contains latest available average Vbus expressed in u16Celsius

It contains latest available average Vbus expressed in digit

Definition at line 53 of file TemperatureSensorPrivate.h.

5.4.2.61 uint16_t Vars_t::hLatestConv

It contains latest Vbus converted value expressed in u16Volts format

Definition at line 51 of file BusVoltageSensorPrivate.h.

5.4.2.62 uint8_t Vars_t::bDriveNum

Total number of MC objects.

Definition at line 52 of file UserInterfacePrivate.h.

Referenced by UI_Init(), and UI_SelectMC().

5.4.2.63 CMCI* Vars_t::pMCI

Pointer of MC interface list.

Definition at line 53 of file UserInterfacePrivate.h.

Referenced by UI_ExecCmd(), UI_ExecSpeedRamp(), UI_ExecTorqueRamp(), UI_GetReg(), UI_Init(), UI_SetCurrentReferences(), and UI_SetReg().

5.4.2.64 CMCT* Vars_t::pMCT

Pointer of MC tuning list.

Definition at line 54 of file UserInterfacePrivate.h.

Referenced by UI_GetCurrentMCT(), UI_GetReg(), UI_GetRevupData(), UI_Init(), UI_SetReg(), and UI_SetRevupData().

5.4.2.65 uint32_t* Vars_t::pUICfg

Pointer of UI configuration list.

Definition at line 55 of file UserInterfacePrivate.h.

Referenced by UI_ExecCmd(), UI_GetReg(), UI_GetSelectedMCConfig(), UI_Init(), and UI_SetReg().

5.4.2.66 uint8_t Vars_t::bSelectedDrive

Current selected MC object in the list.

Definition at line 56 of file UserInterfacePrivate.h.

Referenced by UFC_Init(), UI_ExecCmd(), UI_ExecSpeedRamp(), UI_ExecTorqueRamp(), UI_GetCurrentMCT(), UI_GetReg(), UI_GetRevupData(), UI_GetSelectedMCConfig(), UI_Init(), UI_SelectMC(), UI_SetCurrentReferences(), UI_SetReg(), and UI_SetRevupData().

5.5 _CCLM_t Struct Reference

5.5.1 Detailed Description

Private CircleLimitation class definition.

Definition at line 54 of file CircleLimitationPrivate.h.

```
#include <CircleLimitationPrivate.h>
```

Data Fields

- `pParams_t pParams_str`

5.5.2 Field Documentation

5.5.2.1 `pParams_t _CCLM_t::pParams_str`

Class parameters container

Definition at line 56 of file CircleLimitationPrivate.h.

Referenced by `CLM_NewObject()`.

5.6 `_CDOUT_t` Struct Reference

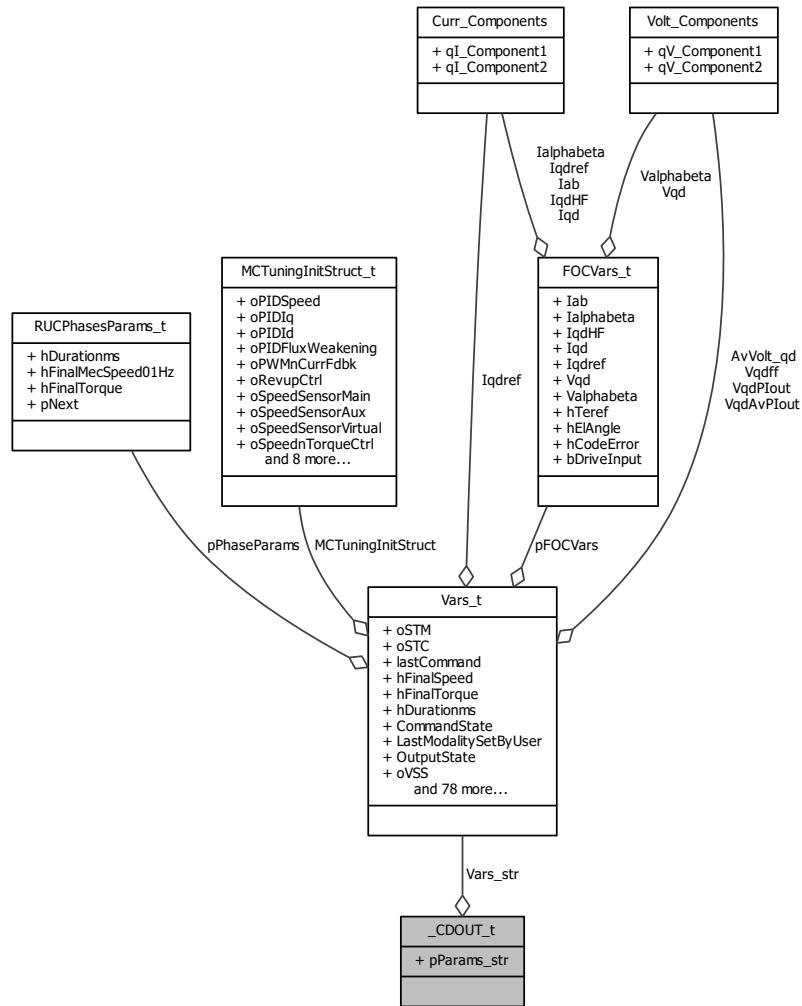
5.6.1 Detailed Description

Private DigitalOutput class definition.

Definition at line 62 of file DigitalOutputPrivate.h.

```
#include <DigitalOutputPrivate.h>
```

Collaboration diagram for `_CDOUT_t`:



Data Fields

- `Vars_t Vars_str`
- `pParams_t pParams_str`

5.6.2 Field Documentation

5.6.2.1 `Vars_t _CDOUT_t::Vars_str`

Class members container

Definition at line 64 of file DigitalOutputPrivate.h.

5.6.2.2 `pParams_t _CDOUT_t::pParams_str`

Class parameters container

Definition at line 65 of file DigitalOutputPrivate.h.

5.7 _CEAC_t Struct Reference

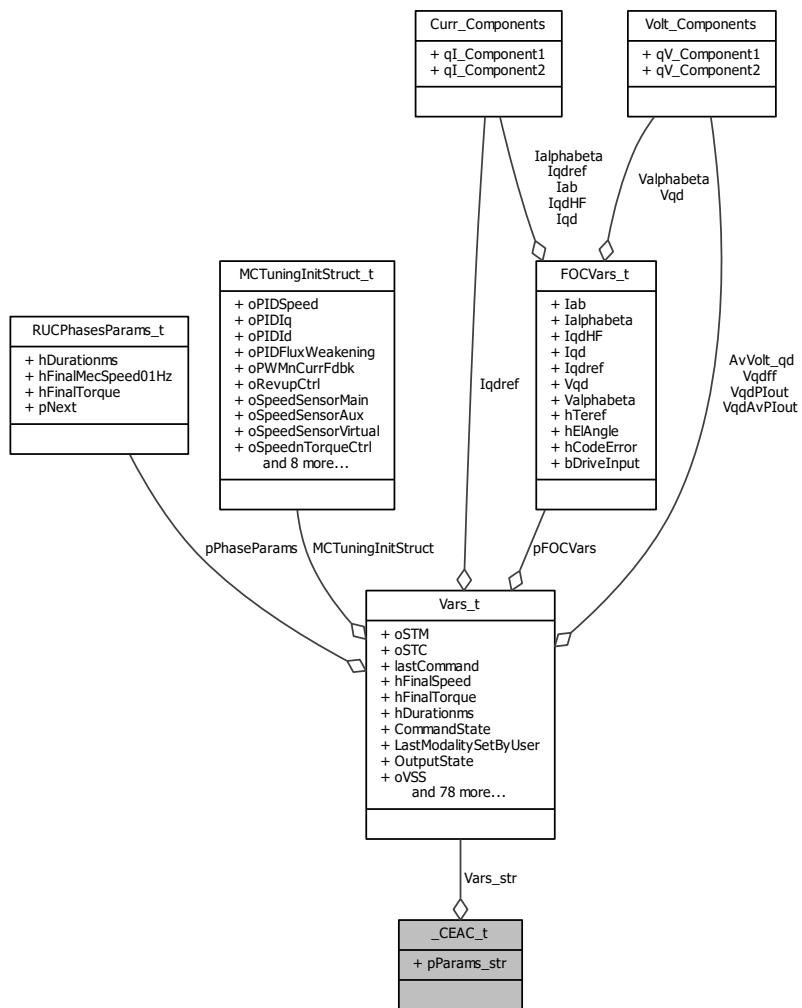
5.7.1 Detailed Description

Private EncAlignCtrl class definition.

Definition at line 73 of file EncAlignCtrlPrivate.h.

```
#include <EncAlignCtrlPrivate.h>
```

Collaboration diagram for _CEAC_t:



Data Fields

- `Vars_t Vars_str`
- `pParams_t pParams_str`

5.7.2 Field Documentation

5.7.2.1 Vars_t_CEAC_t::Vars_str

Class members container

Definition at line 75 of file EncAlignCtrlPrivate.h.

5.7.2.2 pParams_t_CEAC_t::pParams_str

Class parameters container

Definition at line 76 of file EncAlignCtrlPrivate.h.

Referenced by EAC_NewObject().

5.8 _CFF_t Struct Reference

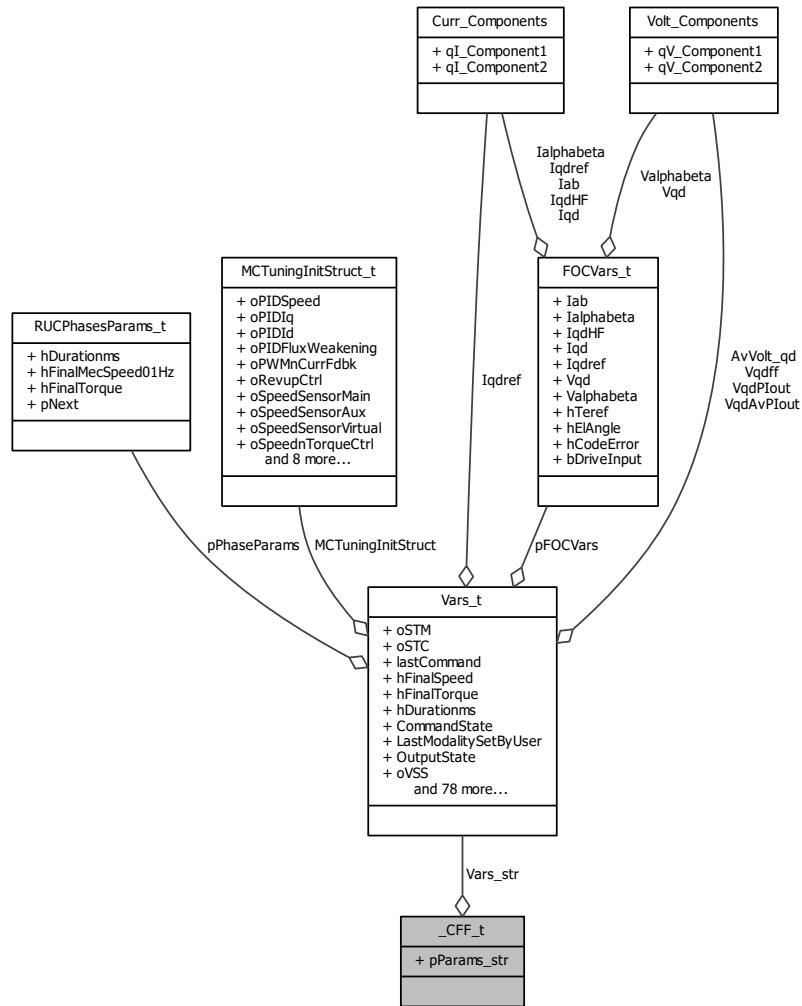
5.8.1 Detailed Description

Private FeedForwardCtrl class definition.

Definition at line 70 of file FeedForwardCtrlPrivate.h.

```
#include <FeedForwardCtrlPrivate.h>
```

Collaboration diagram for `_CFF_t`:



Data Fields

- `Vars_t Vars_str`
- `pParams_t pParams_str`

5.8.2 Field Documentation

5.8.2.1 `Vars_t _CFF_t::Vars_str`

Class members container

Definition at line 72 of file FeedForwardCtrlPrivate.h.

5.8.2.2 `pParams_t _CFF_t::pParams_str`

Class parameters container

Definition at line 73 of file FeedForwardCtrlPrivate.h.

5.9 _CFW_t Struct Reference

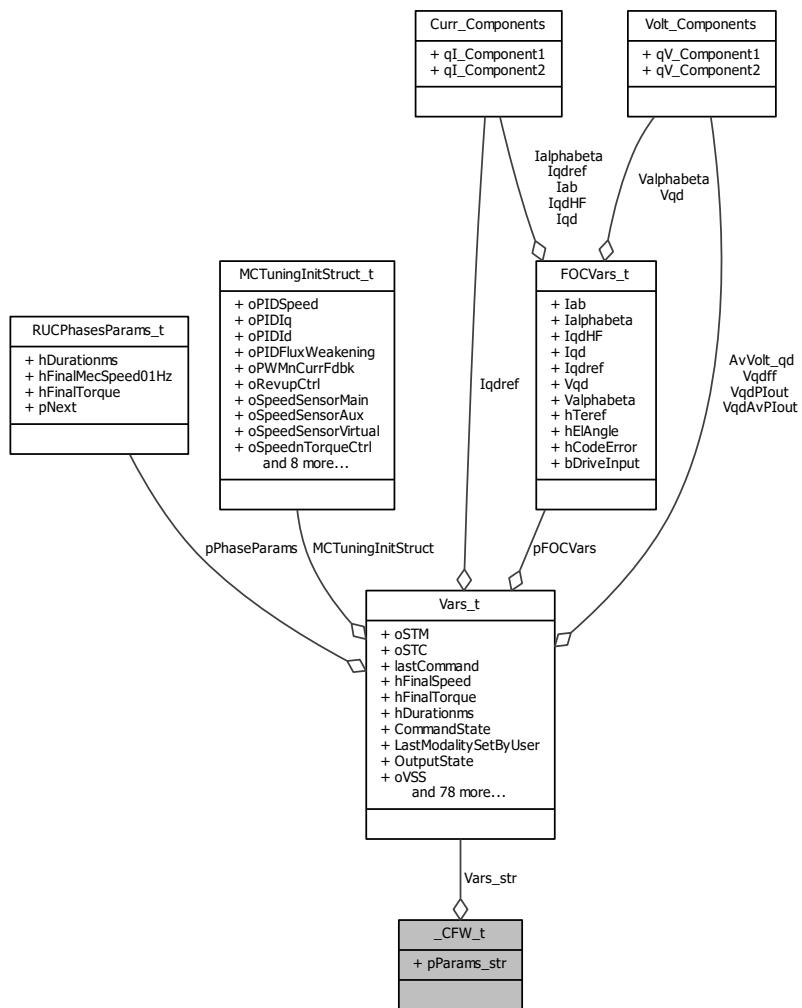
5.9.1 Detailed Description

Private FluxWeakeningCtrl class definition.

Definition at line 69 of file FluxWeakeningCtrlPrivate.h.

```
#include <FluxWeakeningCtrlPrivate.h>
```

Collaboration diagram for _CFW_t:



Data Fields

- `Vars_t Vars_str`
- `pParams_t pParams_str`

5.9.2 Field Documentation

5.9.2.1 `Vars_t_CFW_t::Vars_str`

Class members container

Definition at line 71 of file FluxWeakeningCtrlPrivate.h.

5.9.2.2 `pParams_t_CFW_t::pParams_str`

Class parameters container

Definition at line 72 of file FluxWeakeningCtrlPrivate.h.

Referenced by FW_NewObject().

5.10 `_CMCIRQ_t` Struct Reference

5.10.1 Detailed Description

MCIRQ private module definition.

Definition at line 50 of file MCIRQHandlerPrivate.h.

```
#include <MCIRQHandlerPrivate.h>
```

5.11 `_CMPM_t` Struct Reference

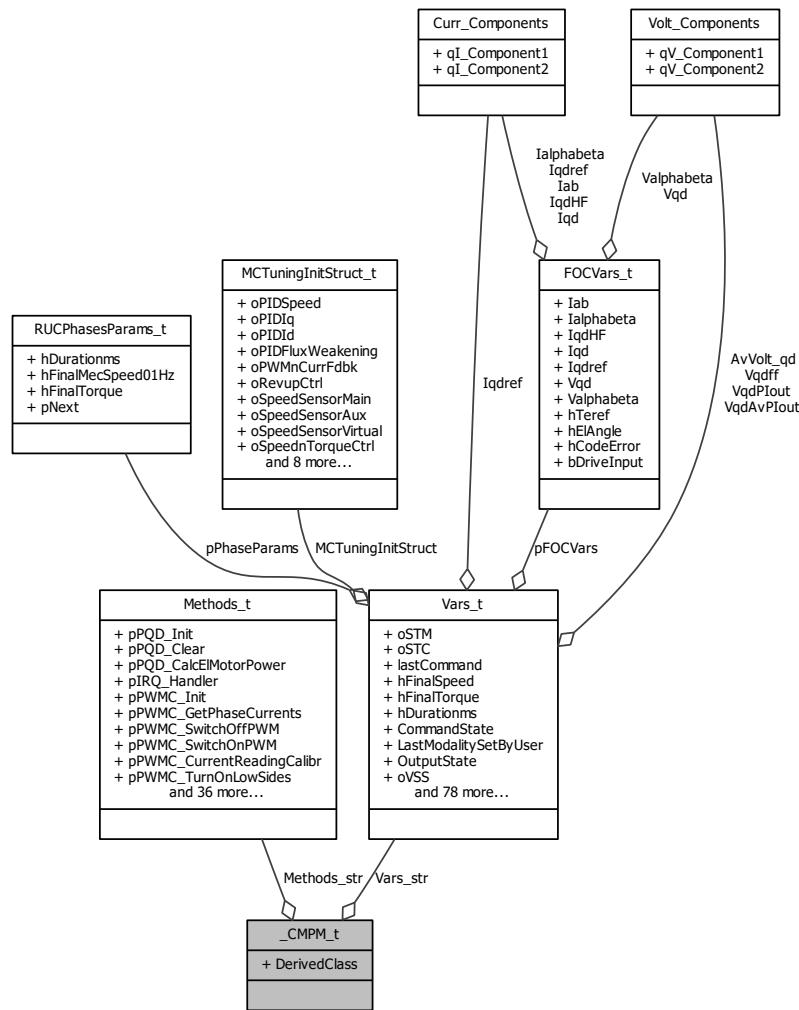
5.11.1 Detailed Description

Private MotorPowerMeasurement class definition.

Definition at line 86 of file MotorPowerMeasurementPrivate.h.

```
#include <MotorPowerMeasurementPrivate.h>
```

Collaboration diagram for `_CMPM_t`:



Data Fields

- `Methods_t Methods_str`
- `Vars_t Vars_str`
- `void * DerivedClass`

5.11.2 Field Documentation

5.11.2.1 `Methods_t _CMPM_t::Methods_str`

Virtual methods container

Definition at line 88 of file MotorPowerMeasurementPrivate.h.

Referenced by `PQD_NewObject()`.

5.11.2.2 `Vars_t _CMPM_t::Vars_str`

Class members container

Definition at line 89 of file MotorPowerMeasurementPrivate.h.

5.11.2.3 `void* _CMPM_t::DerivedClass`

Pointer to derived class

Definition at line 90 of file MotorPowerMeasurementPrivate.h.

Referenced by PQD_NewObject().

5.12 `_CMTPA_t` Struct Reference

5.12.1 Detailed Description

Private MTPACtrl class definition.

Definition at line 52 of file MTPACtrlPrivate.h.

```
#include <MTPACtrlPrivate.h>
```

Data Fields

- `pParams_t pParams_str`

5.12.2 Field Documentation

5.12.2.1 `pParams_t _CMTPA_t::pParams_str`

Class parameters container

Definition at line 54 of file MTPACtrlPrivate.h.

5.13 `_CNTC_TSNS_t` Struct Reference

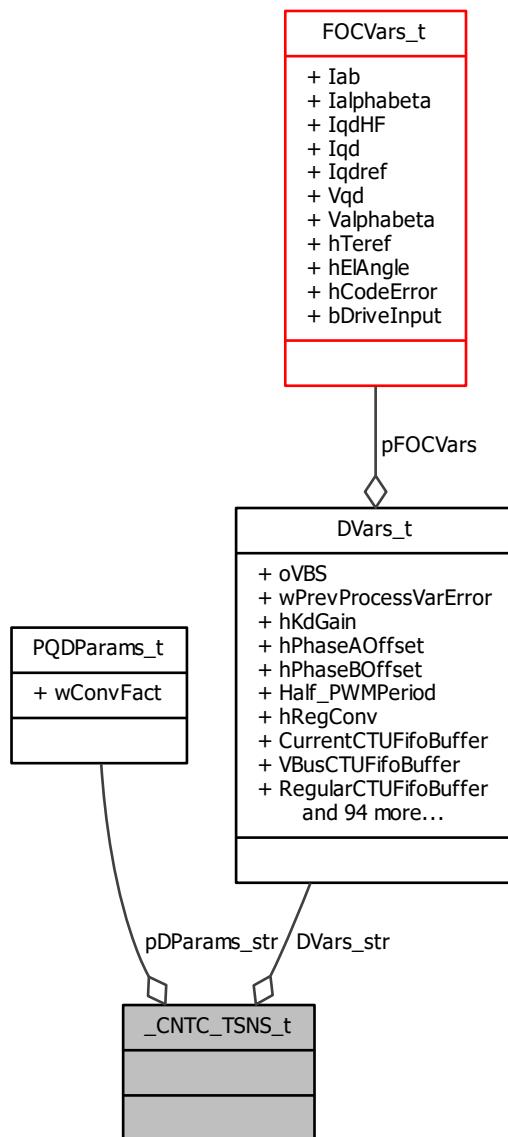
5.13.1 Detailed Description

Private NTC class definition.

Definition at line 65 of file NTC_TemperatureSensorPrivate.h.

```
#include <NTC_TemperatureSensorPrivate.h>
```

Collaboration diagram for `_CNTC_TSNS_t`:



Data Fields

- `DVars_t DVars_str`
- `pDParams_t pDParams_str`

5.13.2 Field Documentation

5.13.2.1 `DVars_t _CNTC_TSNS_t::DVars_str`

Derived class members container

Definition at line 67 of file NTC_TemperatureSensorPrivate.h.

5.13.2.2 pDParams_t _CNTC_TSNS_t::pDParams_str

Derived class parameters container

Definition at line 68 of file NTC_TemperatureSensorPrivate.h.

Referenced by NTC_NewObject().

5.14 _COL_t Struct Reference

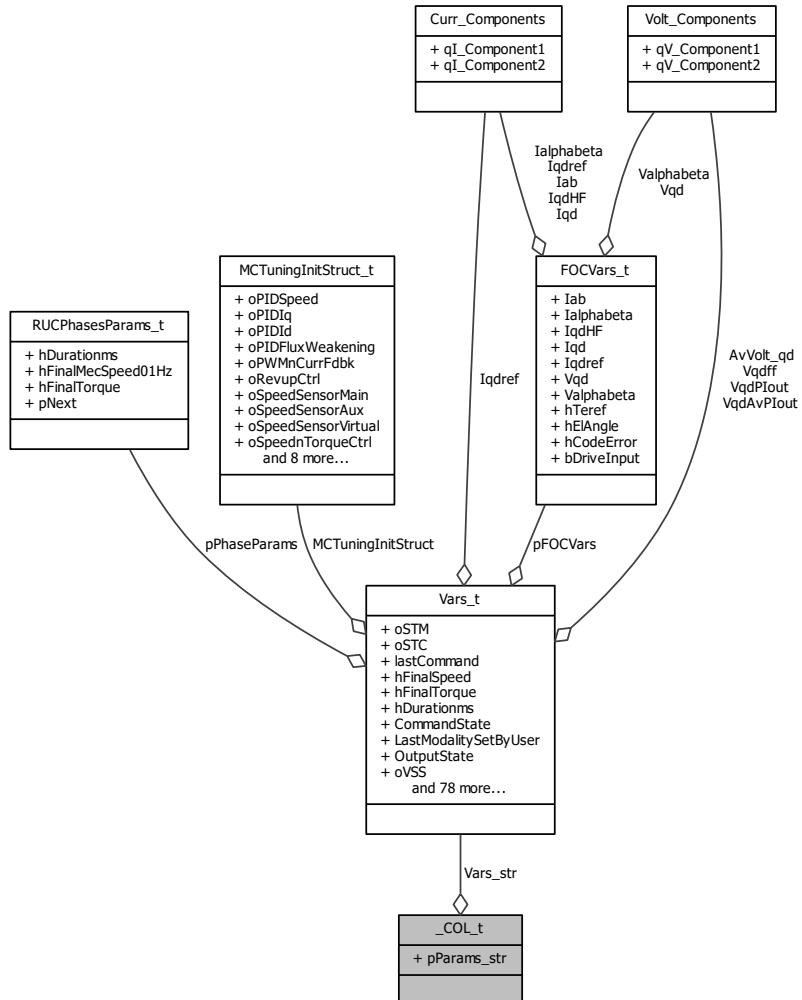
5.14.1 Detailed Description

Private OpenLoop class definition.

Definition at line 64 of file OpenLoopPrivate.h.

```
#include <OpenLoopPrivate.h>
```

Collaboration diagram for _COL_t:



Data Fields

- `Vars_t Vars_str`
- `pParams_t pParams_str`

5.14.2 Field Documentation

5.14.2.1 `Vars_t_COL_t::Vars_str`

Class members container

Definition at line 66 of file OpenLoopPrivate.h.

5.14.2.2 `pParams_t_COL_t::pParams_str`

Class parameters container

Definition at line 67 of file OpenLoopPrivate.h.

Referenced by `OL_NewObject()`.

5.15 _CPI_t Struct Reference

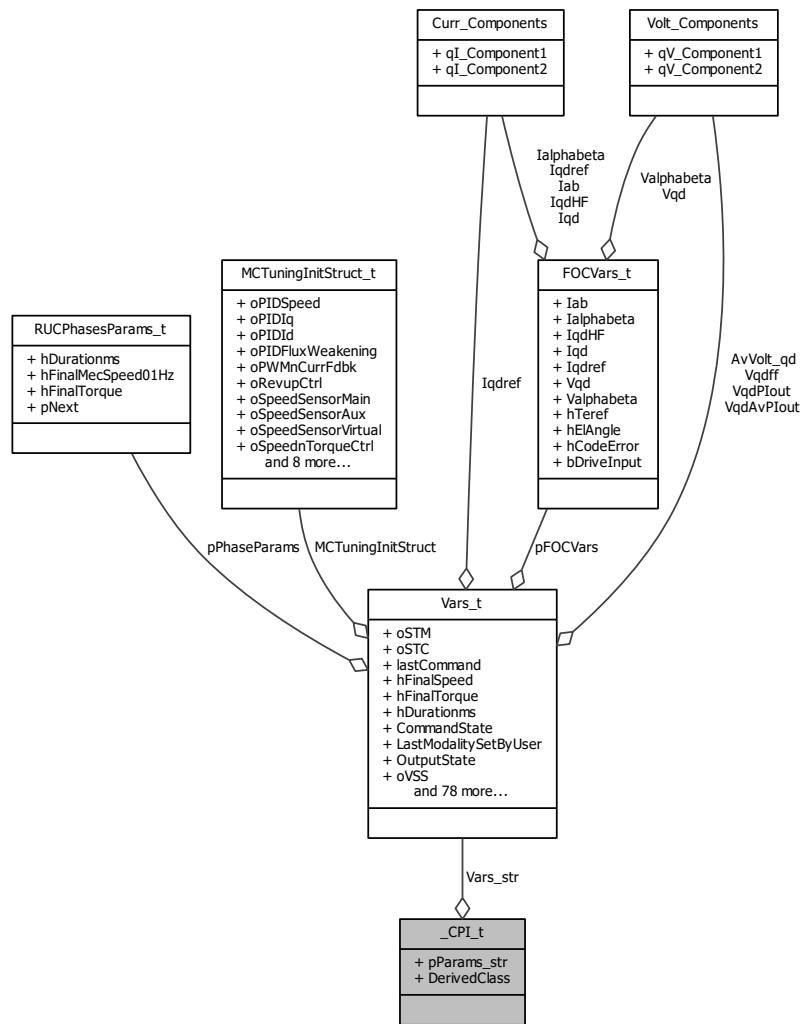
5.15.1 Detailed Description

Private PI regulator class definition.

Definition at line 75 of file PIRegulatorPrivate.h.

```
#include <PIRegulatorPrivate.h>
```

Collaboration diagram for `_CPI_t`:



Data Fields

- `Vars_t Vars_str`
- `pParams_t pParams_str`
- `void * DerivedClass`

5.15.2 Field Documentation

5.15.2.1 `Vars_t _CPI_t::Vars_str`

Class members container

Definition at line 77 of file PIRegulatorPrivate.h.

5.15.2.2 pParams_t _CPI_t::pParams_str

Class parameters container

Definition at line 78 of file PIRegulatorPrivate.h.

Referenced by PI_NewObject().

5.15.2.3 void* _CPI_t::DerivedClass

Pointer to derived class

Definition at line 80 of file PIRegulatorPrivate.h.

Referenced by PI_NewObject(), and PID_NewObject().

5.16 _CPWMC_t Struct Reference

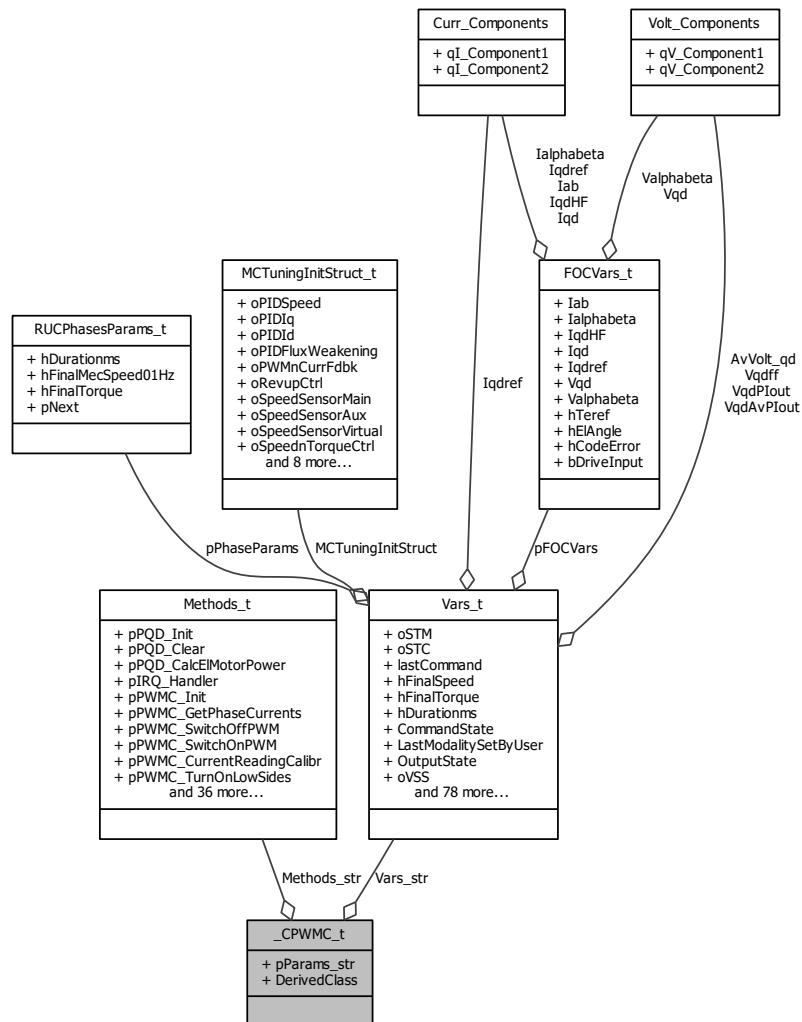
5.16.1 Detailed Description

Private PWMnCurrFdbk class definition.

Definition at line 111 of file PWMnCurrFdbkPrivate.h.

```
#include <PWMnCurrFdbkPrivate.h>
```

Collaboration diagram for `_CPWMC_t`:



Data Fields

- `Methods_t Methods_str`
- `Vars_t Vars_str`
- `pParams_t pParams_str`
- `void * DerivedClass`

5.16.2 Field Documentation

5.16.2.1 `Methods_t _CPWMC_t::Methods_str`

Virtual methods container

Definition at line 113 of file PWMnCurrFdbkPrivate.h.

Referenced by `ICS_NewObject()`, and `R1_NewObject()`.

5.16.2.2 `Vars_t _CPWMC_t::Vars_str`

Class members container

Definition at line 114 of file PWMnCurrFdbkPrivate.h.

5.16.2.3 `pParams_t _CPWMC_t::pParams_str`

Class parameters container

Definition at line 115 of file PWMnCurrFdbkPrivate.h.

5.16.2.4 `void* _CPWMC_t::DerivedClass`

Pointer to derived class

Definition at line 116 of file PWMnCurrFdbkPrivate.h.

Referenced by ICS_NewObject(), and R1_NewObject().

5.17 `_CRUC_t` Struct Reference

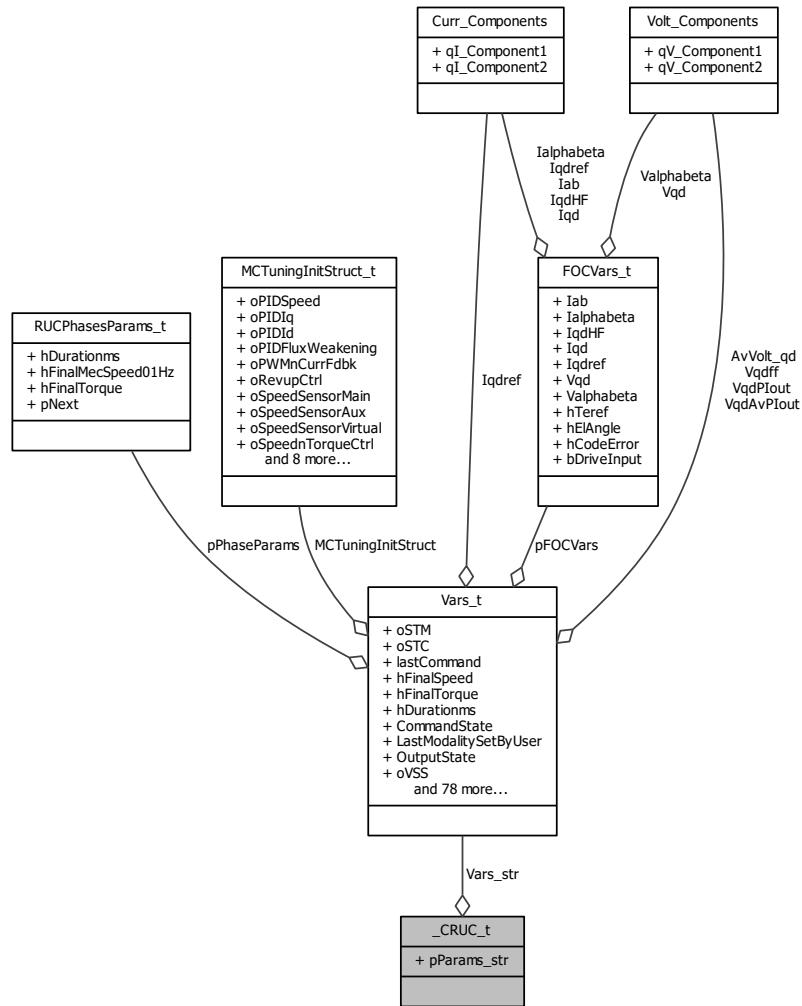
5.17.1 Detailed Description

Private RevupCtrl class definition.

Definition at line 110 of file RevupCtrlPrivate.h.

```
#include <RevupCtrlPrivate.h>
```

Collaboration diagram for `_CRUC_t`:



Data Fields

- `Vars_t Vars_str`
- `pParams_t pParams_str`

5.17.2 Field Documentation

5.17.2.1 `Vars_t _CRUC_t::Vars_str`

Class members container

Definition at line 112 of file RevupCtrlPrivate.h.

5.17.2.2 `pParams_t _CRUC_t::pParams_str`

Class parameters container

Definition at line 113 of file RevupCtrlPrivate.h.

Referenced by RUC_NewObject().

5.18 _CRVBS_VBS_t Struct Reference

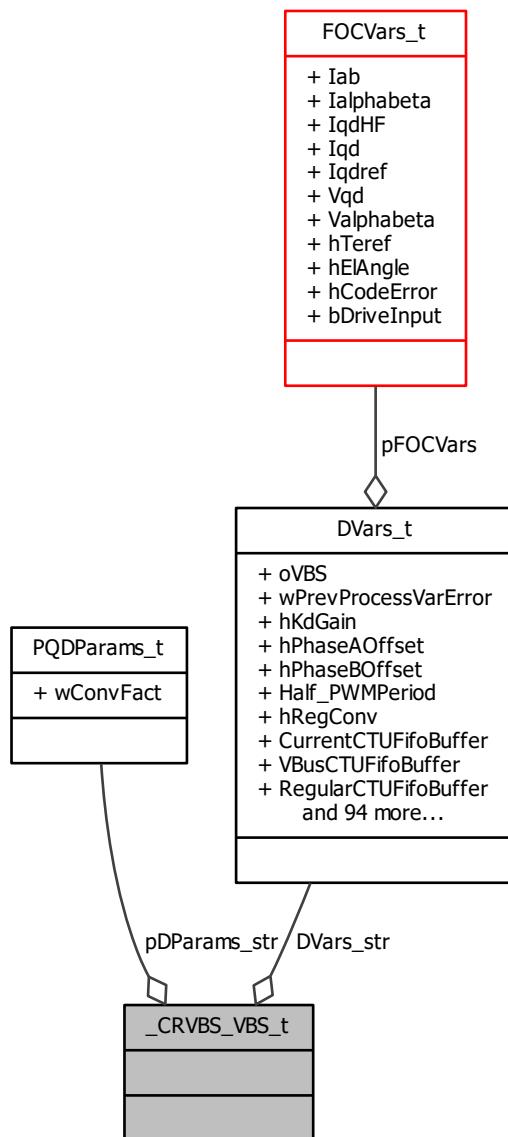
5.18.1 Detailed Description

Private Rdivider class definition.

Definition at line 63 of file Rdivider_BusVoltageSensorPrivate.h.

```
#include <Rdivider_BusVoltageSensorPrivate.h>
```

Collaboration diagram for `_CRVBS_VBS_t`:



Data Fields

- `DVars_t DVars_str`
- `pDParams_t pDParams_str`

5.18.2 Field Documentation

5.18.2.1 `DVars_t _CRVBS_VBS_t::DVars_str`

Derived class members container

Definition at line 65 of file Rdivider_BusVoltageSensorPrivate.h.

5.18.2.2 pDParams_t _CRVBS_VBS_t::pDParams_str

Derived class parameters container

Definition at line 66 of file Rdivider_BusVoltageSensorPrivate.h.

Referenced by RVBS_NewObject().

5.19 _CSPD_t Struct Reference

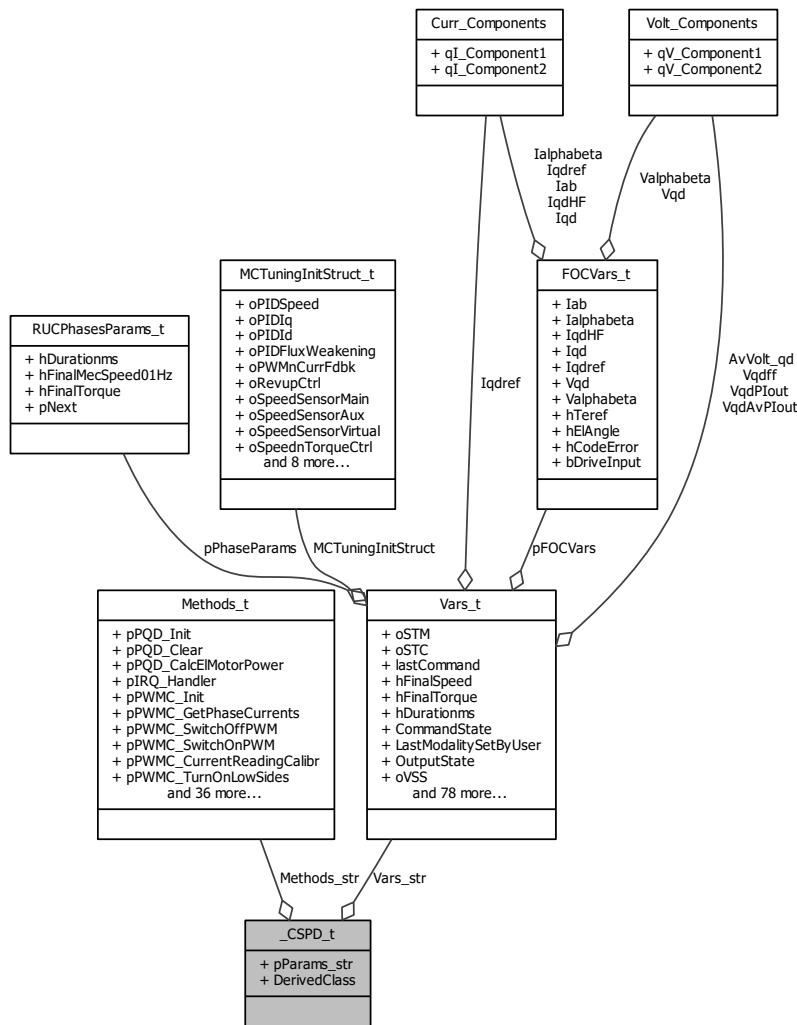
5.19.1 Detailed Description

Private SpeednPosFdbk class definition.

Definition at line 90 of file SpeednPosFdbkPrivate.h.

```
#include <SpeednPosFdbkPrivate.h>
```

Collaboration diagram for `_CSPD_t`:



Data Fields

- `Methods_t Methods_str`
- `Vars_t Vars_str`
- `pParams_t pParams_str`
- `void * DerivedClass`

5.19.2 Field Documentation

5.19.2.1 `Methods_t _CSPD_t::Methods_str`

Virtual methods container

Definition at line 92 of file SpeednPosFdbkPrivate.h.

Referenced by `Enc_NewObject()`, `HALL_NewObject()`, `Res_NewObject()`, `STO_NewObject()`, and `VSS_NewObject()`.

5.19.2.2 `Vars_t_CSPD_t::Vars_str`

Class members container

Definition at line 93 of file SpeednPosFdbkPrivate.h.

5.19.2.3 `pParams_t_CSPD_t::pParams_str`

Class parameters container

Definition at line 94 of file SpeednPosFdbkPrivate.h.

Referenced by `SPD_NewObject()`.

5.19.2.4 `void* _CSPD_t::DerivedClass`

Pointer to derived class

Definition at line 95 of file SpeednPosFdbkPrivate.h.

Referenced by `Enc_NewObject()`, `HALL_NewObject()`, `Res_NewObject()`, `STO_NewObject()`, and `VSS_NewObject()`.

5.20 `_CSTC_t` Struct Reference

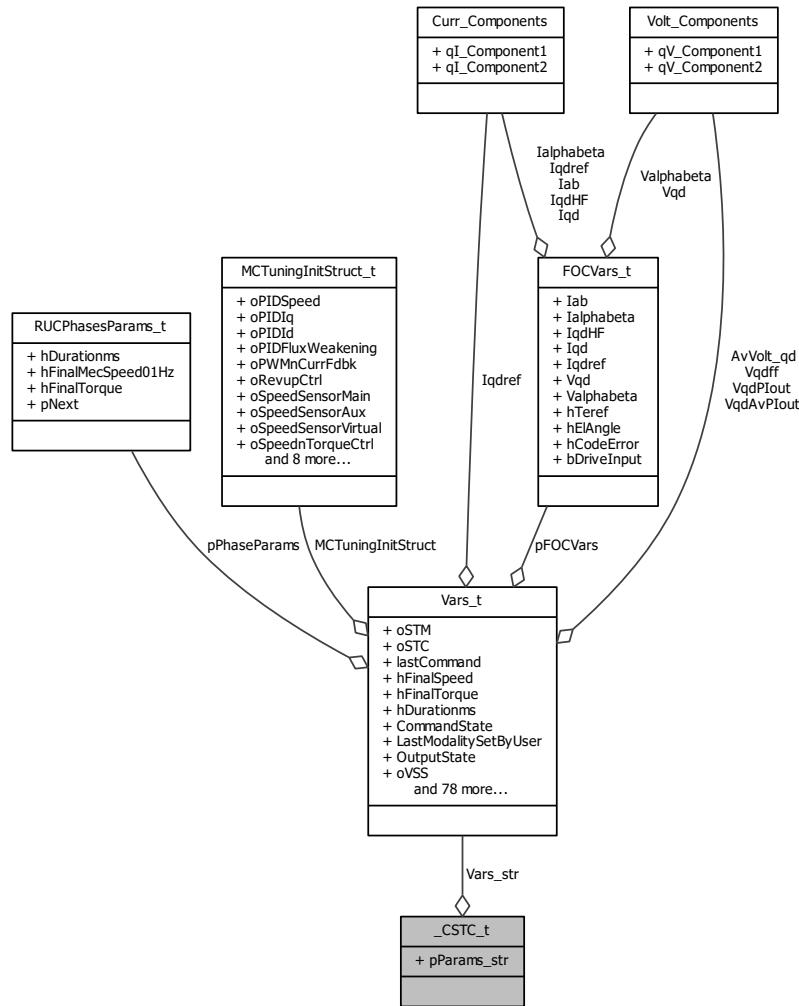
5.20.1 Detailed Description

Private SpeednTorqCtrl class definition.

Definition at line 82 of file SpeednTorqCtrlPrivate.h.

```
#include <SpeednTorqCtrlPrivate.h>
```

Collaboration diagram for `_CSTC_t`:



Data Fields

- `Vars_t Vars_str`
- `pParams_t pParams_str`

5.20.2 Field Documentation

5.20.2.1 `Vars_t _CSTC_t::Vars_str`

Class members container

Definition at line 84 of file SpeednTorqCtrlPrivate.h.

5.20.2.2 `pParams_t _CSTC_t::pParams_str`

Class parameters container

Definition at line 85 of file SpeednTorqCtrlPrivate.h.

Referenced by STC_NewObject().

5.21 _CSTM_t Struct Reference

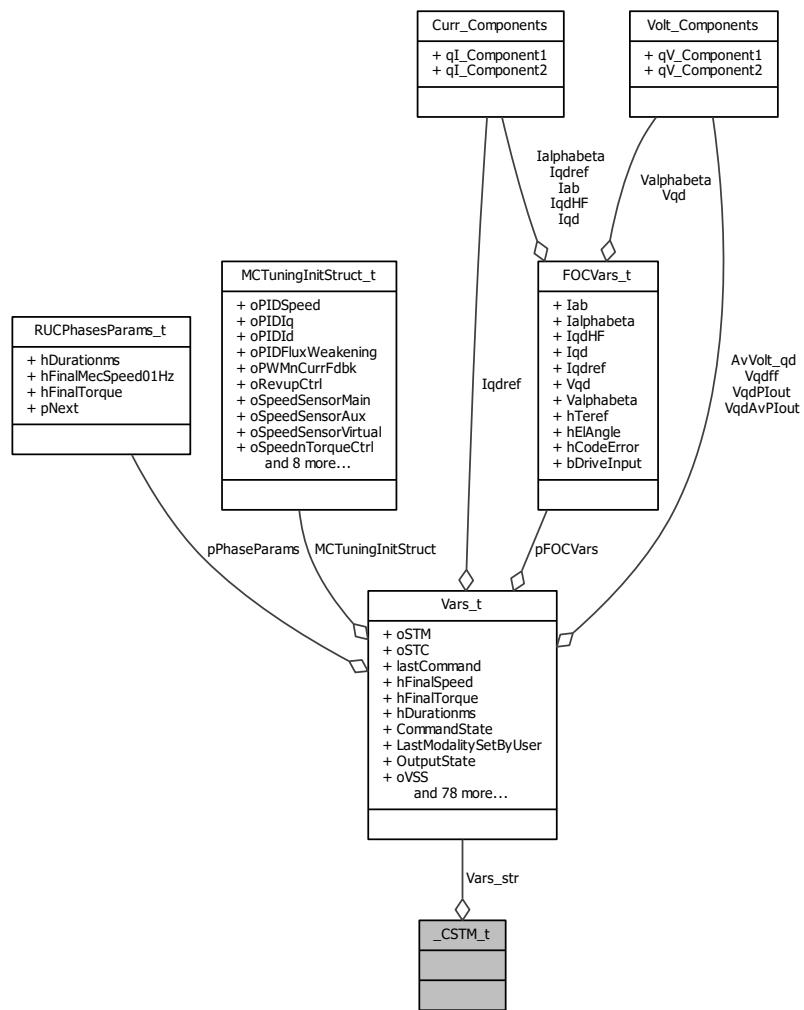
5.21.1 Detailed Description

Private StateMachine class definition.

Definition at line 63 of file StateMachinePrivate.h.

```
#include <StateMachinePrivate.h>
```

Collaboration diagram for _CSTM_t:



Data Fields

- `Vars_t` `Vars_str`

5.21.2 Field Documentation

5.21.2.1 Vars_t _CSTM_t::Vars_str

Class members container

Definition at line 65 of file StateMachinePrivate.h.

5.22 _CTSNS_t Struct Reference

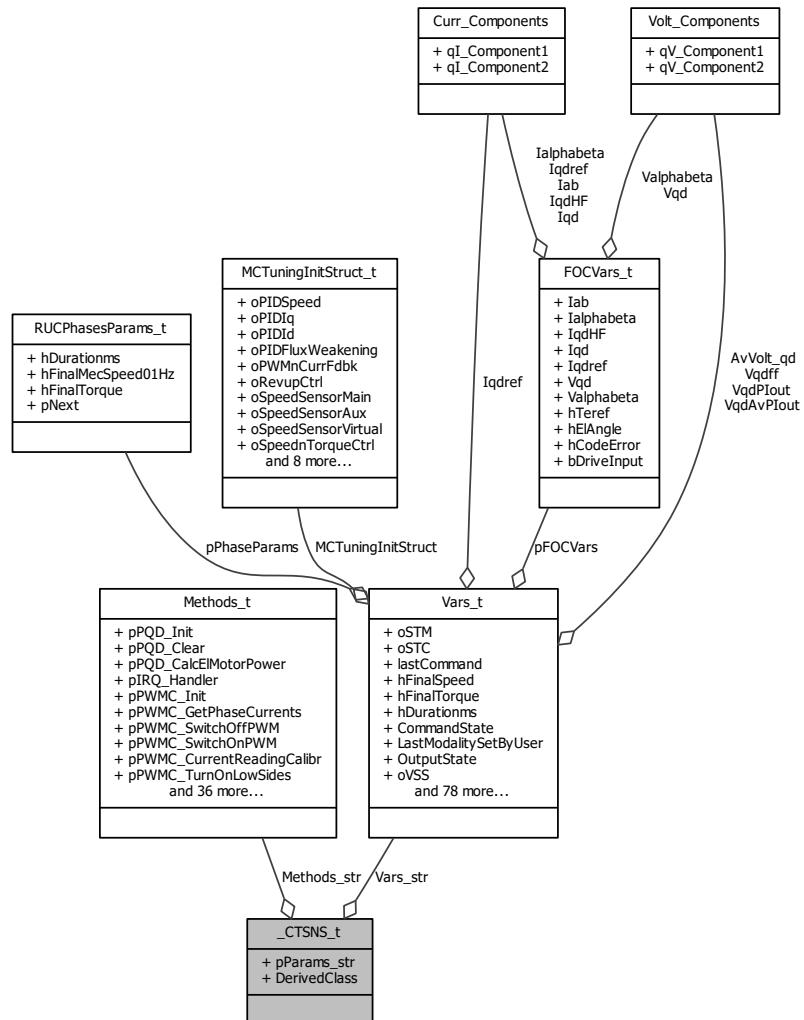
5.22.1 Detailed Description

Private TemperatureSensor class definition.

Definition at line 76 of file TemperatureSensorPrivate.h.

```
#include <TemperatureSensorPrivate.h>
```

Collaboration diagram for _CTSNS_t:



Data Fields

- [Methods_t Methods_str](#)
- [Vars_t Vars_str](#)
- [pParams_t pParams_str](#)
- [void * DerivedClass](#)

5.22.2 Field Documentation

5.22.2.1 Methods_t _CTSNS_t::Methods_str

Virtual methods container

Definition at line 78 of file TemperatureSensorPrivate.h.

Referenced by NTC_NewObject(), and VTS_NewObject().

5.22.2.2 Vars_t _CTSNS_t::Vars_str

Class members container

Definition at line 79 of file TemperatureSensorPrivate.h.

5.22.2.3 pParams_t _CTSNS_t::pParams_str

Class parameters container

Definition at line 80 of file TemperatureSensorPrivate.h.

Referenced by TSNS_NewObject().

5.22.2.4 void* _CTSNS_t::DerivedClass

Pointer to derived class

Definition at line 81 of file TemperatureSensorPrivate.h.

Referenced by NTC_NewObject(), and VTS_NewObject().

5.23 _CVBS_t Struct Reference

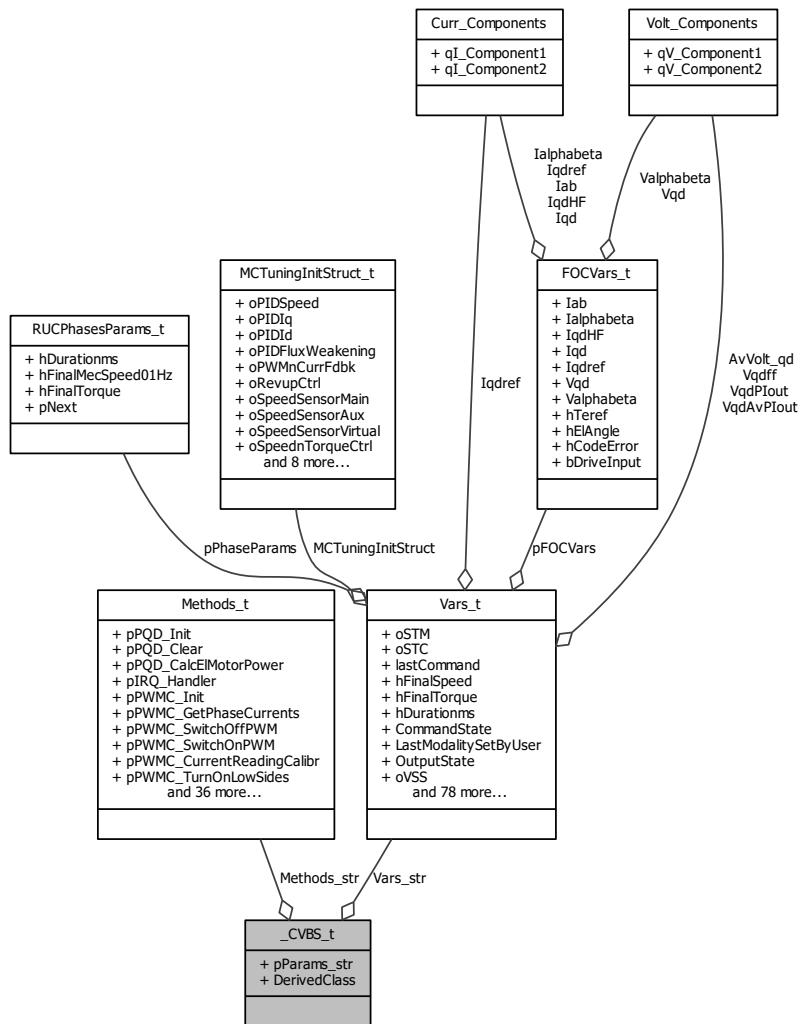
5.23.1 Detailed Description

Private BusVoltageSensor class definition.

Definition at line 77 of file BusVoltageSensorPrivate.h.

```
#include <BusVoltageSensorPrivate.h>
```

Collaboration diagram for `_CVBS_t`:



Data Fields

- `Methods_t Methods_str`
- `Vars_t Vars_str`
- `pParams_t pParams_str`
- `void * DerivedClass`

5.23.2 Field Documentation

5.23.2.1 `Methods_t _CVBS_t::Methods_str`

Virtual methods container

Definition at line 79 of file BusVoltageSensorPrivate.h.

Referenced by `RVBS_NewObject()`, and `VVBS_NewObject()`.

5.23.2.2 `Vars_t_CVBS_t::Vars_str`

Class members container

Definition at line 80 of file BusVoltageSensorPrivate.h.

5.23.2.3 `pParams_t_CVBS_t::pParams_str`

Class parameters container

Definition at line 81 of file BusVoltageSensorPrivate.h.

Referenced by VBS_NewObject().

5.23.2.4 `void* _CVBS_t::DerivedClass`

Pointer to derived class

Definition at line 82 of file BusVoltageSensorPrivate.h.

Referenced by RVBS_NewObject(), and VVBS_NewObject().

5.24 `_CVTS_TSNS_t` Struct Reference

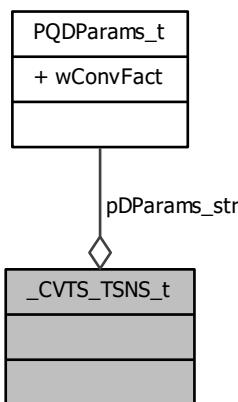
5.24.1 Detailed Description

Private Virtual temperature sensor class definition.

Definition at line 57 of file Virtual_TemperatureSensorPrivate.h.

```
#include <Virtual_TemperatureSensorPrivate.h>
```

Collaboration diagram for `_CVTS_TSNS_t`:



Data Fields

- `pDParams_t pDParams_str`

5.24.2 Field Documentation

5.24.2.1 pDParams_t _CVTS_TSNS_t::pDParams_str

Derived class parameters container

Definition at line 59 of file Virtual_TemperatureSensorPrivate.h.

Referenced by VTS_NewObject().

5.25 _CVVBS_VBS_t Struct Reference

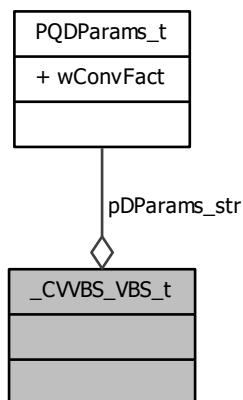
5.25.1 Detailed Description

Private Virtual Vbus sensor class definition.

Definition at line 54 of file Virtual_BusVoltageSensorPrivate.h.

```
#include <Virtual_BusVoltageSensorPrivate.h>
```

Collaboration diagram for _CVVBS_VBS_t:



Data Fields

- pDParams_t pDParams_str

5.25.2 Field Documentation

5.25.2.1 pDParams_t _CVVBS_VBS_t::pDParams_str

Derived class parameters container

Definition at line 56 of file Virtual_BusVoltageSensorPrivate.h.

Referenced by VVBS_NewObject().

5.26 _DCENC_SPD_t Struct Reference

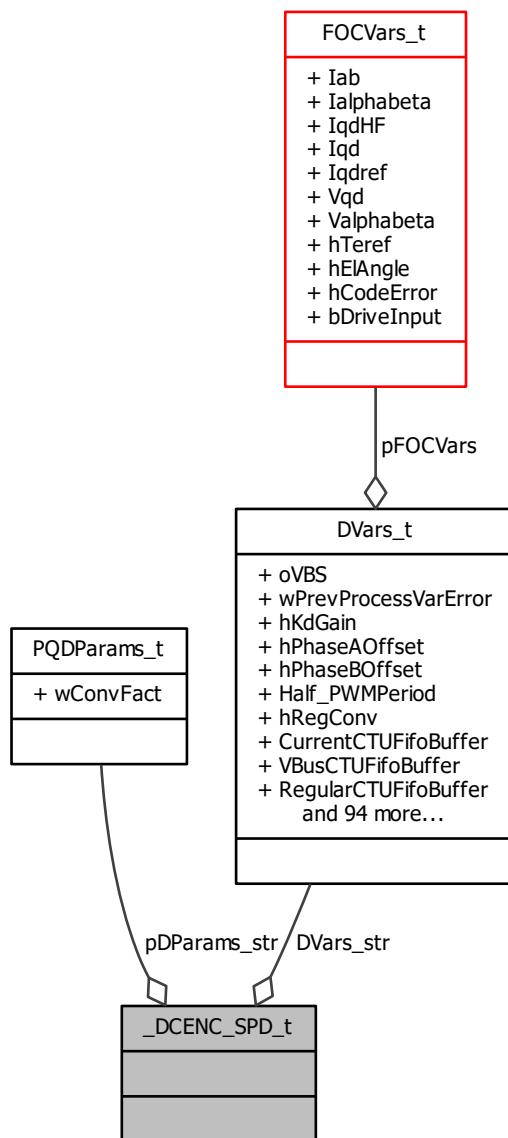
5.26.1 Detailed Description

Private ENCODER class definition.

Definition at line 95 of file ENCODER_SpeednPosFdbkPrivate.h.

```
#include <ENCODER_SpeednPosFdbkPrivate.h>
```

Collaboration diagram for _DCENC_SPD_t:



Data Fields

- DVars_t DVars_str
- pDParams_t pDParams_str

5.26.2 Field Documentation

5.26.2.1 DVars_t _DCENC_SPD_t::DVars_str

Derived class members container

Definition at line 97 of file ENCODER_SpeednPosFdbkPrivate.h.

5.26.2.2 pDParams_t _DCENC_SPD_t::pDParams_str

Derived class parameters container

Definition at line 98 of file ENCODER_SpeednPosFdbkPrivate.h.

Referenced by Enc_NewObject().

5.27 _DCHALL_SPD_t Struct Reference

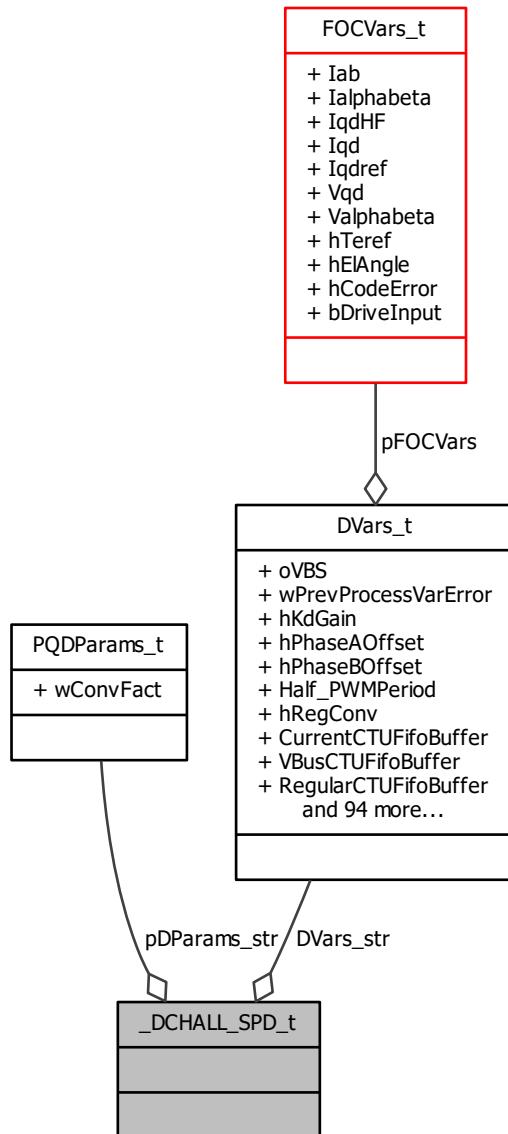
5.27.1 Detailed Description

Private HALL class definition.

Definition at line 100 of file HALL_SpeednPosFdbkPrivate.h.

```
#include <HALL_SpeednPosFdbkPrivate.h>
```

Collaboration diagram for `_DCHALL_SPD_t`:



Data Fields

- `DVars_t DVars_str`
- `pDParams_t pDParams_str`

5.27.2 Field Documentation

5.27.2.1 `DVars_t _DCHALL_SPD_t::DVars_str`

Derived class members container

Definition at line 102 of file HALL_SpeednPosFdbkPrivate.h.

5.27.2.2 pDParams_t _DCHALL_SPD_t::pDParams_str

Derived class parameters container

Definition at line 103 of file HALL_SpeednPosFdbkPrivate.h.

Referenced by HALL_NewObject().

5.28 _DCICS_PWMC_t Struct Reference

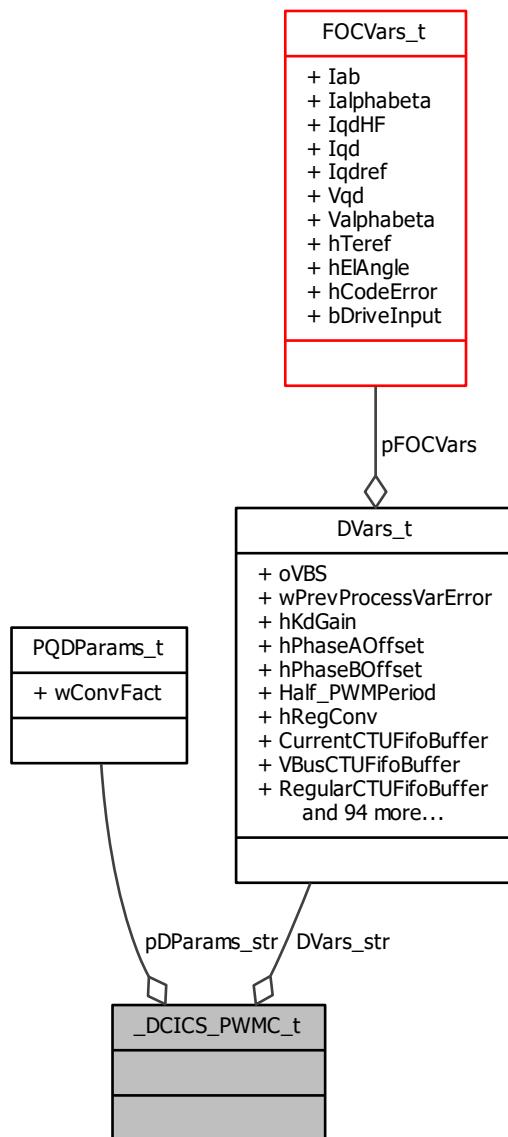
5.28.1 Detailed Description

Private ICS class definition.

Definition at line 73 of file ICS_PWMnCurrFdbkPrivate.h.

```
#include <ICS_PWMnCurrFdbkPrivate.h>
```

Collaboration diagram for `_DCICS_PWMC_t`:



Data Fields

- `DVars_t DVars_str`
- `pDParams_t pDParams_str`

5.28.2 Field Documentation

5.28.2.1 `DVars_t _DCICS_PWMC_t::DVars_str`

Derived class members container

Definition at line 75 of file ICS_PWMnCurrFdbkPrivate.h.

5.28.2.2 pDParams_t _DCICS_PWMC_t::pDParams_str

Derived class parameters container

Definition at line 76 of file ICS_PWMnCurrFdbkPrivate.h.

Referenced by ICS_NewObject().

5.29 _DCPID_t Struct Reference

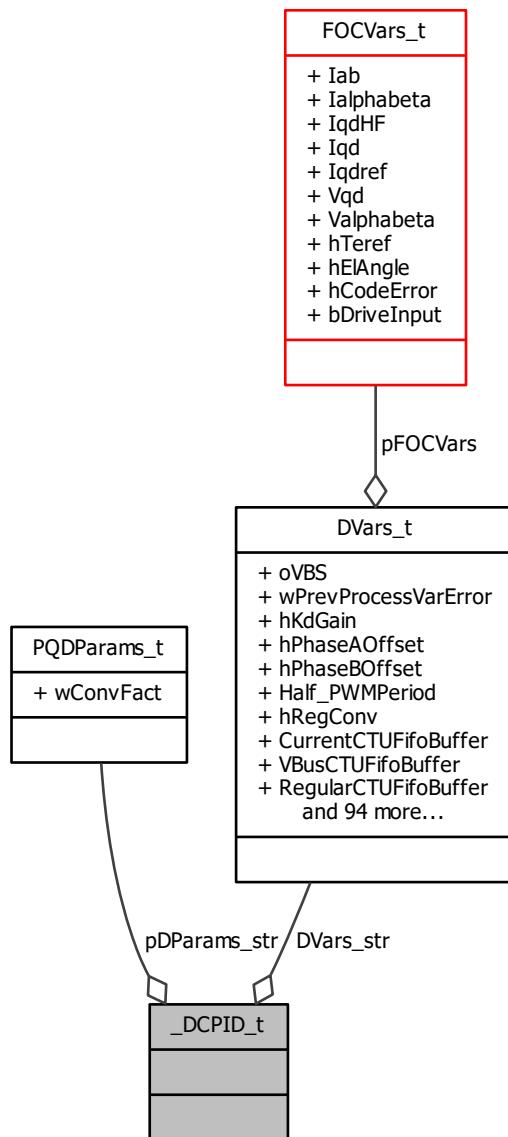
5.29.1 Detailed Description

Private PID class definition.

Definition at line 64 of file PID_PIRegulatorPrivate.h.

```
#include <PID_PIRegulatorPrivate.h>
```

Collaboration diagram for `_DCPID_t`:



5.30 _DCPQD_MPM.t Struct Reference

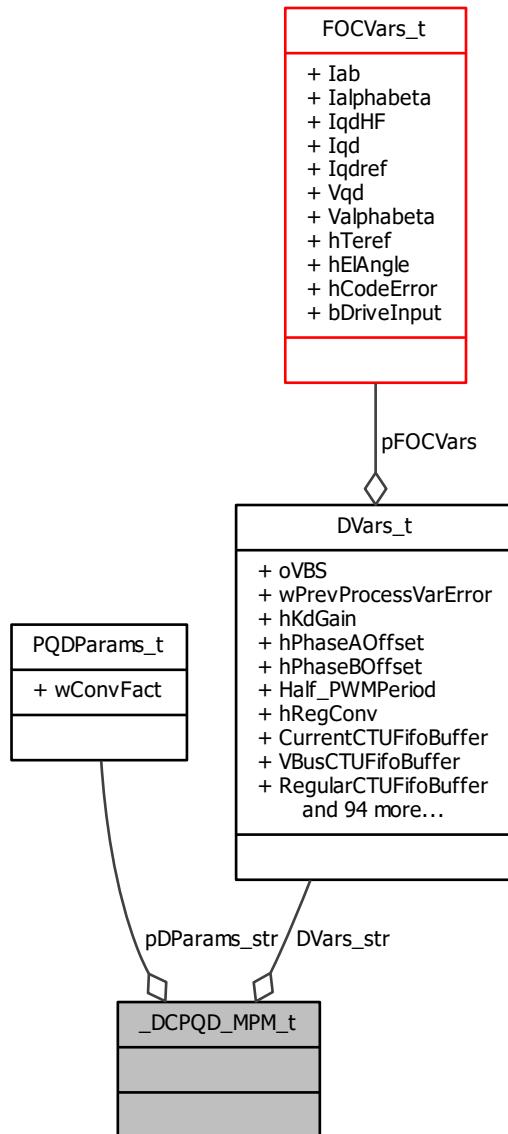
5.30.1 Detailed Description

Private PQD class definition.

Definition at line 63 of file `PQD_MotorPowerMeasurementPrivate.h`.

```
#include <PQD_MotorPowerMeasurementPrivate.h>
```

Collaboration diagram for `_DCPQD_MPM_t`:



Data Fields

- `DVars_t DVars_str`
- `pDParams_t pDParams_str`

5.30.2 Field Documentation

5.30.2.1 `DVars_t _DCPQD_MPM_t::DVars_str`

Derived class members container

Definition at line 65 of file PQD_MotorPowerMeasurementPrivate.h.

5.30.2.2 `pDParams_t _DCPQD_MPM_t::pDParams_str`

Derived class parameters container

Definition at line 66 of file PQD_MotorPowerMeasurementPrivate.h.

Referenced by PQD_NewObject().

5.31 `_DCR1_PWMC_t` Struct Reference

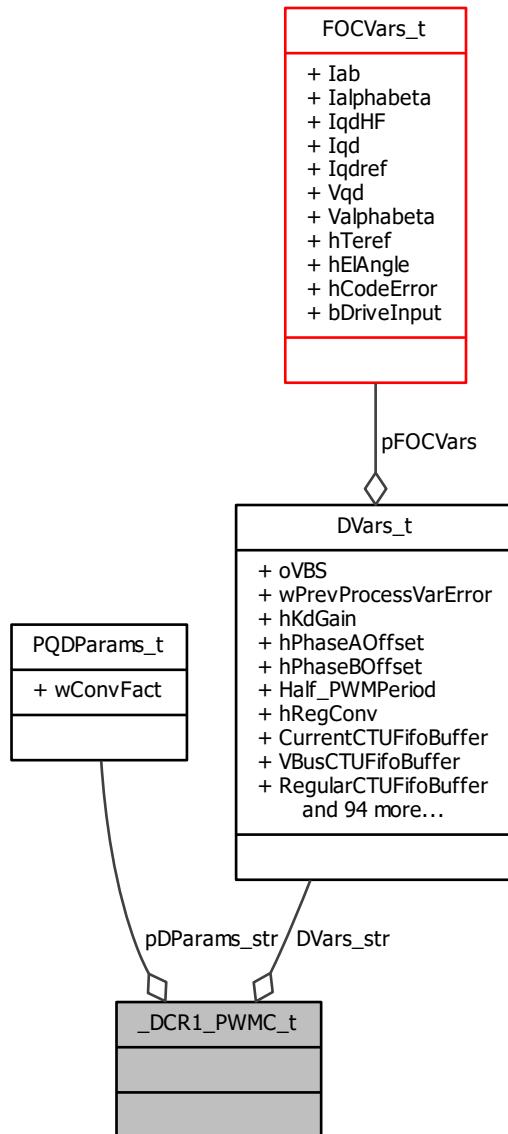
5.31.1 Detailed Description

Private R1 class definition.

Definition at line 106 of file R1_PWMnCurrFdbkPrivate.h.

```
#include <R1_PWMnCurrFdbkPrivate.h>
```

Collaboration diagram for `_DCR1_PWMC_t`:



Data Fields

- `DVars_t DVars_str`
- `pDParams_t pDParams_str`

5.31.2 Field Documentation

5.31.2.1 `DVars_t _DCR1_PWMC_t::DVars_str`

Derived class members container

Definition at line 108 of file R1_PWMnCurrFdbkPrivate.h.

5.31.2.2 pDParams_t _DCR1_PWMC_t::pDParams_str

Derived class parameters container

Definition at line 109 of file R1_PWMnCurrFdbkPrivate.h.

Referenced by R1_NewObject().

5.32 _DCRES_SPD_t Struct Reference

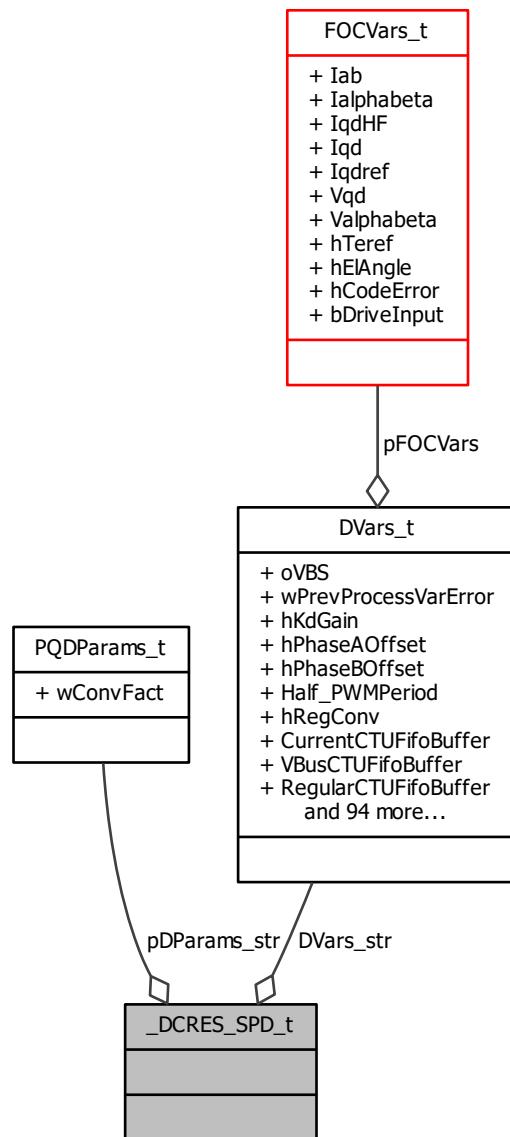
5.32.1 Detailed Description

Private RESOLVER class definition.

Definition at line 83 of file RESOLVER_SpeednPosFdbkPrivate.h.

```
#include <RESOLVER_SpeednPosFdbkPrivate.h>
```

Collaboration diagram for _DCRES_SPD_t:



5.33 _DCSTO_SPD_t Struct Reference

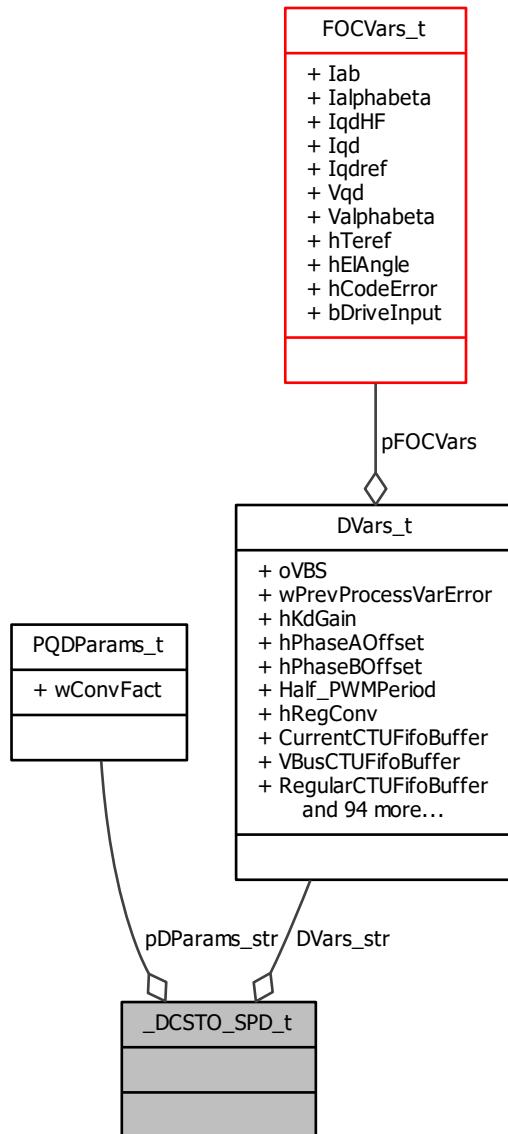
5.33.1 Detailed Description

Private STO class definition.

Definition at line 110 of file `STO_SpeednPosFdbkPrivate.h`.

```
#include <STO_SpeednPosFdbkPrivate.h>
```

Collaboration diagram for `_DCSTO_SPD_t`:



Data Fields

- `DVars_t DVars_str`
- `pDParams_t pDParams_str`

5.33.2 Field Documentation

5.33.2.1 `DVars_t _DCSTO_SPD_t::DVars_str`

Derived class members container

Definition at line 112 of file STO_SpeednPosFdbkPrivate.h.

5.33.2.2 pDParams_t _DCSTO_SPD_t::pDParams_str

Derived class parameters container

Definition at line 113 of file STO_SpeednPosFdbkPrivate.h.

Referenced by STO_NewObject().

5.34 _DCVSS_SPD_t Struct Reference

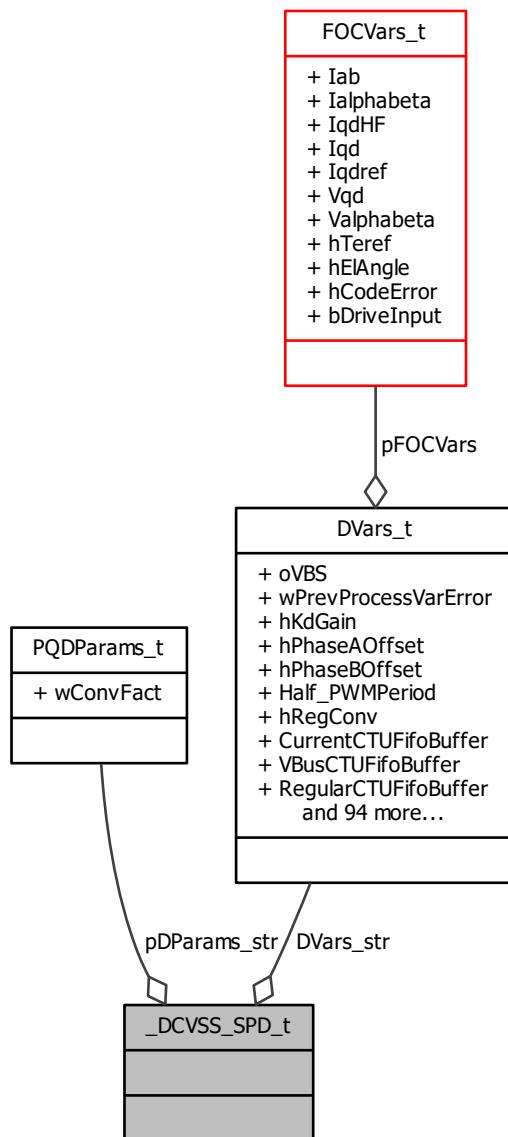
5.34.1 Detailed Description

Private VirtualSpeedSensor class definition.

Definition at line 65 of file VirtualSpeedSensor_SpeednPosFdbkPrivate.h.

```
#include <VirtualSpeedSensor_SpeednPosFdbkPrivate.h>
```

Collaboration diagram for `_DCVSS_SPD_t`:



Data Fields

- `DVars_t DVars_str`
- `pDParams_t pDParams_str`

5.34.2 Field Documentation

5.34.2.1 `DVars_t _DCVSS_SPD_t::DVars_str`

Derived class members container

Definition at line 67 of file VirtualSpeedSensor_SpeednPosFdbkPrivate.h.

5.34.2.2 **pDParams_t _DCVSS_SPD_t::pDParams_str**

Derived class parameters container

Definition at line 68 of file VirtualSpeedSensor_SpeednPosFdbkPrivate.h.

Referenced by VSS_NewObject().

5.35 BusVoltageSensorParams_t Struct Reference

5.35.1 Detailed Description

BusVoltageSensor class parameters definition.

Definition at line 58 of file BusVoltageSensorClass.h.

```
#include <BusVoltageSensorClass.h>
```

Data Fields

- [SensorType_t bSensorType](#)
- [uint16_t hConversionFactor](#)

5.35.2 Field Documentation

5.35.2.1 **SensorType_t BusVoltageSensorParams_t::bSensorType**

It contains the information about the type of instanced bus voltage sensor object. It can be equal to REAL_SENSOR or VIRTUAL_SENSOR

Definition at line 60 of file BusVoltageSensorClass.h.

5.35.2.2 **uint16_t BusVoltageSensorParams_t::hConversionFactor**

It is used to convert bus voltage from u16Volts into real Volts (V). $1 \text{ u16Volt} = 65536/\text{hConversionFactor} \text{ Volts}$ For real sensors hConversionFactor it's equal to the product between the expected MCU voltage and the voltage sensing network attenuation. For virtual sensors it must be equal to 500

Definition at line 64 of file BusVoltageSensorClass.h.

5.36 CircleLimitationParams_t Struct Reference

5.36.1 Detailed Description

CircleLimitation class parameters definition.

Definition at line 57 of file CircleLimitationClass.h.

```
#include <CircleLimitationClass.h>
```

Data Fields

- uint16_t `hMaxModule`
- uint16_t `hCircle_limit_table` [87]
- uint8_t `bStart_index`

5.36.2 Field Documentation

5.36.2.1 uint16_t CircleLimitationParams_t::hMaxModule

Circle limitation maximum allowed module

Definition at line 59 of file CircleLimitationClass.h.

5.36.2.2 uint16_t CircleLimitationParams_t::hCircle_limit_table[87]

Circle limitation table

Definition at line 61 of file CircleLimitationClass.h.

5.36.2.3 uint8_t CircleLimitationParams_t::bStart_index

Circle limitation table indexing start

Definition at line 62 of file CircleLimitationClass.h.

5.37 DigitalOutputParams_t Struct Reference

5.37.1 Detailed Description

DigitalOutput class parameters definition.

Definition at line 82 of file DigitalOutputClass.h.

```
#include <DigitalOutputClass.h>
```

Data Fields

- GPIO_TypeDef * `hDOutputPort`
- uint16_t `hDOutputPin`
- uint8_t `bDOutputPolarity`

5.37.2 Field Documentation

5.37.2.1 GPIO_TypeDef* DigitalOutputParams_t::hDOutputPort

GPIO output port. It must be equal to GPIOx x= A, B, ...

Definition at line 84 of file DigitalOutputClass.h.

5.37.2.2 uint16_t DigitalOutputParams_t::hDOutputPin

GPIO output pin. It must be equal to GPIO_Pin_x x= 0, 1, ...

Definition at line 86 of file DigitalOutputClass.h.

5.37.2.3 uint8_t DigitalOutputParams_t::bDOOutputPolarity

GPIO output polarity. It must be equal to DOOutputActiveHigh or DOOutputActiveLow

Definition at line 88 of file DigitalOutputClass.h.

5.38 DVars_t Struct Reference

5.38.1 Detailed Description

PQD class members definition.

UnidirectionalFastCom class members definition.

DACRCTIMER class members definition.

Rdivider class members definition.

NTC class members definition.

VirtualSpeedSensor class members definition.

STO class members definition.

RESOLVER class members definition.

HALL class members definition.

ENCODER class members definition.

R1 class members definition.

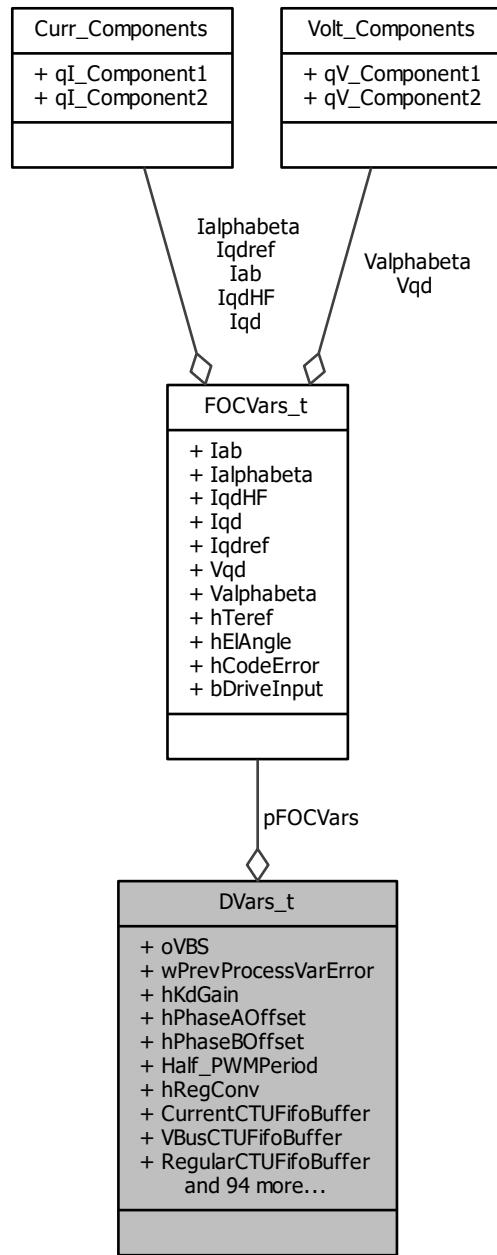
ICS class members definition.

PID class members definition.

Definition at line 49 of file PQD_MotorPowerMeasurementPrivate.h.

```
#include <PQD_MotorPowerMeasurementPrivate.h>
```

Collaboration diagram for DVars_t:



Data Fields

- `pFOCVars_t pFOCVars`
- `CVBS oVBS`
- `uint16_t hPhaseAOOffset`
- `uint16_t hPhaseBOOffset`
- `uint16_t Half_PWMPeriod`

- uint16_t **hRegConv**
- uint32_t **CurrentCTUFifoBuffer** [ICS_ADC_CURR_NUM]
- uint32_t **VBusCTUFifoBuffer**
- uint32_t **RegularCTUFifoBuffer**
- uint16_t **hPhaseOffset**
- uint16_t **hCntSmp1**
- uint16_t **hCntSmp2**
- uint8_t **sampCur1**
- uint8_t **sampCur2**
- int16_t **hCurrAOld**
- int16_t **hCurrBOld**
- int16_t **hCurrCOld**
- uint8_t **bInverted_pwm_new**
- uint16_t **hFlags**
- volatile uint16 **hTimerOverflowNb**
- boolean **SensorIsReliable**
- uint16 **hPreviousCapture**
- sint32 **wDeltaCapturesBuffer** [ENC_SPEED_ARRAY_SIZE]
- uint32 **wU32MAXdivPulseNumber**
- uint16 **hSpeedSamplingFreqHz**
- boolean **TimerOverflowError**
- uint16 **e_angle_first_index_value**
- uint16 **e_angle_current_index_value**
- uint8 **direction**
- boolean **bls_First_Measurement**
- boolean **blIndex_Signal**
- sint16 **hMeasuredElAngle**
- sint16 **hPrev_MeasuredElAngle**
- uint16 **h_u_angle**
- uint32 **speed_buffer** [HALL_AVERAGING_FIFO_DEPTH]
- sint32 **Hall_current_speed**
- uint16 **h_polar_pairs**
- sint8 **bSpeed**
- sint16 **hAvrElSpeedDpp**
- uint8 **bHallState**
- Resolver_Ranges **pole_pair_ranges** [2U]
- uint32 **ResCTUFifoBuffer** [2U]
- int16_t **hC1**
- int16_t **hC2**
- int16_t **hC3**
- int16_t **hC4**
- int16_t **hC5**
- int16_t **hC6**
- int16_t **hF1**
- int16_t **hF2**
- int16_t **hF3**
- uint16_t **hF3POW2**
- CPI oPIRegulator
- int32_t **wlalfa_est**
- int32_t **wlbeta_est**
- int32_t **wBemf_alpha_est**
- int32_t **wBemf_beta_est**
- int16_t **hBemf_alpha_est**
- int16_t **hBemf_beta_est**
- int16_t **hSpeed_Buffer** [64]

- `uint8_t bSpeed_Buffer_Index`
- `bool blsSpeedReliable`
- `uint8_t bConsistencyCounter`
- `uint8_t bReliabilityCounter`
- `bool blsAlgorithmConverged`
- `bool blsBemfConsistent`
- `int32_t wObs_Bemf_Level`
- `int32_t wEst_Bemf_Level`
- `bool bEnableDualCheck`
- `int32_t wDppBufferSum`
- `int16_t hSpeedBufferOldestEI`
- `int32_t wElAccDppP32`
- `int32_t wElSpeedDpp32`
- `uint16_t hRemainingStep`
- `int16_t hFinalMecSpeed01Hz`
- `bool bTransitionStarted`
- `bool bTransitionEnded`
- `int16_t hTransitionRemainingSteps`
- `int16_t hElAngleAccu`
- `bool bTransitionLocked`
- `bool IsOverTempFaultActive`
- `MC_Protocol_REG_t bChannel_variable [DAC_CH_NBR]`
- `int16_t hUserValue [DAC_CH_USER]`
- `bool comON`
- `MC_Protocol_REG_t bChannel [2]`
- `uint8_t bChTransmitted`
- `uint8_t bByteTransmitted`
- `int32_t wBuffer`
- `uint8_t bChByteNum [2]`
- `uint8_t bChNum`

5.38.2 Field Documentation

5.38.2.1 pFOCVars_t DVars_t::pFOCVars

Pointer to FOC vars used by MPM.

Definition at line 51 of file PQD_MotorPowerMeasurementPrivate.h.

5.38.2.2 CVBS DVars_t::oVBS

Bus voltage sensor object used by PQD.

Definition at line 52 of file PQD_MotorPowerMeasurementPrivate.h.

5.38.2.3 uint16_t DVars_t::hPhaseAOffset

Offset of Phase A current sensing network

Definition at line 54 of file ICS_PWMnCurrFdbkPrivate.h.

5.38.2.4 uint16_t DVars_t::hPhaseBOffset

Offset of Phase B current sensing network

Definition at line 55 of file ICS_PWMnCurrFdbkPrivate.h.

5.38.2.5 uint16_t DVars_t::Half_PWMPeriod

Half PWM Period in timer clock counts

Definition at line 56 of file ICS_PWMnCurrFdbkPrivate.h.

5.38.2.6 uint16_t DVars_t::hRegConv

Variable used to store regular conversions result

Temporary variables used to store regular conversions

Definition at line 57 of file ICS_PWMnCurrFdbkPrivate.h.

5.38.2.7 uint32_t DVars_t::CurrentCTUFifoBuffer

Array used to store regular conversions result

Array used to store regular conversions

Definition at line 59 of file ICS_PWMnCurrFdbkPrivate.h.

5.38.2.8 uint32_t DVars_t::VBusCTUFifoBuffer

Voltage Bus CTU buffers

Definition at line 61 of file ICS_PWMnCurrFdbkPrivate.h.

5.38.2.9 uint32_t DVars_t::RegularCTUFifoBuffer

User Conversions CTU buffers

Definition at line 62 of file ICS_PWMnCurrFdbkPrivate.h.

5.38.2.10 uint16_t DVars_t::hPhaseOffset

Offset of current sensing network

Definition at line 69 of file R1_PWMnCurrFdbkPrivate.h.

5.38.2.11 uint16_t DVars_t::hCntSmp1

First sampling point express in timer counts

Definition at line 72 of file R1_PWMnCurrFdbkPrivate.h.

5.38.2.12 uint16_t DVars_t::hCntSmp2

Second sampling point express in timer counts

Definition at line 73 of file R1_PWMnCurrFdbkPrivate.h.

5.38.2.13 uint8_t DVars_t::sampCur1

Current sampled in the first sampling point

Definition at line 74 of file R1_PWMnCurrFdbkPrivate.h.

5.38.2.14 uint8_t DVars_t::sampCur2

Current sampled in the second sampling point

Definition at line 75 of file R1_PWMnCurrFdbkPrivate.h.

5.38.2.15 int16_t DVars_t::hCurrAOld

Previous measured value of phase A current

Definition at line 76 of file R1_PWMnCurrFdbkPrivate.h.

5.38.2.16 int16_t DVars_t::hCurrBOld

Previous measured value of phase B current

Definition at line 77 of file R1_PWMnCurrFdbkPrivate.h.

5.38.2.17 int16_t DVars_t::hCurrCOld

Previous measured value of phase C current

Definition at line 78 of file R1_PWMnCurrFdbkPrivate.h.

5.38.2.18 uint8_t DVars_t::blInverted_pwm_new

This value indicates the type of the current PWM period (Regular, Distort PHA, PHB or PHC)

Definition at line 79 of file R1_PWMnCurrFdbkPrivate.h.

5.38.2.19 uint16_t DVars_t::hFlags

Flags EFOC: Flag to indicate end of FOC duty available STBD3: Flag to indicate which phase has been distorted in boundary 3 zone (A or B) DSTEN: Flag to indicate if the distortion must be performed or not (charge of bootstrap capacitor phase) SOFOC: This flag will be reset to zero at the begin of FOC and will be set in the UP IRQ. If at the end of FOC it is set the software error must be generated CALIB: This flag is used to indicate the ADC calibration phase in order to avoid concurrent regular conversions

Definition at line 81 of file R1_PWMnCurrFdbkPrivate.h.

5.38.2.20 volatile uint16 DVars_t::hTimerOverflowNb

Number of overflows occurred since last speed measurement event

Definition at line 54 of file ENCODER_SpeednPosFdbkPrivate.h.

5.38.2.21 boolean DVars_t::SensorIsReliable

Flag to indicate sensor/decoding is not properly working.

Flag to indicate a wrong configuration of the Hall sensor signanls.

Definition at line 57 of file ENCODER_SpeednPosFdbkPrivate.h.

5.38.2.22 uint16 DVars_t::hPreviousCapture

Timer counter value captured during previous speed measurement event

Definition at line 60 of file ENCODER_SpeednPosFdbkPrivate.h.

5.38.2.23 sint32 DVars_t::wDeltaCapturesBuffer[ENC_SPEED_ARRAY_SIZE]

Buffer used to store captured variations of timer counter

Definition at line 63 of file ENCODER_SpeednPosFdbkPrivate.h.

5.38.2.24 uint32 DVars_t::wU32MAXdivPulseNumber

<Buffer index

Definition at line 68 of file ENCODER_SpeednPosFdbkPrivate.h.

5.38.2.25 uint16 DVars_t::hSpeedSamplingFreqHz

<It stores U32MAX/hPulseNumber

Definition at line 70 of file ENCODER_SpeednPosFdbkPrivate.h.

5.38.2.26 boolean DVars_t::TimerOverflowError

<Frequency (Hz) at which motor speed is to be computed. TRUE if the number of overflow occurred is greater than 'define' ENC_MAX_OVERFLOW_NB

Definition at line 73 of file ENCODER_SpeednPosFdbkPrivate.h.

5.38.2.27 uint16 DVars_t::e_angle_first_index_value

First Electric angle for encoder index signal

Definition at line 76 of file ENCODER_SpeednPosFdbkPrivate.h.

5.38.2.28 uint16 DVars_t::e_angle_current_index_value

Current Electric angle for encoder index signal

Definition at line 78 of file ENCODER_SpeednPosFdbkPrivate.h.

5.38.2.29 uint8 DVars_t::direction

Encoder direction

Definition at line 80 of file ENCODER_SpeednPosFdbkPrivate.h.

5.38.2.30 boolean DVars_t::bls_First_Measurement

First measurement in motor speed calculation.

Definition at line 82 of file ENCODER_SpeednPosFdbkPrivate.h.

5.38.2.31 boolean DVars_t::blIndex_Signal

Index signal captured (used in motor speed calculation)

Definition at line 84 of file ENCODER_SpeednPosFdbkPrivate.h.

5.38.2.32 sint16 DVars_t::hMeasuredEIAngle

This is the electrical angle measured at each Hall sensor signal edge. It is considered the best measurement of electrical rotor angle.

Definition at line 64 of file HALL_SpeednPosFdbkPrivate.h.

5.38.2.33 sint16 DVars_t::hPrev_MeasuredEIAngle

This is the previous electrical angle measured at each Hall sensor signal edge.

Definition at line 68 of file HALL_SpeednPosFdbkPrivate.h.

5.38.2.34 uint16 DVars_t::h_u_angle

Angle captured in unsigned format [0..65535]

Definition at line 71 of file HALL_SpeednPosFdbkPrivate.h.

5.38.2.35 uint32 DVars_t::speed_buffer

Buffer of last measured angle in long format from Hall Sensor

Buffer of last measured angle in long format from Resolver

Definition at line 73 of file HALL_SpeednPosFdbkPrivate.h.

5.38.2.36 sint32 DVars_t::Hall_current_speed

Current computed speed.

Definition at line 75 of file HALL_SpeednPosFdbkPrivate.h.

5.38.2.37 uint16 DVars_t::h_polar_pairs

Index used to fill SensorPeriod buffer in order to manage motor with pole pair > 1

Definition at line 77 of file HALL_SpeednPosFdbkPrivate.h.

5.38.2.38 sint8 DVars_t::bSpeed

Instantaneous direction of rotor between two captures

Definition at line 82 of file HALL_SpeednPosFdbkPrivate.h.

5.38.2.39 sint16 DVars_t::hAvrEISpeedDpp

It is the averaged rotor electrical speed express in s16degree per current control period.

Definition at line 85 of file HALL_SpeednPosFdbkPrivate.h.

5.38.2.40 uint8 DVars_t::bHallState

Current HALL state configuration

Definition at line 88 of file HALL_SpeednPosFdbkPrivate.h.

5.38.2.41 Resolver_Ranges DVars_t::pole_pair_ranges[2U]

Two ranges for max/min values for each channel for each pole pair

Definition at line 69 of file RESOLVER_SpeednPosFdbkPrivate.h.

Referenced by Res_NewObject().

5.38.2.42 uint32 DVars_t::ResCTUFifoBuffer[2U]

Resolver CTU buffers

Definition at line 72 of file RESOLVER_SpeednPosFdbkPrivate.h.

5.38.2.43 int16_t DVars_t::hC1

Variable containing state observer constant C1 to speed-up computations

Definition at line 39 of file STO_SpeednPosFdbkPrivate.h.

5.38.2.44 int16_t DVars_t::hC2

Variable containing state observer constant C2, it can be computed as $F1 * K1 / \text{State observer execution rate [Hz]}$ being K1 one of the two observer gains

Definition at line 41 of file STO_SpeednPosFdbkPrivate.h.

Referenced by STO_GetObserverGains(), and STO_SetObserverGains().

5.38.2.45 int16_t DVars_t::hC3

Variable containing state observer constant C3

Definition at line 45 of file STO_SpeednPosFdbkPrivate.h.

5.38.2.46 int16_t DVars_t::hC4

State Observer constant C4, it can be computed as $K2 * \text{max measurable current (A)} / (\text{Max application speed [rpm]} * \text{Motor B-emf constant [Vllrms/kgpm]} * \sqrt{2} * F2 * \text{State observer execution rate [Hz]})$ being K2 one of the two observer gains

Definition at line 47 of file STO_SpeednPosFdbkPrivate.h.

Referenced by STO_GetObserverGains(), and STO_SetObserverGains().

5.38.2.47 int16_t DVars_t::hC5

Variable containing state observer constant C5

Definition at line 54 of file STO_SpeednPosFdbkPrivate.h.

5.38.2.48 int16_t DVars_t::hC6

State observer constant C6, computed with a specific procedure starting from the other constants

Definition at line 56 of file STO_SpeednPosFdbkPrivate.h.

5.38.2.49 int16_t DVars_t::hF1

Variable containing state observer scaling factor F1

Definition at line 59 of file STO_SpeednPosFdbkPrivate.h.

5.38.2.50 int16_t DVars_t::hF2

Variable containing state observer scaling factor F2

Definition at line 61 of file STO_SpeednPosFdbkPrivate.h.

5.38.2.51 int16_t DVars_t::hF3

State observer scaling factor F3

Definition at line 63 of file STO_SpeednPosFdbkPrivate.h.

5.38.2.52 uint16_t DVars_t::hF3POW2

State observer scaling factor F3 expressed as power of 2. E.g. if gain divisor is 512 the value must be 9 because $2^9 = 512$

Definition at line 64 of file STO_SpeednPosFdbkPrivate.h.

5.38.2.53 CPI DVars_t::oPIRegulator

Pointer to PI regulator object, used for PLL implementation

Definition at line 68 of file STO_SpeednPosFdbkPrivate.h.

Referenced by STO_GetPLLGains(), and STO_SetPLLGains().

5.38.2.54 int32_t DVars_t::wlalpha_est

Estimated lalpha current in int32 format

Definition at line 70 of file STO_SpeednPosFdbkPrivate.h.

Referenced by STO_GetEstimatedCurrent().

5.38.2.55 int32_t DVars_t::wlbeta_est

Estimated lbeta current in int32 format

Definition at line 71 of file STO_SpeednPosFdbkPrivate.h.

Referenced by STO_GetEstimatedCurrent().

5.38.2.56 int32_t DVars_t::wBemf_alfa_est

Estimated B-emf alfa in int32_t format

Definition at line 72 of file STO_SpeednPosFdbkPrivate.h.

5.38.2.57 int32_t DVars_t::wBemf_beta_est

Estimated B-emf beta in int32_t format

Definition at line 73 of file STO_SpeednPosFdbkPrivate.h.

5.38.2.58 int16_t DVars_t::hBemf_alfa_est

Estimated B-emf alfa in int16_t format

Definition at line 74 of file STO_SpeednPosFdbkPrivate.h.

5.38.2.59 int16_t DVars_t::hBemf_beta_est

Estimated B-emf beta in int16_t format

Definition at line 75 of file STO_SpeednPosFdbkPrivate.h.

5.38.2.60 int16_t DVars_t::hSpeed_Buffer[64]

Estimated speed FIFO, it contains latest bSpeedBufferSize speed measurements

Definition at line 76 of file STO_SpeednPosFdbkPrivate.h.

Referenced by STO_CalcAvrgElSpeedDpp().

5.38.2.61 uint8_t DVars_t::bSpeed_Buffer_Index

Position of latest speed estimation in estimated speed FIFO

Definition at line 78 of file STO_SpeednPosFdbkPrivate.h.

Referenced by STO_CalcAvrgElSpeedDpp().

5.38.2.62 bool DVars_t::bIsSpeedReliable

Latest private speed reliability information, updated by SPD_CalcAvrgMecSpeed01Hz, it is TRUE if the speed measurement variance is lower than threshold corresponding to hVariancePercentage

Definition at line 80 of file STO_SpeednPosFdbkPrivate.h.

Referenced by STO_IsObserverConverged().

5.38.2.63 uint8_t DVars_t::bConsistencyCounter

Counter of passed tests for start-up validation

Definition at line 85 of file STO_SpeednPosFdbkPrivate.h.

Referenced by STO_IsObserverConverged().

5.38.2.64 uint8_t DVars_t::bReliabilityCounter

Counter for reliability check internal to derived class

Definition at line 87 of file STO_SpeednPosFdbkPrivate.h.

5.38.2.65 bool DVars_t::bIsAlgorithmConverged

Boolean variable containing observer convergence information

Definition at line 89 of file STO_SpeednPosFdbkPrivate.h.

Referenced by STO_IsObserverConverged().

5.38.2.66 bool DVars_t::bIsBemfConsistent

Sensor reliability information, updated by SPD_CalcAvrgMecSpeed01Hz, it is TRUE if the observed back-emfs are consistent with expectation

Definition at line 91 of file STO_SpeednPosFdbkPrivate.h.

5.38.2.67 int32_t DVars_t::wObs_Bemf_Level

Observed back-emf Level

Definition at line 95 of file STO_SpeednPosFdbkPrivate.h.

5.38.2.68 int32_t DVars_t::wEst_Bemf_Level

Estimated back-emf Level

Definition at line 96 of file STO_SpeednPosFdbkPrivate.h.

5.38.2.69 bool DVars_t::bEnableDualCheck

Consistency check enabler

Definition at line 97 of file STO_SpeednPosFdbkPrivate.h.

5.38.2.70 int32_t DVars_t::wDppBufferSum

summation of speed buffer elements [dpp]

Definition at line 98 of file STO_SpeednPosFdbkPrivate.h.

Referenced by STO_CalcAvrgElSpeedDpp().

5.38.2.71 int16_t DVars_t::hSpeedBufferOldestEl

Oldest element of the speed buffer

Definition at line 99 of file STO_SpeednPosFdbkPrivate.h.

Referenced by STO_CalcAvrgElSpeedDpp().

5.38.2.72 int32_t DVars_t::wElAccDppP32

Delta electrical speed expressed in dpp per speed sampling period to be applied each time is called SPD_CalcAvrg-MecSpeed01Hz multiplied by scaling factor of 65536.

Definition at line 39 of file VirtualSpeedSensor_SpeednPosFdbkPrivate.h.

Referenced by VSPD_SetMecAcceleration().

5.38.2.73 int32_t DVars_t::wElSpeedDpp32

Electrical speed expressed in dpp multiplied by scaling factor 65536.

Definition at line 43 of file VirtualSpeedSensor_SpeednPosFdbkPrivate.h.

Referenced by VSPD_SetMecAcceleration().

5.38.2.74 uint16_t DVars_t::hRemainingStep

Number of steps remaining to reach the final speed.

Definition at line 45 of file VirtualSpeedSensor_SpeednPosFdbkPrivate.h.

Referenced by VSPD_SetMecAcceleration().

5.38.2.75 int16_t DVars_t::hFinalMecSpeed01Hz

Backup of hFinalMecSpeed01Hz to be applied in the last step.

Definition at line 47 of file VirtualSpeedSensor_SpeednPosFdbkPrivate.h.

Referenced by VSPD_GetLastRampFinalSpeed(), and VSPD_SetMecAcceleration().

5.38.2.76 bool DVars_t::bTransitionStarted

Retaining information about Transition status.

Definition at line 49 of file VirtualSpeedSensor_SpeednPosFdbkPrivate.h.

Referenced by VSPD_SetMecAcceleration().

5.38.2.77 bool DVars_t::bTransitionEnded

Retaining information about transition status.

Definition at line 50 of file VirtualSpeedSensor_SpeednPosFdbkPrivate.h.

5.38.2.78 int16_t DVars_t::hTransitionRemainingSteps

Number of steps remaining to end transition from CVSS_SPD to other CSPD

Definition at line 51 of file VirtualSpeedSensor_SpeednPosFdbkPrivate.h.

5.38.2.79 int16_t DVars_t::hElAngleAccu

Electrical angle accumulator

Definition at line 53 of file VirtualSpeedSensor_SpeednPosFdbkPrivate.h.

5.38.2.80 bool DVars_t::bTransitionLocked

Transition acceleration started

Definition at line 54 of file VirtualSpeedSensor_SpeednPosFdbkPrivate.h.

5.38.2.81 bool DVars_t::IsOverTempFaultActive

CPWMC object to be used for regular conversions

Definition at line 53 of file NTC_TemperatureSensorPrivate.h.

5.38.2.82 MC_Protocol_REG_t DVars_t::bChannel_variable[DAC_CH_NBR]

Array of codes of variables to be provided in out to the related channel.

Definition at line 52 of file DACRCTIMER_UserInterfacePrivate.h.

5.38.2.83 int16_t DVars_t::hUserValue[DAC_CH_USER]

Array of user defined DAC values.

Definition at line 56 of file DACRCTIMER_UserInterfacePrivate.h.

5.38.2.84 bool DVars_t::comON

True to establish the communication false to stop it

Definition at line 52 of file UnidirectionalFastCom_UserInterfacePrivate.h.

Referenced by UFC_Init(), UFC_StartCom(), and UFC_StopCom().

5.38.2.85 MC_Protocol_REG_t DVars_t::bChannel[2]

Codes of variables to be sent.

Definition at line 53 of file UnidirectionalFastCom_UserInterfacePrivate.h.

Referenced by UFC_Init().

5.38.2.86 uint8_t DVars_t::bChTransmitted

Current channel to be transmitted.

Definition at line 54 of file UnidirectionalFastCom_UserInterfacePrivate.h.

Referenced by UFC_Init().

5.38.2.87 uint8_t DVars_t::bByteTransmitted

Current byte to be transmitted.

Definition at line 55 of file UnidirectionalFastCom_UserInterfacePrivate.h.

Referenced by UFC_Init().

5.38.2.88 int32_t DVars_t::wBuffer

Transmission buffer 4 bytes.

Definition at line 56 of file UnidirectionalFastCom_UserInterfacePrivate.h.

5.38.2.89 uint8_t DVars_t::bChByteNum[2]

Number of bytes transmitted.

Definition at line 57 of file UnidirectionalFastCom_UserInterfacePrivate.h.

Referenced by UFC_Init().

5.38.2.90 uint8_t DVars_t::bChNum

Number of channel to be transmitted.

Definition at line 58 of file UnidirectionalFastCom_UserInterfacePrivate.h.

Referenced by UFC_Init().

5.39 EncAlignCtrlParams_t Struct Reference

5.39.1 Detailed Description

EncAlignCtrl class parameters definition.

Definition at line 60 of file EncAlignCtrlClass.h.

```
#include <EncAlignCtrlClass.h>
```

Data Fields

- uint16_t hEACFrequencyHz
- int16_t hFinalTorque
- int16_t hEIAngle
- uint16_t hDurationms
- uint8_t bEIToMecRatio

5.39.2 Field Documentation

5.39.2.1 uint16_t EncAlignCtrlParams_t::hEACFrequencyHz

Frequency expressed in Hz at which the user clocks the EAC calling EAC_Exec method

Definition at line 62 of file EncAlignCtrlClass.h.

5.39.2.2 int16_t EncAlignCtrlParams_t::hFinalTorque

Motor torque reference imposed by STC at the end of programmed alignment. This value represents actually the Iq current expressed in digit.

Definition at line 64 of file EncAlignCtrlClass.h.

5.39.2.3 int16_t EncAlignCtrlParams_t::hElAngle

Electrical angle of programmed alignment expressed in s16degrees.

Definition at line 68 of file EncAlignCtrlClass.h.

5.39.2.4 uint16_t EncAlignCtrlParams_t::hDurationms

Duration of the programmed alignment expressed in milliseconds.

Definition at line 70 of file EncAlignCtrlClass.h.

5.39.2.5 uint8_t EncAlignCtrlParams_t::bElToMecRatio

Coefficient used to transform electrical to mechanical quantities and viceversa. It usually coincides with motor pole pairs number

Definition at line 72 of file EncAlignCtrlClass.h.

5.40 ENCODERParams_t Struct Reference

5.40.1 Detailed Description

ENCODER class parameters definition.

Definition at line 68 of file ENCODER_SpeednPosFdbkClass.h.

```
#include <ENCODER_SpeednPosFdbkClass.h>
```

Data Fields

- uint8_t **IRQnb**
MC IRQ number used for eTimer capture index signal event.
- uint16_t **hPulseNumber**
Number of pulses per revolution, provided by each of the two encoder signals, multiplied by 4.
- FunctionalState **RevertSignal**
To be enabled if measured speed is opposite to real one (ENABLE/DISABLE)
- uint16_t **hSpeedSamplingFreq01Hz**
Frequency (01Hz) at which motor speed is to be computed. It must be equal to the frequency at which function SPD_CalcAvrgMecSpeed01Hz is called.
- uint8_t **bSpeedBufferSize**
Size of the buffer used to calculate the average speed. It must be <= 16.
- uint16_t **hInpCaptFilter_chA_B**
Input filter applied to ENCODER sensor capture channels (for channel A and channel B). This value configure the Input Filter register (FILT) of eTimer channel.
- uint16_t **hInpCaptFilter_index_ch**
Input filter applied to index capture channel. This value configure the Input Filter register (FILT) of eTimer channel.
- uint8_t **eTimerModule**
Timer used for ENCODER sensor management.
- uint8_t **quad_eTimer_ch**
eTimer channel configure in quadrature mode
- uint8_t **CH_A_eTimer_ch**
ENCODER CH. A - INPUT eTimer channel.
- uint8_t **CH_B_eTimer_ch**

ENCODER CH. B - INPUT eTimer channel.

- bool `index_enable`

TRUE: The index signal is used. FALSE: The index signal is not used.
- uint8_t `position_eTimer_ch`

eTimer channel used for position acquisition
- uint8_t `index_counter_eTimer_ch`

eTimer channel used for index counter
- uint8_t `index_input_capture_eTimer_ch`

input capture eTimer channel used for reset signal from encoder
- uint8_t `Overflow_eTimer_ch`

eTimer channel used for manage overflow (for speed calculation)
- bool `position_acq_pwm`

TRUE: The rotor electrical and mechanical angle are synchronize with position acquisition with PWM period (CTU trigger) FALSE: It calculates the rotor electrical and mechanical angle, on the basis of the instantaneous value of the timer counter.

5.40.2 Field Documentation

5.40.2.1 uint8_t `ENCODERParams_t::IRQnb`

MC IRQ number used for eTimer capture index signal event.

Definition at line 72 of file `ENCODER_SpeednPosFdbkClass.h`.

Referenced by `Enc_NewObject()`.

5.40.2.2 uint16_t `ENCODERParams_t::hPulseNumber`

Number of pulses per revolution, provided by each of the two encoder signals, multiplied by 4.

Definition at line 75 of file `ENCODER_SpeednPosFdbkClass.h`.

5.40.2.3 FunctionalState `ENCODERParams_t::RevertSignal`

To be enabled if measured speed is opposite to real one (ENABLE/DISABLE)

Definition at line 77 of file `ENCODER_SpeednPosFdbkClass.h`.

5.40.2.4 uint16_t `ENCODERParams_t::hSpeedSamplingFreq01Hz`

Frequency (01Hz) at which motor speed is to be computed. It must be equal to the frequency at which function `SPD_CalcAvgMecSpeed01Hz` is called.

Definition at line 80 of file `ENCODER_SpeednPosFdbkClass.h`.

5.40.2.5 uint8_t `ENCODERParams_t::bSpeedBufferSize`

Size of the buffer used to calculate the average speed. It must be ≤ 16 .

Definition at line 82 of file `ENCODER_SpeednPosFdbkClass.h`.

5.40.2.6 uint16_t ENCODERParams_t::hInpCaptFilter_chA_B

Input filter applied to ENCODER sensor capture channels (for channel A and channel B). This value configure the Input Filter register (FILT) of eTimer channel.

Definition at line 85 of file ENCODER_SpeednPosFdbkClass.h.

5.40.2.7 uint16_t ENCODERParams_t::hInpCaptFilter_index_ch

Input filter applied to index capture channel. This value configure the Input Filter register (FILT) of eTimer channel.

Definition at line 88 of file ENCODER_SpeednPosFdbkClass.h.

5.40.2.8 uint8_t ENCODERParams_t::eTimerModule

Timer used for ENCODER sensor management.

Definition at line 91 of file ENCODER_SpeednPosFdbkClass.h.

5.40.2.9 uint8_t ENCODERParams_t::quad_eTimer_ch

eTimer channel configure in quadrature mode

Definition at line 93 of file ENCODER_SpeednPosFdbkClass.h.

5.40.2.10 uint8_t ENCODERParams_t::CH_A_eTimer_ch

ENCODER CH. A - INPUT eTimer channel.

Definition at line 95 of file ENCODER_SpeednPosFdbkClass.h.

5.40.2.11 uint8_t ENCODERParams_t::CH_B_eTimer_ch

ENCODER CH. B - INPUT eTimer channel.

Definition at line 97 of file ENCODER_SpeednPosFdbkClass.h.

5.40.2.12 bool ENCODERParams_t::index_enable

TRUE: The index signal is used. FALSE: The index signal is not used.

Definition at line 100 of file ENCODER_SpeednPosFdbkClass.h.

5.40.2.13 uint8_t ENCODERParams_t::position_eTimer_ch

eTimer channel used for position acquisition

Definition at line 102 of file ENCODER_SpeednPosFdbkClass.h.

5.40.2.14 uint8_t ENCODERParams_t::index_counter_eTimer_ch

eTimer channel used for index counter

Definition at line 104 of file ENCODER_SpeednPosFdbkClass.h.

5.40.2.15 uint8_t ENCODERParams_t::index_input_capture_eTimer_ch

input capture eTimer channel used for reset signal from encoder

Definition at line 106 of file ENCODER_SpeednPosFdbkClass.h.

5.40.2.16 uint8_t ENCODERParams_t::Overflow_eTimer_ch

eTimer channel used for manage overflow (for speed calculation)

Definition at line 108 of file ENCODER_SpeednPosFdbkClass.h.

5.40.2.17 bool ENCODERParams_t::position_acq_pwm

TRUE: The rotor electrical and mechanical angle are synchronize with position acquisition with PWM period (CTU trigger)
 FALSE: It calculates the rotor electrical and mechanical angle, on the basis of the instantaneous value of the timer counter.

Definition at line 113 of file ENCODER_SpeednPosFdbkClass.h.

5.41 FeedForwardCtrlParams_t Struct Reference

5.41.1 Detailed Description

FeedForwardCtrl class parameters definition.

Definition at line 69 of file FeedForwardCtrlClass.h.

```
#include <FeedForwardCtrlClass.h>
```

Data Fields

- uint16_t hVqdLowPassFilterBW
- int32_t wDefConstant_1D
- int32_t wDefConstant_1Q
- int32_t wDefConstant_2
- uint16_t hVqdLowPassFilterBWLOG

5.41.2 Field Documentation

5.41.2.1 uint16_t FeedForwardCtrlParams_t::hVqdLowPassFilterBW

Use this parameter to configure the Vqd first order software filter bandwidth. hVqdLowPassFilterBW = FOC_Curr-Controller call rate [Hz]/ FilterBandwidth[Hz] in case FULL_MISRA_COMPLIANCY is defined. On the contrary, if FULL_MISRA_COMPLIANCY is not defined, hVqdLowPassFilterBW is equal to log with base two of previous definition

Definition at line 71 of file FeedForwardCtrlClass.h.

5.41.2.2 int32_t FeedForwardCtrlParams_t::wDefConstant_1D

Feed forward default constant for d axes

Definition at line 80 of file FeedForwardCtrlClass.h.

5.41.2.3 int32_t FeedForwardCtrlParams_t::wDefConstant_1Q

Feed forward default constant for q axes

Definition at line 81 of file FeedForwardCtrlClass.h.

5.41.2.4 int32_t FeedForwardCtrlParams_t::wDefConstant_2

Default constant value used by Feed-Forward algorithm

Definition at line 82 of file FeedForwardCtrlClass.h.

5.41.2.5 uint16_t FeedForwardCtrlParams_t::hVqdLowPassFilterBWLOG

hVqdLowPassFilterBW expressed as power of 2. E.g. if gain divisor is 512 the value must be 9 because $2^9 = 512$

Definition at line 84 of file FeedForwardCtrlClass.h.

5.42 FFInit_t Struct Reference

5.42.1 Detailed Description

FeedForwardCtrl class init structure type definition.

Definition at line 54 of file FeedForwardCtrlClass.h.

```
#include <FeedForwardCtrlClass.h>
```

5.43 FluxWeakeningCtrlParams_t Struct Reference

5.43.1 Detailed Description

FluxWeakeningCtrl class parameters definition.

Definition at line 67 of file FluxWeakeningCtrlClass.h.

```
#include <FluxWeakeningCtrlClass.h>
```

Data Fields

- uint16_t hMaxModule
- uint16_t hDefaultFW_V_Ref
- int16_t hDemagCurrent
- int32_t wNominalSqCurr
- uint16_t hVqdLowPassFilterBW
- uint16_t hVqdLowPassFilterBWLOG

5.43.2 Field Documentation

5.43.2.1 uint16_t FluxWeakeningCtrlParams_t::hMaxModule

Circle limitation maximum allowed module

Definition at line 69 of file FluxWeakeningCtrlClass.h.

5.43.2.2 uint16_t FluxWeakeningCtrlParams_t::hDefaultFW_V_Ref

Default flux weakening voltage reference, tenth of percentage points

Definition at line 71 of file FluxWeakeningCtrlClass.h.

5.43.2.3 int16_t FluxWeakeningCtrlParams_t::hDemagCurrent

Demagnetization current in s16A: Current(Amp) = [Current(s16A) * Vdd micro]/ [65536 * Rshunt * Aop]

Definition at line 73 of file FluxWeakeningCtrlClass.h.

5.43.2.4 int32_t FluxWeakeningCtrlParams_t::wNominalSqCurr

Squared motor nominal current in (s16A)² where: Current(Amp) = [Current(s16A) * Vdd micro]/ [65536 * Rshunt * Aop]

Definition at line 76 of file FluxWeakeningCtrlClass.h.

5.43.2.5 uint16_t FluxWeakeningCtrlParams_t::hVqdLowPassFilterBW

Use this parameter to configure the Vqd first order software filter bandwidth. hVqdLowPassFilterBW = FOC_Curr-Controller call rate [Hz]/ FilterBandwidth[Hz] in case FULL_MISRA_COMPLIANCY is defined. On the contrary, if FULL_MISRA_COMPLIANCY is not defined, hVqdLowPassFilterBW is equal to log with base two of previous definition

Definition at line 80 of file FluxWeakeningCtrlClass.h.

5.43.2.6 uint16_t FluxWeakeningCtrlParams_t::hVqdLowPassFilterBWLOG

hVqdLowPassFilterBW expressed as power of 2. E.g. if gain divisor is 512 the value must be 9 because $2^9 = 512$

Definition at line 89 of file FluxWeakeningCtrlClass.h.

5.44 FWInit_t Struct Reference

5.44.1 Detailed Description

FluxWeakeningCtrl class init structure type definition.

Definition at line 53 of file FluxWeakeningCtrlClass.h.

```
#include <FluxWeakeningCtrlClass.h>
```

5.45 HALLParams_t Struct Reference

5.45.1 Detailed Description

HALL class parameters definition.

Definition at line 81 of file HALL_SpeednPosFdbkClass.h.

```
#include <HALL_SpeednPosFdbkClass.h>
```

Data Fields

- **uint8_t IRQnb**
MC IRQ number used for eTimer capture events.
- **uint8_t bSensorPlacement**
here the mechanical position of the sensors with reference to an electrical cycle. Allowed values are: DEGREES_120 or DEGREES_60. N.B. DEGREES_60 not implemented.
- **int16_t hPhaseShift**
Define here in s16degree the electrical phase shift between the low to high transition of signal H1 and the maximum of the Bemf induced on phase A.
- **uint16_t hSpeedSamplingFreqHz**
Frequency (Hz) at which motor speed is to be computed. It must be equal to the frequency at which function SPD_CalcAvrgMecSpeed01Hz is called.
- **uint16_t hSpeedThreshold**
Mechanical speed (01Hz) threshold used to select speed measurement among two consecutive edges of the different or the same hall sensor signals.
- **uint32_t hPwmFrequency**
PWM clock frequency used for Hall sensor class.
- **uint32_t hFocUpdate**
PWM Foc Update.
- **uint16_t hInpCaptFilter_ch1_2_3**
Input filter applied to HALL sensor capture channels (for channel 1 and channel 2 and Channel 3). This value configure the Input Filter register (FILT) of eTimer channel.
- **uint8_t eTimerModule**
Timer used for HALL sensor management.
- **uint8_t S1_eTimer_ch**
HALL SENSOR 1 - INPUT 1 eTimer channel.
- **uint8_t S2_eTimer_ch**
HALL SENSOR 2 - INPUT 2 eTimer channel.
- **uint8_t S3_eTimer_ch**
HALL SENSOR 3 - INPUT 3 eTimer channel.
- **uint8_t ctu_trig_eTimer_ch**
eTimer channel CTU trigger

5.45.2 Field Documentation

5.45.2.1 uint8_t HALLParams_t::IRQnb

MC IRQ number used for eTimer capture events.

Definition at line 85 of file HALL_SpeednPosFdbkClass.h.

Referenced by HALL_NewObject().

5.45.2.2 uint8_t HALLParams_t::bSensorPlacement

here the mechanical position of the sensors with reference to an electrical cycle. Allowed values are: DEGREES_120 or DEGREES_60. N.B. DEGREES_60 not implemented.

Definition at line 89 of file HALL_SpeednPosFdbkClass.h.

5.45.2.3 int16_t HALLParams_t::hPhaseShift

Define here in s16degree the electrical phase shift between the low to high transition of signal H1 and the maximum of the Bemf induced on phase A.

Definition at line 93 of file HALL_SpeednPosFdbkClass.h.

5.45.2.4 uint16_t HALLParams_t::hSpeedSamplingFreqHz

Frequency (Hz) at which motor speed is to be computed. It must be equal to the frequency at which function SPD_CalcAvrgMecSpeed01Hz is called.

Definition at line 97 of file HALL_SpeednPosFdbkClass.h.

5.45.2.5 uint16_t HALLParams_t::hSpeedThreshold

Mechanical speed (01Hz) threshold used to select speed measurement among two consecutive edges of the different or the same hall sensor signals.

Definition at line 100 of file HALL_SpeednPosFdbkClass.h.

5.45.2.6 uint32_t HALLParams_t::hPwmFrequency

PWM clock frequency used for Hall sensor class.

Definition at line 103 of file HALL_SpeednPosFdbkClass.h.

5.45.2.7 uint32_t HALLParams_t::hFocUpdate

PWM Foc Update.

Definition at line 105 of file HALL_SpeednPosFdbkClass.h.

5.45.2.8 uint16_t HALLParams_t::hInpCaptFilter_ch1_2_3

Input filter applied to HALL sensor capture channels (for channel 1 and channel 2 and Channel 3). This value configure the Input Filter register (FILT) of eTimer channel.

Definition at line 109 of file HALL_SpeednPosFdbkClass.h.

5.45.2.9 uint8_t HALLParams_t::eTimerModule

Timer used for HALL sensor management.

Definition at line 111 of file HALL_SpeednPosFdbkClass.h.

5.45.2.10 uint8_t HALLParams_t::S1_eTimer_ch

HALL SENSOR 1 - INPUT 1 eTimer channel.

Definition at line 113 of file HALL_SpeednPosFdbkClass.h.

5.45.2.11 uint8_t HALLParams_t::S2_eTimer_ch

HALL SENSOR 2 - INPUT 2 eTimer channel.

Definition at line 115 of file HALL_SpeednPosFdbkClass.h.

5.45.2.12 uint8_t HALLParams_t::S3_eTimer_ch

HALL SENSOR 3 - INPUT 3 eTimer channel.

Definition at line 117 of file HALL_SpeednPosFdbkClass.h.

5.45.2.13 uint8_t HALLParams_t::ctu_trig_eTimer_ch

eTimer channel CTU trigger

Definition at line 119 of file HALL_SpeednPosFdbkClass.h.

5.46 ICS_Params_t Struct Reference

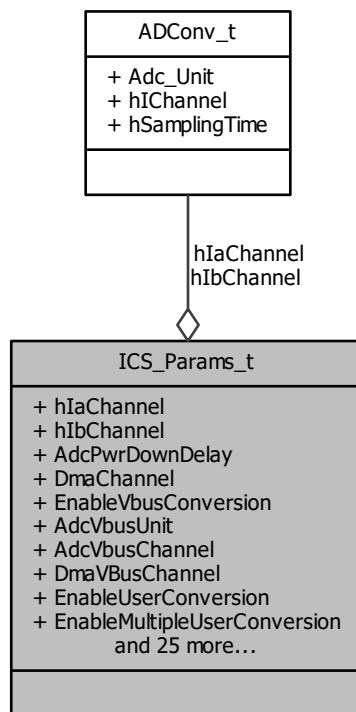
5.46.1 Detailed Description

ICS class parameters definition.

Definition at line 95 of file ICS_PWMnCurrFdbkClass.h.

```
#include <ICS_PWMnCurrFdbkClass.h>
```

Collaboration diagram for ICS_Params_t:



Data Fields

- [ADConv_t hlaChannel](#)

- **ADCConv_t hlbChannel**
ADC channel configuration used for conversion of current Ia.
- **uint8_t AdcPwrDownDelay**
Power Down Exit Delay for ADC Units.
- **uint8_t DmaChannel**
Dma channel for ADC conversion.
- **bool EnableVbusConversion**
VBus conversion enable/disable flag.
- **uint8_t AdcVbusUnit**
Adc unit for VBus conversion.
- **uint8_t AdcVbusChannel**
Adc channel for VBus conversion.
- **uint8_t DmaVBusChannel**
Dma channel for VBus conversion.
- **bool EnableUserConversion**
User conversion enable/disable flag.
- **bool EnableMultipleUserConversion**
Multiple User conversion enable/disable flag.
- **uint8_t AdcUserUnit**
Adc unit for User conversion.
- **uint8_t AdcUserChannel**
Adc channel for User conversion.
- **uint8_t DmaUserChannel**
Dma channel for User conversion.
- **uint8_t IRQnb**
MC IRQ number used for ADC User conversion.
- **uint8_t FlexPWM_Clock_Divider**
PWM clock frequency divide ratio: PWM clock frequency divide ratio: FLEXPWM_CTRL_PRSC_PRESC_1, FLEXPWM_CTRL_PRSC_PRESC_2, FLEXPWM_CTRL_PRSC_PRESC_4, FLEXPWM_CTRL_PRSC_PRESC_8, FLEXPWM_CTRL_PRSC_PRESC_16 FLEXPWM_CTRL_PRSC_PRESC_32 FLEXPWM_CTRL_PRSC_PRESC_64 FLEXPWM_CTRL_PRSC_PRESC_128.
- **uint8_t FlexPWMMModule**
FlexPWM used for PWM generation.
- **uint16_t hDeadTime**
Dead time in number of IPBUS clock cycles regardless of the setting of FlexPWM_Clock_Divider.
- **uint16_t PWMLoadFreq**
PWM load frequency: FLEXPWM_CTRL_LDFQ_EACH1- FLEXPWM_CTRL_LDFQ_EACH16.
- **uint16_t hCh1Polarity**
Channel 1 (high side) output polarity,it must be FLEXPWM_Polarity_High or FLEXPWM_Polarity_Low.
- **uint16_t hCh1IdleState**
Channel 1 (high side) state (low or high) when FlexPWM is in Idle state. It must be FLEXPWM_IDLE_STATE_HIGH or FLEXPWM_IDLE_STATE_LOW.
- **uint16_t hCh2Polarity**
Channel 2 (high side) output polarity,it must be FLEXPWM_Polarity_High or FLEXPWM_Polarity_Low.
- **uint16_t hCh2IdleState**
Channel 2 (high side) state (low or high) when FlexPWM is in Idle state. It must be FLEXPWM_IDLE_STATE_HIGH or FLEXPWM_IDLE_STATE_LOW.
- **uint16_t hCh3Polarity**
Channel 3 (high side) output polarity,it must be FLEXPWM_Polarity_High or FLEXPWM_Polarity_Low.
- **uint16_t hCh3IdleState**

Channel 3 (high side) state (low or high) when FlexPWM is in Idle state. It must be FLEXPWM_IDLE_STATE_HIGH or FLEXPWM_IDLE_STATE_LOW.

- [LowSideOutputsFunction_t LowSideOutputs](#)

Low side or enabling signals generation method definition.

- [ICS_ShuntPosition_t ShuntPosition](#)

Shunt Position on PHASES or LEGS (ICS_SHUNT_PHASES/ICS_SHUNT_LEGS)

- [ICS_SensingSelection_t SensingSelection](#)

Sensing Selection (ICS_SENSING_ON_UV_PHASES/ICS_SENSING_ON_UW_PHASES/ICS_SENSING_ON_VW_PHASES)

- [uint16_t hCh1NPolarity](#)

Channel 1N (low side) output polarity,it must be FLEXPWM_NPolarity_High or FLEXPWM_NPolarity_Low.

- [uint16_t hCh1NIdleState](#)

Channel 1N (low side) state (low or high) when FlexPWM is in Idle state. It must be FLEXPWM_IDLE_STATE_HIGH or FLEXPWM_IDLE_STATE_LOW.

- [uint16_t hCh2NPolarity](#)

Channel 2N (low side) output polarity,it must be FLEXPWM_NPolarity_High or FLEXPWM_NPolarity_Low.

- [uint16_t hCh2NIdleState](#)

Channel 2N (low side) state (low or high) when FlexPWM is in Idle state. It must be FLEXPWM_IDLE_STATE_HIGH or FLEXPWM_IDLE_STATE_LOW.

- [uint16_t hCh3NPolarity](#)

Channel 3N (low side) output polarity,it must be FLEXPWM_NPolarity_High or FLEXPWM_NPolarity_Low.

- [uint16_t hCh3NIdleState](#)

Channel 3N (low side) state (low or high) when FlexPWM is in Idle state. It must be FLEXPWM_IDLE_STATE_HIGH or FLEXPWM_IDLE_STATE_LOW.

- [FunctionalState EmergencyStop](#)

It enable/disable the management of an emergency input instantaneously stopping PWM generation. It must be either equal to ENABLE or DISABLE.

- [uint8_t FaultPIN](#)

Emergency Stop PIN. It must be: FAULT_PIN0-FAULT_PIN3.

- [uint16_t FaultPolarity](#)

Emergency Stop Input polarity, it must be: FAULT_POLARITY_HIGH or FAULT_POLARITY_LOW.

5.46.2 Field Documentation

5.46.2.1 ADCConv_t ICS_Params_t::hlaChannel

ADC channel configuration used for conversion of current Ia.

Definition at line 99 of file ICS_PWMnCurrFdbkClass.h.

5.46.2.2 ADCConv_t ICS_Params_t::hlbChannel

ADC channel configuration used for conversion of current Ib.

Definition at line 101 of file ICS_PWMnCurrFdbkClass.h.

5.46.2.3 uint8_t ICS_Params_t::AdcPwrDownDelay

Power Down Exit Delay for ADC Units.

Definition at line 103 of file ICS_PWMnCurrFdbkClass.h.

5.46.2.4 uint8_t ICS_Params_t::DmaChannel

Dma channel for ADC conversion.

Definition at line 105 of file ICS_PWMnCurrFdbkClass.h.

5.46.2.5 bool ICS_Params_t::EnableVbusConversion

VBus conversion enable/disable flag.

Definition at line 108 of file ICS_PWMnCurrFdbkClass.h.

5.46.2.6 uint8_t ICS_Params_t::AdcVbusUnit

Adc unit for VBus conversion.

Definition at line 110 of file ICS_PWMnCurrFdbkClass.h.

5.46.2.7 uint8_t ICS_Params_t::AdcVbusChannel

Adc channel for VBus conversion.

Definition at line 112 of file ICS_PWMnCurrFdbkClass.h.

5.46.2.8 uint8_t ICS_Params_t::DmaVBusChannel

Dma channel for VBus conversion.

Definition at line 114 of file ICS_PWMnCurrFdbkClass.h.

5.46.2.9 bool ICS_Params_t::EnableUserConversion

User conversion enable/disable flag.

Definition at line 117 of file ICS_PWMnCurrFdbkClass.h.

5.46.2.10 bool ICS_Params_t::EnableMultipleUserConversion

Multiple User conversion enable/disable flag.

Definition at line 119 of file ICS_PWMnCurrFdbkClass.h.

5.46.2.11 uint8_t ICS_Params_t::AdcUserUnit

Adc unit for User conversion.

Definition at line 121 of file ICS_PWMnCurrFdbkClass.h.

5.46.2.12 uint8_t ICS_Params_t::AdcUserChannel

Adc channel for User conversion.

Definition at line 123 of file ICS_PWMnCurrFdbkClass.h.

5.46.2.13 uint8_t ICS_Params_t::DmaUserChannel

Dma channel for User conversion.

Definition at line 125 of file ICS_PWMnCurrFdbkClass.h.

5.46.2.14 uint8_t ICS_Params_t::IRQnb

MC IRQ number used for ADC User conversion.

Definition at line 127 of file ICS_PWMnCurrFdbkClass.h.

Referenced by ICS_NewObject().

5.46.2.15 uint8_t ICS_Params_t::FlexPWM_Clock_Divider

PWM clock frequency divide ratio: PWM clock frequency divide ratio: FLEXPWM_CTRL_PRSC_PRESC_1, FLEXPWM_CTRL_PRSC_PRESC_2, FLEXPWM_CTRL_PRSC_PRESC_4, FLEXPWM_CTRL_PRSC_PRESC_8, FLEXPWM_CTRL_PRSC_PRESC_16 FLEXPWM_CTRL_PRSC_PRESC_32 FLEXPWM_CTRL_PRSC_PRESC_64 FLEXPWM_CTRL_PRSC_PRESC_128.

Definition at line 138 of file ICS_PWMnCurrFdbkClass.h.

5.46.2.16 uint8_t ICS_Params_t::FlexPWMMModule

FlexPWM used for PWM generation.

Definition at line 140 of file ICS_PWMnCurrFdbkClass.h.

5.46.2.17 uint16_t ICS_Params_t::hDeadTime

Dead time in number of IPBUS clock cycles regardless of the setting of FlexPWM_Clock_Divider.

Definition at line 143 of file ICS_PWMnCurrFdbkClass.h.

5.46.2.18 uint16_t ICS_Params_t::PWMLoadFreq

PWM load frequency: FLEXPWM_CTRL_LDFQ_EACH1- FLEXPWM_CTRL_LDFQ_EACH16.

Definition at line 145 of file ICS_PWMnCurrFdbkClass.h.

5.46.2.19 uint16_t ICS_Params_t::hCh1Polarity

Channel 1 (high side) output polarity,it must be FLEXPWM_Polarity_High or FLEXPWM_Polarity_Low.

Definition at line 149 of file ICS_PWMnCurrFdbkClass.h.

5.46.2.20 uint16_t ICS_Params_t::hCh1IdleState

Channel 1 (high side) state (low or high) when FlexPWM is in Idle state. It must be FLEXPWM_IDLE_STATE_HIGH or FLEXPWM_IDLE_STATE_LOW.

Definition at line 152 of file ICS_PWMnCurrFdbkClass.h.

5.46.2.21 uint16_t ICS_Params_t::hCh2Polarity

Channel 2 (high side) output polarity,it must be FLEXPWM_Polarity_High or FLEXPWM_Polarity_Low.

Definition at line 155 of file ICS_PWMnCurrFdbkClass.h.

5.46.2.22 uint16_t ICS_Params_t::hCh2IdleState

Channel 2 (high side) state (low or high) when FlexPWM is in Idle state. It must be FLEXPWM_IDLE_STATE_HIGH or FLEXPWM_IDLE_STATE_LOW.

Definition at line 158 of file ICS_PWMnCurrFdbkClass.h.

5.46.2.23 uint16_t ICS_Params_t::hCh3Polarity

Channel 3 (high side) output polarity,it must be FLEXPWM_Polarity_High or FLEXPWM_Polarity_Low.

Definition at line 161 of file ICS_PWMnCurrFdbkClass.h.

5.46.2.24 uint16_t ICS_Params_t::hCh3IdleState

Channel 3 (high side) state (low or high) when FlexPWM is in Idle state. It must be FLEXPWM_IDLE_STATE_HIGH or FLEXPWM_IDLE_STATE_LOW.

Definition at line 164 of file ICS_PWMnCurrFdbkClass.h.

5.46.2.25 LowSideOutputsFunction_t ICS_Params_t::LowSideOutputs

Low side or enabling signals generation method definition.

Definition at line 166 of file ICS_PWMnCurrFdbkClass.h.

5.46.2.26 ICS_ShuntPosition_t ICS_Params_t::ShuntPosition

Shunt Position on PHASES or LEGS (ICS_SHUNT_PHASES/ICS_SHUNT_LEGS)

Definition at line 168 of file ICS_PWMnCurrFdbkClass.h.

5.46.2.27 ICS_SensingSelection_t ICS_Params_t::SensingSelection

Sensing Selection (ICS_SENSING_ON_UV_PHASES/ICS_SENSING_ON_UW_PHASES/ICS_SENSING_ON_VW_PHASES)

Definition at line 170 of file ICS_PWMnCurrFdbkClass.h.

5.46.2.28 uint16_t ICS_Params_t::hCh1NPolarity

Channel 1N (low side) output polarity,it must be FLEXPWM_NPolarity_High or FLEXPWM_NPolarity_Low.

Definition at line 173 of file ICS_PWMnCurrFdbkClass.h.

5.46.2.29 uint16_t ICS_Params_t::hCh1NIdleState

Channel 1N (low side) state (low or high) when FlexPWM is in Idle state. It must be FLEXPWM_IDLE_STATE_HIGH or FLEXPWM_IDLE_STATE_LOW.

Definition at line 176 of file ICS_PWMnCurrFdbkClass.h.

5.46.2.30 uint16_t ICS_Params_t::hCh2NPolarity

Channel 2N (low side) output polarity,it must be FLEXPWM_NPolarity_High or FLEXPWM_NPolarity_Low.

Definition at line 179 of file ICS_PWMnCurrFdbkClass.h.

5.46.2.31 uint16_t ICS_Params_t::hCh2NIdleState

Channel 2N (low side) state (low or high) when FlexPWM is in Idle state. It must be FLEXPWM_IDLE_STATE_HIGH or FLEXPWM_IDLE_STATE_LOW.

Definition at line 182 of file ICS_PWMnCurrFdbkClass.h.

5.46.2.32 uint16_t ICS_Params_t::hCh3NPolarity

Channel 3N (low side) output polarity,it must be FLEXPWM_NPolarity_High or FLEXPWM_NPolarity_Low.

Definition at line 185 of file ICS_PWMnCurrFdbkClass.h.

5.46.2.33 uint16_t ICS_Params_t::hCh3NIdleState

Channel 3N (low side) state (low or high) when FlexPWM is in Idle state. It must be FLEXPWM_IDLE_STATE_HIGH or FLEXPWM_IDLE_STATE_LOW.

Definition at line 188 of file ICS_PWMnCurrFdbkClass.h.

5.46.2.34 FunctionalState ICS_Params_t::EmergencyStop

It enable/disable the management of an emergency input instantaneously stopping PWM generation. It must be either equal to ENABLE or DISABLE.

Definition at line 192 of file ICS_PWMnCurrFdbkClass.h.

5.46.2.35 uint8_t ICS_Params_t::FaultPIN

Emergency Stop PIN. It must be: FAULT_PIN0-FAULT_PIN3.

Definition at line 194 of file ICS_PWMnCurrFdbkClass.h.

5.46.2.36 uint16_t ICS_Params_t::FaultPolarity

Emergency Stop Input polarity, it must be: FAULT_POLARITY_HIGH or FAULT_POLARITY_LOW.

Definition at line 197 of file ICS_PWMnCurrFdbkClass.h.

5.47 Methods_t Struct Reference

5.47.1 Detailed Description

Virtual methods container.

Definition at line 76 of file MotorPowerMeasurementPrivate.h.

```
#include <MotorPowerMeasurementPrivate.h>
```

5.48 MTPACtrlParams_t Struct Reference

5.48.1 Detailed Description

MTPACtrl class parameters definition.

Definition at line 54 of file MTPACtrlClass.h.

```
#include <MTPACtrlClass.h>
```

Data Fields

- int16_t hSegDiv
- int32_t wAngCoeff [8]
- int32_t wOffset [8]

5.48.2 Field Documentation

5.48.2.1 int16_t MTPACtrlParams_t::hSegDiv

Segments divisor

Definition at line 56 of file MTPACtrlClass.h.

5.48.2.2 int32_t MTPACtrlParams_t::wAngCoeff[8]

Angular coefficients table

Definition at line 57 of file MTPACtrlClass.h.

5.48.2.3 int32_t MTPACtrlParams_t::wOffset[8]

Offsets table

Definition at line 58 of file MTPACtrlClass.h.

5.49 NTCParams_t Struct Reference

5.49.1 Detailed Description

NTC class parameters definition.

Definition at line 54 of file NTC_TemperatureSensorClass.h.

```
#include <NTC_TemperatureSensorClass.h>
```

Data Fields

- uint8_t bTsensADCModule
- uint8_t bTsensADChannel
- uint8_t bTsensSamplingTime
- uint16_t hLowPassFilterBW
- uint16_t hOverTempThreshold
- uint16_t hOverTempDeactThreshold
- int16_t hSensitivity

- `uint32_t wV0`
- `uint16_t hT0`

5.49.2 Field Documentation

5.49.2.1 `uint8_t NTCParams_t::bTsensADCModule`

ADC module used for conversion of Temperature.

Definition at line 56 of file NTC_TemperatureSensorClass.h.

5.49.2.2 `uint8_t NTCParams_t::bTsensADChannel`

ADC channel used for conversion of temperature sensor output. It must be equal to ADC_Channel_x x= 0, ..., 15

Definition at line 58 of file NTC_TemperatureSensorClass.h.

5.49.2.3 `uint8_t NTCParams_t::bTsensSamplingTime`

Sampling time used for bVbusChannel AD conversion. It must be equal to ADC_SampleTime_xCycles5 x= 1, 7, ...

Definition at line 61 of file NTC_TemperatureSensorClass.h.

5.49.2.4 `uint16_t NTCParams_t::hLowPassFilterBW`

Use this number to configure the temperature first order software filter bandwidth $hLowPassFilterBW = TSNS_CalcBusReading$ call rate [Hz]/ FilterBandwidth[Hz]

Definition at line 64 of file NTC_TemperatureSensorClass.h.

5.49.2.5 `uint16_t NTCParams_t::hOverTempThreshold`

It represents the over voltage protection intervention threshold. To be expressed in u16Celsius through formula:
 $hOverTempThreshold[\text{u16Celsius}] = (V0[\text{V}] + dV/dT[\text{V/C}] * (\text{OverTempThreshold}[\text{C}] - T0[\text{C}])) * 65536 / \text{MCU supply voltage}$

- $T0[\text{C}]) * 65536 / \text{MCU supply voltage}$

Definition at line 69 of file NTC_TemperatureSensorClass.h.

5.49.2.6 `uint16_t NTCParams_t::hOverTempDeactThreshold`

It represents the temperature expressed in u16Celsius below which an active over temperature fault is cleared
 $hOverTempDeactThreshold[\text{u16Celsius}] = (V0[\text{V}] + dV/dT[\text{V/C}] * (\text{OverTempDeactThresh}[\text{C}] - T0[\text{C}])) * 65536 / \text{MCU supply voltage}$

- $T0[\text{C}]) * 65536 / \text{MCU supply voltage}$

Definition at line 75 of file NTC_TemperatureSensorClass.h.

5.49.2.7 `int16_t NTCParams_t::hSensitivity`

It is equal to MCU supply voltage [V] / dV/dT [V/C] and represents the NTC sensitivity

Definition at line 81 of file NTC_TemperatureSensorClass.h.

5.49.2.8 uint32_t NTCParams_t::wV0

It is equal to $V0 * 65536 / \text{MCU supply voltage}$ where $V0$ is used to convert the temperature into Volts $V[V] = V0 + dV/dT[V/C] * (T - T0)[C]$

Definition at line 84 of file NTC_TemperatureSensorClass.h.

5.49.2.9 uint16_t NTCParams_t::hT0

It is equal to $T0 [C]$, used to convert the temperature into Volts $V[V] = V0 + dV/dT[V/C] * (T - T0)[C]$

Definition at line 88 of file NTC_TemperatureSensorClass.h.

5.50 Observer_Inputs_t Struct Reference

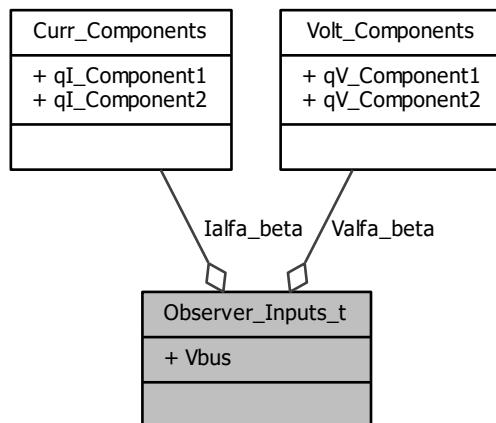
5.50.1 Detailed Description

STO_SPD derived class input structure type definition for SPD_CalcAngle.

Definition at line 50 of file STO_SpeednPosFdbkClass.h.

```
#include <STO_SpeednPosFdbkClass.h>
```

Collaboration diagram for Observer_Inputs_t:



5.51 OpenLoopParams_t Struct Reference

5.51.1 Detailed Description

OpenLoop class parameters definition.

Definition at line 58 of file OpenLoopClass.h.

```
#include <OpenLoopClass.h>
```

Data Fields

- bool `VFMode`
- int16_t `hVFSlope`

5.51.2 Field Documentation

5.51.2.1 bool OpenLoopParams_t::VFMode

Default Open loop voltage. Default mode. TRUE to enable V/F mode otherwise FALSE

Definition at line 61 of file OpenLoopClass.h.

5.51.2.2 int16_t OpenLoopParams_t::hVFSlope

Offset of V/F curve expressed in s16 Voltage applied when frequency is zero.

Definition at line 65 of file OpenLoopClass.h.

5.52 PIDParams_t Struct Reference

5.52.1 Detailed Description

PID class parameters definition.

Definition at line 54 of file PID_PIRegulatorClass.h.

```
#include <PID_PIRegulatorClass.h>
```

Data Fields

- int16_t `hDefKdGain`
- uint16_t `hKdDivisor`
- uint16_t `hKdDivisorPOW2`

5.52.2 Field Documentation

5.52.2.1 int16_t PIDParams_t::hDefKdGain

Default Kd gain, used to initialize Kd gain software variable

Definition at line 56 of file PID_PIRegulatorClass.h.

5.52.2.2 uint16_t PIDParams_t::hKdDivisor

Kd gain divisor, used in conjunction with Kd gain allows obtaining fractional values. If FULL_MISRA_C_COMPLIANCE is not defined the divisor is implemented through algebrical right shifts to speed up PI execution. Only in this case this parameter specifies the number of right shifts to be executed

Definition at line 58 of file PID_PIRegulatorClass.h.

5.52.2.3 uint16_t PIDParams_t::hKdDivisorPOW2

Kd gain divisor expressed as power of 2. E.g. if gain divisor is 512 the value must be 9 because $2^9 = 512$

Definition at line 66 of file PID_RegulatorClass.h.

5.53 PIParams_t Struct Reference

5.53.1 Detailed Description

PI regulator class parameters structure definition.

Definition at line 57 of file PIRegulatorClass.h.

```
#include <PIRegulatorClass.h>
```

Data Fields

- int16_t hDefKpGain
- int16_t hDefKiGain
- uint16_t hKpDivisor
- uint16_t hKiDivisor
- int32_t wDefMaxIntegralTerm
- int32_t wDefMinIntegralTerm
- int16_t hDefMaxOutput
- int16_t hDefMinOutput
- uint16_t hKpDivisorPOW2
- uint16_t hKiDivisorPOW2

5.53.2 Field Documentation

5.53.2.1 int16_t PIParams_t::hDefKpGain

Default Kp gain, used to initialize Kp gain software variable

Definition at line 59 of file PIRegulatorClass.h.

5.53.2.2 int16_t PIParams_t::hDefKiGain

Default Ki gain, used to initialize Ki gain software variable

Definition at line 61 of file PIRegulatorClass.h.

5.53.2.3 uint16_t PIParams_t::hKpDivisor

Kp gain divisor, used in conjunction with Kp gain allows obtaining fractional values. If FULL_MISRA_C_COMPLIANCE is not defined the divisor is implemented through algebrical right shifts to speed up PI execution. Only in this case this parameter specifies the number of right shifts to be executed

Definition at line 63 of file PIRegulatorClass.h.

5.53.2.4 uint16_t PIParams_t::hKiDivisor

Ki gain divisor, used in conjunction with Ki gain allows obtaining fractional values. If FULL_MISRA_C_COMPLIANCY is not defined the divisor is implemented through algebraical right shifts to speed up PI execution. Only in this case this parameter specifies the number of right shifts to be executed

Definition at line 71 of file PIRegulatorClass.h.

5.53.2.5 int32_t PIParams_t::wDefMaxIntegralTerm

Upper limit used to saturate the integral term given by Ki / Ki divisor * integral of process variable error

Definition at line 79 of file PIRegulatorClass.h.

5.53.2.6 int32_t PIParams_t::wDefMinIntegralTerm

Lower limit used to saturate the integral term given by Ki / Ki divisor * integral of process variable error

Definition at line 82 of file PIRegulatorClass.h.

5.53.2.7 int16_t PIParams_t::hDefMaxOutput

Upper limit used to saturate the PI output

Definition at line 85 of file PIRegulatorClass.h.

5.53.2.8 int16_t PIParams_t::hDefMinOutput

Lower limit used to saturate the PI output

Definition at line 87 of file PIRegulatorClass.h.

5.53.2.9 uint16_t PIParams_t::hKpDivisorPOW2

Kp gain divisor expressed as power of 2. E.g. if gain divisor is 512 the value must be 9 because $2^9 = 512$

Definition at line 89 of file PIRegulatorClass.h.

5.53.2.10 uint16_t PIParams_t::hKiDivisorPOW2

Ki gain divisor expressed as power of 2. E.g. if gain divisor is 512 the value must be 9 because $2^9 = 512$

Definition at line 92 of file PIRegulatorClass.h.

5.54 PQD_MPMLInitStruct_t Struct Reference

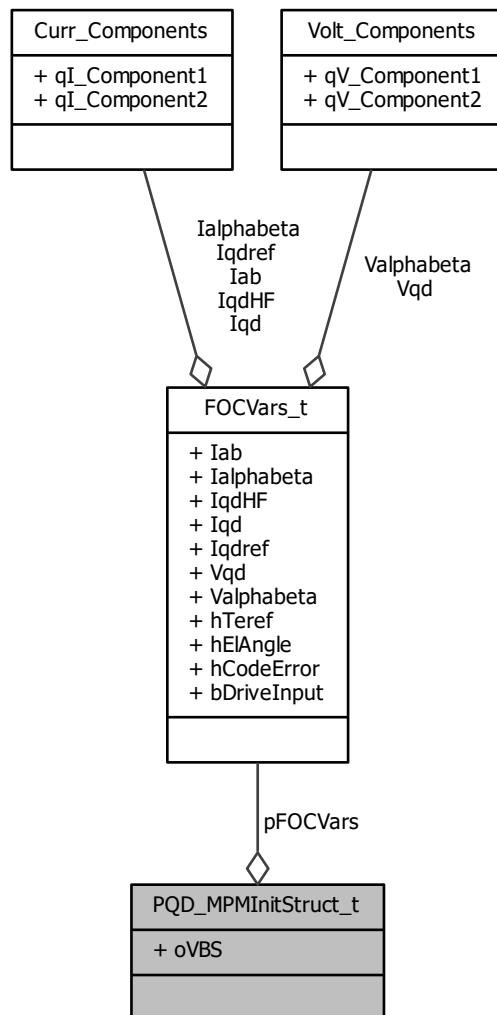
5.54.1 Detailed Description

PQD class init struct definition.

Definition at line 68 of file PQD_MotorPowerMeasurementClass.h.

```
#include <PQD_MotorPowerMeasurementClass.h>
```

Collaboration diagram for PQD_MPMPInitStruct_t:



Data Fields

- `pFOCVars_t pFOCVars`
- `CVBS oVBS`

5.54.2 Field Documentation

5.54.2.1 `pFOCVars_t PQD_MPMPInitStruct_t::pFOCVars`

Pointer to FOC vars used by MPM.

Definition at line 69 of file PQD_MotorPowerMeasurementClass.h.

5.54.2.2 CVBS PQD_MPMLInitStruct_t::oVBS

Bus voltage sensor object used by MPM.

Definition at line 70 of file PQD_MotorPowerMeasurementClass.h.

5.55 PQDParams_t Struct Reference

5.55.1 Detailed Description

PQD class parameters definition.

Definition at line 56 of file PQD_MotorPowerMeasurementClass.h.

```
#include <PQD_MotorPowerMeasurementClass.h>
```

5.56 PWMnCurrFdbkParams_t Struct Reference

5.56.1 Detailed Description

PWMnCurrFdbk class parameters definition.

Definition at line 69 of file PWMnCurrFdbkClass.h.

```
#include <PWMnCurrFdbkClass.h>
```

Data Fields

- `uint16_t hPWMperiod`
It contains the PWM period expressed in timer clock cycles unit: hPWMPPeriod = Timer Fclk / Fpwm.
- `uint16_t hOffCalibrWaitTicks`

5.56.2 Field Documentation

5.56.2.1 `uint16_t PWMnCurrFdbkParams_t::hPWMperiod`

It contains the PWM period expressed in timer clock cycles unit: hPWMPPeriod = Timer Fclk / Fpwm.

Definition at line 73 of file PWMnCurrFdbkClass.h.

5.56.2.2 `uint16_t PWMnCurrFdbkParams_t::hOffCalibrWaitTicks`

Wait time duration before current reading calibration expressed in number of calls of PWMC_CurrentReadingCalibr with action CRC_EXEC

Definition at line 74 of file PWMnCurrFdbkClass.h.

5.57 R1_Params_t Struct Reference

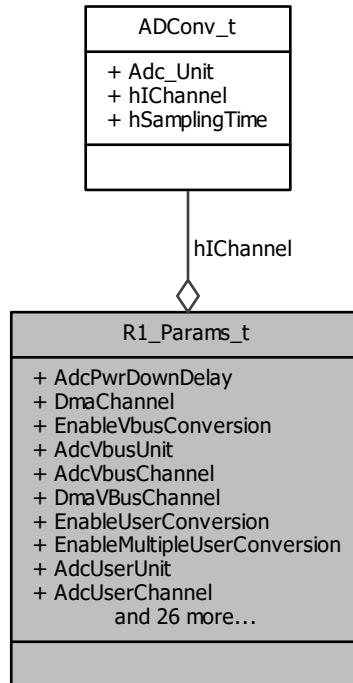
5.57.1 Detailed Description

R1 class parameters definition.

Definition at line 65 of file R1_PWMnCurrFdbkClass.h.

```
#include <R1_PWMnCurrFdbkClass.h>
```

Collaboration diagram for R1_Params_t:



Data Fields

- **ADConv_t hIChannel**
ADC channel configuration used for conversion of current Ia.
- **uint8_t AdcPwrDownDelay**
Power Down Exit Delay for ADC Units.
- **uint8_t DmaChannel**
Dma channel for ADC conversion.
- **bool EnableVbusConversion**
VBus conversion enable/disable flag.
- **uint8_t AdcVbusUnit**
Adc unit for VBus conversion.
- **uint8_t AdcVbusChannel**
Adc channel for VBus conversion.
- **uint8_t DmaVBusChannel**
Dma channel for VBus conversion.
- **bool EnableUserConversion**
User conversion enable/disable flag.
- **bool EnableMultipleUserConversion**
Multiple User conversion enable/disable flag.
- **uint8_t AdcUserUnit**

- **AdcUnit**
Adc unit for User conversion.
- **uint8_t AdcUserChannel**
Adc channel for User conversion.
- **uint8_t DmaUserChannel**
Dma channel for User conversion.
- **uint8_t IRQnb**
MC IRQ number used for R1 management.
- **uint16_t hDeadTime**
Dead time in number of clock cycles.
- **uint16_t hTafter**
It is the sum of dead time plus rise time express in number of.
- **uint16_t hTbefore**
It is the sampling time express in number of FlexPWM clocks.
- **uint16_t hTMin**
It is the sum of dead time plus rise time plus sampling time.
- **uint16_t hHTMin**
It is the half of hTMin value.
- **uint16_t hTSample**
It is the sampling time express in numbers of FlexPWM clocks.
- **uint8_t FlexPWM_Clock_Divider**
PWM clock frequency divide ratio: PWM clock frequency divide ratio: FLEXPWM_CTRL_PRSC_PRESC_1, FLEXPWM_CTRL_PRSC_PRESC_2, FLEXPWM_CTRL_PRSC_PRESC_4, FLEXPWM_CTRL_PRSC_PRESC_8, FLEXPWM_CTRL_PRSC_PRESC_16 FLEXPWM_CTRL_PRSC_PRESC_32 FLEXPWM_CTRL_PRSC_PRESC_64 FLEXPWM_CTRL_PRSC_PRESC_128.
- **uint8_t FlexPWMModule**
FlexPWM used for PWM generation.
- **uint16_t PWMLoadFreq**
Dead time in number of IPBUS clock cycles regardless of the setting of FlexPWM_Clock_Divider.
- **uint16_t hCh1Polarity**
Channel 1 (high side) output polarity,it must be FLEXPWM_Polarity_High or FLEXPWM_Polarity_Low.
- **uint16_t hCh1IdleState**
Channel 1 (high side) state (low or high) when FlexPWM is in Idle state. It must be FLEXPWM_IDLE_STATE_HIGH or FLEXPWM_IDLE_STATE_LOW.
- **uint16_t hCh2Polarity**
Channel 2 (high side) output polarity,it must be FLEXPWM_Polarity_High or FLEXPWM_Polarity_Low.
- **uint16_t hCh2IdleState**
Channel 2 (high side) state (low or high) when FlexPWM is in Idle state. It must be FLEXPWM_IDLE_STATE_HIGH or FLEXPWM_IDLE_STATE_LOW.
- **uint16_t hCh3Polarity**
Channel 3 (high side) output polarity,it must be FLEXPWM_Polarity_High or FLEXPWM_Polarity_Low.
- **uint16_t hCh3IdleState**
Channel 3 (high side) state (low or high) when FlexPWM is in Idle state. It must be FLEXPWM_IDLE_STATE_HIGH or FLEXPWM_IDLE_STATE_LOW.
- **LowSideOutputsFunction_t LowSideOutputs**
Low side or enabling signals generation method definition.
- **uint16_t hCh1NPolarity**
Channel 1N (low side) output polarity,it must be FLEXPWM_NPolarity_High or FLEXPWM_NPolarity_Low.
- **uint16_t hCh1NIdleState**
Channel 1N (low side) state (low or high) when FlexPWM is in Idle state. It must be FLEXPWM_IDLE_STATE_HIGH or FLEXPWM_IDLE_STATE_LOW.
- **uint16_t hCh2NPolarity**
Channel 2N (low side) output polarity,it must be FLEXPWM_NPolarity_High or FLEXPWM_NPolarity_Low.

- **uint16_t hCh2NIdleState**
Channel 2N (low side) state (low or high) when FlexPWM is in Idle state. It must be FLEXPWM_IDLE_STATE_HIGH or FLEXPWM_IDLE_STATE_LOW.
- **uint16_t hCh3NPolarity**
Channel 3N (low side) output polarity,it must be FLEXPWM_NPolarity_High or FLEXPWM_NPolarity_Low.
- **uint16_t hCh3NIdleState**
Channel 3N (low side) state (low or high) when FlexPWM is in Idle state. It must be FLEXPWM_IDLE_STATE_HIGH or FLEXPWM_IDLE_STATE_LOW.
- **FunctionalState EmergencyStop**
It enable/disable the management of an emergency input instantaneously stopping PWM generation. It must be either equal to ENABLE or DISABLE.
- **uint8_t FaultPIN**
Emergency Stop PIN. It must be: FAULT_PIN0-FAULT_PIN3.
- **uint16_t FaultPolarity**
Emergency Stop Input polarity, it must be: FAULT_POLARITY_HIGH or FAULT_POLARITY_LOW.

5.57.2 Field Documentation

5.57.2.1 **ADCConv_t R1_Params_t::hIChannel**

ADC channel configuration used for conversion of current Ia.

Definition at line 70 of file R1_PWMnCurrFdbkClass.h.

5.57.2.2 **uint8_t R1_Params_t::AdcPwrDownDelay**

Power Down Exit Delay for ADC Units.

Definition at line 72 of file R1_PWMnCurrFdbkClass.h.

5.57.2.3 **uint8_t R1_Params_t::DmaChannel**

Dma channel for ADC conversion.

Definition at line 74 of file R1_PWMnCurrFdbkClass.h.

5.57.2.4 **bool R1_Params_t::EnableVbusConversion**

VBus conversion enable/disable flag.

Definition at line 77 of file R1_PWMnCurrFdbkClass.h.

5.57.2.5 **uint8_t R1_Params_t::AdcVbusUnit**

Adc unit for VBus conversion.

Definition at line 79 of file R1_PWMnCurrFdbkClass.h.

5.57.2.6 **uint8_t R1_Params_t::AdcVbusChannel**

Adc channel for VBus conversion.

Definition at line 81 of file R1_PWMnCurrFdbkClass.h.

5.57.2.7 uint8_t R1_Params_t::DmaVBusChannel

Dma channel for VBus conversion.

Definition at line 83 of file R1_PWMnCurrFdbkClass.h.

5.57.2.8 bool R1_Params_t::EnableUserConversion

User conversion enable/disable flag.

Definition at line 85 of file R1_PWMnCurrFdbkClass.h.

5.57.2.9 bool R1_Params_t::EnableMultipleUserConversion

Multiple User conversion enable/disable flag.

Definition at line 87 of file R1_PWMnCurrFdbkClass.h.

5.57.2.10 uint8_t R1_Params_t::AdcUserUnit

Adc unit for User conversion.

Definition at line 89 of file R1_PWMnCurrFdbkClass.h.

5.57.2.11 uint8_t R1_Params_t::AdcUserChannel

Adc channel for User conversion.

Definition at line 91 of file R1_PWMnCurrFdbkClass.h.

5.57.2.12 uint8_t R1_Params_t::DmaUserChannel

Dma channel for User conversion.

Definition at line 93 of file R1_PWMnCurrFdbkClass.h.

5.57.2.13 uint8_t R1_Params_t::IRQnb

MC IRQ number used for R1 management.

Definition at line 95 of file R1_PWMnCurrFdbkClass.h.

Referenced by R1_NewObject().

5.57.2.14 uint16_t R1_Params_t::hDeadTime

Dead time in number of clock cycles.

Definition at line 98 of file R1_PWMnCurrFdbkClass.h.

5.57.2.15 uint16_t R1_Params_t::hTaftter

It is the sum of dead time plus rise time express in number of.

Definition at line 101 of file R1_PWMnCurrFdbkClass.h.

5.57.2.16 uint16_t R1_Params_t::hTbefore

It is the sampling time express in number of FlexPWM clocks.

Definition at line 103 of file R1_PWMnCurrFdbkClass.h.

5.57.2.17 uint16_t R1_Params_t::hTMin

It is the sum of dead time plus rise time plus sampling time.

Definition at line 106 of file R1_PWMnCurrFdbkClass.h.

5.57.2.18 uint16_t R1_Params_t::hHTMin

It is the half of hTMin value.

Definition at line 108 of file R1_PWMnCurrFdbkClass.h.

5.57.2.19 uint16_t R1_Params_t::hTSample

It is the sampling time express in numbers of FlexPWM clocks.

Definition at line 110 of file R1_PWMnCurrFdbkClass.h.

5.57.2.20 uint8_t R1_Params_t::FlexPWM_Clock_Divider

PWM clock frequency divide ratio: PWM clock frequency divide ratio: FLEXPWM_CTRL_PRSC_PRESC_1, FLEXPWM_CTRL_PRSC_PRESC_2, FLEXPWM_CTRL_PRSC_PRESC_4, FLEXPWM_CTRL_PRSC_PRESC_8, FLEXPWM_CTRL_PRSC_PRESC_16 FLEXPWM_CTRL_PRSC_PRESC_32 FLEXPWM_CTRL_PRSC_PRESC_64 FLEXPWM_CTRL_PRSC_PRESC_128.

Definition at line 121 of file R1_PWMnCurrFdbkClass.h.

5.57.2.21 uint8_t R1_Params_t::FlexPWMMModule

FlexPWM used for PWM generation.

Definition at line 123 of file R1_PWMnCurrFdbkClass.h.

5.57.2.22 uint16_t R1_Params_t::PWMLoadFreq

Dead time in number of IPBUS clock cycles regardless of the setting of FlexPWM_Clock_Divider.

PWM load frequency: FLEXPWM_CTRL_LDFQ_EACH1- FLEXPWM_CTRL_LDFQ_EACH16

Definition at line 128 of file R1_PWMnCurrFdbkClass.h.

5.57.2.23 uint16_t R1_Params_t::hCh1Polarity

Channel 1 (high side) output polarity,it must be FLEXPWM_Polarity_High or FLEXPWM_Polarity_Low.

Definition at line 132 of file R1_PWMnCurrFdbkClass.h.

5.57.2.24 uint16_t R1_Params_t::hCh1IdleState

Channel 1 (high side) state (low or high) when FlexPWM is in Idle state. It must be FLEXPWM_IDLE_STATE_HIGH or FLEXPWM_IDLE_STATE_LOW.

Definition at line 135 of file R1_PWMnCurrFdbkClass.h.

5.57.2.25 uint16_t R1_Params_t::hCh2Polarity

Channel 2 (high side) output polarity,it must be FLEXPWM_Polarity_High or FLEXPWM_Polarity_Low.

Definition at line 138 of file R1_PWMnCurrFdbkClass.h.

5.57.2.26 uint16_t R1_Params_t::hCh2IdleState

Channel 2 (high side) state (low or high) when FlexPWM is in Idle state. It must be FLEXPWM_IDLE_STATE_HIGH or FLEXPWM_IDLE_STATE_LOW.

Definition at line 141 of file R1_PWMnCurrFdbkClass.h.

5.57.2.27 uint16_t R1_Params_t::hCh3Polarity

Channel 3 (high side) output polarity,it must be FLEXPWM_Polarity_High or FLEXPWM_Polarity_Low.

Definition at line 144 of file R1_PWMnCurrFdbkClass.h.

5.57.2.28 uint16_t R1_Params_t::hCh3IdleState

Channel 3 (high side) state (low or high) when FlexPWM is in Idle state. It must be FLEXPWM_IDLE_STATE_HIGH or FLEXPWM_IDLE_STATE_LOW.

Definition at line 147 of file R1_PWMnCurrFdbkClass.h.

5.57.2.29 LowSideOutputsFunction_t R1_Params_t::LowSideOutputs

Low side or enabling signals generation method definition.

Definition at line 149 of file R1_PWMnCurrFdbkClass.h.

5.57.2.30 uint16_t R1_Params_t::hCh1NPolarity

Channel 1N (low side) output polarity,it must be FLEXPWM_NPolarity_High or FLEXPWM_NPolarity_Low.

Definition at line 152 of file R1_PWMnCurrFdbkClass.h.

5.57.2.31 uint16_t R1_Params_t::hCh1NIdleState

Channel 1N (low side) state (low or high) when FlexPWM is in Idle state. It must be FLEXPWM_IDLE_STATE_HIGH or FLEXPWM_IDLE_STATE_LOW.

Definition at line 155 of file R1_PWMnCurrFdbkClass.h.

5.57.2.32 uint16_t R1_Params_t::hCh2NPolarity

Channel 2N (low side) output polarity,it must be FLEXPWM_NPolarity_High or FLEXPWM_NPolarity_Low.

Definition at line 158 of file R1_PWMnCurrFdbkClass.h.

5.57.2.33 uint16_t R1_Params_t::hCh2NIdleState

Channel 2N (low side) state (low or high) when FlexPWM is in Idle state. It must be FLEXPWM_IDLE_STATE_HIGH or FLEXPWM_IDLE_STATE_LOW.

Definition at line 161 of file R1_PWMnCurrFdbkClass.h.

5.57.2.34 uint16_t R1_Params_t::hCh3NPolarity

Channel 3N (low side) output polarity, it must be FLEXPWM_NPolarity_High or FLEXPWM_NPolarity_Low.

Definition at line 164 of file R1_PWMnCurrFdbkClass.h.

5.57.2.35 uint16_t R1_Params_t::hCh3NIdleState

Channel 3N (low side) state (low or high) when FlexPWM is in Idle state. It must be FLEXPWM_IDLE_STATE_HIGH or FLEXPWM_IDLE_STATE_LOW.

Definition at line 167 of file R1_PWMnCurrFdbkClass.h.

5.57.2.36 FunctionalState R1_Params_t::EmergencyStop

It enable/disable the management of an emergency input instantaneously stopping PWM generation. It must be either equal to ENABLE or DISABLE.

Definition at line 171 of file R1_PWMnCurrFdbkClass.h.

5.57.2.37 uint8_t R1_Params_t::FaultPIN

Emergency Stop PIN. It must be: FAULT_PIN0-FAULT_PIN3.

Definition at line 173 of file R1_PWMnCurrFdbkClass.h.

5.57.2.38 uint16_t R1_Params_t::FaultPolarity

Emergency Stop Input polarity, it must be: FAULT_POLARITY_HIGH or FAULT_POLARITY_LOW.

Definition at line 176 of file R1_PWMnCurrFdbkClass.h.

5.58 RdividerParams_t Struct Reference

5.58.1 Detailed Description

Rdivider class parameters definition.

Definition at line 54 of file Rdivider_BusVoltageSensorClass.h.

```
#include <Rdivider_BusVoltageSensorClass.h>
```

Data Fields

- uint8_t bVbusADCModule
- uint8_t bVbusADChannel
- uint8_t bVbusSamplingTime
- uint16_t hLowPassFilterBW

- uint16_t **hOverVoltageThreshold**
- uint16_t **hUnderVoltageThreshold**

5.58.2 Field Documentation

5.58.2.1 uint8_t **RdividerParams_t::bVbusADCModule**

ADC module used for conversion of bus voltage.

Definition at line 56 of file Rdivider_BusVoltageSensorClass.h.

5.58.2.2 uint8_t **RdividerParams_t::bVbusADChannel**

ADC channel used for conversion of bus voltage.

Definition at line 58 of file Rdivider_BusVoltageSensorClass.h.

5.58.2.3 uint8_t **RdividerParams_t::bVbusSamplingTime**

Sampling time used for bVbusChannel AD conversion. It must be equal to ADC_SampleTime_xCycles5 x= 1, 7, ...

Definition at line 60 of file Rdivider_BusVoltageSensorClass.h.

5.58.2.4 uint16_t **RdividerParams_t::hLowPassFilterBW**

Use this number to configure the Vbus first order software filter bandwidth. hLowPassFilterBW = VBS_CalcBus-Reading call rate [Hz]/ FilterBandwidth[Hz]

Definition at line 63 of file Rdivider_BusVoltageSensorClass.h.

5.58.2.5 uint16_t **RdividerParams_t::hOverVoltageThreshold**

It represents the over voltage protection intervention threshold. To be expressed in digital value through formula:
hOverVoltageThreshold (digital value) = Over Voltage Threshold (V) * 65536 / hConversionFactor

Definition at line 67 of file Rdivider_BusVoltageSensorClass.h.

5.58.2.6 uint16_t **RdividerParams_t::hUnderVoltageThreshold**

It represents the under voltage protection intervention threshold. To be expressed in digital value through formula:
hUnderVoltageThreshold (digital value)= Under Voltage Threshold (V) * 65536 / hConversionFactor

Definition at line 73 of file Rdivider_BusVoltageSensorClass.h.

5.59 RESOLVERParams_t Struct Reference

5.59.1 Detailed Description

RESOLVER class parameters definition.

Definition at line 73 of file RESOLVER_SpeednPosFdbkClass.h.

```
#include <RESOLVER_SpeednPosFdbkClass.h>
```

Data Fields

- `uint8_t eTimerModule`
Timer used for RESOLVER sensor management.
- `uint8_t RES_Excitation_eTimer_ch`
Resolver excitation eTimer channel.
- `uint16_t RES_Filtering_shift`
Resolver Filtering shift.
- `uint16_t RES_Excitation_Signal_freq`
Resolver Signal Excitation frequency.
- `uint8_t Res_SIN_ADC_Module`
Resolver feedback SIN signal: ADC module.
- `uint8_t Res_SIN_ADC_Channel`
Resolver feedback SIN signal. ADC channel.
- `uint8_t Res_COS_ADC_Module`
Resolver feedback COS signal: ADC module.
- `uint8_t Res_COS_ADC_Channel`
Resolver feedback COS signal. ADC channel.
- `uint16_t hSpeedSamplingFreqHz`
Frequency (Hz) at which motor speed is to be computed. It must be equal to the frequency at which function SPD_CalcAvrgMecSpeed01Hz is called.
- `uint8_t DmaChannel`
Dma channel.
- `int16_t hPhaseShift`
Dma channel.
- `int32_t hResSinPhaseChannelB`
*Sin of Channel B Phase is the sin (multiplied by 1024) of the offset angle between channel A and channel B. e.g. phase = 120 -> RES_SIN_PHASE_CH_B = 0.86 * 1024 = 887.*
- `int32_t hResCosPhaseChannelB`
*Sin of Channel B Phase is the sin (multiplied by 1024) of the offset angle between channel A and channel B. e.g. phase = 120 -> RES_COS_PHASE_CH_B = -0.5 * 1024 = -512.*
- `ResolverSensorType_t RES_Angle_Reading_Type`
This is RES_ELECTRICAL_DEGREE if the resolver is reading electrical degree, otherwise is RES_MECHANICAL_DEGREE.
- `int32_t hResPole2MotorDegreesMultiplier`
This is the multiplier that convert resolver pole pair into motor pole pair.
- `int32_t hResMinADCChannelA`
This is the minimum value ranged for the wave in ADC CH A.
- `int32_t hResMaxADCChannelA`
This is the maximum value ranged for the wave in ADC CH A.
- `int32_t hResMinADCChannelB`
This is the minimum value ranged for the wave in ADC CH B.
- `int32_t hResMaxADCChannelB`
This is the maximum value ranged for the wave in ADC CH B.

5.59.2 Field Documentation

5.59.2.1 `uint8_t RESOLVERParams_t::eTimerModule`

Timer used for RESOLVER sensor management.

Definition at line 77 of file RESOLVER_SpeednPosFdbkClass.h.

5.59.2.2 uint8_t RESOLVERParams_t::RES_Excitation_eTimer_ch

Resolver excitation eTimer channel.

Definition at line 79 of file RESOLVER_SpeednPosFdbkClass.h.

5.59.2.3 uint16_t RESOLVERParams_t::RES_Filtering_shift

Resolver Filtering shift.

Definition at line 81 of file RESOLVER_SpeednPosFdbkClass.h.

5.59.2.4 uint16_t RESOLVERParams_t::RES_Excitation_Signal_freq

Resolver Signal Excitation frequency.

Definition at line 83 of file RESOLVER_SpeednPosFdbkClass.h.

5.59.2.5 uint8_t RESOLVERParams_t::Res_SIN_ADC_Module

Resolver feedback SIN signal: ADC module.

Definition at line 85 of file RESOLVER_SpeednPosFdbkClass.h.

5.59.2.6 uint8_t RESOLVERParams_t::Res_SIN_ADC_Channel

Resolver feedback SIN signal. ADC channel.

Definition at line 87 of file RESOLVER_SpeednPosFdbkClass.h.

5.59.2.7 uint8_t RESOLVERParams_t::Res_COS_ADC_Module

Resolver feedback COS signal: ADC module.

Definition at line 89 of file RESOLVER_SpeednPosFdbkClass.h.

5.59.2.8 uint8_t RESOLVERParams_t::Res_COS_ADC_Channel

Resolver feedback COS signal. ADC channel.

Definition at line 91 of file RESOLVER_SpeednPosFdbkClass.h.

5.59.2.9 uint16_t RESOLVERParams_t::hSpeedSamplingFreqHz

Frequency (Hz) at which motor speed is to be computed. It must be equal to the frequency at which function SPD_CalcAvgMecSpeed01Hz is called.

Definition at line 95 of file RESOLVER_SpeednPosFdbkClass.h.

5.59.2.10 uint8_t RESOLVERParams_t::DmaChannel

Dma channel.

Definition at line 97 of file RESOLVER_SpeednPosFdbkClass.h.

5.59.2.11 int16_t RESOLVERParams_t::hPhaseShift

Dma channel.

Define here in s16degree the electrical phase shift between the Resolver Channel A and the Bemf induced on phase A

Definition at line 101 of file RESOLVER_SpeednPosFdbkClass.h.

5.59.2.12 int32_t RESOLVERParams_t::hResSinPhaseChannelB

Sin of Channel B Phase is the sin (multiplied by 1024) of the offset angle between channel A and channel B. e.g. phase = 120 -> RES_SIN_PHASE_CH_B = 0.86 * 1024 = 887.

Definition at line 105 of file RESOLVER_SpeednPosFdbkClass.h.

5.59.2.13 int32_t RESOLVERParams_t::hResCosPhaseChannelB

Sin of Channel B Phase is the sin (multiplied by 1024) of the offset angle between channel A and channel B. e.g. phase = 120 -> RES_COS_PHASE_CH_B = -0.5 * 1024 = -512.

Definition at line 109 of file RESOLVER_SpeednPosFdbkClass.h.

5.59.2.14 ResolverSensorType_t RESOLVERParams_t::RES_Angle_Reading_Type

This is RES_ELECTRICAL_DEGREE if the resolver is reading electrical degree, otherwise is RES_MECHANICAL_DEGREE.

Definition at line 112 of file RESOLVER_SpeednPosFdbkClass.h.

5.59.2.15 int32_t RESOLVERParams_t::hResPole2MotorDegreesMultiplier

This is the multiplier that convert resolver pole pair into motor pole pair.

Definition at line 114 of file RESOLVER_SpeednPosFdbkClass.h.

5.59.2.16 int32_t RESOLVERParams_t::hResMinADCChannelA

This is the minimum value ranged for the wave in ADC CH A.

Definition at line 116 of file RESOLVER_SpeednPosFdbkClass.h.

5.59.2.17 int32_t RESOLVERParams_t::hResMaxADCChannelA

This is the maximum value ranged for the wave in ADC CH A.

Definition at line 118 of file RESOLVER_SpeednPosFdbkClass.h.

5.59.2.18 int32_t RESOLVERParams_t::hResMinADCChannelB

This is the minimum value ranged for the wave in ADC CH B.

Definition at line 120 of file RESOLVER_SpeednPosFdbkClass.h.

5.59.2.19 int32_t RESOLVERParams_t::hResMaxADCChannelB

This is the maximum value ranged for the wave in ADC CH B.

Definition at line 122 of file RESOLVER_SpeednPosFdbkClass.h.

5.60 RevupCtrlParams_t Struct Reference

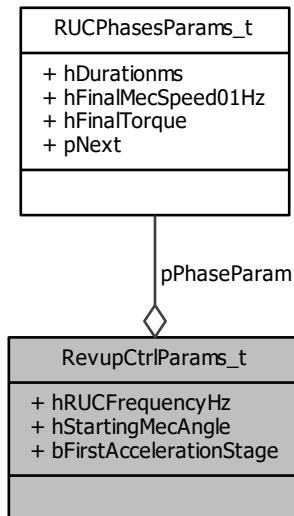
5.60.1 Detailed Description

RevupCtrl class parameters definition.

Definition at line 75 of file RevupCtrlClass.h.

```
#include <RevupCtrlClass.h>
```

Collaboration diagram for RevupCtrlParams_t:



Data Fields

- uint16_t hRUCFrequencyHz
- int16_t hStartingMecAngle
- pRUCPhasesParams_t pPhaseParam

5.60.2 Field Documentation

5.60.2.1 uint16_t RevupCtrlParams_t::hRUCFrequencyHz

Frequency expressed in Hz at which the user clocks the RUC calling RUC_Exec method

Definition at line 77 of file RevupCtrlClass.h.

5.60.2.2 int16_t RevupCtrlParams_t::hStartingMecAngle

Starting angle of programmed rev up.

Definition at line 79 of file RevupCtrlClass.h.

5.60.2.3 pRUCPhasesParams_t RevupCtrlParams_t::pPhaseParam

Pointer of the first element of phase params that constitute the programmed rev up. Each element is of type RUC-
PhasesParams_t.

Definition at line 80 of file RevupCtrlClass.h.

5.61 RUCPhasesParams_t Struct Reference

5.61.1 Detailed Description

Parameters related to each rev up phase.

Definition at line 54 of file RevupCtrlClass.h.

```
#include <RevupCtrlClass.h>
```

Data Fields

- uint16_t hDurationms
- int16_t hFinalMecSpeed01Hz
- int16_t hFinalTorque
- void * pNext

5.61.2 Field Documentation

5.61.2.1 uint16_t RUCPhasesParams_t::hDurationms

Duration of the rev up phase expressed in milliseconds.

Definition at line 56 of file RevupCtrlClass.h.

Referenced by RUC_Exec(), and RUC_Init().

5.61.2.2 int16_t RUCPhasesParams_t::hFinalMecSpeed01Hz

Mechanical speed expressed in 0.1Hz assumed by VSS at the end of the rev up phase.

Definition at line 58 of file RevupCtrlClass.h.

Referenced by RUC_Exec(), and RUC_Init().

5.61.2.3 int16_t RUCPhasesParams_t::hFinalTorque

Motor torque reference imposed by STC at the end of rev up phase. This value represents actually the Iq current expressed in digit.

Definition at line 60 of file RevupCtrlClass.h.

Referenced by RUC_Exec(), and RUC_Init().

5.61.2.4 void* RUCPhasesParams_t::pNext

Pointer of the next element of phase params. It must be MC_NULL for the last element.

Definition at line 63 of file RevupCtrlClass.h.

Referenced by RUC_Clear(), RUC_Exec(), and RUC_Init().

5.62 SpeednPosFdbkParams_t Struct Reference

5.62.1 Detailed Description

SpeednPosFdbk class parameters definition.

Definition at line 80 of file SpeednPosFdbkClass.h.

```
#include <SpeednPosFdbkClass.h>
```

Data Fields

- uint8_t bEIToMecRatio

Coefficient used to transform electrical to mechanical quantities and viceversa. It usually coincides with motor pole pairs number.

- uint16_t hMaxReliableMecSpeed01Hz

Maximum value of measured speed that is considered to be valid. It's expressed in tenth of mechanical Hertz.

- uint16_t hMinReliableMecSpeed01Hz

Minimum value of measured speed that is considered to be valid. It's expressed in tenth of mechanical Hertz.

- uint8_t bMaximumSpeedErrorsNumber

Maximum value of not valid measurements before an error is reported.

- uint16_t hMaxReliableMecAccel01HzP

Maximum value of measured acceleration that is considered to be valid. It's expressed in 01HzP (tenth of Hertz per speed calculation period)

- uint16_t hMeasurementFrequency

Frequency on which the user will request a measurement of the rotor electrical angle. It's also used to convert measured speed from tenth of Hz to dpp and viceversa.

5.62.2 Field Documentation

5.62.2.1 uint8_t SpeednPosFdbkParams_t::bEIToMecRatio

Coefficient used to transform electrical to mechanical quantities and viceversa. It usually coincides with motor pole pairs number.

Definition at line 85 of file SpeednPosFdbkClass.h.

5.62.2.2 uint16_t SpeednPosFdbkParams_t::hMaxReliableMecSpeed01Hz

Maximum value of measured speed that is considered to be valid. It's expressed in tenth of mechanical Hertz.

Definition at line 88 of file SpeednPosFdbkClass.h.

5.62.2.3 uint16_t SpeednPosFdbkParams_t::hMinReliableMecSpeed01Hz

Minimum value of measured speed that is considered to be valid. It's expressed in tenth of mechanical Hertz.

Definition at line 91 of file SpeednPosFdbkClass.h.

5.62.2.4 uint8_t SpeednPosFdbkParams_t::bMaximumSpeedErrorsNumber

Maximum value of not valid measurements before an error is reported.

Definition at line 93 of file SpeednPosFdbkClass.h.

5.62.2.5 uint16_t SpeednPosFdbkParams_t::hMaxReliableMecAccel01HzP

Maximum value of measured acceleration that is considered to be valid. It's expressed in 01HzP (tenth of Hertz per speed calculation period)

Definition at line 96 of file SpeednPosFdbkClass.h.

5.62.2.6 uint16_t SpeednPosFdbkParams_t::hMeasurementFrequency

Frequency on which the user will request a measurement of the rotor electrical angle. It's also used to convert measured speed from tenth of Hz to dpp and viceversa.

Definition at line 100 of file SpeednPosFdbkClass.h.

5.63 SpeednTorqCtrlParams_t Struct Reference

5.63.1 Detailed Description

SpeednTorqCtrl class parameters definition.

Definition at line 59 of file SpeednTorqCtrlClass.h.

```
#include <SpeednTorqCtrlClass.h>
```

Data Fields

- uint16_t hSTCFrequencyHz
- uint16_t hMaxAppPositiveMecSpeed01Hz
- uint16_t hMinAppPositiveMecSpeed01Hz
- int16_t hMaxAppNegativeMecSpeed01Hz
- int16_t hMinAppNegativeMecSpeed01Hz
- uint16_t hMaxPositiveTorque
- int16_t hMinNegativeTorque
- STC_Modality_t bModeDefault
- int16_t hMecSpeedRef01HzDefault
- int16_t hTorqueRefDefault
- int16_t hIdrefDefault

5.63.2 Field Documentation

5.63.2.1 uint16_t SpeednTorqCtrlParams_t::hSTCFrequencyHz

Frequency on which the user updates the torque reference calling STC_CalcTorqueReference method expressed in Hz

Definition at line 61 of file SpeednTorqCtrlClass.h.

5.63.2.2 uint16_t SpeednTorqCtrlParams_t::hMaxAppPositiveMecSpeed01Hz

Application maximum positive value of rotor speed. It's expressed in tenth of mechanical Hertz.

Definition at line 65 of file SpeednTorqCtrlClass.h.

5.63.2.3 uint16_t SpeednTorqCtrlParams_t::hMinAppPositiveMecSpeed01Hz

Application minimum positive value of rotor speed. It's expressed in tenth of mechanical Hertz.

Definition at line 68 of file SpeednTorqCtrlClass.h.

5.63.2.4 int16_t SpeednTorqCtrlParams_t::hMaxAppNegativeMecSpeed01Hz

Application maximum negative value of rotor speed. It's expressed in tenth of mechanical Hertz.

Definition at line 71 of file SpeednTorqCtrlClass.h.

5.63.2.5 int16_t SpeednTorqCtrlParams_t::hMinAppNegativeMecSpeed01Hz

Application minimum negative value of rotor speed. It's expressed in tenth of mechanical Hertz.

Definition at line 74 of file SpeednTorqCtrlClass.h.

5.63.2.6 uint16_t SpeednTorqCtrlParams_t::hMaxPositiveTorque

Maximum positive value of motor torque. This value represents actually the maximum Iq current expressed in digit.

Definition at line 77 of file SpeednTorqCtrlClass.h.

5.63.2.7 int16_t SpeednTorqCtrlParams_t::hMinNegativeTorque

Minimum negative value of motor torque. This value represents actually the maximum Iq current expressed in digit.

Definition at line 81 of file SpeednTorqCtrlClass.h.

5.63.2.8 STC_Modality_t SpeednTorqCtrlParams_t::bModeDefault

Default STC modality.

Definition at line 85 of file SpeednTorqCtrlClass.h.

5.63.2.9 int16_t SpeednTorqCtrlParams_t::hMecSpeedRef01HzDefault

Default mechanical rotor speed reference expressed in tenths of HZ.

Definition at line 86 of file SpeednTorqCtrlClass.h.

5.63.2.10 int16_t SpeednTorqCtrlParams_t::hTorqueRefDefault

Default motor torque reference. This value represents actually the Iq current reference expressed in digit.

Definition at line 89 of file SpeednTorqCtrlClass.h.

5.63.2.11 int16_t SpeednTorqCtrlParams_t::hldrefDefault

Default Id current reference expressed in digit.

Definition at line 93 of file SpeednTorqCtrlClass.h.

5.64 STOParams_t Struct Reference

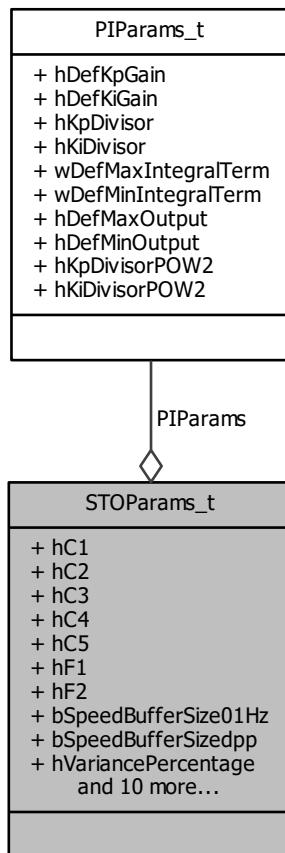
5.64.1 Detailed Description

STO class parameters definition.

Definition at line 64 of file STO_SpeednPosFdbkClass.h.

```
#include <STO_SpeednPosFdbkClass.h>
```

Collaboration diagram for STOParams_t:



Data Fields

- int16_t hC1
- int16_t hC2

- int16_t hC3
- int16_t hC4
- int16_t hC5
- int16_t hF1
- int16_t hF2
- PIParams_t PIParams
- uint8_t bSpeedBufferSize01Hz
- uint8_t bSpeedBufferSizedpp
- uint16_t hVariancePercentage
- uint8_t bSpeedValidationBand_H
- uint8_t bSpeedValidationBand_L
- uint16_t hMinStartUpValidSpeed
- uint8_t bStartUpConsistThreshold
- uint8_t bReliability_hysteresis
- uint8_t bBemfConsistencyCheck
- uint8_t bBemfConsistencyGain
- uint16_t hMaxAppPositiveMecSpeed01Hz
- uint16_t hF1LOG
- uint16_t hF2LOG
- uint16_t bSpeedBufferSizedppLOG

5.64.2 Field Documentation

5.64.2.1 int16_t STOParams_t::hC1

State observer constant C1, it can be computed as $F1 * Rs(\text{ohm}) / (Ls[\text{H}] * \text{State observer execution rate [Hz]})$

Definition at line 66 of file STO_SpeednPosFdbkClass.h.

5.64.2.2 int16_t STOParams_t::hC2

Variable containing state observer constant C2, it can be computed as $F1 * K1 / \text{State observer execution rate [Hz]}$ being K1 one of the two observer gains

Definition at line 69 of file STO_SpeednPosFdbkClass.h.

5.64.2.3 int16_t STOParams_t::hC3

State observer constant C3, it can be computed as $F1 * \text{Max application speed [rpm]} * \text{Motor B-emf constant [Vllrms/kgpm]} * \sqrt{2} / (Ls [\text{H}] * \text{max measurable current (A)} * \text{State observer execution rate [Hz]})$

Definition at line 74 of file STO_SpeednPosFdbkClass.h.

5.64.2.4 int16_t STOParams_t::hC4

State Observer constant C4, it can be computed as $K2 * \text{max measurable current (A)} / (\text{Max application speed [rpm]} * \text{Motor B-emf constant [Vllrms/kgpm]} * \sqrt{2} * F2 * \text{State observer execution rate [Hz]})$ being K2 one of the two observer gains

Definition at line 80 of file STO_SpeednPosFdbkClass.h.

5.64.2.5 int16_t STOParams_t::hC5

State Observer constant C5, it can be computed as $F1 * \text{max measurable voltage} / (2 * Ls [\text{Hz}] * \text{max measurable current} * \text{State observer execution rate [Hz]})$

Definition at line 87 of file STO_SpeednPosFdbkClass.h.

5.64.2.6 int16_t STOParams_t::hF1

State observer scaling factor F1

Definition at line 92 of file STO_SpeednPosFdbkClass.h.

5.64.2.7 int16_t STOParams_t::hF2

State observer scaling factor F2

Definition at line 93 of file STO_SpeednPosFdbkClass.h.

5.64.2.8 PIParams_t STOParams_t::PIParams

It contains the parameters of the PI object necessary for PLL implementation

Definition at line 94 of file STO_SpeednPosFdbkClass.h.

5.64.2.9 uint8_t STOParams_t::bSpeedBufferSize01Hz

Depth of FIFO used to average estimated speed exported by SPD_GetAvrgMecSpeed01Hz. It must be an integer number between 1 and 64

Definition at line 97 of file STO_SpeednPosFdbkClass.h.

5.64.2.10 uint8_t STOParams_t::bSpeedBufferSizedpp

Depth of FIFO used for both averaging estimated speed exported by SPD_GetElSpeedDpp and state observer equations. It must be an integer number between 1 and bSpeedBufferSize01Hz

Definition at line 102 of file STO_SpeednPosFdbkClass.h.

5.64.2.11 uint16_t STOParams_t::hVariancePercentage

Parameter expressing the maximum allowed variance of speed estimation

Definition at line 108 of file STO_SpeednPosFdbkClass.h.

5.64.2.12 uint8_t STOParams_t::bSpeedValidationBand_H

It expresses how much estimated speed can exceed forced stator electrical frequency during start-up without being considered wrong. The measurement unit is 1/16 of forced speed

Definition at line 111 of file STO_SpeednPosFdbkClass.h.

5.64.2.13 uint8_t STOParams_t::bSpeedValidationBand_L

It expresses how much estimated speed can be below forced stator electrical frequency during start-up without being considered wrong. The measurement unit is 1/16 of forced speed

Definition at line 117 of file STO_SpeednPosFdbkClass.h.

5.64.2.14 uint16_t STOParams_t::hMinStartUpValidSpeed

Minimum mechanical speed (expressed in 01Hz required to validate the start-up

Definition at line 123 of file STO_SpeednPosFdbkClass.h.

5.64.2.15 uint8_t STOParams_t::bStartUpConsistThreshold

Number of consecutive tests on speed consistency to be passed before validating the start-up

Definition at line 126 of file STO_SpeednPosFdbkClass.h.

5.64.2.16 uint8_t STOParams_t::bReliability_hysteresys

Number of reliability failed consecutive tests before a speed check fault is returned to base class

Definition at line 129 of file STO_SpeednPosFdbkClass.h.

5.64.2.17 uint8_t STOParams_t::bBemfConsistencyCheck

Degree of consistency of the observed back-emfs, it must be an integer number ranging from 1 (very high consistency) down to 64 (very poor consistency)

Definition at line 133 of file STO_SpeednPosFdbkClass.h.

5.64.2.18 uint8_t STOParams_t::bBemfConsistencyGain

Gain to be applied when checking back-emfs consistency; default value is 64 (neutral), max value 105 (x1.64 amplification), min value 1 (/64 attenuation)

Definition at line 138 of file STO_SpeednPosFdbkClass.h.

5.64.2.19 uint16_t STOParams_t::hMaxAppPositiveMecSpeed01Hz

Maximum positive value of rotor speed. It's expressed in tenth of mechanical Hertz. It can be x1.1 greater than max application speed

Definition at line 143 of file STO_SpeednPosFdbkClass.h.

5.64.2.20 uint16_t STOParams_t::hF1LOG

F1 gain divisor expressed as power of 2. E.g. if gain divisor is 512 the value must be 9 because $2^9 = 512$

Definition at line 148 of file STO_SpeednPosFdbkClass.h.

5.64.2.21 uint16_t STOParams_t::hF2LOG

F2 gain divisor expressed as power of 2. E.g. if gain divisor is 512 the value must be 9 because $2^9 = 512$

Definition at line 151 of file STO_SpeednPosFdbkClass.h.

5.64.2.22 uint16_t STOParams_t::bSpeedBufferSizedppLOG

bSpeedBufferSizedpp expressed as power of 2. E.g. if gain divisor is 512 the value must be 9 because $2^9 = 512$

Definition at line 154 of file STO_SpeednPosFdbkClass.h.

5.65 TempSensorParams_t Struct Reference

5.65.1 Detailed Description

Temperature sensor class parameters definition.

Definition at line 58 of file TemperatureSensorClass.h.

```
#include <TemperatureSensorClass.h>
```

Data Fields

- [SensorType_t bSensorType](#)

5.65.2 Field Documentation

5.65.2.1 SensorType_t TempSensorParams_t::bSensorType

It contains the information about the type of instanced temperature sensor object. It can be equal to REAL_SENSOR or VIRTUAL_SENSOR

Definition at line 60 of file TemperatureSensorClass.h.

5.66 Vars_t Struct Reference

5.66.1 Detailed Description

MCInterface class members definition.

UserInterface class members definition.

BusVoltageSensor class members definition.

TemperatureSensor class members definition.

StateMachine class members definition.

SpeednTorqCtrl class members definition.

SpeednPosFdbk class members definition.

RevupCtrl class members definition.

PWMnCurrFdbk class members definition.

FluxWeakeningCtrl class members definition.

FeedForwardCtrl class members definition.

PI regulator class members definition.

OpenLoop class members definition.

MotorPowerMeasurement class members definition.

EncAlignCtrl class members definition.

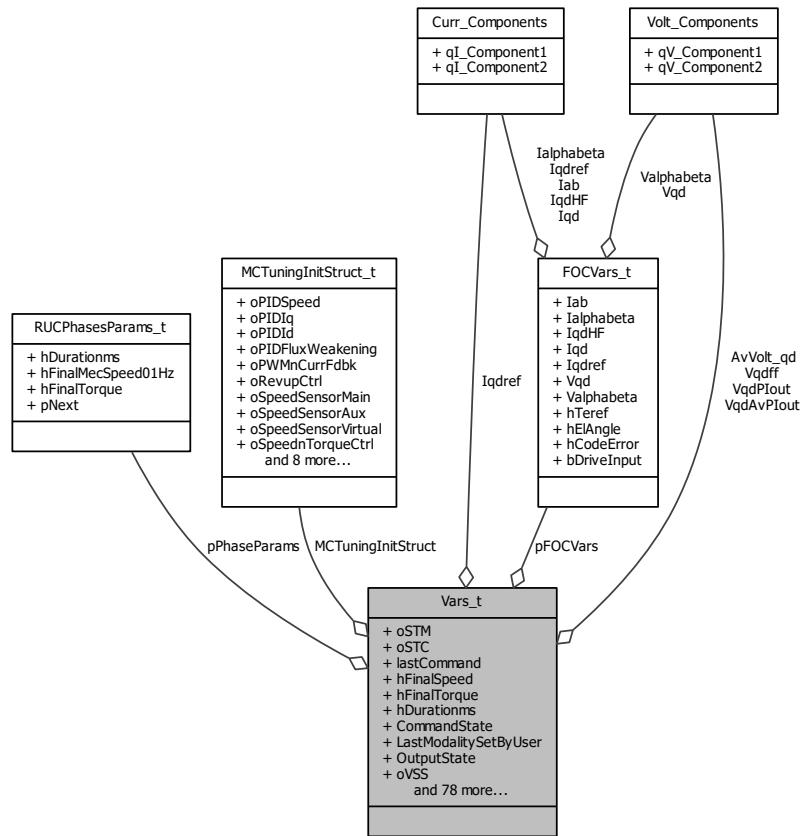
DigitalOutput class members definition.

MCTuning class members definition.

Definition at line 50 of file MCInterfacePrivate.h.

```
#include <MCInterfacePrivate.h>
```

Collaboration diagram for Vars_t:



Data Fields

- `CSTM oSTM`
- `CSTC oSTC`
- `pFOCVars_t pFOCVars`
- `UserCommands_t lastCommand`
- `int16_t hFinalSpeed`
- `int16_t hFinalTorque`
- `Curr_Components Iqref`
- `uint16_t hDurationms`
- `CommandState_t CommandState`
- `STC_Modality_t LastModalitySetByUser`
- `CVSS_SPD oVSS`
- `CENC_SPD oENC`
- `uint16_t hRemainingTicks`
- `bool EncAligned`
- `bool EncRestart`
- `int16_t hMeasBuffer [MPM_BUFFER_LENGTH]`
- `uint16_t hNextMeasBufferIndex`
- `uint16_t hLastMeasBufferIndex`
- `int16_t hAvgElMotorPowerW`
- `int16_t hVoltage`

- **CSPD oVSS**
- bool **VFMode**
- **Volt_Components Vqdff**
- **Volt_Components VqdPlout**
- **Volt_Components VqdAvPlout**
- int32_t **wConstant_1D**
- int32_t **wConstant_1Q**
- int32_t **wConstant_2**
- **CVBS oVBS**
- **CPI oPI_q**
- **CPI oPI_d**
- **CPI oFluxWeakeningPI**
- **CPI oSpeedPI**
- uint16_t **hFW_V_Ref**
- **Volt_Components AvVolt_qd**
- int16_t **AvVoltAmpl**
- int16_t **hldRefOffset**
- uint16_t **hT_Sqrt3**
- uint16_t **hSector**
- bool **bTurnOnLowSidesAction**
- uint16_t **hOffCalibrWaitTimeCounter**
- uint8_t **bMotor**
- uint16_t **hPhaseRemainingTicks**
- **pRUCPhasesParams_t pPhaseParams**
- int16_t **hDirection**
- uint8_t **bStageCnt**
- **RUCPhasesData_t ParamsData [MAX_PHASE_NUMBER]**
- uint8_t **bPhaseNbr**
- **STC_Modality_t bMode**
- int16_t **hTargetFinal**
- int32_t **wSpeedRef01HzExt**
- int32_t **wTorqueRef**
- uint16_t **hRampRemainingStep**
- **CPI oPISpeed**
- **CSPD oSPD**
- int32_t **wIncDecAmount**
- **State_t bState**
- uint16_t **hFaultNow**
- uint16_t **hFaultOccurred**
- uint16_t **hFaultState**
- uint16_t **hLatestConv**
- uint8_t **bDriveNum**
- **CMCI * pMCI**
- **CMCT * pMCT**
- uint32_t * **pUICfg**
- uint8_t **bSelectedDrive**

5.66.2 Field Documentation

5.66.2.1 CSTM Vars_t::oSTM

State machine object used by MCI.

Definition at line 52 of file MCInterfacePrivate.h.

Referenced by MCI_Init().

5.66.2.2 **CSTC Vars_t::oSTC**

Speed and torque controller object used by MCI.

Speed and torque controller object used by EAC.

Speed and torque controller object used by RUC.

Definition at line 53 of file MCInterfacePrivate.h.

Referenced by EAC_Init(), EAC_StartAlignment(), MCI_Clear_Iqdref(), MCI_ExecBufferedCommands(), MCI_GetAvrgMecSpeed01Hz(), MCI_GetMecSpeedRef01Hz(), MCI_GetSpdSensorReliability(), MCI_Init(), MCI_RampCompleted(), RUC_Clear(), RUC_Exec(), and RUC_Init().

5.66.2.3 **pFOCVars_t Vars_t::pFOCVars**

Pointer to FOC vars used by MCI.

Definition at line 54 of file MCInterfacePrivate.h.

Referenced by MCI_Clear_Iqdref(), MCI_ExecBufferedCommands(), MCI_GetElAngledpp(), MCI_Getlab(), MCI_Getlalphabet(), MCI_Getlqd(), MCI_GetlqdHF(), MCI_Getlqdref(), MCI_GetPhaseCurrentAmplitude(), MCI_GetPhaseVoltageAmplitude(), MCI_GetTeref(), MCI_GetValphabet(), MCI_GetVqd(), MCI_Init(), and MCI_SetIdref().

5.66.2.4 **UserCommands_t Vars_t::lastCommand**

Last command coming from the user.

Definition at line 56 of file MCInterfacePrivate.h.

Referenced by MCI_ExecBufferedCommands(), MCI_ExecSpeedRamp(), MCI_ExecTorqueRamp(), MCI_GetImposedMotorDirection(), MCI_GetLastRampFinalSpeed(), MCI_Init(), and MCI_SetCurrentReferences().

5.66.2.5 **int16_t Vars_t::hFinalSpeed**

Final speed of last ExecSpeedRamp command.

Definition at line 57 of file MCInterfacePrivate.h.

Referenced by MCI_ExecBufferedCommands(), MCI_ExecSpeedRamp(), MCI_GetImposedMotorDirection(), MCI_GetLastRampFinalSpeed(), and MCI_Init().

5.66.2.6 **int16_t Vars_t::hFinalTorque**

Final torque of last ExecTorqueRamp command.

Definition at line 58 of file MCInterfacePrivate.h.

Referenced by MCI_ExecBufferedCommands(), MCI_ExecTorqueRamp(), MCI_GetImposedMotorDirection(), and MCI_Init().

5.66.2.7 **Curr_Components Vars_t::lqdref**

Current component of last SetCurrentReferences command.

Definition at line 60 of file MCInterfacePrivate.h.

Referenced by MCI_ExecBufferedCommands(), MCI_GetImposedMotorDirection(), and MCI_SetCurrentReferences().

5.66.2.8 `uint16_t Vars_t::hDurationms`

Duration in ms of last ExecSpeedRamp or ExecTorqueRamp command.

Definition at line 62 of file MCInterfacePrivate.h.

Referenced by MCI_ExecBufferedCommands(), MCI_ExecSpeedRamp(), MCI_ExecTorqueRamp(), and MCI_Init().

5.66.2.9 `CommandState_t Vars_t::CommandState`

The status of the buffered command.

Definition at line 65 of file MCInterfacePrivate.h.

Referenced by MCI_ExecBufferedCommands(), MCI_ExecSpeedRamp(), MCI_ExecTorqueRamp(), MCI_Init(), MCI_IsCommandAcknowledged(), MCI_SetCurrentReferences(), and MCI_StartMotor().

5.66.2.10 `STC_Modality_t Vars_t::LastModalitySetByUser`

The last STC_Modality_t set by the user.

Definition at line 66 of file MCInterfacePrivate.h.

Referenced by MCI_ExecSpeedRamp(), MCI_ExecTorqueRamp(), MCI_GetControlMode(), and MCI_SetCurrentReferences().

5.66.2.11 `CVSS_SPD Vars_t::oVSS`

Virtual speed sensor object used by EAC.

Virtual speed sensor object used by RUC.

Definition at line 53 of file EncAlignCtrlPrivate.h.

Referenced by EAC_Init(), EAC_StartAlignment(), OL_Calc(), OL_Init(), RUC_Clear(), RUC_Exec(), and RUC_Init().

5.66.2.12 `CENC_SPD Vars_t::oENC`

Encoder object used by EAC.

Definition at line 54 of file EncAlignCtrlPrivate.h.

Referenced by EAC_Exec(), and EAC_Init().

5.66.2.13 `uint16_t Vars_t::hRemainingTicks`

Number of clock events remaining to complete the alignment.

Definition at line 55 of file EncAlignCtrlPrivate.h.

Referenced by EAC_Exec(), and EAC_StartAlignment().

5.66.2.14 `bool Vars_t::EncAligned`

This flag is TRUE if the encoder has been aligned at least one time, FALSE if hasn't been never aligned.

Definition at line 57 of file EncAlignCtrlPrivate.h.

Referenced by EAC_Exec(), EAC_Init(), and EAC_IsAligned().

5.66.2.15 bool Vars_t::EncRestart

This flag is used to force a restart of the motor after the encoder alignment. It is TRUE if a restart is programmed else FALSE

Definition at line 60 of file EncAlignCtrlPrivate.h.

Referenced by EAC_GetRestartState(), EAC_Init(), and EAC_SetRestartState().

5.66.2.16 int16_t Vars_t::hMeasBuffer[MPM_BUFFER_LENGTH]

Buffer used by MPM object to store the instantaneous measurements of motor power.

Definition at line 62 of file MotorPowerMeasurementPrivate.h.

Referenced by MPM_CalcElMotorPower(), MPM_Clear(), and MPM_GetElMotorPowerW().

5.66.2.17 uint16_t Vars_t::hNextMeasBufferIndex

Index of the buffer that will contain the next motor power measurement.

Definition at line 65 of file MotorPowerMeasurementPrivate.h.

Referenced by MPM_CalcElMotorPower(), and MPM_Clear().

5.66.2.18 uint16_t Vars_t::hLastMeasBufferIndex

Index of the buffer that contains the last motor power measurement.

Definition at line 67 of file MotorPowerMeasurementPrivate.h.

Referenced by MPM_CalcElMotorPower(), MPM_Clear(), and MPM_GetElMotorPowerW().

5.66.2.19 int16_t Vars_t::hAvrgElMotorPowerW

The average measured motor power expressed in watt.

Definition at line 69 of file MotorPowerMeasurementPrivate.h.

Referenced by MPM_CalcElMotorPower(), and MPM_GetAvrgElMotorPowerW().

5.66.2.20 int16_t Vars_t::hVoltage

Open loop voltage to be applied

Definition at line 51 of file OpenLoopPrivate.h.

Referenced by OL_Calc(), OL_Init(), OL_UpdateVoltage(), and OL_VqdConditioning().

5.66.2.21 CSPD Vars_t::oVSS

Related VSS object used

Definition at line 52 of file OpenLoopPrivate.h.

5.66.2.22 bool Vars_t::VFMode

TRUE to enable V/F mode otherwise FALSE

Definition at line 53 of file OpenLoopPrivate.h.

Referenced by OL_Calc(), OL_Init(), and OL_VF().

5.66.2.23 Volt_Components Vars_t::Vqdff

Feed-forward Iqd controller contribution to Vqd

Definition at line 49 of file FeedForwardCtrlPrivate.h.

5.66.2.24 Volt_Components Vars_t::VqdPlout

Vqd as output by PI controller

Definition at line 51 of file FeedForwardCtrlPrivate.h.

5.66.2.25 Volt_Components Vars_t::VqdAvPlout

Averaged Vqd as output by PI controller

Definition at line 52 of file FeedForwardCtrlPrivate.h.

5.66.2.26 int32_t Vars_t::wConstant_1D

Feed forward default constant for d axes

Definition at line 53 of file FeedForwardCtrlPrivate.h.

5.66.2.27 int32_t Vars_t::wConstant_1Q

Feed forward default constant for q axes

Definition at line 54 of file FeedForwardCtrlPrivate.h.

5.66.2.28 int32_t Vars_t::wConstant_2

Default constant value used by Feed-Forward algorithm

Definition at line 55 of file FeedForwardCtrlPrivate.h.

5.66.2.29 CVBS Vars_t::oVBS

Related bus voltage sensor

Definition at line 57 of file FeedForwardCtrlPrivate.h.

5.66.2.30 CPI Vars_t::oPI_q

Related PI for Iq regulator

Definition at line 58 of file FeedForwardCtrlPrivate.h.

5.66.2.31 CPI Vars_t::oPI_d

Related PI for Id regulator

Definition at line 59 of file FeedForwardCtrlPrivate.h.

5.66.2.32 CPI Vars_t::oFluxWeakeningPI

PI object used for flux weakening

Definition at line 51 of file FluxWeakeningCtrlPrivate.h.

Referenced by FW_CalcCurrRef(), FW_Clear(), and FW_Init().

5.66.2.33 CPI Vars_t::oSpeedPI

PI object used for speed control

Definition at line 52 of file FluxWeakeningCtrlPrivate.h.

Referenced by FW_CalcCurrRef(), and FW_Init().

5.66.2.34 uint16_t Vars_t::hFW_V_Ref

Voltage reference, tenth of percentage points

Definition at line 53 of file FluxWeakeningCtrlPrivate.h.

Referenced by FW_CalcCurrRef(), FW_GetVref(), FW_Init(), and FW_SetVref().

5.66.2.35 Volt_Components Vars_t::AvVolt_qd

Average stator voltage in qd reference frame

Definition at line 55 of file FluxWeakeningCtrlPrivate.h.

Referenced by FW_CalcCurrRef(), FW_Clear(), and FW_DataProcess().

5.66.2.36 int16_t Vars_t::AvVoltAmpl

Average stator voltage amplitude

Definition at line 57 of file FluxWeakeningCtrlPrivate.h.

Referenced by FW_CalcCurrRef(), FW_Clear(), FW_GetAvVAmplitude(), and FW_GetAvVPercentage().

5.66.2.37 int16_t Vars_t::hIdRefOffset

Id reference offset

Definition at line 58 of file FluxWeakeningCtrlPrivate.h.

Referenced by FW_CalcCurrRef(), and FW_Clear().

5.66.2.38 uint16_t Vars_t::hT_Sqrt3

Contains a constant utilized by SVPWM algorithm

Definition at line 64 of file PWMnCurrFdbkPrivate.h.

Referenced by PWMC_Init(), and PWMC_SetPhaseVoltage().

5.66.2.39 uint16_t Vars_t::hSector

Contains the space vector sector number

Definition at line 65 of file PWMnCurrFdbkPrivate.h.

Referenced by PWMC_SetPhaseVoltage().

5.66.2.40 bool Vars_t::bTurnOnLowSidesAction

TRUE if TurnOnLowSides action is active, FALSE otherwise.

Definition at line 70 of file PWMnCurrFdbkPrivate.h.

Referenced by PWMC_GetTurnOnLowSidesAction(), PWMC_Init(), PWMC_SwitchOffPWM(), PWMC_SwitchOnPWM(), and PWMC_TurnOnLowSides().

5.66.2.41 uint16_t Vars_t::hOffCalibrWaitTimeCounter

Counter to wait fixed time before motor current measurement offset calibration.

Definition at line 72 of file PWMnCurrFdbkPrivate.h.

Referenced by PWMC_CurrentReadingCalibr().

5.66.2.42 uint8_t Vars_t::bMotor

Motor reference number

Definition at line 75 of file PWMnCurrFdbkPrivate.h.

5.66.2.43 uint16_t Vars_t::hPhaseRemainingTicks

Number of clock events remaining to complete the phase.

Definition at line 77 of file RevupCtrlPrivate.h.

Referenced by RUC_Clear(), and RUC_Exec().

5.66.2.44 pRUCPhasesParams_t Vars_t::pPhaseParams

Pointer to parameter of the current executed rev up phase.

Definition at line 79 of file RevupCtrlPrivate.h.

Referenced by RUC_Clear(), and RUC_Exec().

5.66.2.45 int16_t Vars_t::hDirection

If it is "1" the programmed revup sequence is performed. If it is "-1" the revup sequence is performed with opposite values of targets (speed, torque).

Definition at line 81 of file RevupCtrlPrivate.h.

Referenced by RUC_Clear(), and RUC_Exec().

5.66.2.46 uint8_t Vars_t::bStageCnt

It counts the rev up stages that have been already started. NOTE: The rev up stage are zero-based indexed so that the fist stage is the number zero.

Definition at line 85 of file RevupCtrlPrivate.h.

Referenced by RUC_Clear(), RUC_Exec(), and RUC_FirstAccelerationStageReached().

5.66.2.47 RUCPhasesData_t Vars_t::ParamsData[MAX_PHASE_NUMBER]

Rev up phases data. It is initialized equal to rev up phases parameters but can be changed in run-time using tuning methods.

Definition at line 93 of file RevupCtrlPrivate.h.

Referenced by RUC_Clear(), RUC_GetPhaseDurationms(), RUC_GetPhaseFinalMecSpeed01Hz(), RUC_GetPhaseFinalTorque(), RUC_Init(), RUC_SetPhaseDurationms(), RUC_SetPhaseFinalMecSpeed01Hz(), and RUC_SetPhaseFinalTorque().

5.66.2.48 uint8_t Vars_t::bPhaseNbr

Number of phases relative to the programmed rev up sequence.

Definition at line 97 of file RevupCtrlPrivate.h.

Referenced by RUC_GetNumberOfPhases(), and RUC_Init().

5.66.2.49 STC_Modality_t Vars_t::bMode

Modality of STC. It can be one of these two settings: STC_TORQUE_MODE to enable the Torque mode or STC_SPEED_MODE to enable the Speed mode.

Definition at line 51 of file SpeednTorqCtrlPrivate.h.

Referenced by STC_CalcTorqueReference(), STC_Clear(), STC_ExecRamp(), STC_GetControlMode(), STC_Init(), and STC_SetControlMode().

5.66.2.50 int16_t Vars_t::hTargetFinal

Backup of hTargetFinal to be applied in the last step.

Definition at line 55 of file SpeednTorqCtrlPrivate.h.

Referenced by STC_CalcTorqueReference(), STC_ExecRamp(), and STC_Init().

5.66.2.51 int32_t Vars_t::wSpeedRef01HzExt

Current mechanical rotor speed reference expressed in tenths of HZ multiplied by

1.

Definition at line 57 of file SpeednTorqCtrlPrivate.h.

Referenced by STC_CalcTorqueReference(), STC_ExecRamp(), and STC_Init().

5.66.2.52 int32_t Vars_t::wTorqueRef

Current motor torque reference. This value represents actually the Iq current expressed in digit multiplied by 65536.

Definition at line 60 of file SpeednTorqCtrlPrivate.h.

Referenced by STC_CalcTorqueReference(), STC_ExecRamp(), and STC_Init().

5.66.2.53 uint16_t Vars_t::hRampRemainingStep

Number of steps remaining to complete the ramp.

Definition at line 63 of file SpeednTorqCtrlPrivate.h.

Referenced by STC_CalcTorqueReference(), STC_ExecRamp(), STC_Init(), STC_RampCompleted(), STC_SetControlMode(), and STC_StopRamp().

5.66.2.54 CPI Vars_t::oPISpeed

The regulator used to perform the speed control loop.

Definition at line 65 of file SpeednTorqCtrlPrivate.h.

Referenced by STC_CalcTorqueReference(), and STC_Init().

5.66.2.55 CSPD Vars_t::oSPD

The speed sensor used to perform the speed regulation.

Definition at line 67 of file SpeednTorqCtrlPrivate.h.

Referenced by STC_CalcTorqueReference(), STC_ExecRamp(), STC_GetSpeedSensor(), STC_Init(), and STC_SetSpeedSensor().

5.66.2.56 int32_t Vars_t::wIncDecAmount

Increment/decrement amount to be applied to the reference value at each CalcTorqueReference.

Definition at line 69 of file SpeednTorqCtrlPrivate.h.

Referenced by STC_CalcTorqueReference(), STC_ExecRamp(), STC_Init(), and STC_StopRamp().

5.66.2.57 State_t Vars_t::bState

Variable containing state machine current state

Definition at line 51 of file StateMachinePrivate.h.

Referenced by STM_FaultAcknowledged(), STM_FaultProcessing(), STM_Init(), and STM_NextState().

5.66.2.58 uint16_t Vars_t::hFaultNow

Bit fields variable containing faults currently present

Definition at line 53 of file StateMachinePrivate.h.

Referenced by STM_FaultProcessing(), STM_GetFaultState(), and STM_Init().

5.66.2.59 uint16_t Vars_t::hFaultOccurred

Bit fields variable containing faults historically occurred since the state machine has been moved to FAULT_NOW state

Definition at line 55 of file StateMachinePrivate.h.

Referenced by STM_FaultAcknowledged(), STM_FaultProcessing(), STM_GetFaultState(), and STM_Init().

5.66.2.60 uint16_t Vars_t::hFaultState

It contains latest available average Vbus expressed in u16Celsius

It contains latest available average Vbus expressed in digit

Definition at line 53 of file TemperatureSensorPrivate.h.

5.66.2.61 uint16_t Vars_t::hLatestConv

It contains latest Vbus converted value expressed in u16Volts format

Definition at line 51 of file BusVoltageSensorPrivate.h.

5.66.2.62 uint8_t Vars_t::bDriveNum

Total number of MC objects.

Definition at line 52 of file UserInterfacePrivate.h.

Referenced by UI_Init(), and UI_SelectMC().

5.66.2.63 CMCI* Vars_t::pMCi

Pointer of MC interface list.

Definition at line 53 of file UserInterfacePrivate.h.

Referenced by UI_ExecCmd(), UI_ExecSpeedRamp(), UI_ExecTorqueRamp(), UI_GetReg(), UI_Init(), UI_SetCurrentReferences(), and UI_SetReg().

5.66.2.64 CMCT* Vars_t::pMCT

Pointer of MC tuning list.

Definition at line 54 of file UserInterfacePrivate.h.

Referenced by UI_GetCurrentMCT(), UI_GetReg(), UI_GetRevupData(), UI_Init(), UI_SetReg(), and UI_SetRevupData().

5.66.2.65 uint32_t* Vars_t::pUICfg

Pointer of UI configuration list.

Definition at line 55 of file UserInterfacePrivate.h.

Referenced by UI_ExecCmd(), UI_GetReg(), UI_GetSelectedMCConfig(), UI_Init(), and UI_SetReg().

5.66.2.66 uint8_t Vars_t::bSelectedDrive

Current selected MC object in the list.

Definition at line 56 of file UserInterfacePrivate.h.

Referenced by UFC_Init(), UI_ExecCmd(), UI_ExecSpeedRamp(), UI_ExecTorqueRamp(), UI_GetCurrentMC(), UI_GetReg(), UI_GetRevupData(), UI_GetSelectedMCConfig(), UI_Init(), UI_SelectMC(), UI_SetCurrentReferences(), UI_SetReg(), and UI_SetRevupData().

5.67 VirtualParams_t Struct Reference

5.67.1 Detailed Description

Virtual Vbus sensor class parameters definition.

Definition at line 54 of file Virtual_BusVoltageSensorClass.h.

```
#include <Virtual_BusVoltageSensorClass.h>
```

Data Fields

- uint16_t `hExpectedVbus_d`

5.67.2 Field Documentation

5.67.2.1 uint16_t `VirtualParams_t::hExpectedVbus_d`

Expected Vbus voltage expressed in digital value hOverVoltageThreshold(digital value)= Over Voltage Threshold (V) * 65536 / 500

Definition at line 56 of file `Virtual_BusVoltageSensorClass.h`.

5.68 VirtualSpeedSensorParams_t Struct Reference

5.68.1 Detailed Description

`VirtualSpeedSensor` class parameters definition.

Definition at line 54 of file `VirtualSpeedSensor_SpeednPosFdbkClass.h`.

```
#include <VirtualSpeedSensor_SpeednPosFdbkClass.h>
```

Data Fields

- uint16_t `hSpeedSamplingFreqHz`

5.68.2 Field Documentation

5.68.2.1 uint16_t `VirtualSpeedSensorParams_t::hSpeedSamplingFreqHz`

Frequency (Hz) at which motor speed is to be computed. It must be equal to the frequency at which function `SPD_CalcAvgMecSpeed01Hz` is called.

Definition at line 56 of file `VirtualSpeedSensor_SpeednPosFdbkClass.h`.

5.69 VirtualITParams_t Struct Reference

5.69.1 Detailed Description

`Virtual temperature sensor` class parameters definition.

Definition at line 54 of file `Virtual_TemperatureSensorClass.h`.

```
#include <Virtual_TemperatureSensorClass.h>
```

Data Fields

- uint16_t `hExpectedTemp_d`
- int16_t `hExpectedTemp_C`

5.69.2 Field Documentation

5.69.2.1 uint16_t VirtualTPParams_t::hExpectedTemp_d

Value returned by base class method TSNS_GetAvTemp_d

Definition at line 56 of file Virtual_TemperatureSensorClass.h.

5.69.2.2 int16_t VirtualTPParams_t::hExpectedTemp_C

Expected temperature in degrees (value returned by base class method TSNS_GetAvTemp_C)

Definition at line 58 of file Virtual_TemperatureSensorClass.h.

5.70 _CUI_t Struct Reference

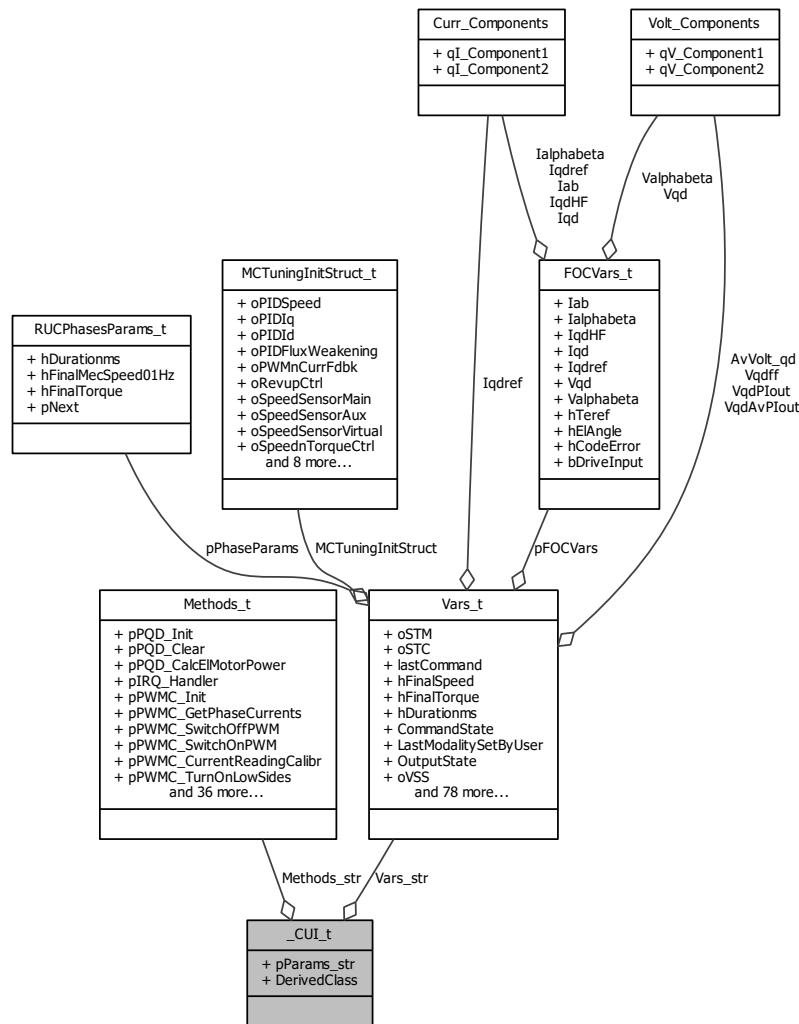
5.70.1 Detailed Description

Private UserInterface class definition.

Definition at line 90 of file UserInterfacePrivate.h.

```
#include <UserInterfacePrivate.h>
```

Collaboration diagram for `_CUI_t`:



Data Fields

- `Methods_t Methods_str`
- `Vars_t Vars_str`
- `pParams_t pParams_str`
- `void * DerivedClass`

5.70.2 Field Documentation

5.70.2.1 `Methods_t _CUI_t::Methods_str`

Virtual methods container

Definition at line 92 of file UserInterfacePrivate.h.

Referenced by `DACT_NewObject()`, and `UFC_NewObject()`.

5.70.2.2 **Vars_t_CUI_t::Vars_str**

Class members container

Definition at line 93 of file UserInterfacePrivate.h.

5.70.2.3 **pParams_t_CUI_t::pParams_str**

Class parameters container

Definition at line 94 of file UserInterfacePrivate.h.

Referenced by UI_NewObject().

5.70.2.4 **void* _CUI_t::DerivedClass**

Pointer to derived class

Definition at line 95 of file UserInterfacePrivate.h.

Referenced by DACT_NewObject(), MCP_NewObject(), and UFC_NewObject().

5.71 CANParams_t Struct Reference

5.71.1 Detailed Description

UserInterface class parameters definition.

Definition at line 71 of file CAN_PhysicalLayerCommunication_Class.h.

```
#include <CAN_PhysicalLayerCommunication_Class.h>
```

5.72 FCPPParams_t Struct Reference

5.72.1 Detailed Description

FrameCommunicationProtocol class parameters definition.

Definition at line 78 of file FrameCommunicationProtocolClass.h.

```
#include <FrameCommunicationProtocolClass.h>
```

5.73 MCPPParams_t Struct Reference

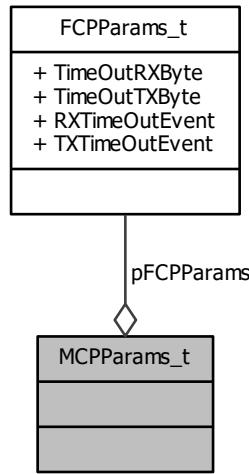
5.73.1 Detailed Description

MotorControlProtocol class parameters definition.

Definition at line 58 of file MotorControlProtocolClass.h.

```
#include <MotorControlProtocolClass.h>
```

Collaboration diagram for MCPParams_t:



5.74 Methods_t Struct Reference

5.74.1 Detailed Description

Virtual methods container.

Definition at line 76 of file MotorPowerMeasurementPrivate.h.

```
#include <MotorPowerMeasurementPrivate.h>
```

5.75 USART_TypeDef Struct Reference

5.75.1 Detailed Description

Universal Synchronous Asynchronous Receiver Transmitter.

Definition at line 71 of file USART_PhysicalLayerCommunication_Class.h.

```
#include <USART_PhysicalLayerCommunication_Class.h>
```

5.76 USARTParams_t Struct Reference

5.76.1 Detailed Description

UserInterface class parameters definition.

Definition at line 122 of file USART_PhysicalLayerCommunication_Class.h.

```
#include <USART_PhysicalLayerCommunication_Class.h>
```

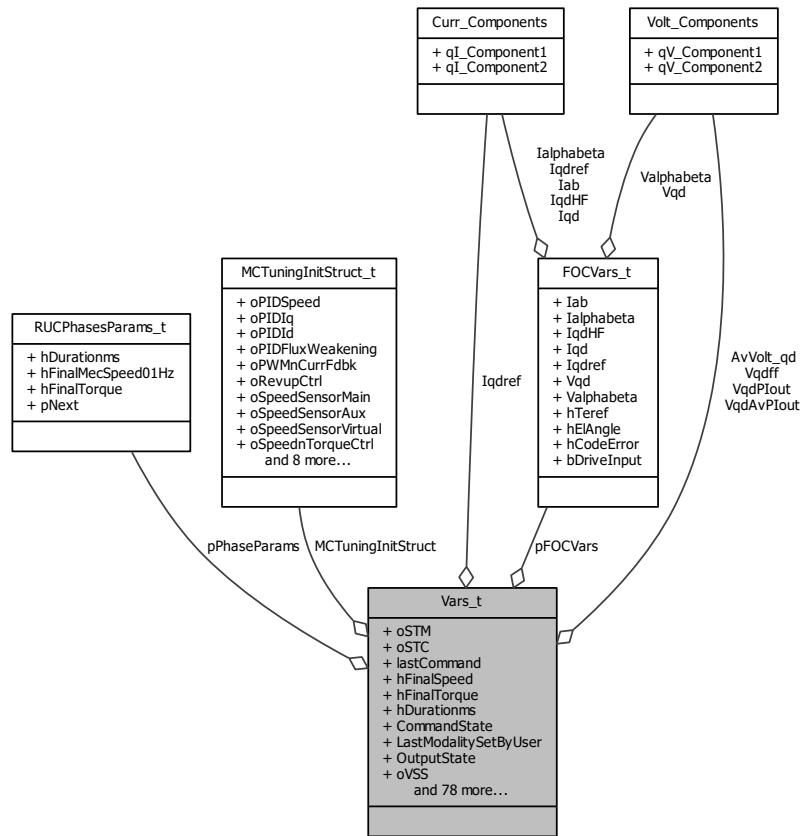
5.77 Vars_t Struct Reference

5.77.1 Detailed Description

MCIInterface class members definition.
UserInterface class members definition.
BusVoltageSensor class members definition.
TemperatureSensor class members definition.
StateMachine class members definition.
SpeednTorqCtrl class members definition.
SpeednPosFdbk class members definition.
RevupCtrl class members definition.
PWMMnCurrFdbk class members definition.
FluxWeakeningCtrl class members definition.
FeedForwardCtrl class members definition.
PI regulator class members definition.
OpenLoop class members definition.
MotorPowerMeasurement class members definition.
EncAlignCtrl class members definition.
DigitalOutput class members definition.
MCTuning class members definition.
Definition at line 50 of file MCInterfacePrivate.h.

```
#include <MCInterfacePrivate.h>
```

Collaboration diagram for Vars_t:



Data Fields

- `CSTM oSTM`
- `CSTC oSTC`
- `pFOCVars_t pFOCVars`
- `UserCommands_t lastCommand`
- `int16_t hFinalSpeed`
- `int16_t hFinalTorque`
- `Curr_Components Iqdref`
- `uint16_t hDurationms`
- `CommandState_t CommandState`
- `STC_Modality_t LastModalitySetByUser`
- `CVSS_SPD oVSS`
- `CENC_SPD oENC`
- `uint16_t hRemainingTicks`
- `bool EncAligned`
- `bool EncRestart`
- `int16_t hMeasBuffer [MPM_BUFFER_LENGTH]`
- `uint16_t hNextMeasBufferIndex`
- `uint16_t hLastMeasBufferIndex`
- `int16_t hAvgElMotorPowerW`
- `int16_t hVoltage`

- **CSPD oVSS**
- bool **VFMode**
- **Volt_Components Vqdff**
- **Volt_Components VqdPlout**
- **Volt_Components VqdAvPlout**
- int32_t **wConstant_1D**
- int32_t **wConstant_1Q**
- int32_t **wConstant_2**
- **CVBS oVBS**
- **CPI oPI_q**
- **CPI oPI_d**
- **CPI oFluxWeakeningPI**
- **CPI oSpeedPI**
- uint16_t **hFW_V_Ref**
- **Volt_Components AvVolt_qd**
- int16_t **AvVoltAmpl**
- int16_t **hldRefOffset**
- uint16_t **hT_Sqrt3**
- uint16_t **hSector**
- bool **bTurnOnLowSidesAction**
- uint16_t **hOffCalibrWaitTimeCounter**
- uint8_t **bMotor**
- uint16_t **hPhaseRemainingTicks**
- **pRUCPhasesParams_t pPhaseParams**
- int16_t **hDirection**
- uint8_t **bStageCnt**
- **RUCPhasesData_t ParamsData [MAX_PHASE_NUMBER]**
- uint8_t **bPhaseNbr**
- **STC_Modality_t bMode**
- int16_t **hTargetFinal**
- int32_t **wSpeedRef01HzExt**
- int32_t **wTorqueRef**
- uint16_t **hRampRemainingStep**
- **CPI oPISpeed**
- **CSPD oSPD**
- int32_t **wIncDecAmount**
- **State_t bState**
- uint16_t **hFaultNow**
- uint16_t **hFaultOccurred**
- uint16_t **hFaultState**
- uint16_t **hLatestConv**
- uint8_t **bDriveNum**
- **CMCI * pMCI**
- **CMCT * pMCT**
- uint32_t * **pUICfg**
- uint8_t **bSelectedDrive**

5.77.2 Field Documentation

5.77.2.1 CSTM Vars_t::oSTM

State machine object used by MCI.

Definition at line 52 of file MCInterfacePrivate.h.

Referenced by MCI_Init().

5.77.2.2 **CSTC Vars_t::oSTC**

Speed and torque controller object used by MCI.

Speed and torque controller object used by EAC.

Speed and torque controller object used by RUC.

Definition at line 53 of file MCInterfacePrivate.h.

Referenced by EAC_Init(), EAC_StartAlignment(), MCI_Clear_Iqdref(), MCI_ExecBufferedCommands(), MCI_GetAvrgMecSpeed01Hz(), MCI_GetMecSpeedRef01Hz(), MCI_GetSpdSensorReliability(), MCI_Init(), MCI_RampCompleted(), RUC_Clear(), RUC_Exec(), and RUC_Init().

5.77.2.3 **pFOCVars_t Vars_t::pFOCVars**

Pointer to FOC vars used by MCI.

Definition at line 54 of file MCInterfacePrivate.h.

Referenced by MCI_Clear_Iqdref(), MCI_ExecBufferedCommands(), MCI_GetElAngledpp(), MCI_Getlab(), MCI_Getlalphabet(), MCI_Getlqd(), MCI_GetlqdHF(), MCI_Getlqdref(), MCI_GetPhaseCurrentAmplitude(), MCI_GetPhaseVoltageAmplitude(), MCI_GetTeref(), MCI_GetValphabet(), MCI_GetVqd(), MCI_Init(), and MCI_SetIdref().

5.77.2.4 **UserCommands_t Vars_t::lastCommand**

Last command coming from the user.

Definition at line 56 of file MCInterfacePrivate.h.

Referenced by MCI_ExecBufferedCommands(), MCI_ExecSpeedRamp(), MCI_ExecTorqueRamp(), MCI_GetImposedMotorDirection(), MCI_GetLastRampFinalSpeed(), MCI_Init(), and MCI_SetCurrentReferences().

5.77.2.5 **int16_t Vars_t::hFinalSpeed**

Final speed of last ExecSpeedRamp command.

Definition at line 57 of file MCInterfacePrivate.h.

Referenced by MCI_ExecBufferedCommands(), MCI_ExecSpeedRamp(), MCI_GetImposedMotorDirection(), MCI_GetLastRampFinalSpeed(), and MCI_Init().

5.77.2.6 **int16_t Vars_t::hFinalTorque**

Final torque of last ExecTorqueRamp command.

Definition at line 58 of file MCInterfacePrivate.h.

Referenced by MCI_ExecBufferedCommands(), MCI_ExecTorqueRamp(), MCI_GetImposedMotorDirection(), and MCI_Init().

5.77.2.7 **Curr_Components Vars_t::lqdref**

Current component of last SetCurrentReferences command.

Definition at line 60 of file MCInterfacePrivate.h.

Referenced by MCI_ExecBufferedCommands(), MCI_GetImposedMotorDirection(), and MCI_SetCurrentReferences().

5.77.2.8 `uint16_t Vars_t::hDurationms`

Duration in ms of last ExecSpeedRamp or ExecTorqueRamp command.

Definition at line 62 of file MCInterfacePrivate.h.

Referenced by MCI_ExecBufferedCommands(), MCI_ExecSpeedRamp(), MCI_ExecTorqueRamp(), and MCI_Init().

5.77.2.9 `CommandState_t Vars_t::CommandState`

The status of the buffered command.

Definition at line 65 of file MCInterfacePrivate.h.

Referenced by MCI_ExecBufferedCommands(), MCI_ExecSpeedRamp(), MCI_ExecTorqueRamp(), MCI_Init(), MCI_IsCommandAcknowledged(), MCI_SetCurrentReferences(), and MCI_StartMotor().

5.77.2.10 `STC_Modality_t Vars_t::LastModalitySetByUser`

The last STC_Modality_t set by the user.

Definition at line 66 of file MCInterfacePrivate.h.

Referenced by MCI_ExecSpeedRamp(), MCI_ExecTorqueRamp(), MCI_GetControlMode(), and MCI_SetCurrentReferences().

5.77.2.11 `CVSS_SPD Vars_t::oVSS`

Virtual speed sensor object used by EAC.

Virtual speed sensor object used by RUC.

Definition at line 53 of file EncAlignCtrlPrivate.h.

Referenced by EAC_Init(), EAC_StartAlignment(), OL_Calc(), OL_Init(), RUC_Clear(), RUC_Exec(), and RUC_Init().

5.77.2.12 `CENC_SPD Vars_t::oENC`

Encoder object used by EAC.

Definition at line 54 of file EncAlignCtrlPrivate.h.

Referenced by EAC_Exec(), and EAC_Init().

5.77.2.13 `uint16_t Vars_t::hRemainingTicks`

Number of clock events remaining to complete the alignment.

Definition at line 55 of file EncAlignCtrlPrivate.h.

Referenced by EAC_Exec(), and EAC_StartAlignment().

5.77.2.14 `bool Vars_t::EncAligned`

This flag is TRUE if the encoder has been aligned at least one time, FALSE if hasn't been never aligned.

Definition at line 57 of file EncAlignCtrlPrivate.h.

Referenced by EAC_Exec(), EAC_Init(), and EAC_IsAligned().

5.77.2.15 bool Vars_t::EncRestart

This flag is used to force a restart of the motor after the encoder alignment. It is TRUE if a restart is programmed else FALSE

Definition at line 60 of file EncAlignCtrlPrivate.h.

Referenced by EAC_GetRestartState(), EAC_Init(), and EAC_SetRestartState().

5.77.2.16 int16_t Vars_t::hMeasBuffer[MPM_BUFFER_LENGTH]

Buffer used by MPM object to store the instantaneous measurements of motor power.

Definition at line 62 of file MotorPowerMeasurementPrivate.h.

Referenced by MPM_CalcElMotorPower(), MPM_Clear(), and MPM_GetElMotorPowerW().

5.77.2.17 uint16_t Vars_t::hNextMeasBufferIndex

Index of the buffer that will contain the next motor power measurement.

Definition at line 65 of file MotorPowerMeasurementPrivate.h.

Referenced by MPM_CalcElMotorPower(), and MPM_Clear().

5.77.2.18 uint16_t Vars_t::hLastMeasBufferIndex

Index of the buffer that contains the last motor power measurement.

Definition at line 67 of file MotorPowerMeasurementPrivate.h.

Referenced by MPM_CalcElMotorPower(), MPM_Clear(), and MPM_GetElMotorPowerW().

5.77.2.19 int16_t Vars_t::hAvrgElMotorPowerW

The average measured motor power expressed in watt.

Definition at line 69 of file MotorPowerMeasurementPrivate.h.

Referenced by MPM_CalcElMotorPower(), and MPM_GetAvrgElMotorPowerW().

5.77.2.20 int16_t Vars_t::hVoltage

Open loop voltage to be applied

Definition at line 51 of file OpenLoopPrivate.h.

Referenced by OL_Calc(), OL_Init(), OL_UpdateVoltage(), and OL_VqdConditioning().

5.77.2.21 CSPD Vars_t::oVSS

Related VSS object used

Definition at line 52 of file OpenLoopPrivate.h.

5.77.2.22 bool Vars_t::VFMode

TRUE to enable V/F mode otherwise FALSE

Definition at line 53 of file OpenLoopPrivate.h.

Referenced by OL_Calc(), OL_Init(), and OL_VF().

5.77.2.23 Volt_Components Vars_t::Vqdff

Feed-forward Iqd controller contribution to Vqd

Definition at line 49 of file FeedForwardCtrlPrivate.h.

5.77.2.24 Volt_Components Vars_t::VqdPlout

Vqd as output by PI controller

Definition at line 51 of file FeedForwardCtrlPrivate.h.

5.77.2.25 Volt_Components Vars_t::VqdAvPlout

Averaged Vqd as output by PI controller

Definition at line 52 of file FeedForwardCtrlPrivate.h.

5.77.2.26 int32_t Vars_t::wConstant_1D

Feed forward default constant for d axes

Definition at line 53 of file FeedForwardCtrlPrivate.h.

5.77.2.27 int32_t Vars_t::wConstant_1Q

Feed forward default constant for q axes

Definition at line 54 of file FeedForwardCtrlPrivate.h.

5.77.2.28 int32_t Vars_t::wConstant_2

Default constant value used by Feed-Forward algorithm

Definition at line 55 of file FeedForwardCtrlPrivate.h.

5.77.2.29 CVBS Vars_t::oVBS

Related bus voltage sensor

Definition at line 57 of file FeedForwardCtrlPrivate.h.

5.77.2.30 CPI Vars_t::oPI_q

Related PI for Iq regulator

Definition at line 58 of file FeedForwardCtrlPrivate.h.

5.77.2.31 CPI Vars_t::oPI_d

Related PI for Id regulator

Definition at line 59 of file FeedForwardCtrlPrivate.h.

5.77.2.32 CPI_Vars_t::oFluxWeakeningPI

PI object used for flux weakening

Definition at line 51 of file FluxWeakeningCtrlPrivate.h.

Referenced by FW_CalcCurrRef(), FW_Clear(), and FW_Init().

5.77.2.33 CPI_Vars_t::oSpeedPI

PI object used for speed control

Definition at line 52 of file FluxWeakeningCtrlPrivate.h.

Referenced by FW_CalcCurrRef(), and FW_Init().

5.77.2.34 uint16_t Vars_t::hFW_V_Ref

Voltage reference, tenth of percentage points

Definition at line 53 of file FluxWeakeningCtrlPrivate.h.

Referenced by FW_CalcCurrRef(), FW_GetVref(), FW_Init(), and FW_SetVref().

5.77.2.35 Volt_Components_Vars_t::AvVolt_qd

Average stator voltage in qd reference frame

Definition at line 55 of file FluxWeakeningCtrlPrivate.h.

Referenced by FW_CalcCurrRef(), FW_Clear(), and FW_DataProcess().

5.77.2.36 int16_t Vars_t::AvVoltAmpl

Average stator voltage amplitude

Definition at line 57 of file FluxWeakeningCtrlPrivate.h.

Referenced by FW_CalcCurrRef(), FW_Clear(), FW_GetAvVAmplitude(), and FW_GetAvVPercentage().

5.77.2.37 int16_t Vars_t::hIdRefOffset

Id reference offset

Definition at line 58 of file FluxWeakeningCtrlPrivate.h.

Referenced by FW_CalcCurrRef(), and FW_Clear().

5.77.2.38 uint16_t Vars_t::hT_Sqrt3

Contains a constant utilized by SVPWM algorithm

Definition at line 64 of file PWMnCurrFdbkPrivate.h.

Referenced by PWMC_Init(), and PWMC_SetPhaseVoltage().

5.77.2.39 uint16_t Vars_t::hSector

Contains the space vector sector number

Definition at line 65 of file PWMnCurrFdbkPrivate.h.

Referenced by PWMC_SetPhaseVoltage().

5.77.2.40 bool Vars_t::bTurnOnLowSidesAction

TRUE if TurnOnLowSides action is active, FALSE otherwise.

Definition at line 70 of file PWMnCurrFdbkPrivate.h.

Referenced by PWMC_GetTurnOnLowSidesAction(), PWMC_Init(), PWMC_SwitchOffPWM(), PWMC_SwitchOnPWM(), and PWMC_TurnOnLowSides().

5.77.2.41 uint16_t Vars_t::hOffCalibrWaitTimeCounter

Counter to wait fixed time before motor current measurement offset calibration.

Definition at line 72 of file PWMnCurrFdbkPrivate.h.

Referenced by PWMC_CurrentReadingCalibr().

5.77.2.42 uint8_t Vars_t::bMotor

Motor reference number

Definition at line 75 of file PWMnCurrFdbkPrivate.h.

5.77.2.43 uint16_t Vars_t::hPhaseRemainingTicks

Number of clock events remaining to complete the phase.

Definition at line 77 of file RevupCtrlPrivate.h.

Referenced by RUC_Clear(), and RUC_Exec().

5.77.2.44 pRUCPhasesParams_t Vars_t::pPhaseParams

Pointer to parameter of the current executed rev up phase.

Definition at line 79 of file RevupCtrlPrivate.h.

Referenced by RUC_Clear(), and RUC_Exec().

5.77.2.45 int16_t Vars_t::hDirection

If it is "1" the programmed revup sequence is performed. If it is "-1" the revup sequence is performed with opposite values of targets (speed, torque).

Definition at line 81 of file RevupCtrlPrivate.h.

Referenced by RUC_Clear(), and RUC_Exec().

5.77.2.46 uint8_t Vars_t::bStageCnt

It counts the rev up stages that have been already started. NOTE: The rev up stage are zero-based indexed so that the fist stage is the number zero.

Definition at line 85 of file RevupCtrlPrivate.h.

Referenced by RUC_Clear(), RUC_Exec(), and RUC_FirstAccelerationStageReached().

5.77.2.47 RUCPhasesData_t Vars_t::ParamsData[MAX_PHASE_NUMBER]

Rev up phases data. It is initialized equal to rev up phases parameters but can be changed in run-time using tuning methods.

Definition at line 93 of file RevupCtrlPrivate.h.

Referenced by RUC_Clear(), RUC_GetPhaseDurationms(), RUC_GetPhaseFinalMecSpeed01Hz(), RUC_GetPhaseFinalTorque(), RUC_Init(), RUC_SetPhaseDurationms(), RUC_SetPhaseFinalMecSpeed01Hz(), and RUC_SetPhaseFinalTorque().

5.77.2.48 uint8_t Vars_t::bPhaseNbr

Number of phases relative to the programmed rev up sequence.

Definition at line 97 of file RevupCtrlPrivate.h.

Referenced by RUC_GetNumberOfPhases(), and RUC_Init().

5.77.2.49 STC_Modality_t Vars_t::bMode

Modality of STC. It can be one of these two settings: STC_TORQUE_MODE to enable the Torque mode or STC_SPEED_MODE to enable the Speed mode.

Definition at line 51 of file SpeednTorqCtrlPrivate.h.

Referenced by STC_CalcTorqueReference(), STC_Clear(), STC_ExecRamp(), STC_GetControlMode(), STC_Init(), and STC_SetControlMode().

5.77.2.50 int16_t Vars_t::hTargetFinal

Backup of hTargetFinal to be applied in the last step.

Definition at line 55 of file SpeednTorqCtrlPrivate.h.

Referenced by STC_CalcTorqueReference(), STC_ExecRamp(), and STC_Init().

5.77.2.51 int32_t Vars_t::wSpeedRef01HzExt

Current mechanical rotor speed reference expressed in tenths of HZ multiplied by

1.

Definition at line 57 of file SpeednTorqCtrlPrivate.h.

Referenced by STC_CalcTorqueReference(), STC_ExecRamp(), and STC_Init().

5.77.2.52 int32_t Vars_t::wTorqueRef

Current motor torque reference. This value represents actually the Iq current expressed in digit multiplied by 65536.

Definition at line 60 of file SpeednTorqCtrlPrivate.h.

Referenced by STC_CalcTorqueReference(), STC_ExecRamp(), and STC_Init().

5.77.2.53 uint16_t Vars_t::hRampRemainingStep

Number of steps remaining to complete the ramp.

Definition at line 63 of file SpeednTorqCtrlPrivate.h.

Referenced by STC_CalcTorqueReference(), STC_ExecRamp(), STC_Init(), STC_RampCompleted(), STC_SetControlMode(), and STC_StopRamp().

5.77.2.54 CPI Vars_t::oPISpeed

The regulator used to perform the speed control loop.

Definition at line 65 of file SpeednTorqCtrlPrivate.h.

Referenced by STC_CalcTorqueReference(), and STC_Init().

5.77.2.55 CSPD Vars_t::oSPD

The speed sensor used to perform the speed regulation.

Definition at line 67 of file SpeednTorqCtrlPrivate.h.

Referenced by STC_CalcTorqueReference(), STC_ExecRamp(), STC_GetSpeedSensor(), STC_Init(), and STC_SetSpeedSensor().

5.77.2.56 int32_t Vars_t::wIncDecAmount

Increment/decrement amount to be applied to the reference value at each CalcTorqueReference.

Definition at line 69 of file SpeednTorqCtrlPrivate.h.

Referenced by STC_CalcTorqueReference(), STC_ExecRamp(), STC_Init(), and STC_StopRamp().

5.77.2.57 State_t Vars_t::bState

Variable containing state machine current state

Definition at line 51 of file StateMachinePrivate.h.

Referenced by STM_FaultAcknowledged(), STM_FaultProcessing(), STM_Init(), and STM_NextState().

5.77.2.58 uint16_t Vars_t::hFaultNow

Bit fields variable containing faults currently present

Definition at line 53 of file StateMachinePrivate.h.

Referenced by STM_FaultProcessing(), STM_GetFaultState(), and STM_Init().

5.77.2.59 uint16_t Vars_t::hFaultOccurred

Bit fields variable containing faults historically occurred since the state machine has been moved to FAULT_NOW state

Definition at line 55 of file StateMachinePrivate.h.

Referenced by STM_FaultAcknowledged(), STM_FaultProcessing(), STM_GetFaultState(), and STM_Init().

5.77.2.60 uint16_t Vars_t::hFaultState

It contains latest available average Vbus expressed in u16Celsius

It contains latest available average Vbus expressed in digit

Definition at line 53 of file TemperatureSensorPrivate.h.

5.77.2.61 uint16_t Vars_t::hLatestConv

It contains latest Vbus converted value expressed in u16Volts format

Definition at line 51 of file BusVoltageSensorPrivate.h.

5.77.2.62 uint8_t Vars_t::bDriveNum

Total number of MC objects.

Definition at line 52 of file UserInterfacePrivate.h.

Referenced by UI_Init(), and UI_SelectMC().

5.77.2.63 CMCI* Vars_t::pMCi

Pointer of MC interface list.

Definition at line 53 of file UserInterfacePrivate.h.

Referenced by UI_ExecCmd(), UI_ExecSpeedRamp(), UI_ExecTorqueRamp(), UI_GetReg(), UI_Init(), UI_SetCurrentReferences(), and UI_SetReg().

5.77.2.64 CMCT* Vars_t::pMCT

Pointer of MC tuning list.

Definition at line 54 of file UserInterfacePrivate.h.

Referenced by UI_GetCurrentMCT(), UI_GetReg(), UI_GetRevupData(), UI_Init(), UI_SetReg(), and UI_SetRevupData().

5.77.2.65 uint32_t* Vars_t::pUICfg

Pointer of UI configuration list.

Definition at line 55 of file UserInterfacePrivate.h.

Referenced by UI_ExecCmd(), UI_GetReg(), UI_GetSelectedMCConfig(), UI_Init(), and UI_SetReg().

5.77.2.66 uint8_t Vars_t::bSelectedDrive

Current selected MC object in the list.

Definition at line 56 of file UserInterfacePrivate.h.

Referenced by UFC_Init(), UI_ExecCmd(), UI_ExecSpeedRamp(), UI_ExecTorqueRamp(), UI_GetCurrentMC-T(), UI_GetReg(), UI_GetRevupData(), UI_GetSelectedMCConfig(), UI_Init(), UI_SelectMC(), UI_SetCurrentReferences(), UI_SetReg(), and UI_SetRevupData().