Rapid IOT SDK API Reference Manual

NXP Semiconductors

Document Number: SLN-RPK-NODE-SDK-API-RM
Rev. 0
Jun 2018

Contents

Chapter 1
Introduction

Chapter 2
A100x_interface

2.1 Overview . . . **3**

2.2 Data Structure Documentation . . . **5**
2.2.1 struct a1006_IoFunc_t . . . 5
2.2.1.1 Field Documentation . . . 5
2.2.1.1.1 I2C_Read . . . 5
2.2.1.1.2 I2C_Write . . . 5
2.2.1.1.3 WaitMsec . . . 5
2.2.1.1.4 i2c_Address . . . 6
2.2.1.1.5 cert_slot . . . 6

2.3 Enumeration Type Documentation . . . **6**
2.3.1 a1006_cert_slot_t . . . 6
2.3.2 a1006_transfer_status_t . . . 6
2.3.3 a1006_init_status_t . . . 6

2.4 Function Documentation . . . **7**
2.4.1 A1006_Init_Driver(a1006_IoFunc_t *pIoFunc) . . . 7
2.4.2 A1006_Deinit_Driver(void) . . . 7
2.4.3 A1006_Soft_Reset(void) . . . 7
2.4.4 A1006_Power_Down(void) . . . 8
2.4.5 A1006_Wake_Up(void) . . . 8
2.4.6 A1006_Get_Status(uint32_t *a1006_status) . . . 8
2.4.7 A1006_Get_Uid(uint8_t *uid) . . . 8
2.4.8 A1006_Get_Cert(uint8_t *cert) . . . 9
2.4.9 A1006_Set_Challenge(uint8_t *challenge) . . . 10
2.4.10 A1006_Get_Response(uint8_t *response) . . . 10

Chapter 3
LCD Backlight

3.1 Overview . . . 11

3.2 Data Structure Documentation . . . 12
3.2.1 struct backlight_fct_t . . . 12
3.2.1.1 Field Documentation . . . 12
3.2.1.1.1 connect_hw . . . 12
3.2.1.1.2 disconnect_hw . . . 12
3.2.1.1.3 set_level . . . 12

3.3 Enumeration Type Documentation . . . 12
3.3.1 backlight_status_t . . . 12
3.3.2 backlight_level_t . . . 13

3.4 Function Documentation . . . 13
3.4.1 Backlight_Init_Driver(backlight_fct_t *fct) . . . 13
3.4.2 Backlight_Deinit_Driver() . . . 13
3.4.3 Backlight_Init_Hw() . . . 13
3.4.4 Backlight_SetLevel(backlight_level_t level) . . . 13
3.4.5 Backlight_GetLevel(backlight_level_t *level) . . . 14

Chapter 4
Battery Sensor

4.1 Overview . . . 15

4.2 Data Structure Documentation . . . 16
4.2.1 struct battery_fct_t . . . 16
4.2.1.1 Field Documentation . . . 16
4.2.1.1.1 WaitMsec . . . 16

4.3 Enumeration Type Documentation . . . 16
4.3.1 battery_status_t . . . 16

4.4 Function Documentation . . . 16
4.4.1 BatterySensor_Init_Driver(ptbattery_fct_t FCT) . . . 16
4.4.2 BatterySensor_Deinit_Driver() . . . 17
4.4.3 BatterySensor_Init_Hw() . . . 17
4.4.4 BatterySensor_GetState(uint8_t *batteryPercentLevel, uint8_t *batteryChargingState) . . . 17

Section number Title Page

Chapter 5
Buzzer

5.1 Overview . . . 19

5.2 Enumeration Type Documentation . . . 20
5.2.1 buzzer_status_t . . . 20

5.3 Function Documentation . . . 20
5.3.1 Buzzer_Init_Driver() . . . 20
5.3.2 Buzzer_Deinit_Driver() . . . 20
5.3.3 Buzzer_Init_Hw() . . . 20
5.3.4 Buzzer_On(void) . . . 20
5.3.5 Buzzer_Off(void) . . . 21

Chapter 6
CCS811 Air Quality Sensor

6.1 Overview . . . 23

6.2 Data Structure Documentation . . . 25
6.2.1 struct CCS811_fct_t . . . 25
6.2.1.1 Field Documentation . . . 25
6.2.1.1.1 connect_hw . . . 25
6.2.1.1.2 disconnect_hw . . . 25
6.2.1.1.3 I2C_Read . . . 25
6.2.1.1.4 I2C_Write . . . 25
6.2.1.1.5 WaitMsec . . . 25

6.3 Enumeration Type Documentation . . . 25
6.3.1 CCS811_status . . . 25

6.4 Function Documentation . . . 26
6.4.1 CCS811_Init_Driver(ptCCS811_fct_t FCT) . . . 26
6.4.2 CCS811_Deinit_Driver() . . . 26
6.4.3 CCS811_Init_Hw(void) . . . 26
6.4.4 CCS811_readAlgorithmResults(void) . . . 26
6.4.5 CCS811_checkForStatusError(uint8_t *StatusError) . . . 27
6.4.6 CCS811_dataAvailable(uint8_t *value) . . . 28
6.4.7 CCS811_appValid(uint8_t *value) . . . 28
6.4.8 CCS811_getErrorRegister(uint8_t *value) . . . 29
6.4.9 CCS811_getBaseline(unsigned int *baseline) . . . 29
6.4.10 CCS811_setBaseline(uint16_t input) . . . 29

Section number | Title | Page

6.4.11 CCS811_setInterrupts(uint8_t fct) . . . 30
6.4.12 CCS811_setThresholds(uint8_t fct) . . . 30
6.4.13 CCS811_setDriveMode(uint8_t mode) . . . 30
6.4.14 CCS811_setEnvironmentalData(float relativeHumidity, float temperature) . . . 32
6.4.15 CCS811_setRefResistance(float input) . . . 32
6.4.16 CCS811_readNTC(void) . . . 32
6.4.17 CCS811_getTVOC(void) . . . 33
6.4.18 CCS811_getCO2(void) . . . 33
6.4.19 CCS811_getResistance(void) . . . 33
6.4.20 CCS811_getTemperature(void) . . . 33

Chapter 7
ENS210 Relative Humidity and Temperature Sensor

7.1 Overview . . . 35

7.2 Data Structure Documentation . . . 38
7.2.1 struct ens210_IoFunc_t . . . 38
7.2.1.1 Field Documentation . . . 38
7.2.1.1.1 I2C_Read . . . 38
7.2.1.1.2 I2C_Write . . . 38
7.2.1.1.3 WaitMsec . . . 38
7.2.2 struct ENS210_Ids_t . . . 38
7.2.3 struct ens210_meas_data_t . . . 38

7.3 Macro Definition Documentation . . . 39
7.3.1 ENS210_OSFREE_DRIVER_VERSION . . . 39
7.3.2 ENS210_T_H_CONVERSION_TIME_MS . . . 39
7.3.3 ENS210_T_CONVERSION_TIME_MS . . . 39
7.3.4 ENS210_BOOTING_TIME_MS . . . 39
7.3.5 ENS210_RESET_WAIT_TIME_MS . . . 39
7.3.6 ENS210_I2C_SLAVE_ADDRESS . . . 39
7.3.7 ENS210_SYSCTRL_LOWPOWER_ENABLE . . . 40
7.3.8 ENS210_SYSCTRL_LOWPOWER_DISABLE . . . 40
7.3.9 ENS210_SYSCTRL_RESET_ENABLE . . . 40
7.3.10 ENS210_SYSCTRL_RESET_DISABLE . . . 40
7.3.11 ENS210_SYSSTAT_MODE_STANDBY . . . 40
7.3.12 ENS210_SYSSTAT_MODE_ACTIVE . . . 40
7.3.13 ENS210_SENSRUN_T_MODE_SINGLE_SHOT . . . 40
7.3.14 ENS210_SENSRUN_T_MODE_CONTINUOUS . . . 40
7.3.15 ENS210_SENSRUN_H_MODE_SINGLE_SHOT . . . 40
7.3.16 ENS210_SENSRUN_H_MODE_CONTINUOUS . . . 41
7.3.17 ENS210_SENSSTART_T_START . . . 41
7.3.18 ENS210_SENSSTART_H_START . . . 41

Section number	Title	Page
7.3.19	ENS210_SENSSTOP_T_STOP	41
7.3.20	ENS210_SENSSTOP_H_STOP	41
7.3.21	ENS210_SENSSTAT_T_STAT_IDLE	41
7.3.22	ENS210_SENSSTAT_T_STAT_ACTIVE	41
7.3.23	ENS210_SENSSTAT_H_STAT_IDLE	41
7.3.24	ENS210_SENSSTAT_H_STAT_ACTIVE	41
7.4	**Enumeration Type Documentation**	**42**
7.4.1	ens210_status_t	42
7.4.2	measurement_mode	42
7.5	**Function Documentation**	**42**
7.5.1	ENS210_Init_Driver(ens210_IoFunc_t *pIoFunc)	42
7.5.2	ENS210_Deinit_Driver()	42
7.5.3	ENS210_Init_Hw(void)	43
7.5.4	ENS210_SysCtrl_Set(uint8_t sysCtrl)	43
7.5.5	ENS210_SysCtrl_Get(uint8_t *sysCtrl)	43
7.5.6	ENS210_SysStat_Get(uint8_t *sysStat)	43
7.5.7	ENS210_SensRun_Set(uint8_t sensRun)	44
7.5.8	ENS210_SensRun_Get(uint8_t *sensRun)	44
7.5.9	ENS210_SensStart_Set(uint8_t sensStart)	44
7.5.10	ENS210_SensStop_Set(uint8_t sensStop)	45
7.5.11	ENS210_SensStat_Get(uint8_t *sensStat)	45
7.5.12	ENS210_TVal_Get(uint32_t *traw)	45
7.5.13	ENS210_HVal_Get(uint32_t *hraw)	46
7.5.14	ENS210_THVal_Get(uint32_t *traw, uint32_t *hraw)	46
7.5.15	ENS210_Ids_Get(ENS210_Ids_t *ids)	46
7.5.16	ENS210_Measure(uint8_t meas_mode, ens210_meas_data_t *results)	47

Chapter 8
FXAS21002 Gyroscope

8.1	**Overview**	**49**
8.2	**Data Structure Documentation**	**50**
8.2.1	struct fxas21002_spi_sensorhandle_t	50
8.2.2	struct fxas21002_i2c_sensorhandle_t	51
8.2.3	struct fxas21002_gyrodata_t	52
8.3	**Macro Definition Documentation**	**52**
8.3.1	FXAS21002_GYRO_DATA_SIZE	52
8.3.2	FXAS21002_SPI_MAX_MSG_SIZE	52
8.3.3	FXAS21002_SPI_CMD_LEN	52
8.3.4	FXAS21002_SS_ACTIVE_VALUE	52

Section number	Title	Page
8.4	**Function Documentation**	**53**
8.4.1	FXAS21002_I2C_Initialize(fxas21002_i2c_sensorhandle_t *pSensorHandle, ARM_DRIVER_I2C *pBus, uint8_t index, uint16_t sAddress, uint8_t whoAmi)	53
8.4.2	FXAS21002_I2C_SetIdleTask(fxas21002_i2c_sensorhandle_t *pSensorHandle, registeridlefunction_t idleTask, void *userParam)	53
8.4.3	FXAS21002_I2C_Configure(fxas21002_i2c_sensorhandle_t *pSensorHandle, const registerwritelist_t *pRegWriteList)	53
8.4.4	FXAS21002_I2C_ReadData(fxas21002_i2c_sensorhandle_t *pSensorHandle, const registerreadlist_t *pReadList, uint8_t *pBuffer)	54
8.4.5	FXAS21002_I2C_Deinit(fxas21002_i2c_sensorhandle_t *pSensorHandle)	54
8.4.6	FXAS21002_SPI_Initialize(fxas21002_spi_sensorhandle_t *pSensorHandle, ARM_DRIVER_SPI *pBus, uint8_t index, void *pSlaveSelect, uint8_t whoAmi)	55
8.4.7	FXAS21002_SPI_SetIdleTask(fxas21002_spi_sensorhandle_t *pSensorHandle, registeridlefunction_t idleTask, void *userParam)	55
8.4.8	FXAS21002_SPI_Configure(fxas21002_spi_sensorhandle_t *pSensorHandle, const registerwritelist_t *pRegWriteList)	56
8.4.9	FXAS21002_SPI_ReadData(fxas21002_spi_sensorhandle_t *pSensorHandle, const registerreadlist_t *pReadList, uint8_t *pBuffer)	56
8.4.10	FXAS21002_SPI_Deinit(fxas21002_spi_sensorhandle_t *pSensorHandle)	56
8.4.11	FXAS21002_SPI_ReadPreprocess(void *pCmdOut, uint32_t offset, uint32_t size)	57
8.4.12	FXAS21002_SPI_WritePreprocess(void *pCmdOut, uint32_t offset, uint32_t size, void *pWritebuffer)	57

Chapter 9
FXOS8700 Combination Accelerometer & Magnetometer

9.1	**Overview**	**59**
9.2	**Data Structure Documentation**	**68**
9.2.1	struct fxos8700_handle_t	68
9.2.2	struct fxos8700_data_t	69
9.3	**Function Documentation**	**69**
9.3.1	FXOS8700_Init(fxos8700_handle_t *fxos8700_handle)	69
9.3.2	FXOS8700_ReadSensorData(fxos8700_handle_t *fxos8700_handle, fxos8700_data_t *sensorData)	70
9.3.3	FXOS8700_WriteReg(fxos8700_handle_t *handle, uint8_t reg, uint8_t val)	71
9.3.4	FXOS8700_ReadReg(fxos8700_handle_t *handle, uint8_t reg, uint8_t *val, uint8_t bytesNumber)	71
9.3.5	FXOS8700_SetStandby(fxos8700_handle_t *fxos8700_handle)	71
9.3.6	FXOS8700_SetActive(fxos8700_handle_t *fxos8700_handle)	72
9.3.7	FXOS8700_MotionDetect_Init(fxos8700_handle_t *fxos8700_handle)	72
9.3.8	FXOS8700_FreefallDetect_Init(fxos8700_handle_t *fxos8700_handle)	73

Section number	Title	Page
9.3.9	FXOS8700_TapDetect_Init(fxos8700_handle_t *fxos8700_handle)	73
9.3.10	FXOS8700_FreefallMotion_DeInit(fxos8700_handle_t *fxos8700_handle)	74
9.3.11	FXOS8700_TapDetect_DeInit(fxos8700_handle_t *fxos8700_handle)	74
9.3.12	FXOS8700_TransientDetect_Init(fxos8700_handle_t *fxos8700_handle)	74
9.3.13	FXOS8700_TransientDetect_DeInit(fxos8700_handle_t *fxos8700_handle)	75

Chapter 10
Image BMP Callbacks

10.1	**Overview**	**77**

Chapter 11
MPL3115 Atmospheric Pressure Sensor

11.1	**Overview**	**79**
11.2	**Data Structure Documentation**	**81**
11.2.1	struct mpl3115_IoFunc_t	81
11.2.1.1	Field Documentation	81
11.2.1.1.1	I2C_Read	81
11.2.1.1.2	I2C_Write	81
11.2.1.1.3	WaitMsec	82
11.2.2	struct settingsMPL_t	82
11.3	**Enumeration Type Documentation**	**82**
11.3.1	mpl_status_t	82
11.3.2	overSampleMPL_t	82
11.3.3	autoAcquisitionTime_t	83
11.3.4	modeMPL_t	83
11.3.5	modeFIFO_t	84
11.3.6	pinINT_t	84
11.4	**Function Documentation**	**84**
11.4.1	MPL3115_Init_Driver(mpl3115_IoFunc_t *pIoFunc)	84
11.4.2	MPL3115_Deinit_Driver()	84
11.4.3	MPL3115_Init_Hw()	84
11.4.4	MPL_SoftReset()	85
11.4.5	MPL_ToggleOneShot()	85
11.4.6	MPL_GetID(uint8_t *sensorID)	85
11.4.7	MPL_SetMode(modeMPL_t mode)	85
11.4.8	MPL_GotoStandby()	86
11.4.9	MPL_SetActive()	86

Section number	Title	Page
11.4.10	MPL_SetOversampleRate(uint8_t sampleRate)	86
11.4.11	MPL_SetAutoAcquisitionTime(uint8_t sampleTime)	87
11.4.12	MPL_EnableEventFlags()	87
11.4.13	MPL_SetOffsetPressure(int8_t pressOffset)	87
11.4.14	MPL_SetOffsetAltitude(int8_t altitudeOffset)	88
11.4.15	MPL_SetOffsetTemperature(int8_t temperatureOffset)	88
11.4.16	MPL_SetFifoMode(modeFIFO_t fMode, uint8_t fWmrk)	89
11.4.17	MPL_SetFifoInterrupt(pinINT_t pinINT)	89
11.4.18	MPL_DisableFifoInterrupt()	89
11.4.19	MPL_GetFifoStatus(uint8_t *fifoStatus)	89
11.4.20	MPL_ReadRawData(modeMPL_t mode, int32_t *sensorData)	90
11.4.21	MPL_Dump(uint8_t *sensorReg)	90

Chapter 12
PCF2123 Real Time Clock/Calendar

12.1	**Overview**	**91**
12.2	**Data Structure Documentation**	**94**
12.2.1	struct pcf2123_IoFunc_t	94
12.2.1.1	Field Documentation	94
12.2.1.1.1	SPI_Read	94
12.2.1.1.2	SPI_Write	94
12.2.1.1.3	WaitMsec	94
12.2.2	struct settingsPCF_t	94
12.3	**Enumeration Type Documentation**	**95**
12.3.1	pcf2123_status_t	95
12.3.2	modePCF_t	95
12.3.3	weekdaysPCF_t	95
12.3.4	monthsPCF_t	96
12.3.5	clockOutputPCF_t	96
12.3.6	cdmodePCF_t	97
12.4	**Function Documentation**	**97**
12.4.1	PCF2123_Init_Driver(pcf2123_IoFunc_t *pIoFunc)	97
12.4.2	PCF2123_Deinit_Driver()	97
12.4.3	PCF2123_Init_Hw(const settingsPCF_t *pcfSettings)	97
12.4.4	PCF2123_Dump(uint8_t *pcfReg)	97
12.4.5	PCF2123_SetOffset(bool mode, uint8_t Offset)	98
12.4.6	PCF2123_GetDateTime(settingsPCF_t *pcfDateTime)	98
12.4.7	PCF2123_SetDateTime(settingsPCF_t *pcfDateTime)	98
12.4.8	PCF2123_SetClockOutputFreq(clockOutputPCF_t pcfClockOutputFreq)	99
12.4.9	PCF2123_SetMinSecInterrupt(bool Minute_Int, bool Second_Int, bool Pulse_Int)	99

Section number	Title	Page
12.4.10	PCF2123_SetAlarmInterrupt(bool Alarm_Int)	99
12.4.11	PCF2123_SetCountdownInterrupt(bool Countdown_Int)	100
12.4.12	PCF2123_SetCountdownMode(bool CountDownEn, cdmodePCF_t CountDownMode)	100
12.4.13	PCF2123_SetCountdownTimer(uint8_t CDT_Value)	100
12.4.14	PCF2123_SetMinuteAlarm(bool AlarmEn, uint8_t Minute)	101
12.4.15	PCF2123_SetHourAlarm(bool AlarmEn, bool AMPM, uint8_t Hour)	101
12.4.16	PCF2123_SetDayAlarm(bool AlarmEn, uint8_t Day)	102
12.4.17	PCF2123_SetWeekdayAlarm(bool AlarmEn, uint8_t Weekday)	102
12.4.18	PCF2123_ClearMinSecInterruptFlag()	102
12.4.19	PCF2123_ClearCountdownInterruptFlag()	103
12.4.20	PCF2123_ClearAlarmInterruptFlag()	103

Chapter 13
Port Interrupts

13.1	**Overview**	**105**
13.2	**Function Documentation**	**106**
13.2.1	PORT_IRQ_EnablePortAIrq(void)	106
13.2.2	PORT_IRQ_EnablePortBIrq(void)	107
13.2.3	PORT_IRQ_EnablePortCIrq(void)	107
13.2.4	PORT_IRQ_EnablePortDIrq(void)	107
13.2.5	PORT_IRQ_EnablePortEIrq(void)	107
13.2.6	PORTA_IRQHandler(void)	107
13.2.7	PORTB_IRQHandler(void)	108
13.2.8	PORTC_IRQHandler(void)	108
13.2.9	PORTD_IRQHandler(void)	108
13.2.10	PORTE_IRQHandler(void)	109
13.2.11	PORT_IRQ_UiEnableIrq(void)	109
13.2.12	PORT_IRQ_UiDisableIrq(void)	109

Chapter 14
RGB LED

14.1	**Overview**	**111**
14.2	**Data Structure Documentation**	**112**
14.2.1	struct rgbled_fct_t	112
14.2.1.1	Field Documentation	112
14.2.1.1.1	connect_hw	112
14.2.1.1.2	disconnect_hw	112

Section number	Title	Page
14.2.1.1.3	set_rgb_colors	113
14.3	**Enumeration Type Documentation**	**113**
14.3.1	rgb_led_brightness_t	113
14.3.2	rgb_led_color_t	113
14.4	**Function Documentation**	**113**
14.4.1	RGB_Led_Init_Driver()	113
14.4.2	RGB_Led_Deinit_Driver()	113
14.4.3	RGB_Led_Init_Hw()	114
14.4.4	RGB_Led_Get_State(uint8_t *brightness, uint8_t *color)	114
14.4.5	RGB_Led_Set_State(uint8_t brightness, uint8_t color)	114

Chapter 15
Base64 String Encoder/Decoder

15.1	**Overview**	**115**
15.2	**Function Documentation**	**116**
15.2.1	RPK_Base64_Get_Encoded_Len(uint32_t len)	116
15.2.2	RPK_Base64_Get_Decoded_Len(uint8_t *buf, uint32_t len)	116
15.2.3	RPK_Base64_Encode(uint8_t *in, uint32_t numInBytes, uint8_t *out, uint32_t numOutBytes)	117
15.2.4	RPK_Base64_Decode(uint8_t *in, uint32_t numInBytes, uint8_t *out, uint32_t numOutBytes)	117

Chapter 16
SX9500 Touch Controller

16.1	**Overview**	**119**
16.2	**Data Structure Documentation**	**121**
16.2.1	union RegIrqSrc_t	121
16.2.2	struct RegIrqSrc_t.bits	121
16.2.3	union RegStat_t	121
16.2.4	struct RegStat_t.bits	121
16.2.5	union RegProxCtrl0_t	122
16.2.6	struct RegProxCtrl0_t.bits	122
16.2.7	union RegProxCtrl3_t	122
16.2.8	struct RegProxCtrl3_t.bits	122
16.2.9	struct sx9500_fct_t	123
16.2.9.1	Field Documentation	123
16.2.9.1.1	connect_hw	123

Section number	Title	Page
16.2.9.1.2	disconnect_hw	123
16.2.9.1.3	I2C_Read	123
16.2.9.1.4	I2C_Write	123
16.2.9.1.5	WaitMs	124
16.3	**Enumeration Type Documentation**	**124**
16.3.1	SX9500_status	124
16.4	**Function Documentation**	**124**
16.4.1	SX9500_Init_Driver(ptsx9500_fct_t FCT)	124
16.4.2	SX9500_Deinit_Driver()	124
16.4.3	SX9500_Init_Hw()	124
16.4.4	SX9500_GetInfo_sensor(char CSn, uint8_t *buf)	124
16.4.5	SX9500_get_active()	125
16.4.6	SX9500_set_active(bool en)	125
16.4.7	SX9500_CSi_Detected(uint8_t *CSi)	125
16.4.8	SX9500_Read_Irq(uint8_t *irqReg)	126
16.4.9	SX9500_Read_Proximity_Sensors(uint8_t *data)	126

Chapter 17
TSL2572 Ambient Light Sensor

17.1	**Overview**	**127**
17.2	**Data Structure Documentation**	**128**
17.2.1	struct tsl2572_IoFunc_t	128
17.2.1.1	Field Documentation	129
17.2.1.1.1	I2C_Read	129
17.2.1.1.2	I2C_Write	129
17.2.1.1.3	WaitMsec	129
17.2.2	struct tsl2x7x_settings_t	129
17.3	**Enumeration Type Documentation**	**130**
17.3.1	tsl2572_status_t	130
17.4	**Function Documentation**	**130**
17.4.1	TSL2572_Init_Driver(tsl2572_IoFunc_t *pIoFunc)	130
17.4.2	TSL2572_Deinit_Driver()	130
17.4.3	TSL2572_Init_Hw(void)	130
17.4.4	TSL2572_ReadAmbientLight(float *lux)	131
17.4.5	TSL2572_SetALSGain(uint8_t AGAIN, bool AGL)	131
17.4.6	TSL2572_SetALSTime(uint8_t ATIME)	131
17.4.7	TSL2572_SetALSThresholds(uint16_t ALS_interrupt_Low_Threshold, uint16_t ALS_interrupt_High_Threshold)	132

Section number	Title	Page
17.4.8	TSL2572_GetALSThresholds(uint16_t *ALS_interrupt_Low_Threshold, uint16_t *ALS_interrupt_High_Threshold)	132
17.4.9	TSL2572_SetALSPersistence(uint8_t APERS)	133
17.4.10	TSL2572_SetWaitTime(uint8_t WTIME, bool WLONG)	134
17.4.11	TSL2572_EnableALSInterrupts(bool AIEN)	134
17.4.12	TSL2572_ClearALSInterrupt(void)	134
17.4.13	TSL2572_Enable_ALS(bool AEN)	134
17.4.14	TSL2572_Enable_Wait(bool WEN)	135
17.4.15	TSL2572_Power_ON(bool PON)	135
17.4.16	TSL2572_ReadAllRegisters(uint8_t *RegData)	135
17.4.17	TSL2572_ReadCH0(uint8_t *RegData)	136

Chapter 18
UI Manager

18.1	**Overview**	**139**
18.2	**Enumeration Type Documentation**	**141**
18.2.1	ui_events_t	141
18.2.2	ui_touch_t	141
18.3	**Function Documentation**	**142**
18.3.1	UiManager_SetEvent(uint32_t event)	142
18.3.2	UiManager_Init(void)	142

Chapter 1
Introduction

The Rapid IOT Software Development Kit (SDK) is based on the MCUXpresso Software Development Kit (MCUXpresso SDK) and builds on it by providing the firmware for interfacing with the hardware built around the K64F and the KW41Z microcontrollers in the Rapid IOT. In this document you will find the API function calls to the various sensors, touch controller, secure authenticator, port interrupts, UI manager, display, lights, buzzer and battery sensor. Please refer to the MCUXpresso SDK for the underlying drivers and middleware on which the Rapid IOT SDK is based.

Chapter 2
A100x_interface

2.1 Overview

The a100x_interface module provides communication with the A1006 secure authenticator via I2C.

This API facilitates authentication between the Client device w/A1006 on board and a remote Host.

Usage

Initialization:

```
a1006_IoFunc_t A1006_auth;
A1006_auth.I2C_Read = App_I2C2_Read;
A1006_auth.I2C_Write = App_I2C2_Write;
A1006_auth.WaitMsec = App_WaitMsec;
A1006_auth.i2c_Address = A1006_I2C_SLAVE_ADDRESS;
A1006_auth.cert_slot = kCertificateOne;

uint8_t ret = A1006_Init_Driver(&A1006_auth);
```

Basic Operation:

```
// Obtaining UID
uint8_t ret = 1;
uint8_t uid[16] = {0};

if (NULL != uid)
{
    ret = A1006_Get_Uid(uid);
}

// Obtaining Certificate
uint8_t cert[128] = {0};

if (NULL != cert)
{
    ret = A1006_Get_Cert(cert);
}

// Setting Challenge from Host
uint8_t challenge[44] = {0};

if (NULL != challenge)
{
    ret = A1006_Set_Challenge(challenge);
}

// Getting channenge Response from A1006
uint8_t response[44] = {0};

if (NULL != response)
{
    ret = A1006_Get_Response(response);
}
```

Memory Considerations

It is the responsibility of the developer to allocate the necessary memory.

The following buffer sizes are needed:

- A1006_Get_Uid:
  - uid buffer must be 16 bytes
- A1006_Get_Cert:
  - cert buffer must be 128 bytes
- A1006_Set_Challenge:
  - challenge buffer must be 44 bytes
- A1006_Get_Response:
  - response buffer must be 44 bytes

Certificate Options

The a100x_interface can be directed to use two different certificates for authentication.

These options are:

1. User Certificate [kCertificateOne]
2. NXP Certificate [kCertificateTwo]

A100x interface routines to communicate with the A100x over I2C.

Files

- file a100x_interface.h

Data Structures

- struct pa1006_IoFunc_t

Enumerations

- enum a1006_cert_slot_t {
  kCertificateOne,
  kCertificateTwo,
  kCertificateInvalid }
- enum a1006_transfer_status_t {
  kTransferOkay,
  kTransferFail,
  kTransferTimeout,
  kInvalidInput }
- enum a1006_init_status_t {
  kInitSuccess,
  kAlreadyInitialized,
  kDeinitSuccess,

kNotInitialized }

Functions

- uint8_t A1006_Init_Driver (a1006_IoFunc_t *pIoFunc)
- uint8_t A1006_Deinit_Driver (void)
- uint8_t A1006_Soft_Reset (void)
- uint8_t A1006_Power_Down (void)
- uint8_t A1006_Wake_Up (void)
- uint8_t A1006_Get_Status (uint32_t *a1006_status)
- uint8_t A1006_Get_Uid (uint8_t *uid)
- uint8_t A1006_Get_Cert (uint8_t *cert)
- uint8_t A1006_Set_Challenge (uint8_t *challenge)
- uint8_t A1006_Get_Response (uint8_t *response)

2.2 Data Structure Documentation

2.2.1 struct a1006_IoFunc_t

A1006 IO Functions.

Data Fields

- uint8_t(* I2C_Read)(uint8_t device_addr, uint8_t *writeBuf, uint32_t writeSize, uint8_t *readBuf, uint32_t readSize)
- uint8_t(* I2C_Write)(uint8_t device_addr, uint8_t *writeBuf, uint32_t writeSize)
- void(* WaitMsec)(uint32_t millisec)
- uint8_t i2c_Address
- a1006_cert_slot_t cert_slot

2.2.1.1 Field Documentation

2.2.1.1.1 uint8_t(* I2C_Read) (uint8_t device_addr, uint8_t *writeBuf, uint32_t writeSize, uint8_t *readBuf, uint32_t readSize)

Function pointer to I2C Read function.

2.2.1.1.2 uint8_t(* I2C_Write) (uint8_t device_addr, uint8_t *writeBuf, uint32_t writeSize)

Function pointer to I2C Write function.

2.2.1.1.3 void(* WaitMsec) (uint32_t millisec)

Function pointer to WaitMsec function.

2.2.1.1.4 uint8_t i2c_Address

7-bit I2C slave address

2.2.1.1.5 a1006_cert_slot_t cert_slot

Enum for slot number to read out credentials.

2.3 Enumeration Type Documentation

2.3.1 enum a1006_cert_slot_t

A1006 Certificate Slots.

Enumerator

kCertificateOne Directs a100x_interface to use User Certificate.
kCertificateTwo Directs a100x_interface to use NXP Certificate.
kCertificateInvalid Represent invalid certificate slot.

2.3.2 enum a1006_transfer_status_t

A1006 Transfer Status.

Enumerator

kTransferOkay Transfer Ok.
kTransferFail Transfer Fail.
kTransferTimeout Transfer Timeout.
kInvalidInput Invalid Input.

2.3.3 enum a1006_init_status_t

A1006 Initialization Status.

Enumerator

kInitSuccess A1006 Initialization Successful.
kAlreadyInitialized A1006 Already Initialized.
kDeinitSuccess A1006 Deinitialization Successful.
kNotInitialized A1006 Not Initialized.

2.4 Function Documentation

2.4.1 uint8_t A1006_Init_Driver (a1006_IoFunc_t * *pIoFunc*)

Initialize interfaceA100x slave.

Note

This API should be called at the beginning of the application before using the driver.

Parameters

pIoFunc	Input I/O function structure

Returns

None

2.4.2 uint8_t A1006_Deinit_Driver (void)

De-Initialize interface to A100x slave.

Note

This API should be called once application has finished using A100x

Returns

None

2.4.3 uint8_t A1006_Soft_Reset (void)

Soft reset using general call (0x00)

Returns

0 on success
!= 0 on error

2.4.4 uint8_t A1006_Power_Down (void)

Power down device with I2C command.

Returns

0 on success
!= 0 on error

2.4.5 uint8_t A1006_Wake_Up (void)

Wake-up device with I2C command.

Returns

0 on success
!= 0 on error

2.4.6 uint8_t A1006_Get_Status (uint32_t * *a1006_status*)

Get device statuc over I2C.

Parameters

a1006_status	Output 32-bit integer status value

Returns

0 on success
!= 0 on error

2.4.7 uint8_t A1006_Get_Uid (uint8_t * *uid*)

Get UID from device over I2C.

Parameters

uid	Output 16 byte buffer containing uid from device

Returns

0 on success
!= 0 on error

2.4.8 uint8_t A1006_Get_Cert (uint8_t * *cert*)

Get raw compressed certificate from device over i2c.

Parameters

cert	Output 128 byte buffer for containing raw certificate data

Returns

0 on success
!= 0 on error

2.4.9 uint8_t A1006_Set_Challenge (uint8_t * *challenge*)

Set challenge on device over I2C.

Parameters

challenge	Input 44 byte buffer containing the challenge to place in device

Returns

0 on success
!= 0 on error

2.4.10 uint8_t A1006_Get_Response (uint8_t * *response*)

Get challenge response from device over I2C.

Parameters

response	Output 44 byte buffer containing the response from the device

Returns

0 on success
!= 0 on error

Chapter 3
LCD Backlight

3.1 Overview

This module provides the API to control the LCD backlight level through callback functions.

The basic steps to operate the LCD backlight are as follows:

1. Initialize the driver with callback functions (Backlight_Init_Driver)
2. Initialize the hardware (Backlight_Init_Hw)
3. Set the backlight level (Backlight_SetLevel)
4. If the backlight is not needed anymore, de-initialize the driver (Backlight_Deinit_Driver). The backlight will be switched off. It allows to eventually release shared resources.

Example - Sample application code to set LCD backlight level without error management

```
#include "backlight.h"

backlight_fct_t backlight_fct;
backlight_level_t level;

backlight_fct.connect_hw = Backlight_Connect;       // callback function to activate backlight
       hardware resource
backlight_fct.disconnect_hw = Backlight_Disconnect; // callback function to deactivate
       backlight hardware resource
backlight_fct.set_level = Backlight_Set_Level;      // callback function to set backlight level

Backlight_Init_Driver(&backlight_fct);
Backlight_Init_Hw();
printf("Set backlight level to medium\n");
Backlight_SetLevel(BLIGHT_LEVEL_MEDIUM);
[..]
Backlight_GetLevel(&level);
printf("The current backlight level is: %d\n", level);
[..]
// if the backlight is not needed anymore, the driver can be de-initialized
printf("Deinitialize backlight (it will be turned off)\n");
Backlight_Deinit_Driver();
```

Data Structures

- struct backlight_fct_t

Enumerations

- enum backlight_status_t {
  backlight_status_success,
  backlight_status_noinit,
  backlight_status_error }

Enumeration Type Documentation

- enum backlight_level_t {
  BLIGHT_LEVEL_OFF,
  BLIGHT_LEVEL_LOW,
  BLIGHT_LEVEL_MEDIUM,
  BLIGHT_LEVEL_HIGH }

Functions

- void Backlight_Init_Driver (backlight_fct_t *fct)
- void Backlight_Deinit_Driver ()
- backlight_status_t Backlight_Init_Hw ()
- backlight_status_t Backlight_SetLevel (backlight_level_t level)
- backlight_status_t Backlight_GetLevel (backlight_level_t *level)

3.2 Data Structure Documentation

3.2.1 struct backlight_fct_t

Structure of external functions or values.

Data Fields

- void(* connect_hw)(void)
- void(* disconnect_hw)(void)
- void(* set_level)(uint8_t level)

3.2.1.1 Field Documentation

3.2.1.1.1 void(* connect_hw) (void)

External function to activate the backlight hardware.

3.2.1.1.2 void(* disconnect_hw) (void)

External function to deactivate the backlight hardware.

3.2.1.1.3 void(* set_level) (uint8_t level)

External function to set the backlight level.

3.3 Enumeration Type Documentation

3.3.1 enum backlight_status_t

Status return codes.

Enumerator

backlight_status_success Success.
backlight_status_noinit Hardware or driver not initialized.
backlight_status_error Internal error.

3.3.2 enum backlight_level_t

Predefined backlight levels.

Enumerator

BLIGHT_LEVEL_OFF Backlight Off.
BLIGHT_LEVEL_LOW Lowest predefined Backlight level.
BLIGHT_LEVEL_MEDIUM Medium predefined Backlight level.
BLIGHT_LEVEL_HIGH Highest predefined Backlight level.

3.4 Function Documentation

3.4.1 void Backlight_Init_Driver (backlight_fct_t * *fct*)

Initialize the LCD backlight driver.

Parameters

fct	Pointer to a structure of external functions or values

3.4.2 void Backlight_Deinit_Driver ( )

De-initialize the LCD backlight driver.

3.4.3 backlight_status_t Backlight_Init_Hw ( )

Initialize the LCD backlight hardware.

Returns

Status value (0 for success)

3.4.4 backlight_status_t Backlight_SetLevel (backlight_level_t *level*)

Set current backlight level.

Function Documentation

Parameters

level	Backlight level (Off=0, Low=1, Medium=2, High=3)

Returns

Status value (0 for success)

3.4.5 backlight_status_t Backlight_GetLevel (backlight_level_t * *level*)

Get current backlight level.

Parameters

level	Pointer to backlight level (Off=0, Low=1, Medium=2, High=3)

Returns

Status value (0 for success)

Chapter 4
Battery Sensor

4.1 Overview

This module provides the API to monitor the battery charging state and level.

The basic steps to monitor the battery are as follows:

1. Initialize the driver with callback functions (BatterySensor_Init_Driver)
2. Initialize the hardware (BatterySensor_Init_Hw)
3. Acquire the current battery state (BatterySensor_GetState)
4. If the battery monitoring is not needed anymore, de-initialize the driver (BatterySensor_Deinit_↩ Driver).

The battery charging state is controlled in hardware directly and shows the actual state.

Example - Sample application code to monitor the battery level without error management

```
#include "battery.h"

battery_fct_t bat_fct;
uint8_t bat_level;
uint8_t bat_state;

bat_fct.WaitMsec = WaitMs;  // wait callback function (in ms)

BatterySensor_Init_Driver(&bat_fct);
BatterySensor_Init_Hw();
// acquire current battery state
BatterySensor_GetState(&bat_level, &bat_state);
printf("Battery level = %d, charging state = %d\n", bat_level, bat_state);
[..]
// if the battery monitoring is not needed anymore, the driver can be de-initialized
printf("Deinitialize battery\n");
BatterySensor_Deinit_Driver();
```

Data Structures

- struct ptbattery_fct_t

Macros

- #define **VBAT_MV_MIN**
- #define **VBAT_MV_MAX**
- #define **HYSTERESIS_MV**

Enumerations

- enum battery_status_t {
  battery_status_success,
  battery_status_noinit,
  battery_status_error }

Functions

- void BatterySensor_Init_Driver (ptbattery_fct_t FCT)
- void BatterySensor_Deinit_Driver ()
- battery_status_t BatterySensor_Init_Hw ()
- battery_status_t BatterySensor_GetState (uint8_t *batteryPercentLevel, uint8_t *batteryCharging↩
  State)

4.2 Data Structure Documentation

4.2.1 struct battery_fct_t

Structure of external functions or values.

Data Fields

- void(* WaitMsec)(uint32_t tms)

4.2.1.1 Field Documentation

4.2.1.1.1 void(* WaitMsec) (uint32_t tms)

Wait function in milliseconds.

4.3 Enumeration Type Documentation

4.3.1 enum battery_status_t

Status return codes.

Enumerator

battery_status_success Success.
battery_status_noinit Hardware or driver not initialized.
battery_status_error Internal error.

4.4 Function Documentation

4.4.1 void BatterySensor_Init_Driver (ptbattery_fct_t *FCT*)

Initialize the battery sensor driver.

Parameters

FCT	Pointer to a structure of external functions or values

4.4.2 void BatterySensor_Deinit_Driver ( )

De-initialize the battery sensor driver.

4.4.3 battery_status_t BatterySensor_Init_Hw ( )

Initialize the battery sensor hardware.

Returns

Status value (0 for success)

4.4.4 battery_status_t BatterySensor_GetState (uint8_t * *batteryPercentLevel,* uint8_t * *batteryChargingState*)

Get current battery state.

Parameters

battery↩ PercentLevel	Pointer to battery level percentage (0-100)
battery↩ ChargingState	Pointer to charging state (0 - not charging, 1 - charging)

Returns

Status value (0 for success)

Chapter 5
Buzzer

5.1 Overview

This module provides the API to control the buzzer.

The basic steps to operate the buzzer are as follows:

1. Initialize the driver (Buzzer_Init_Driver)
2. Initialize the hardware (Buzzer_Init_Hw)
3. Turn on the buzzer (Buzzer_On) or off (Buzzer_Off)
4. If the buzzer is not needed anymore, de-initialize the driver (Buzzer_Deinit_Driver).

Example - Sample application code to turn on or off the buzzer without error management

```
#include "buzzer.h"

Buzzer_Init_Driver();
Buzzer_Init_Hw();
printf("Turn on the buzzer\n");
Buzzer_On();
[..]
printf("Turn off the buzzer\n");
Buzzer_Off();
// if the buzzer is not needed anymore, the driver can be de-initialized
printf("De-initialize buzzer (it will be turned off)\n");
Buzzer_Deinit_Driver();
```

Enumerations

- enum buzzer_status_t {
  buzzer_status_success,
  buzzer_status_noinit,
  buzzer_status_error }

Functions

- void Buzzer_Init_Driver ()
- void Buzzer_Deinit_Driver ()
- buzzer_status_t Buzzer_Init_Hw ()
- buzzer_status_t Buzzer_On (void)
- buzzer_status_t Buzzer_Off (void)

5.2 Enumeration Type Documentation

5.2.1 enum buzzer_status_t

Status return codes.

Enumerator

buzzer_status_success Success.
buzzer_status_noinit Hardware or driver not initialized.
buzzer_status_error Internal error.

5.3 Function Documentation

5.3.1 void Buzzer_Init_Driver ( )

Initialize the buzzer driver.

5.3.2 void Buzzer_Deinit_Driver ( )

De-initialize the buzzer driver.

5.3.3 buzzer_status_t Buzzer_Init_Hw ( )

Initialize the buzzer hardware.

Returns

Status value (0 for success)

5.3.4 buzzer_status_t Buzzer_On (void)

Put the buzzer on.

Returns

Status value (0 for success)

5.3.5 buzzer_status_t Buzzer_Off (void)

Put the buzzer off.

Returns

Status value (0 for success)

Chapter 6
CCS811 Air Quality Sensor

6.1 Overview

This module provides the API to control and monitor the CCS811 air quality sensor through an I2C interface.

The basic steps to operate the CCS811 are as follows:

1. Initialize the driver with callback functions (CCS811_Init_Driver)
2. Initialize the hardware (CCS811_Init_Hw)
3. Check data availability (CCS811_dataAvailable)
4. Trigger all data acquisition (CCS811_readAlgorithmResults)
5. Read the relevant data (for example CCS811_getCO2)
6. If the CCS811 is not needed anymore, de-initialize the driver (CCS811_Deinit_Driver). The CCS811 will be switched off. It allows to eventually release shared resources.

Example - Sample application code to set CCS811 without error management

```
#include "CCS811.h"

CCS811_fct_t ccs811_fct;
uint8_t ready;
uint16_t uCO2;

ccs811_fct.connect_hw = Ccs811_Connect;       // callback function to activate CCS811 hardware
       resource
ccs811_fct.disconnect_hw = Ccs811_Disconnect; // callback function to deactivate CCS811
       hardware resource
ccs811_fct.I2C_Read = I2c_Read;               // callback function for I2C read
ccs811_fct.I2C_Write = I2c_Write;             // callback function for I2C write
ccs811_fct.WaitMsec = WaitMs;                 // wait callback function (in ms)

CCS811_Init_Driver(&ccs811_fct);
CCS811_Init_Hw();
// check new data availability
CCS811_dataAvailable(&ready);
if (ready == 1) {
    CCS811_readAlgorithmResults();
    uCO2 = CCS811_getCO2();
    printf("The current CO2 value is: %d\n", uCO2);
}
[..]
// if the CCS811 is not needed anymore, the driver can be de-initialized
printf("Deinitialize CCS811 (it will be turned off)\n");
CCS811_Deinit_Driver();
```

Data Structures

- struct ptCCS811_fct_t

Macros

- #define **CCS811_I2C_ADDRESS**
- #define **CCS811_STATUS**
- #define **CCS811_MEAS_MODE**
- #define **CCS811_ALG_RESULT_DATA**
- #define **CCS811_RAW_DATA**
- #define **CCS811_ENV_DATA**
- #define **CCS811_NTC**
- #define **CCS811_THRESHOLDS**
- #define **CCS811_BASELINE**
- #define **CCS811_HW_ID**
- #define **CCS811_HW_VERSION**
- #define **CCS811_FW_BOOT_VERSION**
- #define **CCS811_FW_APP_VERSION**
- #define **CCS811_ERROR_ID**
- #define **CCS811_APP_START**
- #define **CCS811_SW_RESET**
- #define **CCS811_INTERRUPT_DRIVEN**
- #define **CCS811_THRESHOLDS_ENABLED**

Enumerations

- enum CCS811_status {
  CCS811_SUCCESS,
  CCS811_ID_ERROR,
  CCS811_I2C_ERROR,
  CCS811_INTERNAL_ERROR,
  CCS811_NOINIT_ERROR,
  CCS811_GENERIC_ERROR }

Functions

- void CCS811_Init_Driver (ptCCS811_fct_t FCT)
- void CCS811_Deinit_Driver ()
- CCS811_status CCS811_Init_Hw (void)
- CCS811_status CCS811_readAlgorithmResults (void)
- CCS811_status CCS811_checkForStatusError (uint8_t *StatusError)
- CCS811_status CCS811_dataAvailable (uint8_t *value)
- CCS811_status CCS811_appValid (uint8_t *value)
- CCS811_status CCS811_getErrorRegister (uint8_t *value)
- CCS811_status CCS811_getBaseline (unsigned int *baseline)
- CCS811_status CCS811_setBaseline (uint16_t input)
- CCS811_status CCS811_setInterrupts (uint8_t fct)
- CCS811_status CCS811_setThresholds (uint8_t fct)
- CCS811_status CCS811_setDriveMode (uint8_t mode)
- CCS811_status CCS811_setEnvironmentalData (float relativeHumidity, float temperature)
- void CCS811_setRefResistance (float input)
- CCS811_status CCS811_readNTC (void)
- uint16_t CCS811_getTVOC (void)
- uint16_t CCS811_getCO2 (void)
- float CCS811_getResistance (void)
- float CCS811_getTemperature (void)

6.2 Data Structure Documentation

6.2.1 struct CCS811_fct_t

Structure of external functions or values.

Data Fields

- void(∗ connect_hw)(void)
- void(∗ disconnect_hw)(void)
- uint8_t(∗ I2C_Read)(uint8_t device_addr, uint8_t ∗writeBuf, uint32_t writeSize, uint8_t ∗readBuf, uint32_t readSize)
- uint8_t(∗ I2C_Write)(uint8_t device_addr, uint8_t ∗writeBuf, uint32_t writeSize)
- void(∗ WaitMsec)(uint32_t tms)

6.2.1.1 Field Documentation

6.2.1.1.1 void(∗ connect_hw) (void)

External function to activate the CCS811 hardware.

6.2.1.1.2 void(∗ disconnect_hw) (void)

External function to deactivate the CCS811 hardware.

6.2.1.1.3 uint8_t(∗ I2C_Read) (uint8_t device_addr, uint8_t ∗writeBuf, uint32_t writeSize, uint8_t ∗readBuf, uint32_t readSize)

External I2C read function.

6.2.1.1.4 uint8_t(∗ I2C_Write) (uint8_t device_addr, uint8_t ∗writeBuf, uint32_t writeSize)

External I2C write function.

6.2.1.1.5 void(∗ WaitMsec) (uint32_t tms)

Wait function in milliseconds.

6.3 Enumeration Type Documentation

6.3.1 enum CCS811_status

Status return codes.

Enumerator

CCS811_SUCCESS Success.
CCS811_ID_ERROR Bad hardware identifier.
CCS811_I2C_ERROR I2C read/write error.
CCS811_INTERNAL_ERROR Internal hardware error.
CCS811_NOINIT_ERROR Hardware or driver not initialized.
CCS811_GENERIC_ERROR Generic error (settings not allowed)

6.4 Function Documentation

6.4.1 void CCS811_Init_Driver (ptCCS811_fct_t *FCT*)

Initialize CCS811 driver.

Parameters

FCT	Pointer to a structure of external functions or values

6.4.2 void CCS811_Deinit_Driver ( )

De-initialize CCS811 driver.

6.4.3 CCS811_status CCS811_Init_Hw (void)

Initialize CCS811 hardware.

Returns

Status value (0 for success)

6.4.4 CCS811_status CCS811_readAlgorithmResults (void)

Read algorithm results.

Note

Update the total volatile organic compounds (TVOC) in parts per billion (PPB) and the CO2 value.

Returns

Status value (0 for success)

6.4.5 CCS811_status CCS811_checkForStatusError (uint8_t * *StatusError*)

Check if error bit is set.

Parameters

StatusError	Pointer to error status bit value

Returns

Status value (0 for success)

6.4.6 CCS811_status CCS811_dataAvailable (uint8_t * *value*)

Check if data is available.

Note

Based on DATA_READ flag in the status register.

Parameters

value	Pointer to available data bit value

Returns

Status value (0 for success)

6.4.7 CCS811_status CCS811_appValid (uint8_t * *value*)

Check if APP_VALID is set.

Note

Based on APP_VALID flag in the status register.

Parameters

value	Pointer to APP_VALID bit value

Returns

Status value (0 for success)

6.4.8 CCS811_status CCS811_getErrorRegister (uint8_t * *value*)

Get the error register value.

Note

Based on ERROR_ID register.

Parameters

value	Pointer to error value

Returns

Status value (0 for success)

6.4.9 CCS811_status CCS811_getBaseline (unsigned int * *baseline*)

Get the baseline value.

Note

It is used for telling sensor what 'clean' air is.
Put the sensor in clean air and record this value.

Parameters

baseline	Pointer to baseline value

Returns

Status value (0 for success)

6.4.10 CCS811_status CCS811_setBaseline (uint16_t *input*)

Set the baseline value.

Note

It is used for telling sensor what 'clean' air is.
Put the sensor in clean air and record this value.

Parameters

input	Baseline value

Returns

Status value (0 for success)

6.4.11 CCS811_status CCS811_setInterrupts (uint8_t *fct*)

Enable or disable the interrupts.

Note

It clears/sets the nINT signal.

Parameters

fct	Interrupt mode (0 for disable, 1 for enable)

Returns

Status value (0 for success)

6.4.12 CCS811_status CCS811_setThresholds (uint8_t *fct*)

Enable or disable the interrupt thresholds.

Parameters

fct	Threshold mode (0 for disable, 1 for enable)

Returns

Status value (0 for success)

6.4.13 CCS811_status CCS811_setDriveMode (uint8_t *mode*)

Set the drive mode.

Note

Mode 0 = Idle
Mode 1 = read every 1s
Mode 2 = every 10s
Mode 3 = every 60s
Mode 4 = RAW mode

Parameters

mode	Drive mode

Returns

Status value (0 for success)

6.4.14 CCS811_status CCS811_setEnvironmentalData (float *relativeHumidity,* float *temperature*)

Set environmental data.

Note

Given a temperature and humidity, use these data for better compensation.

Parameters

relative↩ Humidity	Relative humidity, value within [0,100]
temperature	Temperature (Celsius), value within [-25,+50]

Returns

Status value (0 for success)

6.4.15 void CCS811_setRefResistance (float *input*)

Set ref resistance.

Parameters

input	Ref resistance

6.4.16 CCS811_status CCS811_readNTC (void)

Read NTC.

Returns

Status value (0 for success)

6.4.17 uint16_t CCS811_getTVOC (void)

Get Total Volatile Organic Compound (TVOC) value.

Returns

TVOC value (0ppb to 1187ppb)

6.4.18 uint16_t CCS811_getCO2 (void)

Get the equivalent CO2 value.

Returns

eCO2 value (400ppm to 8192ppm)

6.4.19 float CCS811_getResistance (void)

Get resistance value.

Returns

Resistance value

6.4.20 float CCS811_getTemperature (void)

Get temperature value.

Returns

Temperature value (Celsius)

Chapter 7
ENS210 Relative Humidity and Temperature Sensor

7.1 Overview

This module provides the API to operate an ENS210 relative humidity and temperature sensor with I2C interface.

Basic steps to operate the sensor are as follows:

1. Set Run mode (ENS210_SensRun_Set)
2. Start measurement (ENS210_SensStart_Set)
3. Wait for measurement to complete
4. Read measurement (ENS210_TVal_Get, ENS210_HVal_Get, ENS210_THVal_Get)

Please refer to ENS210 Reference Driver and Porting Guide for more details on platform porting. In this module, names T and H have been used to refer to temperature and relative humidity respectively to comply with ENS210 datasheet naming convention.

Example 1 - Sample application code to measure temperature and relative humidity without error checking

```
uint32_t T_Raw, H_Raw;
int32_t T_mCelsius, T_mFahrenheit, T_mKelvin, H_Percent;

//Set runmode, start measurement, wait, read measurement (for both T and H)
ENS210_SensRun_Set(ENS210_SENSRUN_T_MODE_SINGLE_SHOT |
      ENS210_SENSRUN_H_MODE_SINGLE_SHOT);
ENS210_SensStart_Set(ENS210_SENSSTART_T_START |
      ENS210_SENSSTART_H_START);
WaitMsec(ENS210_T_H_CONVERSION_TIME_MS);
ENS210_THVal_Get(&T_Raw,&H_Raw);

//Convert the raw temperature to milli Kelvin
T_mKelvin = ENS210_ConvertRawToKelvin(T_Raw, 1000);
//Convert the raw temperature to milli Celsius
T_mCelsius = ENS210_ConvertRawToCelsius(T_Raw, 1000);
//Convert the raw temperature to milli Fahrenheit
T_mFahrenheit = ENS210_ConvertRawToFahrenheit(T_Raw, 1000);
printf("T crc ok = %s\n", ENS210_IsCrcOk(T_Raw)  ? "yes" : "no");
printf("T valid = %s \n", ENS210_IsDataValid(T_Raw) ? "yes" : "no");
//Update the int32_t format specifier (%ld) based on platform word-size
printf("T = %ld mK %ld mC %ld mF \n", T_mKelvin, T_mCelsius, T_mFahrenheit);

//Convert the raw relative humidity to milli %
H_Percent = ENS210_ConvertRawToPercentageH(H_Raw, 1000);
printf("H crc ok = %s\n", ENS210_IsCrcOk(H_Raw)  ? "yes" : "no");
printf("H valid = %s \n", ENS210_IsDataValid(H_Raw) ? "yes" : "no");
//Update the int32_t format specifier (%ld) based on platform word-size
printf("H = %ld m%%\n", H_Percent);
```

Example 2 - Sample application code to measure relative humidity with error checking

```
uint32_t H_Raw;
int32_t H_Percent;
int status;
bool i2cOk;

i2cOk = true; //Start accumulating I2C transaction errors

status = ENS210_SensRun_Set(ENS210_SENSRUN_H_MODE_SINGLE_SHOT
      );
i2cOk &= status==I2C_RESULT_OK;

status = ENS210_SensStart_Set(ENS210_SENSSTART_H_START);
i2cOk &= status==I2C_RESULT_OK;

WaitMsec(ENS210_T_H_CONVERSION_TIME_MS);

status = ENS210_HVal_Get(&H_Raw);
i2cOk &= status==I2C_RESULT_OK;

if( !i2cOk ) {
    printf("H i2c error\n")
} else if( !ENS210_IsCrcOk(H_Raw) ) {
    printf("H crc error\n")
} else if( !ENS210_IsDataValid(H_Raw) ) {
    printf("H data invalid\n")
} else {
    //Convert the raw relative humidity to milli %
    H_Percent = ENS210_ConvertRawToPercentageH(H_Raw,1000);
    //Update the int32_t format specifier (%ld) based on platform word-size
    printf("H = %ld m%%\n", H_Percent);
}
```

Data Structures

- struct pens210_IoFunc_t
- struct ENS210_Ids_t
- struct ens210_meas_data_t

Macros

- #define ENS210_OSFREE_DRIVER_VERSION
- #define ENS210_T_H_CONVERSION_TIME_MS
- #define ENS210_T_CONVERSION_TIME_MS
- #define ENS210_BOOTING_TIME_MS
- #define ENS210_RESET_WAIT_TIME_MS
- #define ENS210_I2C_SLAVE_ADDRESS
- #define ENS210_SYSCTRL_LOWPOWER_ENABLE
- #define ENS210_SYSCTRL_LOWPOWER_DISABLE
- #define ENS210_SYSCTRL_RESET_ENABLE
- #define ENS210_SYSCTRL_RESET_DISABLE
- #define ENS210_SYSSTAT_MODE_STANDBY
- #define ENS210_SYSSTAT_MODE_ACTIVE
- #define ENS210_SENSRUN_T_MODE_SINGLE_SHOT
- #define ENS210_SENSRUN_T_MODE_CONTINUOUS
- #define ENS210_SENSRUN_H_MODE_SINGLE_SHOT
- #define ENS210_SENSRUN_H_MODE_CONTINUOUS
- #define ENS210_SENSSTART_T_START
- #define ENS210_SENSSTART_H_START

- #define ENS210_SENSSTOP_T_STOP
- #define ENS210_SENSSTOP_H_STOP
- #define ENS210_SENSSTAT_T_STAT_IDLE
- #define ENS210_SENSSTAT_T_STAT_ACTIVE
- #define ENS210_SENSSTAT_H_STAT_IDLE
- #define ENS210_SENSSTAT_H_STAT_ACTIVE

Enumerations

- enum ens210_status_t {
  ens210_success,
  ens210_I2C_error,
  ens210_invalid_ID,
  ens210_Tdata_CRC_error,
  ens210_Hdata_CRC_error,
  ens210_T_invalid_data,
  ens210_H_invalid_data,
  ens210_wrong_parameter,
  ens210_noinit }
- enum measurement_mode {
  mode_TH,
  mode_Tonly,
  mode_Honly }

Functions

- void ENS210_Init_Driver (ens210_IoFunc_t *pIoFunc)
- void ENS210_Deinit_Driver ()
- ens210_status_t ENS210_Init_Hw (void)
- ens210_status_t ENS210_SysCtrl_Set (uint8_t sysCtrl)
- ens210_status_t ENS210_SysCtrl_Get (uint8_t *sysCtrl)
- ens210_status_t ENS210_SysStat_Get (uint8_t *sysStat)
- ens210_status_t ENS210_SensRun_Set (uint8_t sensRun)
- ens210_status_t ENS210_SensRun_Get (uint8_t *sensRun)
- ens210_status_t ENS210_SensStart_Set (uint8_t sensStart)
- ens210_status_t ENS210_SensStop_Set (uint8_t sensStop)
- ens210_status_t ENS210_SensStat_Get (uint8_t *sensStat)
- ens210_status_t ENS210_TVal_Get (uint32_t *traw)
- ens210_status_t ENS210_HVal_Get (uint32_t *hraw)
- ens210_status_t ENS210_THVal_Get (uint32_t *traw, uint32_t *hraw)
- ens210_status_t ENS210_Ids_Get (ENS210_Ids_t *ids)
- ens210_status_t ENS210_Measure (uint8_t meas_mode, ens210_meas_data_t *results)

7.2 Data Structure Documentation

7.2.1 struct ens210_IoFunc_t

Data Fields

- uint8_t(∗ I2C_Read)(uint8_t device_addr, uint8_t ∗writeBuf, uint32_t writeSize, uint8_t ∗readBuf, uint32_t readSize)
- uint8_t(∗ I2C_Write)(uint8_t device_addr, uint8_t ∗writeBuf, uint32_t writeSize)
- void(∗ WaitMsec)(uint32_t millisec)

7.2.1.1 Field Documentation

7.2.1.1.1 uint8_t(∗ I2C_Read) (uint8_t device_addr, uint8_t ∗writeBuf, uint32_t writeSize, uint8_t ∗readBuf, uint32_t readSize)

External I2C read function.

7.2.1.1.2 uint8_t(∗ I2C_Write) (uint8_t device_addr, uint8_t ∗writeBuf, uint32_t writeSize)

External I2C write function.

7.2.1.1.3 void(∗ WaitMsec) (uint32_t millisec)

Wait function in milliseconds.

7.2.2 struct ENS210_Ids_t

ENS210 ID block structure.

Data Fields

uint16_t	partId	Part ID.
uint8_t	uId[8]	Unique Identifier 8 bytes.

7.2.3 struct ens210_meas_data_t

Structure of measurement data.

Data Fields

int32_t	T_Celsius	Temperature in Celsius.
int32_t	T_Fahrenheit	Temperature in Fahrenheit.
int32_t	T_Kelvin	Temperature in Kelvin.
int32_t	T_mCelsius	Temperature in milliCelsius.
int32_t	T_mFahrenheit	Temperature in milliFahrenheit.
int32_t	T_mKelvin	Temperature in milliKelvin.
int32_t	H_Percent	Relative Humidity to %.
int32_t	H_mPercent	Relative Humidity to milli%.

7.3 Macro Definition Documentation

7.3.1 #define ENS210_OSFREE_DRIVER_VERSION

ENS210 os-free driver version info.

7.3.2 #define ENS210_T_H_CONVERSION_TIME_MS

ENS210 T and H conversion time in milliseconds.

Refer to ENS210 data sheet for timing information.

7.3.3 #define ENS210_T_CONVERSION_TIME_MS

ENS210 T conversion time in milliseconds.

7.3.4 #define ENS210_BOOTING_TIME_MS

ENS210 Booting time in milliseconds.

7.3.5 #define ENS210_RESET_WAIT_TIME_MS

ENS210 Reset time in milliseconds.

7.3.6 #define ENS210_I2C_SLAVE_ADDRESS

ENS210 I2C slave address.

7.3.7 #define ENS210_SYSCTRL_LOWPOWER_ENABLE

ENS210 SysCtrl register: Low power enable.

7.3.8 #define ENS210_SYSCTRL_LOWPOWER_DISABLE

ENS210 SysCtrl register: Low power disable.

7.3.9 #define ENS210_SYSCTRL_RESET_ENABLE

ENS210 SysCtrl register: Reset enable.

7.3.10 #define ENS210_SYSCTRL_RESET_DISABLE

ENS210 SysCtrl register: Reset disable.

7.3.11 #define ENS210_SYSSTAT_MODE_STANDBY

ENS210 SysStat register: Standby or Booting mode.

7.3.12 #define ENS210_SYSSTAT_MODE_ACTIVE

ENS210 SysStat register: Active mode.

7.3.13 #define ENS210_SENSRUN_T_MODE_SINGLE_SHOT

ENS210 SensRun register: temperature single shot mode.

7.3.14 #define ENS210_SENSRUN_T_MODE_CONTINUOUS

ENS210 SensRun register: temperature continuous mode.

7.3.15 #define ENS210_SENSRUN_H_MODE_SINGLE_SHOT

ENS210 SensRun register: relative humidity single shot mode.

7.3.16 #define ENS210_SENSRUN_H_MODE_CONTINUOUS

ENS210 SensRun register: relative humidity continuous mode.

7.3.17 #define ENS210_SENSSTART_T_START

ENS210 SensStart register: T sensor start.

7.3.18 #define ENS210_SENSSTART_H_START

ENS210 SensStart register: H sensor start.

7.3.19 #define ENS210_SENSSTOP_T_STOP

ENS210 SensStop register: T sensor stop.

7.3.20 #define ENS210_SENSSTOP_H_STOP

ENS210 SensStop register: H sensor stop.

7.3.21 #define ENS210_SENSSTAT_T_STAT_IDLE

ENS210 SensStat register: T sensor idle.

7.3.22 #define ENS210_SENSSTAT_T_STAT_ACTIVE

ENS210 SensStat register: T sensor active.

7.3.23 #define ENS210_SENSSTAT_H_STAT_IDLE

ENS210 SensStat register: H sensor idle.

7.3.24 #define ENS210_SENSSTAT_H_STAT_ACTIVE

ENS210 SensStat register: H sensor active.

7.4 Enumeration Type Documentation

7.4.1 enum ens210_status_t

Status return codes.

Enumerator

ens210_success Function returned successfully.
ens210_I2C_error I2C Error.
ens210_invalid_ID Invalid ID.
ens210_Tdata_CRC_error CRC error for temperature data.
ens210_Hdata_CRC_error CRC error for humidity data.
ens210_T_invalid_data Temperature Data is invalid.
ens210_H_invalid_data Humidity Data is invalid.
ens210_wrong_parameter Wrong Parameter entered.
ens210_noinit ENS210 was not initialized.

7.4.2 enum measurement_mode

Measurement mode of Sensor.

Enumerator

mode_TH ENS210 set to measure both temperature and humidity.
mode_Tonly ENS210 set to measure temperature only.
mode_Honly ENS210 set to measure humidity only.

7.5 Function Documentation

7.5.1 void ENS210_Init_Driver (ens210_IoFunc_t * *ploFunc*)

Initialize ENS210 driver.

Parameters

pIoFunc	: Pointer to a structure of external functions or values

7.5.2 void ENS210_Deinit_Driver ( )

De-initialize ENS210 driver.

7.5.3 ens210_status_t ENS210_Init_Hw (void)

Initialize ENS210 hardware.

Returns

The return status value (0 for success)

7.5.4 ens210_status_t ENS210_SysCtrl_Set (uint8_t *sysCtrl*)

Set ENS210 SysCtrl register; enabling reset and/or low power.

Parameters

sysCtrl	: Mask composed of ENS210_SYSCTRL_xxx macros.

Returns

The return status value (0 for success)

7.5.5 ens210_status_t ENS210_SysCtrl_Get (uint8_t * *sysCtrl*)

Get ENS210 SysCtrl register.

Parameters

sysCtrl	: Pointer to receive value of the register. Must not be null.

Returns

The return status value (0 for success)

Note

If the return indicates I2C failure, the value of the out parameter is undefined.

7.5.6 ens210_status_t ENS210_SysStat_Get (uint8_t * *sysStat*)

Get ENS210 SysStat register.

Function Documentation

Parameters

sysStat	: Pointer to receive value of the register. Must not be null.

Returns

The return status value (0 for success)

Note

If the return indicates I2C failure, the value of the out parameter is undefined.

7.5.7 ens210_status_t ENS210_SensRun_Set (uint8_t *sensRun*)

Set ENS210 SensRun register; set the run mode single shot/continuous for T and H sensors.

Parameters

sensRun	: Mask composed of ENS210_SENSRUN_xxx macros.

Returns

The return status value (0 for success)

7.5.8 ens210_status_t ENS210_SensRun_Get (uint8_t * *sensRun*)

Get ENS210 SensRun register.

Parameters

sensRun	: Pointer to receive value of the register. Must not be null.

Returns

The return status value (0 for success)

Note

If the return indicates I2C failure, the value of the out parameter is undefined.

7.5.9 ens210_status_t ENS210_SensStart_Set (uint8_t *sensStart*)

Set ENS210 SensStart register; starts the measurement for T and/or H sensors.

Parameters

sensStart	: Mask composed of ENS210_SENSSTART_xxx macros.

Returns

The return status value (0 for success)

7.5.10 ens210_status_t ENS210_SensStop_Set (uint8_t *sensStop*)

Set ENS210 SensStop register; stops the measurement for T and/or H sensors.

Parameters

sensStop	: Mask composed of ENS210_SENSSTOP_xxx macros.

Returns

The return status value (0 for success)

7.5.11 ens210_status_t ENS210_SensStat_Get (uint8_t * *sensStat*)

Get ENS210 SensStat register.

Parameters

sensStat	: Pointer to receive value of the register. Must not be null.

Returns

The return status value (0 for success)

7.5.12 ens210_status_t ENS210_TVal_Get (uint32_t * *traw*)

Get ENS210 TVal register; raw measurement data as well as CRC and valid indication.

Parameters

traw	: Pointer to receive value of the register. Must not be null.

Returns

The return status value (0 for success)

Note

Use ENS210_IsCrcOk and ENS210_IsDataValid before using the measurement data.
Use ENS210_ConvertRawToXXX to convert raw data to standard units.
If the return indicates I2C failure, the value of the out parameter is undefined.

7.5.13 ens210_status_t ENS210_HVal_Get (uint32_t * *hraw*)

Get ENS210 HVal register; raw measurement data as well as CRC and valid indication.

Parameters

hraw	: Pointer to receive value of register. Must not be null.

Returns

The return status value (0 for success)

Note

Use ENS210_IsCrcOk and ENS210_IsDataValid before using the measurement data.
If the return indicates I2C failure, the value of the out parameter is undefined.

7.5.14 ens210_status_t ENS210_THVal_Get (uint32_t * *traw,* uint32_t * *hraw*)

Get ENS210 TVal and HVal registers; raw measurement data as well as CRC and valid indication.

Parameters

traw	: Pointer to receive value of TVal register. Must not be null.
hraw	: Pointer to receive value of HVal register. Must not be null.

Returns

The return status value (0 for success)

Note

Use ENS210_IsCrcOk and ENS210_IsDataValid before using the measurement data.
If the return indicates I2C failure, the value of the out parameter is undefined.

7.5.15 ens210_status_t ENS210_Ids_Get (ENS210_Ids_t * *ids*)

Get ENS210 Part ID and UID.

Parameters

ids	: Pointer to receive ids. Must not be null.

Returns

The return status value (0 for success)

Note

If this function returns an error, it is suggested to reset the device to bring it to a known state.

7.5.16 ens210_status_t ENS210_Measure (uint8_t *meas_mode,* ens210_meas_data_t * *results*)

Get ENS210 temperature and humidity measurements.

Parameters

meas_mode	: Measurement mode
results	: Pointer to the result structure

Returns

The return status value (0 for success)

Note

mode = 0: temperature and humidity
mode = 1: temperature only
mode = 2: humidity only

Chapter 8
FXAS21002 Gyroscope

8.1 Overview

This module provides the API to operate the FXAS21002 gyroscope sensor through an I2C interface.

The basic steps to operate the FXAS21002 are as follows:

1. Initialize the driver (FXAS21002_I2C_Initialize)
2. Initialize the hardware (FXAS21002_I2C_Configure)
3. Read the raw sensor data (FXAS21002_I2C_ReadData)
4. If the FXAS21002 is not needed anymore, de-initialize the driver (FXAS21002_I2C_Deinit).

Example - Sample application code to set FXAS21002 without error management

```
#include "fxas21002_drv.h"
#include "fsl_i2c_cmsis.h"

#define I2C_S1_DRIVER Driver_I2C1
#define I2C_S1_DEVICE_INDEX I2C1_INDEX
#define FXAS21002_I2C_SLAVE_ADDRESS 0x20
fxas21002_i2c_sensorhandle_t FXAS21002drv;
const registerreadlist_t fxas21002_Output_Values[] = {
    {.readFrom = FXAS21002_OUT_X_MSB, .numBytes = FXAS21002_GYRO_DATA_SIZE},
      __END_READ_DATA__};
uint8_t data[FXAS21002_GYRO_DATA_SIZE];
int16_t rotspeed[3] = {0, 0, 0};

// initialization
FXAS21002_I2C_Initialize(&FXAS21002drv, &I2C_S1_DRIVER, I2C_S1_DEVICE_INDEX,
      FXAS21002_I2C_SLAVE_ADDRESS,
            FXAS21002_WHO_AM_I_WHOAMI_PROD_VALUE);
FXAS21002_I2C_Configure(&FXAS21002drv, fxas21002_Config_Normal);
// read raw data
FXAS21002_I2C_ReadData(&FXAS21002drv, fxas21002_Output_Values, data);
// convert raw data to signed 16-bit container
rotspeed[0] = ((int16_t)data[0] << 8) | data[1];
rotspeed[1] = ((int16_t)data[2] << 8) | data[3];
rotspeed[2] = ((int16_t)data[4] << 8) | data[5];
printf("gyroscope data: %d, %d, %d\n", rotspeed[0], rotspeed[1], rotspeed[2]);
[..]
// if the FXAS21002 is not needed anymore, the driver can be de-initialized
FXAS21002_I2C_Deinit();
```

Data Structures

- struct fxas21002_spi_sensorhandle_t
- struct fxas21002_i2c_sensorhandle_t
- struct fxas21002_gyrodata_t

Macros

- #define FXAS21002_GYRO_DATA_SIZE

- #define FXAS21002_SPI_MAX_MSG_SIZE
- #define FXAS21002_SPI_CMD_LEN
- #define FXAS21002_SS_ACTIVE_VALUE

Functions

- int32_t FXAS21002_I2C_Initialize (fxas21002_i2c_sensorhandle_t *pSensorHandle, ARM_DRIVER_I2C *pBus, uint8_t index, uint16_t sAddress, uint8_t whoAmi)
- void FXAS21002_I2C_SetIdleTask (fxas21002_i2c_sensorhandle_t *pSensorHandle, registeridlefunction_t idleTask, void *userParam)
- int32_t FXAS21002_I2C_Configure (fxas21002_i2c_sensorhandle_t *pSensorHandle, const registerwritelist_t *pRegWriteList)
- int32_t FXAS21002_I2C_ReadData (fxas21002_i2c_sensorhandle_t *pSensorHandle, const registerreadlist_t *pReadList, uint8_t *pBuffer)
- int32_t FXAS21002_I2C_Deinit (fxas21002_i2c_sensorhandle_t *pSensorHandle)
- int32_t FXAS21002_SPI_Initialize (fxas21002_spi_sensorhandle_t *pSensorHandle, ARM_DRIVER_SPI *pBus, uint8_t index, void *pSlaveSelect, uint8_t whoAmi)
- void FXAS21002_SPI_SetIdleTask (fxas21002_spi_sensorhandle_t *pSensorHandle, registeridlefunction_t idleTask, void *userParam)
- int32_t FXAS21002_SPI_Configure (fxas21002_spi_sensorhandle_t *pSensorHandle, const registerwritelist_t *pRegWriteList)
- int32_t FXAS21002_SPI_ReadData (fxas21002_spi_sensorhandle_t *pSensorHandle, const registerreadlist_t *pReadList, uint8_t *pBuffer)
- int32_t FXAS21002_SPI_Deinit (fxas21002_spi_sensorhandle_t *pSensorHandle)
- void FXAS21002_SPI_ReadPreprocess (void *pCmdOut, uint32_t offset, uint32_t size)
- void FXAS21002_SPI_WritePreprocess (void *pCmdOut, uint32_t offset, uint32_t size, void *pWritebuffer)

8.2 Data Structure Documentation

8.2.1 struct fxas21002_spi_sensorhandle_t

This defines the sensor specific information for SPI.

Data Fields

register_DeviceInfo_t	deviceInfo	SPI device context.
ARM_DRIVER_SPI *	pCommDrv	Pointer to the spi driver.
bool	isInitialized	Whether sensor is intialized or not.
spiSlave_Specific_Params_t	slaveParams	Slave Specific Params.

8.2.2 struct fxas21002_i2c_sensorhandle_t

This defines the sensor specific information for I2C.

Data Fields

register↩ DeviceInfo_t	deviceInfo	I2C device context.
ARM_DRIV↩ ER_I2C *	pCommDrv	Pointer to the i2c driver.
bool	isInitialized	whether sensor is initialized or not.
uint16_t	slaveAddress	slave address.

8.2.3 struct fxas21002_gyrodata_t

This structure defines the fxas21002 raw data buffer.

Data Fields

uint32_t	timestamp	
int16_t	gyro[3]	The time, this sample was recorded. The gyro data

8.3 Macro Definition Documentation

8.3.1 #define FXAS21002_GYRO_DATA_SIZE

The size of the FXAS21002 gyro data.

8.3.2 #define FXAS21002_SPI_MAX_MSG_SIZE

The MAX size of SPI message.

8.3.3 #define FXAS21002_SPI_CMD_LEN

The size of the Sensor specific SPI Header.

8.3.4 #define FXAS21002_SS_ACTIVE_VALUE

Is the Slave Select Pin Active Low or High.

8.4 Function Documentation

8.4.1 int32_t FXAS21002_I2C_Initialize (fxas21002_i2c_sensorhandle_t * *pSensorHandle,* ARM_DRIVER_I2C * *pBus,* uint8_t *index,* uint16_t *sAddress,* uint8_t *whoAmi*)

The interface function to initialize the sensor.

This function initialize the sensor and sensor handle.

Parameters

in	*pSensorHandle*	handle to the sensor.
in	*pBus*	pointer to the CMSIS API compatible I2C bus object.
in	*index*	the I2C device number.
in	*sAddress*	slave address of the device on the bus.
in	*whoAmi*	WHO_AM_I value of the device. This should be the first API to be called. Application has to ensure that previous instances of these APIs have exited before invocation. No

Returns

FXAS21002_I2C_Initialize() returns the status .

8.4.2 void FXAS21002_I2C_SetIdleTask (fxas21002_i2c_sensorhandle_t * *pSensorHandle,* registeridlefunction_t *idleTask,* void * *userParam*)

The interface function to set the I2C Idle Task.

Parameters

in	*pSensorHandle*	handle to the sensor handle.
in	*idleTask*	function pointer to the function to execute on I2C Idle Time.
in	*userParam*	the pointer to the user idle ftask parameters.

Returns

void. This can be called any number of times only after FXAS21002_I2C_Initialize(). Application has to ensure that previous instances of these APIs have exited before invocation. No

8.4.3 int32_t FXAS21002_I2C_Configure (fxas21002_i2c_sensorhandle_t * *pSensorHandle,* const registerwritelist_t * *pRegWriteList*)

The interface function to configure he sensor.

This function configure the sensor with requested ODR, Range and registers in the regsiter pair array.

Parameters

in	*pSensorHandle*	handle to the sensor.
in	*pRegWriteList*	pointer to the register list. This can be called any number of times only after FXAS21002_I2C_Initialize(). Application has to ensure that previous instances of these APIs have exited before invocation. No

Returns

FXAS21002_I2C_Configure() returns the status .

8.4.4 int32_t FXAS21002_I2C_ReadData (fxas21002_i2c_sensorhandle_t * *pSensorHandle*, const registerreadlist_t * *pReadList*, uint8_t * *pBuffer*)

The interface function to read the sensor data.

This function read the sensor data out from the device and returns raw data in a byte stream.

Parameters

in	*pSensorHandle*	handle to the sensor.
in	*pReadList*	pointer to the list of device registers and values to read.
out	*pBuffer*	buffer which holds raw sensor data.This buffer may be back to back databuffer based command read in the list. This can be called any number of times only after FXAS21002_I2C_Initialize(). Application has to ensure that previous instances of these APIs have exited before invocation. No

Returns

FXAS21002_I2C_ReadData() returns the status .

8.4.5 int32_t FXAS21002_I2C_Deinit (fxas21002_i2c_sensorhandle_t * *pSensorHandle*)

The interface function to De Initialize sensor.

This function made sensor in a power safe state and de initialize its handle.

Parameters

in	*pSensorHandle*	handle to the sensor. This can be called only after FXAS21002_I2C_Initialize(). Application has to ensure that previous instances of these APIs have exited before invocation. No

Returns

FXAS21002_I2C_Deinit() returns the status .

8.4.6 int32_t FXAS21002_SPI_Initialize (fxas21002_spi_sensorhandle_t * *pSensorHandle*, ARM_DRIVER_SPI * *pBus*, uint8_t *index*, void * *pSlaveSelect*, uint8_t *whoAmi*)

The interface function to initialize the sensor.

This function initializes the sensor and sensor handle.

Parameters

in	*pSensorHandle*	handle to the sensor.
in	*pBus*	pointer to the CMSIS API compatible SPI bus object.
in	*index*	the I2C device number.
in	*pSlaveSelect*	slave select hndle of the device on the bus.
in	*whoAmi*	WHO_AM_I value of the device. This should be the first API to be called. Application has to ensure that previous instances of these APIs have exited before invocation. No

Returns

FXAS21002_SPI_Initialize() returns the status .

8.4.7 void FXAS21002_SPI_SetIdleTask (fxas21002_spi_sensorhandle_t * *pSensorHandle*, registeridlefunction_t *idleTask*, void * *userParam*)

The interface function to set the SPI Idle Task.

Parameters

in	*pSensorHandle*	handle to the sensor handle.
in	*idleTask*	function pointer to the function to execute on SPI Idle Time.
in	*userParam*	the pointer to the user idle ftask parameters.

Returns

void. This can be called any number of times only after FXAS21002_SPI_Initialize(). Application has to ensure that previous instances of these APIs have exited before invocation. No

8.4.8 int32_t FXAS21002_SPI_Configure (fxas21002_spi_sensorhandle_t * *pSensorHandle,* const registerwritelist_t * *pRegWriteList*)

The interface function to configure he sensor.

This function configure the sensor with requested ODR, Range and registers in the regsiter pair array.

Parameters

in	*pSensorHandle*	handle to the sensor.
in	*pRegWriteList*	pointer to the register list. This can be called any number of times only after FXAS21002_SPI_Initialize(). Application has to ensure that previous instances of these APIs have exited before invocation. No

Returns

FXAS21002_SPI_Configure() returns the status .

8.4.9 int32_t FXAS21002_SPI_ReadData (fxas21002_spi_sensorhandle_t * *pSensorHandle,* const registerreadlist_t * *pReadList,* uint8_t * *pBuffer*)

The interface function to read the sensor data.

This function read the sensor data out from the device and returns raw data in a byte stream.

Parameters

in	*pSensorHandle*	handle to the sensor.
in	*pReadList*	pointer to the list of device registers and values to read.
out	*pBuffer*	buffer which holds raw sensor data.This buffer may be back to back databuffer based command read in the list. This can be called any number of times only after FXAS21002_SPI_Initialize(). Application has to ensure that previous instances of these APIs have exited before invocation. No

Returns

FXAS21002_SPI_ReadData() returns the status .

8.4.10 int32_t FXAS21002_SPI_Deinit (fxas21002_spi_sensorhandle_t * *pSensorHandle*)

The interface function to De Initialize sensor.

This function made sensor in a power safe state and de initialize its handle.

Parameters

in	*pSensorHandle*	handle to the sensor. This can be called only after after FXAS21002_SPI_Initialize(). Application has to ensure that previous instances of these APIs have exited before invocation. No

Returns

FXAS21002_SPI_Deinit() returns the status .

8.4.11 void FXAS21002_SPI_ReadPreprocess (void * *pCmdOut*, uint32_t *offset*, uint32_t *size*)

The SPI Read Pre-Process function to generate Sensor specific SPI Message Header.

This function prepares the SPI Read Command Header with register address and R/W bit encoded as the Sensor.

Parameters

out	*pCmdOut*	handle to the output buffer.
in	*offset*	the address of the register to start reading from.
in	*size*	number of bytes to read. None Application has to ensure that previous instances of these APIs have exited before invocation. No

Returns

:: None.

8.4.12 void FXAS21002_SPI_WritePreprocess (void * *pCmdOut*, uint32_t *offset*, uint32_t *size*, void * *pWritebuffer*)

The SPI Write Pre-Process function to generate Sensor specific SPI Message Header.

This function prepares the SPI Write Command Header with register address and R/W bit encoded as the Sensor.

Parameters

out	*pCmdOut*	handle to the output buffer.

in	*offset*	the address of the register to start writing from.
in	*size*	number of bytes to write.
in	*pWritebuffer*	buffer of bytes to write. None Application has to ensure that previous instances of these APIs have exited before invocation. No

Returns

:: None.

Chapter 9
FXOS8700 Combination Accelerometer & Magnetometer

9.1 Overview

This module provides the API to operate the FXOS8700 combination accelerometer/magnetometer sensor through an I2C interface.

The FXOS8700CQ is a small, low-power, 3-axis, linear accelerometer and 3-axis, magnetometer combined into a single package. The device features a 14-bit accelerometer and 16-bit magnetometer ADC resolution along with smart-embedded functions. FXOS8700CQ has dynamically selectable acceleration full-scale ranges of ±2 g/±4 g/±8 g and a fixed magnetic measurement range of ±1200 T. Output data rates (ODR) from 1.563 Hz to 800 Hz are selectable by the user for each sensor. Interleaved magnetic and acceleration data is available at ODR rates of up to 400 Hz.

Usage

Initialization:

```
 #include "fxos8700.h"
 fxos8700_handle_t g_fxosHandle;
 fxos8700_data_t fxos8700_data;

 g_fxosHandle.xfer.slaveAddress = FXOS8700_I2C_SLAVE_ADDRESS;
 g_fxosHandle.base = I2C1;
 g_fxosHandle.i2cHandle = &g_mi2c_handle;

 uint8_t g_sensorRange = 0;
 float g_dataScale =0;

 // Init sensor
if (FXOS8700_Init(&g_fxosHandle) != kStatus_Success)
{
  return kStatus_Fail;
}

 // Get sensor range
 if (FXOS8700_ReadReg(&g_fxosHandle, FXOS8700_XYZ_DATA_CFG_REG, &g_sensorRange, 1) !=
      kStatus_Success)
 {
     return -1;
 }
 if(g_sensorRange == 0x00)
 {
     // 0.244 mg/LSB
     g_dataScale = 0.000244;
 }
 else if(g_sensorRange == 0x01)
 {
     // 0.488 mg/LSB
     g_dataScale = 0.000488;
 }
else if(g_sensorRange == 0x10)
```

```
{
    // 0.976 mg/LSB
    g_dataScale = 0.000976;
}
```

Basic Operation:

```
float accel[3]= {0};
float mag [3] = {0};

if (FXOS8700_ReadSensorData(&g_fxosHandle, &fxos8700_data) == kStatus_Success)
{


    //converting to units of  g
    accel[0] =  ((float)((int16_t)(((fxos8700_data.accelXMSB*256) + (fxos8700_data.
     accelXLSB)))>> 2));
    accel[1] =  ((float)((int16_t)(((fxos8700_data.accelYMSB*256) + (fxos8700_data.
     accelYLSB)))>> 2));
    accel[2] =  ((float)((int16_t)(((fxos8700_data.accelZMSB*256) + (fxos8700_data.
     accelZLSB)))>> 2));

    //apply scaling
    accel[0] *= g_dataScale;
    accel[1] *= g_dataScale;
    accel[2] *= g_dataScale;

    //Converting to units of uT
    mag[0] =  (float)((int16_t)((fxos8700_data.magXMSB*256) + (fxos8700_data.
     magXLSB))) * 0.1;
    mag[1] =  (float)((int16_t)((fxos8700_data.magYMSB*256) + (fxos8700_data.
     magYLSB))) * 0.1;
    mag[2] =  (float)((int16_t)((fxos8700_data.magZMSB*256) + (fxos8700_data.
     magZLSB))) * 0.1;

}
```

Data Structures

- struct fxos8700_handle_t
- struct fxos8700_data_t

Macros

- #define **SINGLE_TAP**
- #define **DOUBLE_TAP**
- #define **FXOS8700_STATUS_00_REG**
- #define **FXOS8700_ZYXOW_MASK**
- #define **FXOS8700_ZYXOW_MASK**
- #define **FXOS8700_ZOW_MASK**
- #define **FXOS8700_ZOW_MASK**
- #define **FXOS8700_YOW_MASK**
- #define **FXOS8700_YOW_MASK**
- #define **FXOS8700_XOW_MASK**
- #define **FXOS8700_XOW_MASK**
- #define **FXOS8700_ZYXDR_MASK**
- #define **FXOS8700_ZYXDR_MASK**
- #define **FXOS8700_ZDR_MASK**
- #define **FXOS8700_ZDR_MASK**

- #define **FXOS8700_YDR_MASK**
- #define **FXOS8700_YDR_MASK**
- #define **FXOS8700_XDR_MASK**
- #define **FXOS8700_XDR_MASK**
- #define **FXOS8700_F_STATUS_REG**
- #define **FXOS8700_F_OVF_MASK**
- #define **FXOS8700_F_WMRK_FLAG_MASK**
- #define **FXOS8700_F_CNT5_MASK**
- #define **FXOS8700_F_CNT4_MASK**
- #define **FXOS8700_F_CNT3_MASK**
- #define **FXOS8700_F_CNT2_MASK**
- #define **FXOS8700_F_CNT1_MASK**
- #define **FXOS8700_F_CNT0_MASK**
- #define **FXOS8700_F_CNT_MASK**
- #define **FXOS8700_OUT_X_MSB_REG**
- #define **FXOS8700_OUT_X_LSB_REG**
- #define **FXOS8700_OUT_Y_MSB_REG**
- #define **FXOS8700_OUT_Y_LSB_REG**
- #define **FXOS8700_OUT_Z_MSB_REG**
- #define **FXOS8700_OUT_Z_LSB_REG**
- #define **FXOS8700_F_SETUP_REG**
- #define **FXOS8700_F_MODE1_MASK**
- #define **FXOS8700_F_MODE0_MASK**
- #define **FXOS8700_F_WMRK5_MASK**
- #define **FXOS8700_F_WMRK4_MASK**
- #define **FXOS8700_F_WMRK3_MASK**
- #define **FXOS8700_F_WMRK2_MASK**
- #define **FXOS8700_F_WMRK1_MASK**
- #define **FXOS8700_F_WMRK0_MASK**
- #define **FXOS8700_F_MODE_MASK**
- #define **FXOS8700_F_WMRK_MASK**
- #define **FXOS8700_F_MODE_DISABLED**
- #define **FXOS8700_F_MODE_CIRCULAR**
- #define **FXOS8700_F_MODE_FILL**
- #define **FXOS8700_F_MODE_TRIGGER**
- #define **FXOS8700_TRIG_CFG_REG**
- #define **FXOS8700_TRIG_TRANS_MASK**
- #define **FXOS8700_TRIG_LNDPRT_MASK**
- #define **FXOS8700_TRIG_PULSE_MASK**
- #define **FXOS8700_TRIG_FF_MT_MASK**
- #define **FXOS8700_SYSMOD_REG**
- #define **FXOS8700_FGERR_MASK**
- #define **FXOS8700_FGT_4_MASK**
- #define **FXOS8700_FGT_3_MASK**
- #define **FXOS8700_FGT_2_MASK**
- #define **FXOS8700_FGT_1_MASK**
- #define **FXOS8700_FGT_0_MASK**
- #define **FXOS8700_FGT_MASK**
- #define **FXOS8700_SYSMOD1_MASK**
- #define **FXOS8700_SYSMOD0_MASK**
- #define **FXOS8700_SYSMOD_MASK**
- #define **FXOS8700_SYSMOD_STANDBY**
- #define **FXOS8700_SYSMOD_WAKE**
- #define **FXOS8700_SYSMOD_SLEEP**
- #define **FXOS8700_INT_SOURCE_REG**
- #define **FXOS8700_SRC_ASLP_MASK**

- #define **FXOS8700_SRC_FIFO_MASK**
- #define **FXOS8700_SRC_TRANS_MASK**
- #define **FXOS8700_SRC_LNDPRT_MASK**
- #define **FXOS8700_SRC_PULSE_MASK**
- #define **FXOS8700_SRC_FF_MT_MASK**
- #define **FXOS8700_SRC_DRDY_MASK**
- #define **FXOS8700_WHO_AM_I_REG**
- #define **FXOS8700_kFXOS_WHO_AM_I_Device_ID**
- #define **FXOS8700_XYZ_DATA_CFG_REG**
- #define **FXOS8700_HPF_OUT_MASK**
- #define **FXOS8700_FS1_MASK**
- #define **FXOS8700_FS0_MASK**
- #define **FXOS8700_FS_MASK**
- #define **FXOS8700_FULL_SCALE_2G**
- #define **FXOS8700_FULL_SCALE_4G**
- #define **FXOS8700_FULL_SCALE_8G**
- #define **FXOS8700_HP_FILTER_CUTOFF_REG**
- #define **FXOS8700_PULSE_HPF_BYP_MASK**
- #define **FXOS8700_PULSE_LPF_EN_MASK**
- #define **FXOS8700_SEL1_MASK**
- #define **FXOS8700_SEL0_MASK**
- #define **FXOS8700_SEL_MASK**
- #define **FXOS8700_PL_STATUS_REG**
- #define **FXOS8700_NEWLP_MASK**
- #define **FXOS8700_LO_MASK**
- #define **FXOS8700_LAPO1_MASK**
- #define **FXOS8700_LAPO0_MASK**
- #define **FXOS8700_BAFRO_MASK**
- #define **FXOS8700_LAPO_MASK**
- #define **FXOS8700_PL_CFG_REG**
- #define **FXOS8700_DBCNTM_MASK**
- #define **FXOS8700_DBCNTM_MASK**
- #define **FXOS8700_PL_EN_MASK**
- #define **FXOS8700_PL_COUNT_REG**
- #define **FXOS8700_PL_BF_ZCOMP_REG**
- #define **FXOS8700_BKFR1_MASK**
- #define **FXOS8700_BKFR0_MASK**
- #define **FXOS8700_ZLOCK2_MASK**
- #define **FXOS8700_ZLOCK1_MASK**
- #define **FXOS8700_ZLOCK0_MASK**
- #define **FXOS8700_BKFR_MASK**
- #define **FXOS8700_ZLOCK_MASK**
- #define **FXOS8700_PL_P_L_THS_REG**
- #define **FXOS8700_P_L_THS4_MASK**
- #define **FXOS8700_P_L_THS3_MASK**
- #define **FXOS8700_P_L_THS2_MASK**
- #define **FXOS8700_P_L_THS1_MASK**
- #define **FXOS8700_P_L_THS0_MASK**
- #define **FXOS8700_HYS2_MASK**
- #define **FXOS8700_HYS1_MASK**
- #define **FXOS8700_HYS0_MASK**
- #define **FXOS8700_P_L_THS_MASK**
- #define **FXOS8700_HYS_MASK**
- #define **FXOS8700_FF_MT_CFG_REG**
- #define **FXOS8700_ELE_MASK**
- #define **FXOS8700_OAE_MASK**

- #define **FXOS8700_ZEFE_MASK**
- #define **FXOS8700_YEFE_MASK**
- #define **FXOS8700_XEFE_MASK**
- #define **FXOS8700_FF_MT_SRC_REG**
- #define **FXOS8700_EA_MASK**
- #define **FXOS8700_ZHE_MASK**
- #define **FXOS8700_ZHP_MASK**
- #define **FXOS8700_YHE_MASK**
- #define **FXOS8700_YHP_MASK**
- #define **FXOS8700_XHE_MASK**
- #define **FXOS8700_XHP_MASK**
- #define **FXOS8700_FT_MT_THS_REG**
- #define **FXOS8700_TRANSIENT_THS_REG**
- #define **FXOS8700_THS6_MASK**
- #define **FXOS8700_THS5_MASK**
- #define **FXOS8700_THS4_MASK**
- #define **FXOS8700_THS3_MASK**
- #define **FXOS8700_THS2_MASK**
- #define **FXOS8700_TXS1_MASK**
- #define **FXOS8700_THS0_MASK**
- #define **FXOS8700_THS_MASK**
- #define **FXOS8700_FF_MT_COUNT_REG**
- #define **FXOS8700_TRANSIENT_CFG_REG**
- #define **FXOS8700_TELE_MASK**
- #define **FXOS8700_ZTEFE_MASK**
- #define **FXOS8700_YTEFE_MASK**
- #define **FXOS8700_XTEFE_MASK**
- #define **FXOS8700_HPF_BYP_MASK**
- #define **FXOS8700_TRANSIENT_SRC_REG**
- #define **FXOS8700_TEA_MASK**
- #define **FXOS8700_ZTRANSE_MASK**
- #define **FXOS8700_Z_TRANS_POL_MASK**
- #define **FXOS8700_YTRANSE_MASK**
- #define **FXOS8700_Y_TRANS_POL_MASK**
- #define **FXOS8700_XTRANSE_MASK**
- #define **FXOS8700_X_TRANS_POL_MASK**
- #define **FXOS8700_TRANSIENT_COUNT_REG**
- #define **FXOS8700_PULSE_CFG_REG**
- #define **FXOS8700_DPA_MASK**
- #define **FXOS8700_PELE_MASK**
- #define **FXOS8700_ZDPEFE_MASK**
- #define **FXOS8700_ZSPEFE_MASK**
- #define **FXOS8700_YDPEFE_MASK**
- #define **FXOS8700_YSPEFE_MASK**
- #define **FXOS8700_XDPEFE_MASK**
- #define **FXOS8700_XSPEFE_MASK**
- #define **FXOS8700_PULSE_SRC_REG**
- #define **FXOS8700_PEA_MASK**
- #define **FXOS8700_AXZ_MASK**
- #define **FXOS8700_AXY_MASK**
- #define **FXOS8700_AXX_MASK**
- #define **FXOS8700_DPE_MASK**
- #define **FXOS8700_POLZ_MASK**
- #define **FXOS8700_POLY_MASK**
- #define **FXOS8700_POLX_MASK**
- #define **FXOS8700_PULSE_THSX_REG**

- #define **FXOS8700_PULSE_THSY_REG**
- #define **FXOS8700_PULSE_THSZ_REG**
- #define **FXOS8700_PTHS_MASK**
- #define **FXOS8700_PULSE_TMLT_REG**
- #define **FXOS8700_PULSE_LTCY_REG**
- #define **FXOS8700_PULSE_WIND_REG**
- #define **FXOS8700_ASLP_COUNT_REG**
- #define **FXOS8700_CTRL_REG1**
- #define **FXOS8700_ASLP_RATE1_MASK**
- #define **FXOS8700_ASLP_RATE0_MASK**
- #define **FXOS8700_DR2_MASK**
- #define **FXOS8700_DR1_MASK**
- #define **FXOS8700_DR0_MASK**
- #define **FXOS8700_LNOISE_MASK**
- #define **FXOS8700_FREAD_MASK**
- #define **FXOS8700_ACTIVE_MASK**
- #define **FXOS8700_ASLP_RATE_MASK**
- #define **FXOS8700_DR_MASK**
- #define **FXOS8700_ASLP_RATE_20MS**
- #define **FXOS8700_ASLP_RATE_80MS**
- #define **FXOS8700_ASLP_RATE_160MS**
- #define **FXOS8700_ASLP_RATE_640MS**
- #define **FXOS8700_ASLP_RATE_50HZ**
- #define **FXOS8700_ASLP_RATE_12_5HZ**
- #define **FXOS8700_ASLP_RATE_6_25HZ**
- #define **FXOS8700_ASLP_RATE_1_56HZ**
- #define **FXOS8700_HYB_ASLP_RATE_25HZ**
- #define **FXOS8700_HYB_ASLP_RATE_6_25HZ**
- #define **FXOS8700_HYB_ASLP_RATE_1_56HZ**
- #define **FXOS8700_HYB_ASLP_RATE_0_8HZ**
- #define **FXOS8700_DATA_RATE_1250US**
- #define **FXOS8700_DATA_RATE_2500US**
- #define **FXOS8700_DATA_RATE_5MS**
- #define **FXOS8700_DATA_RATE_10MS**
- #define **FXOS8700_DATA_RATE_20MS**
- #define **FXOS8700_DATA_RATE_80MS**
- #define **FXOS8700_DATA_RATE_160MS**
- #define **FXOS8700_DATA_RATE_640MS**
- #define **FXOS8700_DATA_RATE_800HZ**
- #define **FXOS8700_DATA_RATE_400HZ**
- #define **FXOS8700_DATA_RATE_200HZ**
- #define **FXOS8700_DATA_RATE_100HZ**
- #define **FXOS8700_DATA_RATE_50HZ**
- #define **FXOS8700_DATA_RATE_12_5HZ**
- #define **FXOS8700_DATA_RATE_6_25HZ**
- #define **FXOS8700_DATA_RATE_1_56HZ**
- #define **FXOS8700_HYB_DATA_RATE_400HZ**
- #define **FXOS8700_HYB_DATA_RATE_200HZ**
- #define **FXOS8700_HYB_DATA_RATE_100HZ**
- #define **FXOS8700_HYB_DATA_RATE_50HZ**
- #define **FXOS8700_HYB_DATA_RATE_25HZ**
- #define **FXOS8700_HYB_DATA_RATE_6_25HZ**
- #define **FXOS8700_HYB_DATA_RATE_3_15HZ**
- #define **FXOS8700_HYB_DATA_RATE_0_8HZ**
- #define **FXOS8700_ACTIVE**
- #define **FXOS8700_STANDBY**

- #define **FXOS8700_CTRL_REG2**
- #define **FXOS8700_ST_MASK**
- #define **FXOS8700_RST_MASK**
- #define **FXOS8700_SMODS1_MASK**
- #define **FXOS8700_SMODS0_MASK**
- #define **FXOS8700_SLPE_MASK**
- #define **FXOS8700_MODS1_MASK**
- #define **FXOS8700_MODS0_MASK**
- #define **FXOS8700_SMODS_MASK**
- #define **FXOS8700_MODS_MASK**
- #define **FXOS8700_SMOD_NORMAL**
- #define **FXOS8700_SMOD_LOW_NOISE**
- #define **FXOS8700_SMOD_HIGH_RES**
- #define **FXOS8700_SMOD_LOW_POWER**
- #define **FXOS8700_MOD_NORMAL**
- #define **FXOS8700_MOD_LOW_NOISE**
- #define **FXOS8700_MOD_HIGH_RES**
- #define **FXOS8700_MOD_LOW_POWER**
- #define **FXOS8700_CTRL_REG3**
- #define **FXOS8700_FIFO_GATE_MASK**
- #define **FXOS8700_WAKE_TRANS_MASK**
- #define **FXOS8700_WAKE_LNDPRT_MASK**
- #define **FXOS8700_WAKE_PULSE_MASK**
- #define **FXOS8700_WAKE_FF_MT_MASK**
- #define **FXOS8700_IPOL_MASK**
- #define **FXOS8700_PP_OD_MASK**
- #define **FXOS8700_CTRL_REG4**
- #define **FXOS8700_INT_EN_ASLP_MASK**
- #define **FXOS8700_INT_EN_FIFO_MASK**
- #define **FXOS8700_INT_EN_TRANS_MASK**
- #define **FXOS8700_INT_EN_LNDPRT_MASK**
- #define **FXOS8700_INT_EN_PULSE_MASK**
- #define **FXOS8700_INT_EN_FF_MT_MASK**
- #define **FXOS8700_INT_EN_DRDY_MASK**
- #define **FXOS8700_CTRL_REG5**
- #define **FXOS8700_INT_CFG_ASLP_MASK**
- #define **FXOS8700_INT_CFG_FIFO_MASK**
- #define **FXOS8700_INT_CFG_TRANS_MASK**
- #define **FXOS8700_INT_CFG_LNDPRT_MASK**
- #define **FXOS8700_INT_CFG_PULSE_MASK**
- #define **FXOS8700_INT_CFG_FF_MT_MASK**
- #define **FXOS8700_INT_CFG_DRDY_MASK**
- #define **FXOS8700_OFF_X_REG**
- #define **FXOS8700_OFF_Y_REG**
- #define **FXOS8700_OFF_Z_REG**
- #define **FXOS8700_M_DR_STATUS_REG**
- #define **FXOS8700_M_OUT_X_MSB_REG**
- #define **FXOS8700_M_OUT_X_LSB_REG**
- #define **FXOS8700_M_OUT_Y_MSB_REG**
- #define **FXOS8700_M_OUT_Y_LSB_REG**
- #define **FXOS8700_M_OUT_Z_MSB_REG**
- #define **FXOS8700_M_OUT_Z_LSB_REG**
- #define **FXOS8700_CMP_X_MSB_REG**
- #define **FXOS8700_CMP_X_LSB_REG**
- #define **FXOS8700_CMP_Y_MSB_REG**
- #define **FXOS8700_CMP_Y_LSB_REG**

- #define **FXOS8700_CMP_Z_MSB_REG**
- #define **FXOS8700_CMP_Z_LSB_REG**
- #define **FXOS8700_M_OFF_X_MSB_REG**
- #define **FXOS8700_M_OFF_X_LSB_REG**
- #define **FXOS8700_M_OFF_Y_MSB_REG**
- #define **FXOS8700_M_OFF_Y_LSB_REG**
- #define **FXOS8700_M_OFF_Z_MSB_REG**
- #define **FXOS8700_M_OFF_Z_LSB_REG**
- #define **FXOS8700_MAX_X_MSB_REG**
- #define **FXOS8700_MAX_X_LSB_REG**
- #define **FXOS8700_MAX_Y_MSB_REG**
- #define **FXOS8700_MAX_Y_LSB_REG**
- #define **FXOS8700_MAX_Z_MSB_REG**
- #define **FXOS8700_MAX_Z_LSB_REG**
- #define **FXOS8700_MIN_X_MSB_REG**
- #define **FXOS8700_MIN_X_LSB_REG**
- #define **FXOS8700_MIN_Y_MSB_REG**
- #define **FXOS8700_MIN_Y_LSB_REG**
- #define **FXOS8700_MIN_Z_MSB_REG**
- #define **FXOS8700_MIN_Z_LSB_REG**
- #define **FXOS8700_TEMP_REG**
- #define **FXOS8700_M_THS_CFG_REG**
- #define **FXOS8700_M_THS_SRC_REG**
- #define **FXOS8700_M_THS_X_MSB_REG**
- #define **FXOS8700_M_THS_X_LSB_REG**
- #define **FXOS8700_M_THS_Y_MSB_REG**
- #define **FXOS8700_M_THS_Y_LSB_REG**
- #define **FXOS8700_M_THS_Z_MSB_REG**
- #define **FXOS8700_M_THS_Z_LSB_REG**
- #define **FXOS8700_M_THS_COUNT**
- #define **FXOS8700_M_CTRL_REG1**
- #define **FXOS8700_M_ACAL_MASK**
- #define **FXOS8700_M_RST_MASK**
- #define **FXOS8700_M_OST_MASK**
- #define **FXOS8700_M_OSR2_MASK**
- #define **FXOS8700_M_OSR1_MASK**
- #define **FXOS8700_M_OSR0_MASK**
- #define **FXOS8700_M_HMS1_MASK**
- #define **FXOS8700_M_HMS0_MASK**
- #define **FXOS8700_M_OSR_MASK**
- #define **FXOS8700_M_HMS_MASK**
- #define **FXOS8700_M_OSR_1_56_HZ**
- #define **FXOS8700_M_OSR_6_25_HZ**
- #define **FXOS8700_M_OSR_12_5_HZ**
- #define **FXOS8700_M_OSR_50_HZ**
- #define **FXOS8700_M_OSR_100_HZ**
- #define **FXOS8700_M_OSR_200_HZ**
- #define **FXOS8700_M_OSR_400_HZ**
- #define **FXOS8700_M_OSR_800_HZ**
- #define **FXOS8700_ACCEL_ACTIVE**
- #define **FXOS8700_MAG_ACTIVE**
- #define **FXOS8700_HYBRID_ACTIVE**
- #define **FXOS8700_M_CTRL_REG2**
- #define **FXOS8700_M_HYB_AUTOINC_MASK**
- #define **FXOS8700_M_MAXMIN_DIS_MASK**
- #define **FXOS8700_M_MAXMIN_DIS_THS_MASK**

- #define **FXOS8700_M_MAXMIN_RST_MASK**
- #define **FXOS8700_M_RST_CNT1_MASK**
- #define **FXOS8700_M_RST_CNT0_MASK**
- #define **FXOS8700_RST_ODR_CYCLE**
- #define **FXOS8700_RST_16_ODR_CYCLE**
- #define **FXOS8700_RST_512_ODR_CYCLE**
- #define **FXOS8700_RST_DISABLED**
- #define **FXOS8700_M_CTRL_REG3**
- #define **FXOS8700_M_RAW_MASK**
- #define **FXOS8700_M_ASLP_OS_2_MASK**
- #define **FXOS8700_M_ASLP_OS_1_MASK**
- #define **FXOS8700_M_ASLP_OS_0_MASK**
- #define **FXOS8700_M_THS_XYZ_MASK**
- #define **FXOS8700_M_ST_Z_MASK**
- #define **FXOS8700_M_ST_XY1_MASK**
- #define **FXOS8700_M_ST_XY0_MASK**
- #define **FXOS8700_M_ASLP_OSR_MASK**
- #define **FXOS8700_M_ST_XY_MASK**
- #define **FXOS8700_M_ASLP_OSR_1_56_HZ**
- #define **FXOS8700_M_ASLP_OSR_6_25_HZ**
- #define **FXOS8700_M_ASLP_OSR_12_5_HZ**
- #define **FXOS8700_M_ASLP_OSR_50_HZ**
- #define **FXOS8700_M_ASLP_OSR_100_HZ**
- #define **FXOS8700_M_ASLP_OSR_200_HZ**
- #define **FXOS8700_M_ASLP_OSR_400_HZ**
- #define **FXOS8700_M_ASLP_OSR_800_HZ**
- #define **FXOS8700_M_INT_SOURCE**
- #define **FXOS8700_SRC_M_DRDY_MASK**
- #define **FXOS8700_SRC_M_VECM_MASK**
- #define **FXOS8700_SRC_M_THS_MASK**
- #define **FXOS8700_A_VECM_CFG**
- #define **FXOS8700_A_VECM_INIT_CFG_MASK**
- #define **FXOS8700_A_VECM_INIT_EN_MASK**
- #define **FXOS8700_A_VECM_WAKE_EN_MASK**
- #define **FXOS8700_A_VECM_EN_MASK**
- #define **FXOS8700_A_VECM_UPDM_MASK**
- #define **FXOS8700_A_VECM_INITM_MASK**
- #define **FXOS8700_A_VECM_ELE_MASK**
- #define **FXOS8700_A_VECM_THS_MSB**
- #define **FXOS8700_A_VECM_DBCNTM_MASK**
- #define **FXOS8700_A_VECM_THS_LSB**
- #define **FXOS8700_A_VECM_CNT**
- #define **FXOS8700_A_VECM_INITX_MSB**
- #define **FXOS8700_A_VECM_INITX_LSB**
- #define **FXOS8700_A_VECM_INITY_MSB**
- #define **FXOS8700_A_VECM_INITY_LSB**
- #define **FXOS8700_A_VECM_INITZ_MSB**
- #define **FXOS8700_A_VECM_INITZ_LSB**
- #define **FXOS8700_M_VECM_CFG**
- #define **FXOS8700_M_VECM_INIT_CFG_MASK**
- #define **FXOS8700_M_VECM_INIT_EN_MASK**
- #define **FXOS8700_M_VECM_WAKE_EN_MASK**
- #define **FXOS8700_M_VECM_EN_MASK**
- #define **FXOS8700_M_VECM_UPDM_MASK**
- #define **FXOS8700_M_VECM_INITM_MASK**
- #define **FXOS8700_M_VECM_ELE_MASK**

- #define **FXOS8700_M_VECM_THS_MSB**
- #define **FXOS8700_M_VECM_DBCNTM_MASK**
- #define **FXOS8700_M_VECM_THS_LSB**
- #define **FXOS8700_M_VECM_CNT**
- #define **FXOS8700_M_VECM_INITX_MSB**
- #define **FXOS8700_M_VECM_INITX_LSB**
- #define **FXOS8700_M_VECM_INITY_MSB**
- #define **FXOS8700_M_VECM_INITY_LSB**
- #define **FXOS8700_M_VECM_INITZ_MSB**
- #define **FXOS8700_M_VECM_INITZ_LSB**
- #define **FXOS8700_A_FFMT_THS_X_MSB**
- #define **FXOS8700_A_FFMT_THS_XYZ_EN_MASK**
- #define **FXOS8700_A_FFMT_THS_X_LSB**
- #define **FXOS8700_A_FFMT_THS_X_LSB_MASK**
- #define **FXOS8700_A_FFMT_THS_Y_MSB**
- #define **FXOS8700_A_FFMT_THS_Y_EN_MASK**
- #define **FXOS8700_A_FFMT_THS_Y_LSB**
- #define **FXOS8700_A_FFMT_THS_Y_LSB_MASK**
- #define **FXOS8700_A_FFMT_THS_Z_MSB**
- #define **FXOS8700_A_FFMT_THS_Z_EN_MASK**
- #define **FXOS8700_A_FFMT_THS_Z_LSB**
- #define **FXOS8700_A_FFMT_THS_Z_LSB_MASK**
- #define **FXOS8700_A_TRAN_INIT_XYZ_MSB**
- #define **FXOS8700_A_TRAN_INIT_X_LSB**
- #define **FXOS8700_A_TRAN_INIT_Y_LSB**
- #define **FXOS8700_A_TRAN_INIT_Z_LSB**

Functions

- status_t FXOS8700_Init (fxos8700_handle_t *fxos8700_handle)
- status_t FXOS8700_ReadSensorData (fxos8700_handle_t *fxos8700_handle, fxos8700_data_t *sensorData)
- status_t FXOS8700_WriteReg (fxos8700_handle_t *handle, uint8_t reg, uint8_t val)
- status_t FXOS8700_ReadReg (fxos8700_handle_t *handle, uint8_t reg, uint8_t *val, uint8_t bytesNumber)
- status_t FXOS8700_SetStandby (fxos8700_handle_t *fxos8700_handle)
- status_t FXOS8700_SetActive (fxos8700_handle_t *fxos8700_handle)
- status_t FXOS8700_MotionDetect_Init (fxos8700_handle_t *fxos8700_handle)
- status_t FXOS8700_FreefallDetect_Init (fxos8700_handle_t *fxos8700_handle)
- status_t FXOS8700_TapDetect_Init (fxos8700_handle_t *fxos8700_handle)
- status_t FXOS8700_FreefallMotion_DeInit (fxos8700_handle_t *fxos8700_handle)
- status_t FXOS8700_TapDetect_DeInit (fxos8700_handle_t *fxos8700_handle)
- status_t FXOS8700_TransientDetect_Init (fxos8700_handle_t *fxos8700_handle)
- status_t FXOS8700_TransientDetect_DeInit (fxos8700_handle_t *fxos8700_handle)

9.2 Data Structure Documentation

9.2.1 struct fxos8700_handle_t

fxos8700cq configure definition.

This structure should be global.

Data Fields

I2C_Type *	base	I2C base.
i2c_master_↩ handle_t *	i2cHandle	I2C master transfer context.
i2c_master_↩ transfer_t	xfer	I2C master xfer.

9.2.2 struct fxos8700_data_t

Data Fields

uint8_t	accelXMSB	X-axis Accelerometer Reading: [7:0] are 8 MSBs of 14-bit sample.
uint8_t	accelXLSB	X-axis Accelerometer Reading: [7:2] are 6 LSBs of 14-bit sample.
uint8_t	accelYMSB	Y-axis Accelerometer Reading: [7:0] are 8 MSBs of 14-bit sample.
uint8_t	accelYLSB	Y-axis Accelerometer Reading: [7:2] are 6 LSBs of 14-bit sample.
uint8_t	accelZMSB	Z-axis Accelerometer Reading: [7:0] are 8 MSBs of 14-bit sample.
uint8_t	accelZLSB	Z-axis Accelerometer Reading: [7:2] are 6 LSBs of 14-bit sample.
uint8_t	magXMSB	X-axis Magnetometer Reading: [7:0] are 8 MSBs of 16-bit sample.
uint8_t	magXLSB	X-axis Magnetometer Reading: [7:0] are 8 LSBs of 16-bit sample.
uint8_t	magYMSB	Y-axis Magnetometer Reading: [7:0] are 8 MSBs of 16-bit sample.
uint8_t	magYLSB	Y-axis Magnetometer Reading: [7:0] are 8 LSBs of 16-bit sample.
uint8_t	magZMSB	Z-axis Magnetometer Reading: [7:0] are 8 MSBs of 16-bit sample.
uint8_t	magZLSB	Z-axis Magnetometer Reading: [7:0] are 8 LSBs of 16-bit sample.

9.3 Function Documentation

9.3.1 status_t FXOS8700_Init (fxos8700_handle_t * *fxos8700_handle*)

Verify and initialize fxos8700_handle ice: Hybrid mode with ODR=200Hz, Mag OSR=32, Acc O↩ SR=Normal.

Interrupt for data ready can be set using #define EN_FXOS_DATAREADY_INTERRUPT

Parameters

fxos8700_↩ handle	The pointer to fxos8700cq driver handle.

Returns

kStatus_Success if success or kStatus_Fail if error.

9.3.2 status_t FXOS8700_ReadSensorData (fxos8700_handle_t * *fxos8700_handle*, fxos8700_data_t * *sensorData*)

Read data from sensors, assumes hyb_autoinc_mode is set in M_CTRL_REG2.

Parameters

fxos8700_handle	The pointer to fxos8700cq driver handle.
sensorData	The pointer to the buffer to hold sensor data

Returns

kStatus_Success if success or kStatus_Fail if error.

9.3.3 status_t FXOS8700_WriteReg (fxos8700_handle_t * *handle*, uint8_t *reg*, uint8_t *val*)

Write value to register of sensor.

Parameters

handle	The pointer to fxos8700cq driver handle.
reg	Register address.
val	Data want to write.

Returns

kStatus_Success if success or kStatus_Fail if error.

9.3.4 status_t FXOS8700_ReadReg (fxos8700_handle_t * *handle*, uint8_t *reg*, uint8_t * *val*, uint8_t *bytesNumber*)

Read n bytes start at register from sensor.

Parameters

handle	The pointer to fxos8700cq driver handle.
reg	Register address.
val	The pointer to address which store data.
bytesNumber	Number of bytes receiver.

Returns

kStatus_Success if success or kStatus_Fail if error.

9.3.5 status_t FXOS8700_SetStandby (fxos8700_handle_t * *fxos8700_handle*)

Puts the FXOS8700CQ into Standby Mode.

Function Documentation

Parameters

fxos8700_↩ handle	The pointer to fxos8700cq driver handle.

Returns

kStatus_Success if success or kStatus_Fail if error.

9.3.6 status_t FXOS8700_SetActive (fxos8700_handle_t * *fxos8700_handle*)

Puts the FXOS8700CQ into Active Mode.

Parameters

fxos8700_↩ handle	The pointer to fxos8700cq driver handle.

Returns

kStatus_Success if success or kStatus_Fail if error.

9.3.7 status_t FXOS8700_MotionDetect_Init (fxos8700_handle_t * *fxos8700_handle*)

Sets up Motion Detection on x and y axis with 0.5g Threshold.

Motion Detection detects both dynamic acceleration and static acceleration(gravity). Use Transient Detection if only interested in detecting dynamic acceleration changes.

NOTE: Motion Detect and FreeFall Detect shares the same hardware block so only one function can be initialized/used at a time. Please refer to NXP Application Note AN4070: Motion and Freefall Detection Using the MMA8451, 2, 3Q on how to configure the Motion Detection parameters. FXOS8700CQ shares the same registers and hardware blocks as the MMA8451 so AN4070 is applicable for this sensor.

Interrupts can be set with #define EN_FFMT_INTERRUPT. Interrupts when enabled goes to INT2 pin, set to INT1 pin using #define EN_FFMT_INT1_PIN

Parameters

fxos8700_↩ handle	The pointer to fxos8700cq driver handle.

Returns

kStatus_Success if success or kStatus_Fail if error.

9.3.8 status_t FXOS8700_FreefallDetect_Init (fxos8700_handle_t * *fxos8700_handle*)

Sets up Freefall Detection on X and Y and Z axis with 0.25g threshold.

Combined X,Y,Z magnitude must be less than 0.25g to trigger. NOTE: Motion Detect and FreeFall Detect shares the same hardware block so only one function can be initialized/used at a time. Please refer to NXP Application Note AN4070: Motion and Freefall Detection Using the MMA8451, 2, 3Q on how to configure the Freefall Detection parameters. FXOS8700CQ shares the same registers and hardware blocks as the MMA8451 so AN4070 is applicable for this sensor.

Interrupts can be set with #define EN_FFMT_INTERRUPT. Interrupts when enabled goes to INT2 pin, set to INT1 pin using #define EN_FFMT_INT1_PIN

Parameters

fxos8700_handle	The pointer to fxos8700cq driver handle.

Returns

kStatus_Success if success or kStatus_Fail if error.

9.3.9 status_t FXOS8700_TapDetect_Init (fxos8700_handle_t * *fxos8700_handle*)

Sets up Tap Detection (also called Pulse Detection) Pulse Thresholds: X:2g Y:2g Z:3g Pulse Time Window: 6 counts Pulse Latency Timer: 40 counts Pulse 2nd Time Window: 15 counts.

Please refer to NXP Application Note AN4072: MMA8451, 2, 3Q Single/Double and Directional Tap Detection on how to configure the Tap Detection parameters. FXOS8700CQ shares the same registers and hardware blocks as the MMA8451 so AN4072 is applicable for this sensor.

Interrupts can be set with #define EN_TAP_INTERRUPT. Interrupts when enabled goes to INT2 pin, set to INT1 pin using #define EN_TAP_INT1_PIN

Parameters

fxos8700_handle	The pointer to fxos8700cq driver handle.

Returns

kStatus_Success if success or kStatus_Fail if error.

9.3.10 status_t FXOS8700_FreefallMotion_DeInit (fxos8700_handle_t * *fxos8700_handle*)

Disables either freefall or motion detection.

Disables the associated interrupts if enabled

Parameters

fxos8700_handle	The pointer to fxos8700cq driver handle.

Returns

kStatus_Success if success or kStatus_Fail if error.

9.3.11 status_t FXOS8700_TapDetect_DeInit (fxos8700_handle_t * *fxos8700_handle*)

Disables Tap Detection.

Disables the associated interrupts if enabled

Parameters

fxos8700_handle	The pointer to fxos8700cq driver handle.

Returns

kStatus_Success if success or kStatus_Fail if error.

9.3.12 status_t FXOS8700_TransientDetect_Init (fxos8700_handle_t * *fxos8700_handle*)

Sets up Transient Detection on x,y,z axis with 0.25g Threshold and High-Pass Filter to 2 Hz (OSR= 200Hz,High Res) Transient Detection is similar to Motion Detection except it only detects dynamic acceleration.

It will not register static acceleration such as gravity.

Please refer to NXP Application Note AN4071: High-Pass Filtered Data and Transient Detection Using the MMA8451, 2, 3Q on how to configure the Transient Detection parameters. It also lists when to use transient detection and when to use motion detection. FXOS8700CQ shares the same registers and hardware blocks as the MMA8451 so AN4071 is applicable for this sensor.

Interrupts can be set with #define EN_TRANS_INTERRUPT. Interrupts when enabled goes to INT2 pin, set to INT1 pin using #define EN_TRANS_INT1_PIN

Parameters

fxos8700_handle	The pointer to fxos8700cq driver handle.

Returns

kStatus_Success if success or kStatus_Fail if error.

9.3.13 status_t FXOS8700_TransientDetect_DeInit (fxos8700_handle_t * *fxos8700_handle*)

Disables Transient detection.

Disables the associated interrupts if enabled

Parameters

fxos8700_handle	The pointer to fxos8700cq driver handle.

Returns

kStatus_Success if success or kStatus_Fail if error.

Chapter 10
Image BMP Callbacks

10.1 Overview

The img_bmp_callbacks module provdes a callback function that facilitates loading BMP images directly from External SPI Flash using GUI_BMP_DrawEx function.

Dependencies

The img_bmp_callbacks module depends on the following:

- img_program_ext
- emWin library
  - NOTE: Designed specifically for use with emWin GUI_BMP_DrawEx.

Usage

```
// Image address in External SPI Flash
// See img_program_ext & ui_manager modules for storing images
uint32_t imgBaseAddr = 0x00C00000;

// Draw image from External memory
GUI_BMP_DrawEx(RPK_GUI_get_data, &imgBaseAddr, 0, 0);
```

Callback function to be used with emWin GUI_BMP_DrawEx function to draw BMP from External SPI Flash

Functions

- int **RPK_GUI_get_data** (void ∗extBaseAddr, const unsigned char ∗∗ppData, unsigned numBytes, long unsigned offset)

Chapter 11
MPL3115 Atmospheric Pressure Sensor

11.1 Overview

This module provides the API to operate the MPL3115 atmospheric pressure sensor through an I2C interface.

The MPL3115A2 is a compact, piezoresistive, absolute pressure sensor with an I2C digital interface. M↩ PL3115A2 has a wide operating range of 20 kPa to 110 kPa, a range that covers all surface elevations on earth. The MEMS is temperature compensated utilizing an on-chip temperature sensor. The pressure and temperature data is fed into a high resolution ADC to provide fully compensated and digitized outputs for pressure in Pascals and temperature in °C.

Usage

Initialization:

```
#include "mpl3115.h"

mpl3115_IoFunc_t MPL3115_sensor;
MPL3115_sensor.I2C_Read = App_I2C1_Read;
MPL3115_sensor.I2C_Write = App_I2C1_Write;
MPL3115_sensor.WaitMsec = App_WaitMsec;

MPL3115_Init_Driver(&MPL3115_sensor);
MPL3115_Init_Hw();
```

Basic Operation:

```
int32_t data;

if (MPL_ReadRawData (MPL_MODE_PRESSURE, &data) == 0)
{
      data /= 400; // in HPa (LSB = 0.25Pa)
}
```

Data Structures

- struct pmpl3115_IoFunc_t
- struct settingsMPL_t

Macros

- #define **MPL3115_I2C_SLAVE_ADDRESS**

Typedefs

- typedef int16_t **mE_t**

Enumerations

- enum mpl_status_t {
  MPL_SUCCESS,
  MPL_ERROR,
  MPL_PROTOCOL_ERROR,
  MPL_INIT_ERROR,
  MPL_TIMEOUT,
  MPL_NOT_SUPPORTED }
- enum overSampleMPL_t {
  MPL_OS_0,
  MPL_OS_1,
  MPL_OS_2,
  MPL_OS_3,
  MPL_OS_4,
  MPL_OS_5,
  MPL_OS_6,
  MPL_OS_7 }
- enum autoAcquisitionTime_t {
  MPL_ST_0,
  MPL_ST_1,
  MPL_ST_2,
  MPL_ST_3,
  MPL_ST_4,
  MPL_ST_5,
  MPL_ST_6,
  MPL_ST_7,
  MPL_ST_8,
  MPL_ST_9,
  MPL_ST_10,
  MPL_ST_11,
  MPL_ST_12,
  MPL_ST_13,
  MPL_ST_14,
  MPL_ST_15 }
- enum modeMPL_t {
  MPL_MODE_PRESSURE,
  MPL_MODE_ALTITUDE,
  MPL_MODE_TEMPERATURE,
  MPL_MODE_CURRENT }
- enum modeFIFO_t {
  FIFO_DISABLED,
  FIFO_CIRCULAR,
  FIFO_STOP_OVERFLOW }
- enum pinINT_t {
  FIFO_INT1,

FIFO_INT2 }

Functions

- void MPL3115_Init_Driver (mpl3115_IoFunc_t *pIoFunc)
- void MPL3115_Deinit_Driver ()
- mpl_status_t MPL3115_Init_Hw ()
- mpl_status_t MPL_SoftReset ()
- mpl_status_t MPL_ToggleOneShot ()
- mpl_status_t MPL_GetID (uint8_t *sensorID)
- mpl_status_t MPL_SetMode (modeMPL_t mode)
- mpl_status_t MPL_GotoStandby ()
- mpl_status_t MPL_SetActive ()
- mpl_status_t MPL_SetOversampleRate (uint8_t sampleRate)
- mpl_status_t MPL_SetAutoAcquisitionTime (uint8_t sampleTime)
- mpl_status_t MPL_EnableEventFlags ()
- mpl_status_t MPL_SetOffsetPressure (int8_t pressOffset)
- mpl_status_t MPL_SetOffsetAltitude (int8_t altitudeOffset)
- mpl_status_t MPL_SetOffsetTemperature (int8_t temperatureOffset)
- mpl_status_t MPL_SetFifoMode (modeFIFO_t fMode, uint8_t fWmrk)
- mpl_status_t MPL_SetFifoInterrupt (pinINT_t pinINT)
- mpl_status_t MPL_DisableFifoInterrupt ()
- mpl_status_t MPL_GetFifoStatus (uint8_t *fifoStatus)
- mpl_status_t MPL_ReadRawData (modeMPL_t mode, int32_t *sensorData)
- mpl_status_t MPL_Dump (uint8_t *sensorReg)

11.2 Data Structure Documentation

11.2.1 struct mpl3115_IoFunc_t

Structure of external functions or values.

Data Fields

- uint8_t(* I2C_Read)(uint8_t device_addr, uint8_t *writeBuf, uint32_t writeSize, uint8_t *readBuf, uint32_t readSize)
- uint8_t(* I2C_Write)(uint8_t device_addr, uint8_t *writeBuf, uint32_t writeSize)
- void(* WaitMsec)(uint32_t millisec)

11.2.1.1 Field Documentation

11.2.1.1.1 uint8_t(* I2C_Read) (uint8_t device_addr, uint8_t *writeBuf, uint32_t writeSize, uint8_t *readBuf, uint32_t readSize)

Function pointer to I2C Read function.

11.2.1.1.2 uint8_t(* I2C_Write) (uint8_t device_addr, uint8_t *writeBuf, uint32_t writeSize)

Function pointer to I2C Write function.

11.2.1.1.3 void(* WaitMsec) (uint32_t millisec)

Function pointer to waitMsec function.

11.2.2 struct settingsMPL_t

Data Fields

modeMPL_t	mode	device mode, altimeter or barometer
overSample↩ MPL_t	oversample	oversampling ratio
auto↩ Acquisition↩ Time_t	auto↩ Acquisition↩ Time	Auto acquisition time step.
int8_t	pressureOffset	Offset pressure correction (signed: 4 Pa/LSB)
int8_t	altitudeOffset	Offset altitude correction (signed: 1m/LSB)
int8_t	tempOffset	Offset temperature correction -8°C (signed: 0.0625°C/LSB)
modeFIFO_t	fifoMode	FIFO mode.
uint8_t	fifoWatermark	These (6) bits set the number of FIFO samples required to trigger a watermark interrupt.
pinINT_t	fifoINTpin	Pin to route FIFO interrupt.

11.3 Enumeration Type Documentation

11.3.1 enum mpl_status_t

Status return codes.

Enumerator

MPL_SUCCESS Function ran successfully.
MPL_ERROR Error in running function.
MPL_PROTOCOL_ERROR Protocol error has occurred.
MPL_INIT_ERROR Initialization error has occurred.
MPL_TIMEOUT MPL function has timed out.
MPL_NOT_SUPPORTED Not Supported.

11.3.2 enum overSampleMPL_t

Oversampling factor.

Enumerator

MPL_OS_0 Oversample Ratio = 1.

MPL_OS_1 Oversample Ratio = 2.
MPL_OS_2 Oversample Ratio = 4.
MPL_OS_3 Oversample Ratio = 8.
MPL_OS_4 Oversample Ratio = 16.
MPL_OS_5 Oversample Ratio = 32.
MPL_OS_6 Oversample Ratio = 64.
MPL_OS_7 Oversample Ratio = 128.

11.3.3 enum autoAcquisitionTime_t

Auto acquisition time step : power(2; MPL_ST_X)

Enumerator

MPL_ST_0 Auto acquistion time step = 1 second.
MPL_ST_1 Auto acquistion time step = 2 seconds.
MPL_ST_2 Auto acquistion time step = 4 seconds.
MPL_ST_3 Auto acquistion time step = 8 seconds.
MPL_ST_4 Auto acquistion time step = 16 seconds.
MPL_ST_5 Auto acquistion time step = 32 seconds.
MPL_ST_6 Auto acquistion time step = 64 seconds.
MPL_ST_7 Auto acquistion time step = 128 seconds.
MPL_ST_8 Auto acquistion time step = 256 seconds.
MPL_ST_9 Auto acquistion time step = 512 seconds.
MPL_ST_10 Auto acquistion time step = 1024 seconds.
MPL_ST_11 Auto acquistion time step = 2048 seconds.
MPL_ST_12 Auto acquistion time step = 4096 seconds.
MPL_ST_13 Auto acquistion time step = 8192 seconds.
MPL_ST_14 Auto acquistion time step = 16384 seconds.
MPL_ST_15 Auto acquistion time step = 32768 seconds.

11.3.4 enum modeMPL_t

Device Mode.

Enumerator

MPL_MODE_PRESSURE Device is in barometer mode. It reports an absolute pressure.
MPL_MODE_ALTITUDE Device is in altimeter mode. The pressure data is converted to equivalent altitude based on US standard atmosphere
MPL_MODE_TEMPERATURE This mode provides temperature from a high resolution temperature sensor.

11.3.5 enum modeFIFO_t

Fifo Mode.

Enumerator

FIFO_DISABLED FIFO is disabled (reset value)
FIFO_CIRCULAR FIFO contains the most recent samples when overflowed (circular buffer). Oldest sample is discarded to be replaced by new sample
FIFO_STOP_OVERFLOW FIFO stops accepting new samples when overflowed.

11.3.6 enum pinINT_t

Pin to route FIFO interrupt.

Enumerator

FIFO_INT1 FIFO Interrupt routed to INT1 pin.
FIFO_INT2 FIFO Interrupt routed to INT2 pin.

11.4 Function Documentation

11.4.1 void MPL3115_Init_Driver (mpl3115_IoFunc_t * *ploFunc*)

Initialize MPL3115 driver.

Note

Wrap the low level functions (I2C write, I2C read, WaitMsec).

Parameters

pIoFunc	Pointer to a structure with external functions

11.4.2 void MPL3115_Deinit_Driver ( )

De-initialize MPL3115 driver.

11.4.3 mpl_status_t MPL3115_Init_Hw ( )

Initialize MPL3115 hardware.

Returns

Status value (0 for success)

11.4.4 mpl_status_t MPL_SoftReset ( )

Soft reset.

Note

The reset mechanism can be enabled in standby and active mode.
When this bit is enabled, the reset mechanism resets all functional block registers and loads the respective internal registers with default values If the system was already in standby mode, the reboot process will
immediately begin; else if the system was in active mode, the boot mechanism will automatically transition the system from active mode to standby mode, and only then can the reboot process begin.

Returns

Status value (0 for success)

11.4.5 mpl_status_t MPL_ToggleOneShot ( )

Toggle the OST bit.

Note

Clears then sets the OST bit which causes the sensor to immediately take another reading, necessary to sample faster than 1Hz.

Returns

Status value (0 for success)

11.4.6 mpl_status_t MPL_GetID (uint8_t * *sensorID*)

Read the chip ID.

Parameters

sensorID	Chip ID value

Returns

Status value (0 for success)

11.4.7 mpl_status_t MPL_SetMode (modeMPL_t *mode*)

Set the device mode (barometer or altimeter).

Parameters

mode	1 - barometer 0 - altimeter

Returns

Status value (0 for success)

11.4.8 mpl_status_t MPL_GotoStandby ( )

Put the sensor in stand-by mode.

Note

It is needed to modify major control registers.

Returns

Status value (0 for success)

11.4.9 mpl_status_t MPL_SetActive ( )

Put the sensor in active mode.

Returns

Status value (0 for success)

11.4.10 mpl_status_t MPL_SetOversampleRate (uint8_t *sampleRate*)

Set the over-sample rate.

Note

Datasheet calls for 128, but you can set it from 1 to 128 samples. The higher the oversample rate, the greater the time between data samples.

Parameters

sampleRate	Over-sample rate value

Returns

Status value (0 for success)

11.4.11 mpl_status_t MPL_SetAutoAcquisitionTime (uint8_t *sampleTime*)

Set the auto-acquisition time step.

Note

Reset value = 0.
Step is power(2; sampleTime).

Parameters

sampleTime	Sample time value

Returns

Status value (0 for success)

11.4.12 mpl_status_t MPL_EnableEventFlags ( )

Enable pressure and temperature measurement event flags.

Note

This is recommended in datasheet during setup.

Returns

Status value (0 for success)

11.4.13 mpl_status_t MPL_SetOffsetPressure (int8_t *pressOffset*)

Set the offset pressure correction.

Note

Pressure user accessible offset trim value number.
The user offset registers may be adjusted to enhance accuracy and optimize the system performance.
Range is from 512 to +508 Pa, 4 Pa/LSB.

Parameters

pressOffset	Pressure offset correction value.

Returns

Status value (0 for success)

11.4.14 mpl_status_t MPL_SetOffsetAltitude (int8_t *altitudeOffset*)

Set the offset altitude correction.

Note

Altitude user accessible offset trim value number.
The user offset register provides user adjustment to the vertical height of the altitude output.
The range of values are from 128 to +127 meters.

Parameters

altitudeOffset	Altitude offset correction value.

Returns

Status value (0 for success)

11.4.15 mpl_status_t MPL_SetOffsetTemperature (int8_t *temperatureOffset*)

Set the offset temperature correction.

Note

Temperature user accessible offset trim value number.
The range of values is from 8 to +7.9375 °C, 0.0625 °C/LSB.

Parameters

temperature↩ Offset	Temperature offset correction value.

Returns

Status value (0 for success)

11.4.16 mpl_status_t MPL_SetFifoMode (modeFIFO_t *fMode,* uint8_t *fWmrk*)

Set the FIFO mode.

Note

It can be configured in either circular buffer or in overflow mode.
In circular buffer mode, a watermark can be set to trigger a flag event. Exceeding the watermark, count does not stop the FIFO from accepting new data, the oldest data is overwritten.

Parameters

fMode	FIFO mode
fWmrk	Watermark

Returns

Status value (0 for success)

11.4.17 mpl_status_t MPL_SetFifoInterrupt (pinINT_t *pinINT*)

Setup the FIFO interrupt and route it to pin INT1 or INT2.

Parameters

pinINT	Interrupt pin selection (INT1 or INT2)

Returns

Status value (0 for success)

11.4.18 mpl_status_t MPL_DisableFifoInterrupt ( )

Disable the FIFO interrupt.

Returns

Status value (0 for success)

11.4.19 mpl_status_t MPL_GetFifoStatus (uint8_t * *fifoStatus*)

Read the FIFO status register.

Parameters

fifoStatus	Pointer to FIFO status

Returns

Status value (0 for success)

11.4.20 mpl_status_t MPL_ReadRawData (modeMPL_t *mode*, int32_t * *sensorData*)

Read sensor raw data.

Parameters

mode	Sensor mode (pressure, altitude...)
sensorData	Pointer to the sensor data

Returns

Status value (0 for success)

11.4.21 mpl_status_t MPL_Dump (uint8_t * *sensorReg*)

Read all the chip registers.

Parameters

sensorReg	Pointer to the register dump

Returns

Status value (0 for success)

Chapter 12
PCF2123 Real Time Clock/Calendar

12.1 Overview

This module provides the API to operate the PCF2123 real time clock/calendar through an SPI interface.

The basic steps to operate the PCF2123 are as follows:

1. Initialize the driver with callback functions (PCF2123_Init_Driver)
2. Initialize the hardware with initial settings (PCF2123_Init_Hw)
3. Get the current date or time (PCF2123_GetDateTime)
4. Enable an interrupt for an alarm or a countdown
5. If the PCF2123 is not needed anymore, de-initialize the driver (PCF2123_Deinit_Driver). The PC↩ F2123 will be switched off. It allows to eventually release shared resources.

Example - Sample application code to set PCF2123 without error management

```
#include "pcf2123.h"

pcf2123_IoFunc_t pcf2123_fct;
settingsPCF_t pcfSettings = {
    PCF2123_MODE_24HOURS,
    true,
    0x15,     // 15 seconds in BCD
    0x31,     // 31 minutes in BCD
    0x21,     // 21 hours in BCD
    12,       // 12th day of the month
    Thursday,
    April,
    0x18,     // year (20)18 in BCD
    clkout_High_Z,
    false,
    false,
    false
};
settingsPCF_t* pcfDateTime;

pcf2123_fct.SPI_Read = Spi_Read;    // callback function for SPI read
pcf2123_fct.SPI_Write = Spi_Write;  // callback function for SPI write
pcf2123_fct.WaitMsec = WaitMs;      // wait callback function (in ms)

PCF2123_Init_Driver(&pcf2123_fct);
PCF2123_Init_Hw(&pcfSettings);
// read new time
PCF2123_GetDateTime(&pcfDateTime);
printf("The current time is: %d:%d\n", pcfDateTime.hours, pcfDateTime.
      minutes);
// enable an alarm every day at 7:40am
PCF2123_SetHourAlarm(true, 0, 0x07);
PCF2123_SetMinuteAlarm(true, 0x40);
PCF2123_SetDayAlarm(false, 0);
PCF2123_SetWeekdayAlarm(false,0);
PCF2123_SetAlarmInterrupt(true);
[..]
```

```
// once alarm interrupt is triggered
PCF2123_ClearAlarmInterruptFlag();
[..]
// enable countdown for 60 seconds
PCF2123_SetCountdownTimer(60);
PCF2123_SetCountdownMode(true, CD_TMR_1HZ); // every second
[..]
// once countdown interrupt is triggered
PCF2123_ClearCountdownInterruptFlag();
PCF2123_SetCountdownMode(false, CD_TMR_1HZ); // stop countdown
[..]
// if the PCF2123 is not needed anymore, the driver can be de-initialized
printf("Deinitialize PCF2123 (it will be turned off)\n");
PCF2123_Deinit_Driver();
```

Data Structures

- struct ppcf2123_IoFunc_t
- struct settingsPCF_t

Macros

- #define **PCF2123_NB_REG**
- #define **PCF2123_WRITE**
- #define **PCF2123_READ**
- #define **PCF2123_ADDRESS_MASK**

Enumerations

- enum pcf2123_status_t {
  PCF2123_SUCCESS,
  PCF2123_I2C_ERROR,
  PCF2123_INIT_ERROR,
  PCF2123_TIMEOUT }
- enum modePCF_t {
  PCF2123_MODE_24HOURS,
  PCF2123_MODE_12HOURS }
- enum weekdaysPCF_t {
  Sunday,
  Monday,
  Tuesday,
  Wednesday,
  Thursday,
  Friday,
  Saturday }
- enum monthsPCF_t {

January,
February,
March,
April,
May,
June,
July,
August,
September,
October,
November,
December }

- enum clockOutputPCF_t {
clkoutFreq_32768,
clkoutfreq_16384,
clkoutFreq_8192,
clkoutFreq_4096,
clkoutFreq_2048,
clkoutFreq_1024,
clkoutFreq_1,
clkout_High_Z }
- enum cdmodePCF_t {
CD_TMR_4096KHZ,
CD_TMR_64HZ,
CD_TMR_1HZ,
CD_TMR_60th_HZ }

Functions

- void PCF2123_Init_Driver (pcf2123_IoFunc_t *pIoFunc)
- void PCF2123_Deinit_Driver ()
- pcf2123_status_t PCF2123_Init_Hw (const settingsPCF_t *pcfSettings)
- pcf2123_status_t PCF2123_Dump (uint8_t *pcfReg)
- pcf2123_status_t PCF2123_SetOffset (bool mode, uint8_t Offset)
- pcf2123_status_t PCF2123_GetDateTime (settingsPCF_t *pcfDateTime)
- pcf2123_status_t PCF2123_SetDateTime (settingsPCF_t *pcfDateTime)
- pcf2123_status_t PCF2123_SetClockOutputFreq (clockOutputPCF_t pcfClockOutputFreq)
- pcf2123_status_t PCF2123_SetMinSecInterrupt (bool Minute_Int, bool Second_Int, bool Pulse_↩ Int)
- pcf2123_status_t PCF2123_SetAlarmInterrupt (bool Alarm_Int)
- pcf2123_status_t PCF2123_SetCountdownInterrupt (bool Countdown_Int)
- pcf2123_status_t PCF2123_SetCountdownMode (bool CountDownEn, cdmodePCF_t Count↩ DownMode)
- pcf2123_status_t PCF2123_SetCountdownTimer (uint8_t CDT_Value)
- pcf2123_status_t PCF2123_SetMinuteAlarm (bool AlarmEn, uint8_t Minute)
- pcf2123_status_t PCF2123_SetHourAlarm (bool AlarmEn, bool AMPM, uint8_t Hour)
- pcf2123_status_t PCF2123_SetDayAlarm (bool AlarmEn, uint8_t Day)
- pcf2123_status_t PCF2123_SetWeekdayAlarm (bool AlarmEn, uint8_t Weekday)
- pcf2123_status_t PCF2123_ClearMinSecInterruptFlag ()

- pcf2123_status_t PCF2123_ClearCountdownInterruptFlag ()
- pcf2123_status_t PCF2123_ClearAlarmInterruptFlag ()

12.2 Data Structure Documentation

12.2.1 struct pcf2123_IoFunc_t

Structure of external functions or values.

Data Fields

- uint8_t(∗ SPI_Read)(uint8_t ∗writeBuf, uint8_t ∗readBuf, uint32_t readSize)
- uint8_t(∗ SPI_Write)(uint8_t ∗writeBuf, uint32_t writeSize)
- void(∗ WaitMsec)(uint32_t millisec)

12.2.1.1 Field Documentation

12.2.1.1.1 uint8_t(∗ SPI_Read) (uint8_t ∗writeBuf, uint8_t ∗readBuf, uint32_t readSize)

External SPI read function.

12.2.1.1.2 uint8_t(∗ SPI_Write) (uint8_t ∗writeBuf, uint32_t writeSize)

External SPI read function.

12.2.1.1.3 void(∗ WaitMsec) (uint32_t millisec)

Wait function in milliseconds.

12.2.2 struct settingsPCF_t

Real Time Clock Settings.

Data Fields

modePCF_t	mode12_24	12H or 24H mode
bool	Softreset	Force software reset => Date, time and interrupts need to be set after.

uint8_t	seconds	seconds in BCD
uint8_t	minutes	minutes in BCD
uint8_t	hours	hours in BCD. 24H mode: bit 5,4 used for first digit 12H mode: bit 5=AM/PM, bit 4 used for first digit
uint8_t	days	day of the month in BCD
weekdaysPCF_t	weekdays	weekday coded in BCD
monthsPCF_t	months	actual month coded in BCD
uint8_t	years	2 digits
clockOutputPCF_t	clockOutputFreq	Clock output pin frequency from 1H to 32768Hz or Disabled = high Z.
bool	MinInterrupt	Minute interrupt.
bool	SecInterrupt	Second interrupt.
bool	PulseInterrupt	Interrupt generate a pulse when true. Follows timer flag when false

12.3 Enumeration Type Documentation

12.3.1 enum pcf2123_status_t

Status return codes.

Enumerator

PCF2123_SUCCESS Success.
PCF2123_I2C_ERROR I2C read/write error.
PCF2123_INIT_ERROR Hardware or driver not initialized.
PCF2123_TIMEOUT Timeout.

12.3.2 enum modePCF_t

12 or 24 Hour Mode

Enumerator

PCF2123_MODE_24HOURS 24 Hour Mode
PCF2123_MODE_12HOURS 12 Hour Mode

12.3.3 enum weekdaysPCF_t

Day.

Enumerator

Sunday Sunday.
Monday Monday.
Tuesday Tuesday.
Wednesday Wednesday.
Thursday Thursday.
Friday Friday.
Saturday Saturday.

12.3.4 enum monthsPCF_t

Month.

Enumerator

January January.
February February.
March March.
April April.
May May.
June June.
July July.
August August.
September September.
October October.
November November.
December December.

12.3.5 enum clockOutputPCF_t

Clock Output Frequency.

Enumerator

clkoutFreq_32768 Clock output frequency = 32768 Hz.
clkoutfreq_16384 Clock output frequency = 16384 Hz.
clkoutFreq_8192 Clock output frequency = 8192 Hz.
clkoutFreq_4096 Clock output frequency = 4096 Hz.
clkoutFreq_2048 Clock output frequency = 2048 Hz.
clkoutFreq_1024 Clock output frequency = 1024 Hz.
clkoutFreq_1 Clock output frequency = 1 Hz.
clkout_High_Z Clock output is High Z.

12.3.6 enum cdmodePCF_t

Countdown Timer.

Enumerator

CD_TMR_4096KHZ 4096 KHz countdown timer
CD_TMR_64HZ 64 Hz countdown timer
CD_TMR_1HZ 1 Hz countdown timer
CD_TMR_60th_HZ 60th Hz countdown timer

12.4 Function Documentation

12.4.1 void PCF2123_Init_Driver (pcf2123_IoFunc_t * *pIoFunc*)

Initialize PCF2123 driver.

Parameters

pIoFunc	Pointer to a structure of external functions or values

12.4.2 void PCF2123_Deinit_Driver ( )

De-initialize PCF2123 driver.

12.4.3 pcf2123_status_t PCF2123_Init_Hw (const settingsPCF_t * *pcfSettings*)

Initialize PCF2123 hardware.

Parameters

pcfSettings	Pointer to a configuration structure

Returns

The return status value (0 for success)

12.4.4 pcf2123_status_t PCF2123_Dump (uint8_t * *pcfReg*)

Read all chip registers.

Parameters

pcfReg	Pointer to register dump.

Returns

The return status value (0 for success)

12.4.5 pcf2123_status_t PCF2123_SetOffset (bool *mode,* uint8_t *Offset*)

Set Offset register The offset is made once every two hours in the normal mode, or once every hour in the course mode.

Each LSB will introduce an offset of 2.17 ppm for normal mode and 4.34 ppm for course mode. The values of 2.17 ppm and 4.34 ppm are based on a nominal 32.768 kHz clock. The offset value is coded in two's complement giving a range of +63 LSB to -64 LSB. Normal mode = 0, Course mode = 1

Parameters

mode	(normal = 0, course = 1)
Offset	(+63..-64, two's complement)

Returns

The return status value (0 for success)

12.4.6 pcf2123_status_t PCF2123_GetDateTime (settingsPCF_t * *pcfDateTime*)

Read Date and Time.

Parameters

pcfDateTime	Pointer to the updated structure

Returns

The return status value (0 for success)

12.4.7 pcf2123_status_t PCF2123_SetDateTime (settingsPCF_t * *pcfDateTime*)

Set Date and Time.

Parameters

pcfDateTime	Pointer to the reference structure

Returns

The return status value (0 for success)

12.4.8 pcf2123_status_t PCF2123_SetClockOutputFreq (clockOutputPCF_t *pcfClockOutputFreq*)

Set Clock output pin frequency from 1Hz up to 32768Hz, or disabled = High Z.

Parameters

pcfClock↩ OutputFreq	Pointer to the clock output frequency value

Returns

The return status value (0 for success)

12.4.9 pcf2123_status_t PCF2123_SetMinSecInterrupt (bool *Minute_Int,* bool *Second_Int,* bool *Pulse_Int*)

Set Minute and second interrupt The minute and second interrupts (bits MI and SI) are pre-defined timers for generating periodic interrupts.

The timers can be enabled independently from one another. However, a minute interrupt enabled on top of a second interrupt will not be distinguishable since it will occur at the same time

Parameters

Minute_Int	Enable the minute interrupt
Second_Int	Enable the second interrupt
Pulse_Int	Enable the pulse interrupt

Returns

The return status value (0 for success)

12.4.10 pcf2123_status_t PCF2123_SetAlarmInterrupt (bool *Alarm_Int*)

Set Alarm interrupt Interrupt generated when alarm flag set.

Parameters

Alarm_Int	Enable the alarm interrupt

Returns

The return status value (0 for success)

12.4.11 pcf2123_status_t PCF2123_SetCountdownInterrupt (bool *Countdown_Int*)

Set Countdown interrupt Interrupt generated when countdown flag set.

Parameters

Countdown_Int	Enable the countdown interrupt

Returns

The return status value (0 for success)

12.4.12 pcf2123_status_t PCF2123_SetCountdownMode (bool *CountDownEn,* cdmodePCF_t *CountDownMode*)

Set Countdown timer source clock.

Parameters

CountDownEn	Enable the countdown timer source clock
CountDown↩ Mode	Countdown mode (CD_TMR_4096KHZ, CD_TMR_64HZ, CD_TMR_1HZ, CD_↩ TMR_60th_HZ)

Returns

The return status value (0 for success)

12.4.13 pcf2123_status_t PCF2123_SetCountdownTimer (uint8_t *CDT_Value*)

Set Countdown timer value.

Parameters

CDT_Value	Countdown timer value

Returns

The return status value (0 for success)

12.4.14 pcf2123_status_t PCF2123_SetMinuteAlarm (bool *AlarmEn,* uint8_t *Minute*)

Set Minute alarm Minute alarm information coded in ∗∗∗ BCD format ∗∗∗ When the register Minute↩ _alarm is loaded with a valid minute and it's corresponding alarm bit is logic 0 that information will be compared with the current minute.

Parameters

AlarmEn	Enable alarm
Minute	Number of minutes

Returns

The return status value (0 for success)

12.4.15 pcf2123_status_t PCF2123_SetHourAlarm (bool *AlarmEn,* bool *AMPM,* uint8_t *Hour*)

Set Hour alarm Hour alarm information coded in ∗∗∗ BCD format ∗∗∗.

When the register Hour is loaded with a valid hour and it's corresponding alarm bit is logic 0 that information will be compared with the current hour. AMPM = 0 indicates AM (only valid in 12 hours mode)

Parameters

AlarmEn	Enable alarm
AMPM	AMPM mode (0 is AM, 1 is PM)
Hour	Number of hours

Returns

The return status value (0 for success)

12.4.16 pcf2123_status_t PCF2123_SetDayAlarm (bool *AlarmEn,* uint8_t *Day*)

Set Day alarm Day alarm information coded in ∗∗∗ BCD format ∗∗∗.

When the register Day_alarm is loaded with a valid day and it's corresponding alarm bit is logic 0 that information will be compared with the current Day.

Parameters

AlarmEn	Enable alarm
Day	Number of days

Returns

The return status value (0 for success)

12.4.17 pcf2123_status_t PCF2123_SetWeekdayAlarm (bool *AlarmEn,* uint8_t *Weekday*)

Set Weekday alarm Weekday alarm information coded in ∗∗∗ BCD format ∗∗∗.

When the register weekday_alarm is loaded with a valid day and it's corresponding alarm bit is logic 0 that information will be compared with the current weekday.

Parameters

AlarmEn	Enable alarm
Weekday	Day of the week (0-6: 0 is Sunday, etc.)

Returns

The return status value (0 for success)

12.4.18 pcf2123_status_t PCF2123_ClearMinSecInterruptFlag ( )

Clear minute or second interrupt flag MSF.

Returns

The return status value (0 for success)

12.4.19 pcf2123_status_t PCF2123_ClearCountdownInterruptFlag ( )

Clear Timer countdown flag TF Flag set as the end of a timer countdown.

Returns

The return status value (0 for success)

12.4.20 pcf2123_status_t PCF2123_ClearAlarmInterruptFlag ( )

Clear Alarm flag AF Flag set by alarm event.

Returns

The return status value (0 for success)

Chapter 13
Port Interrupts

13.1 Overview

The port_interrupts module provides a set of turn-key interrupt handlers and enablement functions.

The following interrupts are captured by the port_interrupts module:

- PORTA:
  - User switch SW1 (hw label: USER_SW1)
  - Touch interrupt (hw label: TOUCH_INT)
  - Air quality interrupt (hw label: AIR_INTN)
  - NTAG field detect (hw label: NTAG_FD)
  - Charging state (hw label: CHG_STATE)
- PORTB:
  - External module 1 interrupt (hw label: MB1_INT)
  - External module 2 interrupt (hw label: MB2_INT)
  - External module 3 interrupt (hw label: MB3_INT)
- PORTC:
  - Ambient light interrupt (hw label: AMB_INT)
  - Accelerometer interrupt INT1 (hw label: ACCEL_INT1)
  - Gyroscope interrupt INT2 (hw label: GYRO_INT2)
- PORTD:
  - Real Time Clock interrupt (hw label: RTC_INT)
  - Gyroscope interrupt INT1 (hw label: GYRO_INT1)
  - Pressure interrupt INT2 (hw label: PRESSURE_INT2)
  - Pressure interrupt INT1 (hw label: PRESSURE_INT1)
  - Accelerometer interrupt INT2 (hw label: ACCEL_INT2)
- PORTE:
  - User switch SW2 (hw label: USER_SW2)
  - User switch SW3 (hw label: USER_SW3)
  - User switch SW4 (hw label: USER_SW4)

Usage

Initialization:

```
// Enable all Rapid IoT PORT Interrupts
PORT_IRQ_EnablePortAIrq();
PORT_IRQ_EnablePortBIrq();
PORT_IRQ_EnablePortCIrq();
PORT_IRQ_EnablePortDIrq();
PORT_IRQ_EnablePortEIrq();
```

Function Documentation

Adding Callbacks:

```
// Create a non-blocking callback function
// NOTE: callback functions's signature can vary
void myAmbLightCallback(void)
{
    // Ambient Light interrupt Callback code
    ...
}

// In port_interrupts.c edit associated interrupt handler
void PORTC_IRQHandler(void)
{
    uint32_t pin_nb;
    pin_nb = PORT_GetPinsInterruptFlags(PORTC);

    if (pin_nb & (1 << BOARD_INITPINS_AMB_INT_GPIO_PIN))
    {
        //Add callback for Ambient Light Interrupt
         myAmbLightCallback();
         GPIO_ClearPinsInterruptFlags(BOARD_INITPINS_AMB_INT_GPIO, 1U << BOARD_INITPINS_AMB_INT_GPIO_PIN);
    }
    ...
}
```

Port Interrupt enablement and handling functions

Functions

- void PORT_IRQ_EnablePortAIrq (void)
- void PORT_IRQ_EnablePortBIrq (void)
- void PORT_IRQ_EnablePortCIrq (void)
- void PORT_IRQ_EnablePortDIrq (void)
- void PORT_IRQ_EnablePortEIrq (void)
- void PORTA_IRQHandler (void)
- void PORTB_IRQHandler (void)
- void PORTC_IRQHandler (void)
- void PORTD_IRQHandler (void)
- void PORTE_IRQHandler (void)
- void PORT_IRQ_UiEnableIrq (void)
- void PORT_IRQ_UiDisableIrq (void)

13.2 Function Documentation

13.2.1 void PORT_IRQ_EnablePortAIrq (void)

Enable IRQs on PORTA for sensors with interrupt lines and push button switch SW1.

Returns

None

13.2.2 void PORT_IRQ_EnablePortBIrq (void)

Enable IRQs on PORTB for external modules interrupt lines.

Returns

None

13.2.3 void PORT_IRQ_EnablePortCIrq (void)

Enable IRQs on PORTC for sensors interrupt lines.

Returns

None

13.2.4 void PORT_IRQ_EnablePortDIrq (void)

Enable IRQs on PORTD for sensors interrupt lines.

Returns

None

13.2.5 void PORT_IRQ_EnablePortEIrq (void)

Enable IRQs on PORTE for push button switches SW2, SW3, SW4.

Returns

None

13.2.6 void PORTA_IRQHandler (void)

PORTA_IRQHandler IRQ handler for all PORT A interrupts:

- User switch SW1 (hw label: USER_SW1)
- Touch interrupt (hw label: TOUCH_INT)
- Air quality interrupt (hw label: AIR_INTN)
- NTAG field detect (hw label: NTAG_FD)
- Charging state (hw label: CHG_STATE)

Returns

None

13.2.7 void PORTB_IRQHandler (void)

PORTB_IRQHandler IRQ handler for all PORT B interrupts:

- External module 1 interrupt (hw label: MB1_INT)
- External module 2 interrupt (hw label: MB2_INT)
- External module 3 interrupt (hw label: MB3_INT)

Returns

None

13.2.8 void PORTC_IRQHandler (void)

PORTC_IRQHandler IRQ handler for all PORT C interrupts:

- Ambient light interrupt (hw label: AMB_INT)
- Accelerometer interrupt INT1 (hw label: ACCEL_INT1)
- Gyroscope interrupt INT2 (hw label: GYRO_INT2)

Returns

None

13.2.9 void PORTD_IRQHandler (void)

PORTD_IRQHandler IRQ handler for all PORT D interrupts:

- Real Time Clock interrupt (hw label: RTC_INT)
- Gyroscope interrupt INT1 (hw label: GYRO_INT1)
- Pressure interrupt INT2 (hw label: PRESSURE_INT2)
- Pressure interrupt INT1 (hw label: PRESSURE_INT1)
- Accelerometer interrupt INT2 (hw label: ACCEL_INT2)

Returns

None

13.2.10 void PORTE_IRQHandler (void)

PORTE_IRQHandler IRQ handler for all PORT E interrupts:

- User switch SW2 (hw label: USER_SW2)
- User switch SW3 (hw label: USER_SW3)
- User switch SW4 (hw label: USER_SW4)

Returns

None

13.2.11 void PORT_IRQ_UiEnableIrq (void)

Enable IRQ for All UI related interrupt lines.

Returns

None

13.2.12 void PORT_IRQ_UiDisableIrq (void)

Disable IRQ for All UI related interrupt lines.

Returns

None

Chapter 14
RGB LED

14.1 Overview

This module provides the API to operate the RGB LED.

The basic steps to operate the RGB LED are as follows:

1. Initialize the driver with callback functions (RGB_Led_Init_Driver)
2. Initialize the hardware (RGB_Led_Init_Hw)
3. Set the new RGB LED brightness and color (RGB_Led_Set_State)
4. If the RGB LED is not needed anymore, de-initialize the driver (RGB_Led_Deinit_Driver). The RGB LED will be switched off. It allows to eventually release shared resources.

Example - Sample application code to set RGB LED without error management

```
#include "rgb_led.h"

rgbled_fct_t rgbled_fct;

rgbled_fct.connect_hw = Rgb_Led_Connect;        // callback function to activate RGB LED hardware
       resource
rgbled_fct.disconnect_hw = Rgb_Led_Disconnect;  // callback function to deactivate RGB LED
       hardware resource
rgbled_fct.set_rgb_colors = Rgb_Led_Set_Colors; // callback function to control separately
       the R/G/B colors

RGB_Led_Init_Driver(&rgbled_fct);
RGB_Led_Init_Hw();
// set the new RGB LED brightness and color
RGB_Led_Set_State(RGB_LED_BRIGHT_MEDIUM,
      RGB_LED_COLOR_GREEN);
[..]
// if the RGB LED is not needed anymore, the driver can be de-initialized
printf("Deinitialize RGB LED (it will be turned off)\n");
RGB_Led_Deinit_Driver();
```

Data Structures

- struct rgbled_fct_t

Typedefs

- typedef enum _generic_status **RgbLedErrorCode_t**

Enumerations

- enum rgb_led_brightness_t {
  RGB_LED_BRIGHT_OFF,
  RGB_LED_BRIGHT_LOW,
  RGB_LED_BRIGHT_MEDIUM,
  RGB_LED_BRIGHT_HIGH }
- enum rgb_led_color_t {
  RGB_LED_COLOR_RED,
  RGB_LED_COLOR_GREEN,
  RGB_LED_COLOR_BLUE,
  RGB_LED_COLOR_WHITE,
  RGB_LED_COLOR_YELLOW,
  RGB_LED_COLOR_CYAN,
  RGB_LED_COLOR_PURPLE,
  RGB_LED_COLOR_BLACK }

Functions

- void RGB_Led_Init_Driver ()
- void RGB_Led_Deinit_Driver ()
- RgbLedErrorCode_t RGB_Led_Init_Hw ()
- RgbLedErrorCode_t RGB_Led_Get_State (uint8_t *brightness, uint8_t *color)
- RgbLedErrorCode_t RGB_Led_Set_State (uint8_t brightness, uint8_t color)

14.2 Data Structure Documentation

14.2.1 struct rgbled_fct_t

Structure of external functions or values.

Data Fields

- void(* connect_hw)(void)
- void(* disconnect_hw)(void)
- void(* set_rgb_colors)(uint8_t R, uint8_t G, uint8_t B)

14.2.1.1 Field Documentation

14.2.1.1.1 void(* connect_hw) (void)

External function to activate the RGB LED hardware.

14.2.1.1.2 void(* disconnect_hw) (void)

External function to deactivate the RGB LED hardware.

14.2.1.1.3 void(* set_rgb_colors) (uint8_t R, uint8_t G, uint8_t B)

External function to set R/G/B colors.

14.3 Enumeration Type Documentation

14.3.1 enum rgb_led_brightness_t

LED Brightness.

Enumerator

RGB_LED_BRIGHT_OFF LED Off.
RGB_LED_BRIGHT_LOW LED Low Brightness.
RGB_LED_BRIGHT_MEDIUM LED Medium Brightness.
RGB_LED_BRIGHT_HIGH LED High Brightness.

14.3.2 enum rgb_led_color_t

LED Color.

Enumerator

RGB_LED_COLOR_RED LED Color: Red.
RGB_LED_COLOR_GREEN LED Color: Green.
RGB_LED_COLOR_BLUE LED Color: Blue.
RGB_LED_COLOR_WHITE LED Color: White.
RGB_LED_COLOR_YELLOW LED Color: Yellow.
RGB_LED_COLOR_CYAN LED Color: Cyan.
RGB_LED_COLOR_PURPLE LED Color: Purple.
RGB_LED_COLOR_BLACK LED Color: Black /OFF.

14.4 Function Documentation

14.4.1 void RGB_Led_Init_Driver ( )

Initialize RGB LED driver.

14.4.2 void RGB_Led_Deinit_Driver ( )

De-initialize RGB LED driver.

Function Documentation

14.4.3 RgbLedErrorCode_t RGB_Led_Init_Hw ( )

Initialize RGB LED hardware.

Returns

Status value (0 for success)

14.4.4 RgbLedErrorCode_t RGB_Led_Get_State (uint8_t * *brightness,* uint8_t * *color*)

Get current RGB LED brightness and color values.

Note

The values come from the internal sw settings, not from hw acquisition.

Parameters

brightness	Pointer to a preselected brightness value: 0 (Off), 1 (Low), 2 (Medium) or 3 (High)
color	Pointer to a preselected color value: 0 (Red), 1 (Green), 2 (Blue) or 3 (White)

Returns

Status value (0 for success)

14.4.5 RgbLedErrorCode_t RGB_Led_Set_State (uint8_t *brightness,* uint8_t *color*)

Set RGB LED based on preselected brightness and color values.

Parameters

brightness	Preselected brightness value: 0 (Off), 1 (Low), 2 (Medium) or 3 (High)
color	Preselected color value: 0 (Red), 1 (Green), 2 (Blue) or 3 (White)

Returns

Status value (0 for success)

Chapter 15
Base64 String Encoder/Decoder

15.1 Overview

The rpk_base64 module provides an efficient method of encoding and decoding Base64 strings.

Usage

Encoding:

```
const char msg = "Hello world";

uint32_t encLength = RPK_Base64_Get_Encoded_Len(strlen(msg));

uint8_t encBuffer = (uint8_t*)malloc(encLength);

uint32_t ret = RPK_Base64_Encode((uint8_t*)msg, strlen(msg), encBuffer, encLength);

if (encLength != ret)
{
    // Something went wrong
}
```

Decoding:

```
const char* msg = "SGVsbG8gd29ybGQ=";

uint32_t decLength = RPK_Base64_Get_Decoded_Len((uint8_t*)msg, strlen(msg));

uint8_t decBuffer = (uint8_t*)malloc(decLength);

uint32_t ret = RPK_Base64_Decode((uint8_t*)msg, strlen(msg), decBuffer, decLength);

if (decLength != ret)
{
    // Something went wrong
}
```

Memory Considerations

It is the responsibility of the developer to allocate the necessary memory. See 'Usage' part above for methods on obtaining accurate buffer lengths needed. Base64 Encoding/Decoding

Files

- file rpk_base64.h

Functions

- uint32_t RPK_Base64_Get_Encoded_Len (uint32_t len)
- uint32_t RPK_Base64_Get_Decoded_Len (uint8_t *buf, uint32_t len)
- uint32_t RPK_Base64_Encode (uint8_t *in, uint32_t numInBytes, uint8_t *out, uint32_t numOut↩ Bytes)
- uint32_t RPK_Base64_Decode (uint8_t *in, uint32_t numInBytes, uint8_t *out, uint32_t numOut↩ Bytes)

15.2 Function Documentation

15.2.1 uint32_t RPK_Base64_Get_Encoded_Len (uint32_t *len*)

Get Encoded Length - return buffer length needed to store encoded version of data.

Note

Call before dynamic allocation of memory for storing base64 encoded buffer

Parameters

len	Length of raw data buffer in bytes

Returns

Length in bytes of base64 encoded version of given data buffer

15.2.2 uint32_t RPK_Base64_Get_Decoded_Len (uint8_t * *buf,* uint32_t *len*)

Get Decoded Length - return buffer length needed to store decoded version of data.

Note

Call before dynamic allocation of memory for storing decoded buffer from base64 data

Parameters

buf	Pointer to encoded data buffer
len	Length of encoded data buffer in bytes

Returns

Length in bytes of decoded version of given base64 data buffer; 0 if NULL pointer detected

15.2.3 uint32_t RPK_Base64_Encode (uint8_t * *in*, uint32_t *numInBytes*, uint8_t * *out*, uint32_t *numOutBytes*)

Encode data with base64 encoding.

Note

RPK_Base64_Get_Encoded_Len should be called before hand to set valid numOutBytes

Parameters

in	Pointer to raw data to encode
numInBytes	Length of raw data to encode
out	Pointer to encoded data that will be output
numOutBytes	Length of resulting encoded data

Returns

Length in bytes of encoded data; 0 if NULL pointer detected

15.2.4 uint32_t RPK_Base64_Decode (uint8_t * *in*, uint32_t *numInBytes*, uint8_t * *out*, uint32_t *numOutBytes*)

Decode base64 into raw binary data.

Note

RPK_Base64_Get_Decoded_Len should be called before hand to set valid numOutBytes

Parameters

in	Pointer to encoded data
numInBytes	Length of encoded data
out	Pointer to raw data that will be output
numOutBytes	Length of resulting raw binary data

Returns

Length in bytes of decoded data; 0 if NULL pointer detected

Chapter 16
SX9500 Touch Controller

16.1 Overview

This module provides the API to operate the SX9500 4-channel capacitive controller (touch) through an I2C interface.

Usage

Initialization:

```
sx9500_fct_t FCT_SX9500;
FCT_SX9500.connect_hw = Touch_Controller_Connect;
FCT_SX9500.disconnect_hw = Touch_Controller_Disconnect;
FCT_SX9500.I2C_Write = App_I2C1_Write;
FCT_SX9500.I2C_Read = App_I2C1_Read;
FCT_SX9500.WaitMs = App_WaitMsec;

SX9500_Init_Driver(&FCT_SX9500);

SX9500_status state = SX9500_Init_Hw();
```

Interrupt Handling:

```
RegStat_t dir;
RegIrqSrc_t irq;

if (SX9500_Read_Irq(&irq.octet) == SX9500_SUCCESS)
{
    if (irq.bits.close || irq.bits.far)
    {
        SX9500_Read_Proximity_Sensors(&dir.octet);

        if (dir.bits.proxstat0)
        {
            // Sensor CS0 event handling
        }

        if (dir.bits.proxstat1)
        {
            // Sensor CS1 event handling
        }

        if (dir.bits.proxstat2)
        {
            // Sensor CS2 event handling
        }

        if (dir.bits.proxstat3)
        {
            // Sensor CS3 event handling
        }
    }
}
```

Data Structures

- union RegIrqSrc_t
- struct RegIrqSrc_t.bits
- union RegStat_t
- struct RegStat_t.bits
- union RegProxCtrl0_t
- struct RegProxCtrl0_t.bits
- union RegProxCtrl3_t
- struct RegProxCtrl3_t.bits
- struct ptsx9500_fct_t

Macros

- #define **SX9500_I2C_ADDRESS**
- #define **SX9500_REG_IRQSRC**
- #define **SX9500_REG_STAT**
- #define **SX9500_REG_IRQMSK**
- #define **SX9500_REG_PROXCTRL0**
- #define **SX9500_REG_PROXCTRL1**
- #define **SX9500_REG_PROXCTRL2**
- #define **SX9500_REG_PROXCTRL3**
- #define **SX9500_REG_PROXCTRL4**
- #define **SX9500_REG_PROXCTRL5**
- #define **SX9500_REG_PROXCTRL6**
- #define **SX9500_REG_PROXCTRL7**
- #define **SX9500_REG_PROXCTRL8**
- #define **SX9500_REG_SENSORSEL**
- #define **SX9500_REG_USEMSB**
- #define **SX9500_REG_USELSB**
- #define **SX9500_REG_AVGMSB**
- #define **SX9500_REG_AVGLSB**
- #define **SX9500_REG_DIFFMSB**
- #define **SX9500_REG_DIFFLSB**
- #define **SX9500_REG_OFFSETMSB**
- #define **SX9500_REG_OFFSETLSB**
- #define **SX9500_REG_RESET**
- #define **SX9500_RESET_CMD**

Enumerations

- enum SX9500_status {
  SX9500_SUCCESS,
  SX9500_I2C_ERROR,
  SX9500_INTERNAL_ERROR,
  SX9500_NOINIT_ERROR }

Functions

- void SX9500_Init_Driver (ptsx9500_fct_t FCT)
- void SX9500_Deinit_Driver ()
- SX9500_status SX9500_Init_Hw ()
- SX9500_status SX9500_GetInfo_sensor (char CSn, uint8_t *buf)
- bool SX9500_get_active ()

- SX9500_status SX9500_set_active (bool en)
- SX9500_status SX9500_CSi_Detected (uint8_t *CSi)
- SX9500_status SX9500_Read_Irq (uint8_t *irqReg)
- SX9500_status SX9500_Read_Proximity_Sensors (uint8_t *data)

16.2 Data Structure Documentation

16.2.1 union RegIrqSrc_t

Union for register 0x00 RegIrqSrc.

Data Fields

struct RegIrqSrc_t	bits	RegIrqSrc bit fields.
uint8_t	octet	RegIrqSrc byte value.

16.2.2 struct RegIrqSrc_t.bits

Data Fields

uint8_t	txen_stat: 1	TXENSTAT: Inidcates current TXEN pin status.
uint8_t	reserved: 2	Reserved bits.
uint8_t	conv_done: 1	CONVDONEIRQ: Conversion interrupt source status.
uint8_t	comp_done: 1	COMPDONEIRQ: Compensation interrupt source status.
uint8_t	far: 1	FARIRQ: Far interrupt source status.
uint8_t	close: 1	CLOSEIRQ: Close interrupt source status.
uint8_t	reset: 1	RESETIRQ: Reset interrupt source status.

16.2.3 union RegStat_t

Union for register 0x01 RegStat.

Data Fields

struct RegStat_t	bits	RegStat bit fields.
uint8_t	octet	RegStat byte value.

16.2.4 struct RegStat_t.bits

Data Fields

uint8_t	compstat: 4	COMPSTAT: Indicates which capacitive sensor(s) has a compensation pending.
uint8_t	proxstat0: 1	PROXSTAT0: Indicates if proximity is being detected for CS0.
uint8_t	proxstat1: 1	PROXSTAT1: Indicates if proximity is being detected for CS1.
uint8_t	proxstat2: 1	PROXSTAT2: Indicates if proximity is being detected for CS2.
uint8_t	proxstat3: 1	PROXSTAT3: Indicates if proximity is being detected for CS3.

16.2.5 union RegProxCtrl0_t

Union for register 0x06 RegProxCtrl0.

Data Fields

struct RegProxCtrl0_t	bits	RegStat bit fields.
uint8_t	octet	RegStat byte value.

16.2.6 struct RegProxCtrl0_t.bits

Data Fields

uint8_t	sensor_en: 4	SENSOREN: Enables sensor pins.
uint8_t	scan_period: 3	SCANPERIOD: Defines the Active scan period.
uint8_t	reserved: 1	Reserved bits.

16.2.7 union RegProxCtrl3_t

Union for register 0x09 RegProxCtrl3.

Data Fields

struct RegProxCtrl3_t	bits	RegStat bit fields.
uint8_t	octet	RegStat byte value.

16.2.8 struct RegProxCtrl3_t.bits

Data Fields

uint8_t	raw_filt: 2	RAWFILT: Defines PROXRAW filter strength.
uint8_t	reserved: 2	Reserved bits.
uint8_t	doze_period: 2	DOZEPERIOD: When DOZEN=1, defines the Doze scan period.
uint8_t	doze_en: 1	DOZEEN: Enables Doze mode.
uint8_t	res7: 1	Reserved bits.

16.2.9 struct sx9500_fct_t

Structure of external functions or values.

Data Fields

- void(∗ connect_hw)(void)
- void(∗ disconnect_hw)(void)
- uint8_t(∗ I2C_Read)(uint8_t device_addr, uint8_t ∗writeBuf, uint32_t writeSize, uint8_t ∗readBuf, uint32_t readSize)
- uint8_t(∗ I2C_Write)(uint8_t device_addr, uint8_t ∗writeBuf, uint32_t writeSize)
- void(∗ WaitMs)(uint32_t tms)

16.2.9.1 Field Documentation

16.2.9.1.1 void(∗ connect_hw) (void)

Exit Reset then Enable Transmit function.

16.2.9.1.2 void(∗ disconnect_hw) (void)

Disable Transmit and Enter Reset SX9500 device.

16.2.9.1.3 uint8_t(∗ I2C_Read) (uint8_t device_addr, uint8_t ∗writeBuf, uint32_t writeSize, uint8_t ∗readBuf, uint32_t readSize)

Read Hardware function.

16.2.9.1.4 uint8_t(∗ I2C_Write) (uint8_t device_addr, uint8_t ∗writeBuf, uint32_t writeSize)

Write hardware register, 8bit aligned function.

16.2.9.1.5 void(∗ WaitMs) (uint32_t tms)

Wait function.

16.3 Enumeration Type Documentation

16.3.1 enum SX9500_status

Status return codes.

Enumerator

SX9500_SUCCESS Success status code.
SX9500_I2C_ERROR I2C Error status code.
SX9500_INTERNAL_ERROR SX9500 Internal Error status code.
SX9500_NOINIT_ERROR SX9500 Not Initialized status code.

16.4 Function Documentation

16.4.1 void SX9500_Init_Driver (ptsx9500_fct_t *FCT*)

Initialize SX9500 driver.

Parameters

FCT	Pointer to a structure with external functions

16.4.2 void SX9500_Deinit_Driver ( )

De-initialize SX9500 driver.

16.4.3 SX9500_status SX9500_Init_Hw ( )

Initialize SX9500 hardware.

Returns

Status value (0 for success)

16.4.4 SX9500_status SX9500_GetInfo_sensor (char *CSn,* uint8_t ∗ *buf*)

Get information for selected sensor.

Parameters

CSn	Selected sensor (0-3)
buf	Pointer to an array with PROXUSEFUL/PROXAVG/PROXDIFF values

Returns

Status value (0 for success)

16.4.5 bool SX9500_get_active ( )

Get power mode.

Returns

Power mode: active (true/1) or low power (false/0)

16.4.6 SX9500_status SX9500_set_active (bool *en*)

Set power mode.

Parameters

en	Power mode: active (true/1) or low power (false/0)

Returns

Status value (0 for success)

16.4.7 SX9500_status SX9500_CSi_Detected (uint8_t * *CSi*)

Function returning the triggered capacitive sensing interface.

Note

Returns 0xFF if no interface has been triggered.

Parameters

CSi	Pointer to sensing interface index

Returns

Status value (0 for success)

16.4.8 SX9500_status SX9500_Read_Irq (uint8_t * *irqReg*)

Read IRQ Source.

Note

Used to clear interrupts on SX9500

Parameters

irqReg	Pointer to store value from IRQ register

Returns

Status value (0 for success)

16.4.9 SX9500_status SX9500_Read_Proximity_Sensors (uint8_t * *data*)

Read Proximity Sensors.

Parameters

data	Pointer to store data from proximity sensor data

Returns

Status value (0 for success)

Chapter 17
TSL2572 Ambient Light Sensor

17.1 Overview

This module provides the API to operate the TSL2572 ambient light sensor through an I2C interface.

The basic steps to operate the TSL2572 are as follows:

1. Initialize the driver with callback functions (TSL2572_Init_Driver)
2. Initialize the hardware (TSL2572_Init_Hw)
3. Eventually wait for an interrupt
4. Read data (TSL2572_ReadAmbientLight)
5. Eventually clear the interrupt
6. If the TSL2572 is not needed anymore, de-initialize the driver (TSL2572_Deinit_Driver). The TS↩ L2572 will be switched off.

Example - Sample application code to set TSL2572 without error management

```
#include "tsl2572.h"

tsl2572_IoFunc_t tsl2572_fct;
float fAmbLight;

tsl2572_fct.I2C_Read = I2c_Read;    // callback function for I2C read
tsl2572_fct.I2C_Write = I2c_Write;  // callback function for I2C write
tsl2572_fct.WaitMsec = WaitMs;      // wait callback function (in ms)

TSL2572_Init_Driver(&tsl2572_fct);
// Initialize TSL2572 with default settings
TSL2572_Init_Hw();
// read data
TSL2572_ReadAmbientLight(&fAmbLight);
printf("The current value is: %d lux\n", fAmbLight);
// clear interrupt
TSL2572_ClearALSInterrupt();
[..]
// if the TSL2572 is not needed anymore, the driver can be de-initialized
printf("Deinitialize TSL2572 (it will be turned off)\n");
TSL2572_Deinit_Driver();
```

Data Structures

- struct ptsl2572_IoFunc_t
- struct tsl2x7x_settings_t

Enumerations

- enum tsl2572_status_t {
  tsl2572_success,
  tsl2572_I2C_error,
  tsl2572_invalid_ID,
  tsl2572_wrong_parameter,
  tsl2572_not_initialized }

Functions

- void TSL2572_Init_Driver (tsl2572_IoFunc_t *pIoFunc)
- void TSL2572_Deinit_Driver ()
- tsl2572_status_t TSL2572_Init_Hw (void)
- tsl2572_status_t TSL2572_ReadAmbientLight (float *lux)
- tsl2572_status_t TSL2572_SetALSGain (uint8_t AGAIN, bool AGL)
- tsl2572_status_t TSL2572_SetALSTime (uint8_t ATIME)
- tsl2572_status_t TSL2572_SetALSThresholds (uint16_t ALS_interrupt_Low_Threshold, uint16_t ALS_interrupt_High_Threshold)
- tsl2572_status_t TSL2572_GetALSThresholds (uint16_t *ALS_interrupt_Low_Threshold, uint16_t *ALS_interrupt_High_Threshold)
- tsl2572_status_t TSL2572_SetALSPersistence (uint8_t APERS)
- tsl2572_status_t TSL2572_SetWaitTime (uint8_t WTIME, bool WLONG)
- tsl2572_status_t TSL2572_EnableALSInterrupts (bool AIEN)
- tsl2572_status_t TSL2572_ClearALSInterrupt (void)
- tsl2572_status_t TSL2572_Enable_ALS (bool AEN)
- tsl2572_status_t TSL2572_Enable_Wait (bool WEN)
- tsl2572_status_t TSL2572_Power_ON (bool PON)
- tsl2572_status_t TSL2572_ReadAllRegisters (uint8_t *RegData)
- tsl2572_status_t TSL2572_ReadCH0 (uint8_t *RegData)

17.2 Data Structure Documentation

17.2.1 struct tsl2572_IoFunc_t

Structure of external functions or values.

Data Fields

- uint8_t(* I2C_Read)(uint8_t device_addr, uint8_t *writeBuf, uint32_t writeSize, uint8_t *readBuf, uint32_t readSize)
- uint8_t(* I2C_Write)(uint8_t device_addr, uint8_t *writeBuf, uint32_t writeSize)
- void(* WaitMsec)(uint32_t millisec)

17.2.1.1 Field Documentation

17.2.1.1.1 uint8_t(∗ I2C_Read) (uint8_t device_addr, uint8_t ∗writeBuf, uint32_t writeSize, uint8_t ∗readBuf, uint32_t readSize)

External I2C read function.

17.2.1.1.2 uint8_t(∗ I2C_Write) (uint8_t device_addr, uint8_t ∗writeBuf, uint32_t writeSize)

External I2C write function.

17.2.1.1.3 void(∗ WaitMsec) (uint32_t millisec)

Wait function in milliseconds.

17.2.2 struct tsl2x7x_settings_t

Internal structure for TSL2x7x settings.

Note

struct tsl2x7x_default_settings - power on defaults unless overridden by platform data.

Data Fields

uint8_t	als_time	ALS Integration time - multiple of 50ms.
uint8_t	als_gain	Index into the ALS gain table.
bool	als_gain_level	ALS gain level (when asserted, the 1× and 8× ALS gain (AGAIN) modes are scaled by 0.16)
uint8_t	wait_time	Time between PRX and ALS cycles in 2.7 periods.
bool	wlong	When asserted, the wait cycles are increased by a factor 12× from that programmed in the WTIME register.
bool	interrupts_↩ enable	Enable/Disable als interrupts.
uint8_t	persistence	H/W Filters, number of 'out of limits' ADC readings ALS.
uint16_t	als_thresh_low	CH0 'low' count to trigger interrupt.
uint16_t	als_thresh_↩ high	CH0 'high' count to trigger interrupt.

bool	als_enable	This bit activates the two ADC channels.
bool	wait_enable	This bit activates the wait feature.
bool	power_on	This bit activates the internal oscillator to permit the timers and ADC channels to operate.
float	glass_↩ attenuation	Scaling factor referred to as glass attenuation (GA), can be used to compensate for attenuation.

17.3 Enumeration Type Documentation

17.3.1 enum tsl2572_status_t

Status return codes.

Enumerator

tsl2572_success Success.
tsl2572_I2C_error I2C read/write error.
tsl2572_invalid_ID Bad hardware identifier.
tsl2572_wrong_parameter Parameter not allowed.
tsl2572_not_initialized Hardware or driver not initialized.

17.4 Function Documentation

17.4.1 void TSL2572_Init_Driver (tsl2572_IoFunc_t * *pIoFunc*)

Initialize TSL2572 driver.

Note

Wrap the low level functions (I2C write, I2C read, WaitMsec)

Parameters

pIoFunc	Pointer to a structure with external functions

17.4.2 void TSL2572_Deinit_Driver ( )

De-initialize TSL2572 driver.

17.4.3 tsl2572_status_t TSL2572_Init_Hw (void)

Initialize TSL2572 hardware.

Returns

Status value (0 for success)

17.4.4 tsl2572_status_t TSL2572_ReadAmbientLight (float * *lux*)

Read ambient light.

Note

Sample CH0 and CH1 photodiodes and compute the human eye response to light intensity (in lux)

Parameters

lux	Ambient light value (in lux)

Returns

Status value (0 for success)

17.4.5 tsl2572_status_t TSL2572_SetALSGain (uint8_t *AGAIN,* bool *AGL*)

Set ALS gain.

Note

AGAIN = 0 ALS Gain value = 1 * gain
AGAIN = 1 ALS Gain value = 8 * gain
AGAIN = 2 ALS Gain value = 16 * gain
AGAIN = 3 ALS Gain value = 120 * gain
AGL = 0 AGAIN = 0 or 1 or 2 or 3 -> scaling by 1
AGL = 1 AGAIN = 0 or 1 -> scaling by 0.16
Do not use AGL = 1 with AGAIN = 2 or 3

Parameters

AGAIN	ALS gain control value (AGAIN)
AGL	ALS gain level value (AGL)

Returns

Status value (0 for success)

17.4.6 tsl2572_status_t TSL2572_SetALSTime (uint8_t *ATIME*)

Set ALS ADC integration time time.

Note

The ALS time is the ALS ADC integration time.
ATIME = 0xFF ALS integration cycles = 1, time = 2.73ms
ATIME = 0xF6 ALS integration cycles = 10, time = 27.3ms
ATIME = 0xDB ALS integration cycles = 37, time = 101ms
ATIME = 0xC0 ALS integration cycles = 64, time = 175ms
ATIME = 0x00 ALS integration cycles = 256, time = 699ms

Parameters

ATIME	ALS time (ATIME)

Returns

Status value (0 for success)

17.4.7 tsl2572_status_t TSL2572_SetALSThresholds (uint16_t *ALS_interrupt_Low_Threshold,* uint16_t *ALS_interrupt_High_Threshold*)

Set ALS interrupt threshold low and threshold high.

Note

The thresholds refer to C0 photodiode only (C1 is not used to trigger interrupts).

Parameters

ALS_interrupt_Low_Threshold	ALS interrupt Low Threshold
ALS_interrupt_High_Threshold	ALS interrupt High Threshold

Returns

Status value (0 for success)

17.4.8 tsl2572_status_t TSL2572_GetALSThresholds (uint16_t * *ALS_interrupt_Low_Threshold,* uint16_t * *ALS_interrupt_High_Threshold*)

Get ALS interrupt threshold low and threshold high.

Parameters

ALS_interrupt_Low_Threshold	Pointer to ALS interrupt Low Threshold
ALS_interrupt_High_Threshold	Pointer to ALS interrupt High Threshold

Returns

Status value (0 for success)

17.4.9 tsl2572_status_t TSL2572_SetALSPersistence (uint8_t *APERS*)

Set ALS interrupt persistence filter.

Note

APERS = 0 every ALS cycle generates an interrupt
APERS = 1 1 value outside of threshold range generates an interrupt
APERS = 2 2 consecutive values out of range generates an interrupt
APERS = 3 3 consecutive values out of range generates an interrupt
APERS = 4 5 consecutive values out of range generates an interrupt
APERS = 5 10 consecutive values out of range generates an interrupt
APERS = 6 15 consecutive values out of range generates an interrupt
APERS = 7 20 consecutive values out of range generates an interrupt
APERS = 8 25 consecutive values out of range generates an interrupt
APERS = 9 30 consecutive values out of range generates an interrupt
APERS = 10 35 consecutive values out of range generates an interrupt
APERS = 11 40 consecutive values out of range generates an interrupt
APERS = 12 45 consecutive values out of range generates an interrupt
APERS = 13 50 consecutive values out of range generates an interrupt
APERS = 14 55 consecutive values out of range generates an interrupt
APERS = 15 60 consecutive values out of range generates an interrupt

Parameters

APERS	ALS persistence value (APERS)

Returns

Status value (0 for success)

Function Documentation

17.4.10 tsl2572_status_t TSL2572_SetWaitTime (uint8_t *WTIME,* bool *WLONG*)

Set wait time.

Note

WTIME = 0xFF Wait time = 2.73ms (WLONG = 0), 0.033s (WLONG = 1)
WTIME = 0xB6 Wait time = 202ms (WLONG = 0), 2.4s (WLONG = 1)
WTIME = 0x00 Wait time = 699ms (WLONG = 0), 8.4s (WLONG = 1)
The Wait time register should be configured before TSL2572_Enable_ALS(true)

Parameters

WTIME	Wait time value (WTIME)
WLONG	Wait time extension by a factor of 12 (WLONG)

Returns

Status value (0 for success)

17.4.11 tsl2572_status_t TSL2572_EnableALSInterrupts (bool *AIEN*)

Enable ALS interrupts.

Parameters

AIEN	ALS interrupts enable (AIEN)

Returns

Status value (0 for success)

17.4.12 tsl2572_status_t TSL2572_ClearALSInterrupt (void)

Clear ALS interrupts.

Returns

Status value (0 for success)

17.4.13 tsl2572_status_t TSL2572_Enable_ALS (bool *AEN*)

Enable ALS.

Parameters

AEN	ALS enable (AEN)

Returns

Status value (0 for success)

17.4.14 tsl2572_status_t TSL2572_Enable_Wait (bool *WEN*)

Enable Wait.

Parameters

WEN	Wait enable (WEN)

Returns

Status value (0 for success)

17.4.15 tsl2572_status_t TSL2572_Power_ON (bool *PON*)

Power ON.

Parameters

PON	Power ON enable (PON)

Returns

Status value (0 for success)

17.4.16 tsl2572_status_t TSL2572_ReadAllRegisters (uint8_t * *RegData*)

Read all registers.

Parameters

RegData	Pointer to an array of 16 * uint8_t

Returns

Status value (0 for success)

17.4.17 tsl2572_status_t TSL2572_ReadCH0 (uint8_t * *RegData*)

Read CH0.

Parameters

RegData	Pointer to an array of 2 ∗ uint8_t

Returns

Status value (0 for success)

Chapter 18
UI Manager

18.1 Overview

The ui_manager manages user interactions with the Rapid IoT device.

This includes event handling and display updates for the following:

- Touch Pads
- Push Buttons
- UI Lock for BLE OTA Updates

The ui_manager is intended to be used with the port_interrupts module. If not used with port_interrupts module, the developer must create their own interrupt enablement and interrupt handlers.

Dependencies

The following modules are necessary for the ui_manager:

- emWin GUI Library
- port_interrupts for handling touch and push button interrupts
- osaSemaphoreId_t gOtaSem (defined globaly)
- If LOAD_EXT_IMG is set to 1:
  - img_program_ext module for saving/loading BMP data from External SPI Flash
  - img_bmp_callbacks module for loading images from External SPI Flash to display
  - up_arrow.h, down_arrow.h, right_arrow.h, and left_arrow.h

Behavior

See local README.md

Usage

Initialization:

```
// Define semaphore globaly
osaSemaphoreId_t gOtaSem;

Display_Connect(); // Triggers GUI_Init()
Backlight_SetLevel(BLIGHT_LEVEL_LOW);

// Create semaphore to disable (lock) UI manager while running OTA
gOtaSem = OSA_SemaphoreCreate(0U);
```

```
if (NULL == gOtaSem)
{
    panic(0,0,0,0);
}

UiManager_Init();

PORT_IRQ_EnablePortAIrq();
PORT_IRQ_EnablePortEIrq();
```

Triggering UI Lock:

```
// Trigger UI Lock
UiManager_SetEvent(kUiLock);

// Perform critical task, i.e., BLE OTA

// Unlock UI with semaphore
OSA_SemaphorePost(gOtaSem);
```

Example Interrupt Handler:

```
void PORTA_IRQHandler(void)
{
    uint32_t pin_nb;
    pin_nb = PORT_GetPinsInterruptFlags(PORTA);

    if (pin_nb & (1 << BOARD_INITPINS_USER_SW1_GPIO_PIN))
    {
        UiManager_SetEvent(kUiSwOne);
        GPIO_ClearPinsInterruptFlags(BOARD_INITPINS_USER_SW1_GPIO, 1U << BOARD_INITPINS_USER_SW1_GPIO_PIN);
    }
}
```

User Interface manager for demonstration purposes

Macros

- #define **UI_TASK_STACK_SIZE**
- #define **UI_TASK_PRIO**

Enumerations

- enum ui_events_t {
  kUiTouchEv,
  kUiSwOne,
  kUiSwTwo,
  kUiSwThree,
  kUiSwFour,
  kUiSwMask,
  kUiAllHmiMask,
  kUiLock }

- enum ui_touch_t {
  kUiTouchDn,
  kUiTouchRt,
  kUiTouchUp,
  kUiTouchLt,
  kUiTouchDnRt,
  kUiTouchDnUp,
  kUiTouchDnLt,
  kUiTouchUpRt,
  kUiTouchUpLt,
  kUiTouchRtLt }

Functions

- void UiManager_SetEvent (uint32_t event)
- void UiManager_Init (void)

18.2 Enumeration Type Documentation

18.2.1 enum ui_events_t

UI Event Flags and Masks.

Event flags to be used in IRQ handlers to wake UI Manager task.

Enumerator

kUiTouchEv Touch event flag.
kUiSwOne Switch one event flag.
kUiSwTwo Switch two event flag.
kUiSwThree Switch three event flag.
kUiSwFour Switch four event flag.
kUiSwMask Switch event mask.
kUiAllHmiMask All UI HMI events mask.
kUiLock Lock UI to prevent user interaction.

18.2.2 enum ui_touch_t

UI Touch states.

States read from touch controller (SX9500), processed by UI Manager.

Enumerator

kUiTouchDn Single touch on down pad.
kUiTouchRt Single touch on right pad.

kUiTouchUp Single touch on up pad.
kUiTouchLt Single touch on left pad.
kUiTouchDnRt Dual touch on down and right pads.
kUiTouchDnUp Dual touch on down and up pads.
kUiTouchDnLt Dual touch on down and left pads.
kUiTouchUpRt Dual touch on up and right pads.
kUiTouchUpLt Dual touch on up and left pads.
kUiTouchRtLt Dual touch on right and left pads.

18.3 Function Documentation

18.3.1 void UiManager_SetEvent (uint32_t *event*)

Function to set event, can be called from ISR or task.

Will ignore new events until manager processes previous event.

Parameters

in	*event*	Event to trigger in UI manager

Returns

None

18.3.2 void UiManager_Init (void)

Initialization function for creating UI Manager task and event group.

Returns

None

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer’s technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.