

DevLab_ICM20948

1.0.2

Generated on Wed Jun 17 2026 22:32:52 for DevLab_ICM20948 by Doxygen 1.9.8

Wed Jun 17 2026 22:32:52

1 DevLab ICM20948 Arduino Library	1
1.1 Features	1
1.2 Connections / Wiring	2
1.3 I ² C Connection	2
1.3.1 I ² C Notes	3
1.4 SPI Connection	3
1.5 Installation	3
1.5.1 Arduino Library Manager	3
1.5.2 Manual Installation	3
1.6 Library Overview	3
1.6.1 Initialize Sensor	3
1.6.2 Enable Sensors	4
1.7 Reading Accelerometer	4
1.8 Reading Gyroscope	4
1.9 Reading Temperature	4
1.10 Reading Magnetometer	4
1.11 Setting Accelerometer Scale	4
1.11.1 Setting Gyroscope Scale	4
1.11.2 Setting Sample Rate	5
1.11.3 DLPF Configuration	5
1.11.4 Averaging Configuration	5
1.12 Applications	5
1.13 License	5
2 Hierarchical Index	7
2.1 Class Hierarchy	7
3 Class Index	9
3.1 Class List	9
4 File Index	11
4.1 File List	11
5 Class Documentation	13
5.1 BusIO_7Semi< Bus > Class Template Reference	13
5.1.1 Detailed Description	13
5.1.2 Constructor & Destructor Documentation	13
5.1.2.1 BusIO_7Semi()	13
5.1.3 Member Function Documentation	14
5.1.3.1 read() [1/3]	14
5.1.3.2 read() [2/3]	14
5.1.3.3 read() [3/3]	15
5.1.3.4 readBit() [1/2]	15
5.1.3.5 readBit() [2/2]	16

5.1.3.6 readBits() [1/2]	16
5.1.3.7 readBits() [2/2]	17
5.1.3.8 write() [1/3]	17
5.1.3.9 write() [2/3]	18
5.1.3.10 write() [3/3]	18
5.1.3.11 writeBit() [1/2]	19
5.1.3.12 writeBit() [2/2]	20
5.1.3.13 writeBits() [1/2]	21
5.1.3.14 writeBits() [2/2]	22
5.2 configDLPF Struct Reference	23
5.2.1 Detailed Description	23
5.2.2 Member Data Documentation	23
5.2.2.1 div	23
5.2.2.2 dlpf_idx	23
5.2.2.3 nbw_hz	23
5.2.2.4 nombre	23
5.2.2.5 odr_hz	23
5.3 DevLab_ICM20948 Class Reference	24
5.3.1 Detailed Description	25
5.3.2 Constructor & Destructor Documentation	25
5.3.2.1 DevLab_ICM20948()	25
5.3.3 Member Function Documentation	25
5.3.3.1 applyBasicDefaults()	25
5.3.3.2 auxConfigSlave()	26
5.3.3.3 auxMasterEnable()	26
5.3.3.4 auxRead12bit()	27
5.3.3.5 auxReadByte()	28
5.3.3.6 auxReadResponse()	28
5.3.3.7 auxReadSensorData()	29
5.3.3.8 auxWriteByte()	30
5.3.3.9 auxWriteCommand()	30
5.3.3.10 beginI2C()	31
5.3.3.11 beginSPI()	32
5.3.3.12 checkIntStatus()	32
5.3.3.13 getAccelAveraging()	33
5.3.3.14 getAccelDLPF()	33
5.3.3.15 getAccelOffset()	34
5.3.3.16 getAccelSampleRate()	35
5.3.3.17 getAccelScale()	35
5.3.3.18 getClock()	35
5.3.3.19 getDLPF()	36
5.3.3.20 getGyroOffset()	36

5.3.3.21	getGyroSampleRate()	37
5.3.3.22	getGyroScale()	37
5.3.3.23	getLowPower()	38
5.3.3.24	getSensors()	38
5.3.3.25	initMag()	39
5.3.3.26	intEnableConfig()	40
5.3.3.27	intInit()	40
5.3.3.28	readAccel()	41
5.3.3.29	readGyro()	42
5.3.3.30	readMag()	43
5.3.3.31	readTemperature()	43
5.3.3.32	readWhoAmI()	44
5.3.3.33	selectBank()	45
5.3.3.34	selfTestAccel()	45
5.3.3.35	selfTestGyro()	46
5.3.3.36	setAccelAveraging()	46
5.3.3.37	setAccelDivRate()	47
5.3.3.38	setAccelDLPF()	48
5.3.3.39	setAccelOffset()	49
5.3.3.40	setAccelSampleRate()	50
5.3.3.41	setAccelScale()	50
5.3.3.42	setClock()	51
5.3.3.43	setDLPF()	52
5.3.3.44	setGyroAveraging()	52
5.3.3.45	setGyroDivRate()	53
5.3.3.46	setGyroOffset()	53
5.3.3.47	setGyroSampleRate()	54
5.3.3.48	setGyroScale()	54
5.3.3.49	setLowPower()	55
5.3.3.50	setMagOpMode()	55
5.3.3.51	setSensors()	56
5.3.3.52	sleep()	57
5.3.3.53	softReset()	58
5.4	I2C_Interface Class Reference	58
5.4.1	Detailed Description	59
5.4.2	Member Function Documentation	60
5.4.2.1	beginI2C()	60
5.4.2.2	beginSPI()	60
5.4.2.3	read()	60
5.4.2.4	write()	61
5.4.3	Member Data Documentation	61
5.4.3.1	address	61

5.4.3.2 i2c	61
5.5 ICM20948_IntEnableConfig Struct Reference	61
5.5.1 Detailed Description	62
5.5.2 Member Data Documentation	62
5.5.2.1 dmpInt1En	62
5.5.2.2 fifoOvfEn	62
5.5.2.3 fifoWmEn	62
5.5.2.4 i2cMstIntEn	62
5.5.2.5 pllRdyEn	62
5.5.2.6 rawDataRdyEn	62
5.5.2.7 wofEn	62
5.5.2.8 womIntEn	62
5.6 ICM20948_IntPinConfig Struct Reference	62
5.6.1 Detailed Description	63
5.6.2 Member Data Documentation	63
5.6.2.1 activeLevel	63
5.6.2.2 bypassEn	63
5.6.2.3 clearMode	63
5.6.2.4 driveMode	63
5.6.2.5 fsyncActLevel	63
5.6.2.6 fsyncIntEn	63
5.6.2.7 latchMode	63
5.7 ICM20948_IntStatus Struct Reference	64
5.7.1 Detailed Description	64
5.7.2 Member Data Documentation	64
5.7.2.1 dmpInt1	64
5.7.2.2 fifoOvf	64
5.7.2.3 fifoWm	64
5.7.2.4 i2cMstInt	64
5.7.2.5 pllRdyInt	64
5.7.2.6 rawDataRdy	64
5.7.2.7 womInt	65
5.8 icm_plat_t Struct Reference	65
5.8.1 Detailed Description	65
5.8.2 Member Data Documentation	65
5.8.2.1 delay_ms	65
5.8.2.2 i2c_addr	65
5.8.2.3 read	65
5.8.2.4 spi_cs	66
5.8.2.5 spi_reg_rw_bits	66
5.8.2.6 user	66
5.8.2.7 write	66

5.9 Interface_7Semi Class Reference	66
5.9.1 Detailed Description	67
5.9.2 Constructor & Destructor Documentation	67
5.9.2.1 ~Interface_7Semi()	67
5.9.3 Member Function Documentation	67
5.9.3.1 beginI2C()	67
5.9.3.2 beginSPI()	67
5.9.3.3 delay_us()	68
5.9.3.4 read()	68
5.9.3.5 write()	68
5.10 SPI_Interface Class Reference	68
5.10.1 Detailed Description	70
5.10.2 Member Function Documentation	70
5.10.2.1 beginI2C()	70
5.10.2.2 beginSPI()	70
5.10.2.3 read()	71
5.10.2.4 write()	71
5.10.3 Member Data Documentation	71
5.10.3.1 cs	71
5.10.3.2 speed	71
5.10.3.3 spi	71
6 File Documentation	73
6.1 examples/gyr_test/gyr_test.ino File Reference	73
6.1.1 Detailed Description	74
6.1.2 Macro Definition Documentation	74
6.1.2.1 I2C_FREQ	74
6.1.2.2 ICM_ADDR	75
6.1.2.3 SCL_PIN	75
6.1.2.4 SDA_PIN	75
6.1.3 Function Documentation	75
6.1.3.1 applySettings()	75
6.1.3.2 firstStage()	75
6.1.3.3 loop()	76
6.1.3.4 printDobleLinea()	76
6.1.3.5 printLinea()	76
6.1.3.6 runSweep()	76
6.1.3.7 setup()	77
6.1.4 Variable Documentation	77
6.1.4.1 configsDLPF	77
6.1.4.2 etiquetasGyroAVGCfg	78
6.1.4.3 etiquetasGyroFSCfg	78
6.1.4.4 gyroAVGCfg	78

6.1.4.5 gyroDLPF	78
6.1.4.6 gyroFSCfg	78
6.1.4.7 imu	79
6.1.4.8 N_AVG	79
6.1.4.9 N_DLPF	79
6.1.4.10 N_FS	79
6.2 gyr_test.ino	79
6.3 examples/i2c_accel/i2c_accel.ino File Reference	82
6.3.1 Macro Definition Documentation	82
6.3.1.1 I2C_FREQ	82
6.3.1.2 ICM_ADDR	82
6.3.1.3 SCL_PIN	83
6.3.1.4 SDA_PIN	83
6.3.2 Function Documentation	83
6.3.2.1 loop()	83
6.3.2.2 setup()	83
6.3.3 Variable Documentation	84
6.3.3.1 imu	84
6.4 i2c_accel.ino	84
6.5 examples/i2c_basic/i2c_basic.ino File Reference	85
6.5.1 Macro Definition Documentation	86
6.5.1.1 I2C_FREQ	86
6.5.1.2 ICM_ADDR	86
6.5.1.3 SCL_PIN	86
6.5.1.4 SDA_PIN	86
6.5.2 Function Documentation	86
6.5.2.1 loop()	86
6.5.2.2 setup()	87
6.5.3 Variable Documentation	87
6.5.3.1 imu	87
6.6 i2c_basic.ino	87
6.7 examples/i2c_gyro/i2c_gyro.ino File Reference	89
6.7.1 Macro Definition Documentation	90
6.7.1.1 I2C_FREQ	90
6.7.1.2 ICM_ADDR	90
6.7.1.3 SCL_PIN	90
6.7.1.4 SDA_PIN	90
6.7.2 Function Documentation	90
6.7.2.1 loop()	90
6.7.2.2 setup()	90
6.7.3 Variable Documentation	91
6.7.3.1 imu	91

6.8 i2c_gyro.ino	91
6.9 examples/i2c_magneto/i2c_magneto.ino File Reference	93
6.9.1 Macro Definition Documentation	93
6.9.1.1 I2C_FREQ	93
6.9.1.2 ICM_ADDR	93
6.9.1.3 SCL_PIN	94
6.9.1.4 SDA_PIN	94
6.9.2 Function Documentation	94
6.9.2.1 loop()	94
6.9.2.2 setup()	94
6.9.3 Variable Documentation	95
6.9.3.1 imu	95
6.10 i2c_magneto.ino	95
6.11 examples/interrupt/interrupt.ino File Reference	96
6.11.1 Detailed Description	97
6.11.2 Macro Definition Documentation	97
6.11.2.1 I2C_FREQ	97
6.11.2.2 ICM_ADDR	97
6.11.2.3 PIN_INT	98
6.11.2.4 SCL_PIN	98
6.11.2.5 SDA_PIN	98
6.11.3 Function Documentation	98
6.11.3.1 isrHandler()	98
6.11.3.2 loop()	98
6.11.3.3 printDobleLinea()	98
6.11.3.4 printLinea()	98
6.11.3.5 setup()	98
6.11.4 Variable Documentation	99
6.11.4.1 imu	99
6.11.4.2 intEn	99
6.11.4.3 isrCount	99
6.11.4.4 pinCfg	99
6.12 interrupt.ino	99
6.13 examples/mag_test/mag_test.ino File Reference	101
6.13.1 Detailed Description	102
6.13.2 Macro Definition Documentation	102
6.13.2.1 I2C_FREQ	102
6.13.2.2 ICM_ADDR	102
6.13.2.3 SCL_PIN	102
6.13.2.4 SDA_PIN	102
6.13.3 Function Documentation	102
6.13.3.1 applySettings()	102

6.13.3.2 loop()	103
6.13.3.3 printDobleLinea()	103
6.13.3.4 printLinea()	103
6.13.3.5 setup()	103
6.13.4 Variable Documentation	104
6.13.4.1 etiquetasOpModeCfg	104
6.13.4.2 imu	104
6.13.4.3 N_OPM	104
6.13.4.4 opMode	104
6.14 mag_test.ino	104
6.15 examples/spi_accel/spi_accel.ino File Reference	106
6.15.1 Macro Definition Documentation	107
6.15.1.1 CS_PIN	107
6.15.1.2 SPI_FAST_SPEED	107
6.15.2 Function Documentation	107
6.15.2.1 loop()	107
6.15.2.2 setup()	107
6.15.2.3 spi_bus()	108
6.15.3 Variable Documentation	108
6.15.3.1 imu	108
6.16 spi_accel.ino	108
6.17 examples/spi_basic/spi_basic.ino File Reference	110
6.17.1 Macro Definition Documentation	111
6.17.1.1 CS_PIN	111
6.17.1.2 SPI_FAST_SPEED	111
6.17.2 Function Documentation	111
6.17.2.1 loop()	111
6.17.2.2 setup()	111
6.17.2.3 spi_bus()	112
6.17.3 Variable Documentation	112
6.17.3.1 imu	112
6.18 spi_basic.ino	112
6.19 examples/spi_gyro/spi_gyro.ino File Reference	114
6.19.1 Macro Definition Documentation	114
6.19.1.1 CS_PIN	114
6.19.1.2 SPI_FAST_SPEED	114
6.19.2 Function Documentation	114
6.19.2.1 loop()	114
6.19.2.2 setup()	115
6.19.2.3 spi_bus()	115
6.19.3 Variable Documentation	116
6.19.3.1 imu	116

6.20 spi_gyro.ino	116
6.21 examples/test/test.ino File Reference	117
6.21.1 Detailed Description	119
6.21.2 Macro Definition Documentation	120
6.21.2.1 I2C_FREQ	120
6.21.2.2 ICM_ADDR	120
6.21.2.3 SCL_PIN	120
6.21.2.4 SDA_PIN	120
6.21.3 Function Documentation	120
6.21.3.1 applySettings()	120
6.21.3.2 firstStage()	120
6.21.3.3 loop()	121
6.21.3.4 printDobleLinea()	121
6.21.3.5 printLinea()	121
6.21.3.6 runSweep()	122
6.21.3.7 setup()	122
6.21.4 Variable Documentation	123
6.21.4.1 accelAVG	123
6.21.4.2 accelCfg	123
6.21.4.3 accelDLPF	123
6.21.4.4 accelSR	123
6.21.4.5 configsDLPF	123
6.21.4.6 etiquetasAccelAVG	124
6.21.4.7 etiquetasAccelCfg	124
6.21.4.8 etiquetasAccelDLPFCFG	124
6.21.4.9 etiquetasSR	124
6.21.4.10 imu	124
6.21.4.11 N_DLPF	124
6.21.4.12 N_FS	124
6.21.4.13 total_fail	124
6.21.4.14 total_pass	125
6.21.4.15 total_warn	125
6.22 test.ino	125
6.23 examples/test2/test2.ino File Reference	128
6.23.1 Detailed Description	129
6.23.2 Macro Definition Documentation	129
6.23.2.1 I2C_FREQ	129
6.23.2.2 ICM_ADDR	130
6.23.2.3 SCL_PIN	130
6.23.2.4 SDA_PIN	130
6.23.3 Function Documentation	130
6.23.3.1 applySettings()	130

6.23.3.2 firstStage()	130
6.23.3.3 loop()	131
6.23.3.4 printDobleLinea()	131
6.23.3.5 printLinea()	131
6.23.3.6 runSweep()	131
6.23.3.7 setup()	132
6.23.4 Variable Documentation	132
6.23.4.1 accelAVG	132
6.23.4.2 accelCfg	133
6.23.4.3 accelDLPF	133
6.23.4.4 accelSR	133
6.23.4.5 configsDLPF	133
6.23.4.6 etiquetasAccelAVG	133
6.23.4.7 etiquetasAccelCfg	134
6.23.4.8 etiquetasSR	134
6.23.4.9 imu	134
6.23.4.10 N_AVG	134
6.23.4.11 N_DLPF	134
6.23.4.12 N_FS	134
6.23.4.13 total_fail	134
6.23.4.14 total_pass	134
6.23.4.15 total_warn	134
6.24 test2.ino	135
6.25 README.md File Reference	138
6.26 src/7Semi_I2C_Interface.h File Reference	138
6.27 7Semi_I2C_Interface.h	139
6.28 src/7Semi_Interface.h File Reference	140
6.29 7Semi_Interface.h	141
6.30 src/7Semi_SPI_Interface.h File Reference	142
6.31 7Semi_SPI_Interface.h	143
6.32 src/BusIO_7Semi.h File Reference	145
6.33 BusIO_7Semi.h	145
6.34 src/DevLab_ICM20948.cpp File Reference	147
6.35 DevLab_ICM20948.cpp	147
6.36 src/DevLab_ICM20948.h File Reference	164
6.36.1 Enumeration Type Documentation	165
6.36.1.1 ICM20948_Accel_Average	165
6.36.1.2 ICM20948_Accel_DLPCFG	165
6.36.1.3 ICM20948_Accel_FullScale	166
6.36.1.4 ICM20948_Clock_Source	166
6.36.1.5 ICM20948_Gyro_Average	166
6.36.1.6 ICM20948_Gyro_DLPF	166

6.36.1.7 ICM20948_Gyro_FullScale	167
6.36.1.8 ICM20948_Op_Mode	167
6.37 DevLab_ICM20948.h	167
6.38 src/ICM20948_platform.h File Reference	178
6.39 ICM20948_platform.h	178
6.40 src/ICM20948_regs.h File Reference	179
6.40.1 Macro Definition Documentation	184
6.40.1.1 ACCEL_BYPASS_3DB_BW_HZ	184
6.40.1.2 ACCEL_BYPASS_NBW_HZ	184
6.40.1.3 ACCEL_BYPASS_RATE_HZ	185
6.40.1.4 ACCEL_CONFIG	185
6.40.1.5 ACCEL_CONFIG_2	185
6.40.1.6 ACCEL_DEC3_AVG_16	185
6.40.1.7 ACCEL_DEC3_AVG_32	185
6.40.1.8 ACCEL_DEC3_AVG_4	185
6.40.1.9 ACCEL_DEC3_AVG_8	185
6.40.1.10 ACCEL_DLPF0_3DB_BW_HZ	185
6.40.1.11 ACCEL_DLPF0_NBW_HZ	185
6.40.1.12 ACCEL_DLPF1_3DB_BW_HZ	185
6.40.1.13 ACCEL_DLPF1_NBW_HZ	185
6.40.1.14 ACCEL_DLPF2_3DB_BW_HZ	185
6.40.1.15 ACCEL_DLPF2_NBW_HZ	186
6.40.1.16 ACCEL_DLPF3_3DB_BW_HZ	186
6.40.1.17 ACCEL_DLPF3_NBW_HZ	186
6.40.1.18 ACCEL_DLPF4_3DB_BW_HZ	186
6.40.1.19 ACCEL_DLPF4_NBW_HZ	186
6.40.1.20 ACCEL_DLPF5_3DB_BW_HZ	186
6.40.1.21 ACCEL_DLPF5_NBW_HZ	186
6.40.1.22 ACCEL_DLPF6_3DB_BW_HZ	186
6.40.1.23 ACCEL_DLPF6_NBW_HZ	186
6.40.1.24 ACCEL_DLPF7_3DB_BW_HZ	186
6.40.1.25 ACCEL_DLPF7_NBW_HZ	186
6.40.1.26 ACCEL_DLPF_BASE_HZ	186
6.40.1.27 ACCEL_DLPF_ODR_HZ	187
6.40.1.28 ACCEL_DLPFCFG_0	187
6.40.1.29 ACCEL_DLPFCFG_1	187
6.40.1.30 ACCEL_DLPFCFG_2	187
6.40.1.31 ACCEL_DLPFCFG_3	187
6.40.1.32 ACCEL_DLPFCFG_4	187
6.40.1.33 ACCEL_DLPFCFG_5	187
6.40.1.34 ACCEL_DLPFCFG_6	187
6.40.1.35 ACCEL_DLPFCFG_7	187

6.40.1.36 ACCEL_DLPFCFG_MASK	187
6.40.1.37 ACCEL_DLPFCFG_SHIFT	187
6.40.1.38 ACCEL_FCHOICE	188
6.40.1.39 ACCEL_FCHOICE_BYPASS	188
6.40.1.40 ACCEL_FCHOICE_DLPF	188
6.40.1.41 ACCEL_FS_16G	188
6.40.1.42 ACCEL_FS_2G	188
6.40.1.43 ACCEL_FS_4G	188
6.40.1.44 ACCEL_FS_8G	188
6.40.1.45 ACCEL_FS_SEL_MASK	188
6.40.1.46 ACCEL_FS_SEL_SHIFT	188
6.40.1.47 ACCEL_INTEL_CTRL	188
6.40.1.48 ACCEL_INTEL_EN	188
6.40.1.49 ACCEL_INTEL_MODE_INT	189
6.40.1.50 ACCEL_SMPLRT_DIV_1	189
6.40.1.51 ACCEL_SMPLRT_DIV_2	189
6.40.1.52 ACCEL_SMPLRT_DIV_MAX	189
6.40.1.53 ACCEL_SMPLRT_DIV_MIN	189
6.40.1.54 ACCEL_WOM_THR	189
6.40.1.55 ACCEL_XOUT_H	189
6.40.1.56 ACCEL_XOUT_L	189
6.40.1.57 ACCEL_YOUT_H	189
6.40.1.58 ACCEL_YOUT_L	189
6.40.1.59 ACCEL_ZOUT_H	189
6.40.1.60 ACCEL_ZOUT_L	189
6.40.1.61 AK09916_I2C_ADDR	190
6.40.1.62 AK_CNTL2	190
6.40.1.63 AK_CNTL3	190
6.40.1.64 AK_HXL	190
6.40.1.65 AK_ST1	190
6.40.1.66 AK_ST2	190
6.40.1.67 AK_WIA2	190
6.40.1.68 AK_WIA2_VAL	190
6.40.1.69 AUTO_SEL	190
6.40.1.70 AX_ST_EN_REG	190
6.40.1.71 AY_ST_EN_REG	190
6.40.1.72 AZ_ST_EN_REG	190
6.40.1.73 BANK1_REG_BANK_SEL	191
6.40.1.74 BANK2_REG_BANK_SEL	191
6.40.1.75 BANK3_REG_BANK_SEL	191
6.40.1.76 BYPASS_EN	191
6.40.1.77 CLK_STOP	191

6.40.1.78 DEC3_CFG_MASK	191
6.40.1.79 DEC3_CFG_SHIFT	191
6.40.1.80 DELAY_ES_SHADOW	191
6.40.1.81 DELAY_TIME_EN	191
6.40.1.82 DELAY_TIMEH	191
6.40.1.83 DELAY_TIMEL	191
6.40.1.84 dps1000	191
6.40.1.85 dps2000	192
6.40.1.86 dps250	192
6.40.1.87 dps500	192
6.40.1.88 EnhancedRegular	192
6.40.1.89 EXT_SLV_SENS_DATA_00	192
6.40.1.90 EXT_SLV_SENS_DATA_23	192
6.40.1.91 EXT_SYNC_SET_MASK	192
6.40.1.92 FIFO_CFG	192
6.40.1.93 FIFO_COUNTH	192
6.40.1.94 FIFO_COUNTL	192
6.40.1.95 FIFO_EN_1	192
6.40.1.96 FIFO_EN_2	192
6.40.1.97 FIFO_MODE	193
6.40.1.98 FIFO_R_W	193
6.40.1.99 FIFO_RST	193
6.40.1.100 FSYNC_CONFIG	193
6.40.1.101 FSYNC_INT_MODE_EN	193
6.40.1.102 g16	193
6.40.1.103 g2	193
6.40.1.104 g4	193
6.40.1.105 g8	193
6.40.1.106 GYRO_AVGCFG_MASK	193
6.40.1.107 GYRO_AVGCFG_SHIFT	193
6.40.1.108 GYRO_BW_MEDIUM	193
6.40.1.109 GYRO_BW_NARROW	194
6.40.1.110 GYRO_BW_ULTRA_WIDE	194
6.40.1.111 GYRO_BW_WIDE	194
6.40.1.112 GYRO_CONFIG_1	194
6.40.1.113 GYRO_CONFIG_2	194
6.40.1.114 GYRO_DLPF_BASE_HZ	194
6.40.1.115 GYRO_DLPF_ODR_HZ	194
6.40.1.116 GYRO_DLPFCFG_0	194
6.40.1.117 GYRO_DLPFCFG_1	194
6.40.1.118 GYRO_DLPFCFG_2	194
6.40.1.119 GYRO_DLPFCFG_3	194

6.40.1.120 GYRO_DLPFCFG_4	195
6.40.1.121 GYRO_DLPFCFG_5	195
6.40.1.122 GYRO_DLPFCFG_6	195
6.40.1.123 GYRO_DLPFCFG_7	195
6.40.1.124 GYRO_DLPFCFG_MASK	195
6.40.1.125 GYRO_DLPFCFG_SHIFT	195
6.40.1.126 GYRO_FCHOICE	195
6.40.1.127 GYRO_FCHOICE_BYPASS	195
6.40.1.128 GYRO_FCHOICE_DLPF	195
6.40.1.129 GYRO_FS_1000DPS	195
6.40.1.130 GYRO_FS_2000DPS	195
6.40.1.131 GYRO_FS_250DPS	196
6.40.1.132 GYRO_FS_500DPS	196
6.40.1.133 GYRO_FS_SEL_MASK	196
6.40.1.134 GYRO_FS_SEL_SHIFT	196
6.40.1.135 GYRO_SMPLRT_DIV	196
6.40.1.136 GYRO_SMPLRT_MIN_HZ	196
6.40.1.137 GYRO_XOUT_H	196
6.40.1.138 GYRO_XOUT_L	196
6.40.1.139 GYRO_YOUT_H	196
6.40.1.140 GYRO_YOUT_L	196
6.40.1.141 GYRO_ZOUT_H	196
6.40.1.142 GYRO_ZOUT_L	196
6.40.1.143 HighAccuracy	197
6.40.1.144 Hz10	197
6.40.1.145 Hz15	197
6.40.1.146 Hz2	197
6.40.1.147 Hz20	197
6.40.1.148 Hz25	197
6.40.1.149 Hz30	197
6.40.1.150 Hz6	197
6.40.1.151 Hz8	197
6.40.1.152 I2C_MST_CLK_MASK	197
6.40.1.153 I2C_MST_CTRL	197
6.40.1.154 I2C_MST_DELAY_CTRL	197
6.40.1.155 I2C_MST_ODR_CONFIG	198
6.40.1.156 I2C_MST_P_NSR	198
6.40.1.157 I2C_MST_STATUS	198
6.40.1.158 I2C_SLV0_ADDR	198
6.40.1.159 I2C_SLV0_CTRL	198
6.40.1.160 I2C_SLV0_DELAY_EN	198
6.40.1.161 I2C_SLV0_DO	198

6.40.1.162 I2C_SLV0_REG	198
6.40.1.163 I2C_SLV1_ADDR	198
6.40.1.164 I2C_SLV1_CTRL	198
6.40.1.165 I2C_SLV1_DELAY_EN	198
6.40.1.166 I2C_SLV1_DO	198
6.40.1.167 I2C_SLV1_REG	199
6.40.1.168 I2C_SLV2_ADDR	199
6.40.1.169 I2C_SLV2_CTRL	199
6.40.1.170 I2C_SLV2_DELAY_EN	199
6.40.1.171 I2C_SLV2_DO	199
6.40.1.172 I2C_SLV2_REG	199
6.40.1.173 I2C_SLV3_ADDR	199
6.40.1.174 I2C_SLV3_CTRL	199
6.40.1.175 I2C_SLV3_DELAY_EN	199
6.40.1.176 I2C_SLV3_DO	199
6.40.1.177 I2C_SLV3_REG	199
6.40.1.178 I2C_SLV4_ADDR	199
6.40.1.179 I2C_SLV4_CTRL	200
6.40.1.180 I2C_SLV4_DELAY_EN	200
6.40.1.181 I2C_SLV4_DI	200
6.40.1.182 I2C_SLV4_DLY_MASK	200
6.40.1.183 I2C_SLV4_DO	200
6.40.1.184 I2C_SLV4_REG	200
6.40.1.185 I2C_SLVx_BYTE_SW	200
6.40.1.186 I2C_SLVx_EN	200
6.40.1.187 I2C_SLVx_GRP	200
6.40.1.188 I2C_SLVx LENG_MASK	200
6.40.1.189 I2C_SLVx_REG_DIS	200
6.40.1.190 I2C_SLVx_RNW	200
6.40.1.191 INT1_ACTL	201
6.40.1.192 INT1_LATCH_INT_EN	201
6.40.1.193 INT1_OPEN	201
6.40.1.194 INT_ACTL_FSYNC	201
6.40.1.195 INT_ANYRD_2CLEAR	201
6.40.1.196 INT_DMP_INT1_EN	201
6.40.1.197 INT_ENABLE	201
6.40.1.198 INT_ENABLE_1	201
6.40.1.199 INT_ENABLE_2	201
6.40.1.200 INT_ENABLE_3	201
6.40.1.201 INT_I2C_MST_INT_EN	201
6.40.1.202 INT_PIN_CFG	201
6.40.1.203 INT_PLL_RDY_EN	202

6.40.1.204 INT_RAW_DATA_0_RDY_EN	202
6.40.1.205 INT_REG_WOF_EN	202
6.40.1.206 INT_STATUS	202
6.40.1.207 INT_STATUS_1	202
6.40.1.208 INT_STATUS_2	202
6.40.1.209 INT_STATUS_3	202
6.40.1.210 INT_WOM_INT_EN	202
6.40.1.211 INTERNAL_20MHZ	202
6.40.1.212 LowPower	202
6.40.1.213 LP_ACCEL_CYCLE	202
6.40.1.214 LP_CONFIG	202
6.40.1.215 LP_GYRO_CYCLE	203
6.40.1.216 LP_I2C_MST_CYCLE	203
6.40.1.217 MOD_CTRL_USR	203
6.40.1.218 MST_LOST_ARB	203
6.40.1.219 MST_ODR_CFG_MASK	203
6.40.1.220 MST_PASS_THROUGH	203
6.40.1.221 MST_SLV0_NACK	203
6.40.1.222 MST_SLV1_NACK	203
6.40.1.223 MST_SLV2_NACK	203
6.40.1.224 MST_SLV3_NACK	203
6.40.1.225 MST_SLV4_DONE	203
6.40.1.226 MST_SLV4_NACK	203
6.40.1.227 MULT_MST_EN	204
6.40.1.228 ODR_ALIGN_EN	204
6.40.1.229 ODR_ALIGN_EN_BIT	204
6.40.1.230 PWR_CLKSEL_AUTO	204
6.40.1.231 PWR_CLKSEL_INT_20MHZ	204
6.40.1.232 PWR_CLKSEL_MASK	204
6.40.1.233 PWR_DEVICE_RESET	204
6.40.1.234 PWR_DISABLE_ACCEL	204
6.40.1.235 PWR_DISABLE_GYRO	204
6.40.1.236 PWR_LP_EN	204
6.40.1.237 PWR_MGMT_1	204
6.40.1.238 PWR_MGMT_2	204
6.40.1.239 PWR_SLEEP	205
6.40.1.240 PWR_TEMP_DIS	205
6.40.1.241 REG_BANK_SEL	205
6.40.1.242 REG_BANK_SEL_USER_BANK_MASK	205
6.40.1.243 REG_BANK_SEL_USER_BANK_SHIFT	205
6.40.1.244 REG_LP_DMP_EN	205
6.40.1.245 Regular	205

6.40.1.246 SELF_TEST_X_ACCEL	205
6.40.1.247 SELF_TEST_X_GYRO	205
6.40.1.248 SELF_TEST_Y_ACCEL	205
6.40.1.249 SELF_TEST_Y_GYRO	205
6.40.1.250 SELF_TEST_Z_ACCEL	205
6.40.1.251 SELF_TEST_Z_GYRO	206
6.40.1.252 STS_DMP_INT1	206
6.40.1.253 STS_I2C_MST_INT	206
6.40.1.254 STS_PLL_RDY_INT	206
6.40.1.255 STS_RAW_DATA_0_RDY_INT	206
6.40.1.256 STS_WOM_INT	206
6.40.1.257 TEMP_CONFIG	206
6.40.1.258 TEMP_DLPFCFG_MASK	206
6.40.1.259 TEMP_DLPFCFG_SHIFT	206
6.40.1.260 TEMP_OUT_H	206
6.40.1.261 TEMP_OUT_L	206
6.40.1.262 TIMEBASE_CORRECTION_PLL	206
6.40.1.263 USER_BANK_0	207
6.40.1.264 USER_BANK_1	207
6.40.1.265 USER_BANK_2	207
6.40.1.266 USER_BANK_3	207
6.40.1.267 USER_CTRL	207
6.40.1.268 USER_CTRL_DMP_EN	207
6.40.1.269 USER_CTRL_DMP_RST	207
6.40.1.270 USER_CTRL_FIFO_EN	207
6.40.1.271 USER_CTRL_I2C_IF_DIS	207
6.40.1.272 USER_CTRL_I2C_MST_EN	207
6.40.1.273 USER_CTRL_I2C_MST_RST	207
6.40.1.274 USER_CTRL_SRAM_RST	207
6.40.1.275 WHO_AM_I	208
6.40.1.276 WHO_AM_I_VAL	208
6.40.1.277 WOF_DEGLITCH_EN	208
6.40.1.278 WOF_EDGE_INT	208
6.40.1.279 XA_OFFS_H	208
6.40.1.280 XA_OFFS_L	208
6.40.1.281 XG_OFFS_USRH	208
6.40.1.282 XG_OFFS_USRL	208
6.40.1.283 XGYRO_CTEN	208
6.40.1.284 YA_OFFS_H	209
6.40.1.285 YA_OFFS_L	209
6.40.1.286 YG_OFFS_USRH	209
6.40.1.287 YG_OFFS_USRL	209

6.40.1.288 YGYRO_CTEN	209
6.40.1.289 ZA_OFFS_H	209
6.40.1.290 ZA_OFFS_L	209
6.40.1.291 ZG_OFFS_USRH	209
6.40.1.292 ZG_OFFS_USRL	209
6.40.1.293 ZGYRO_CTEN	209
6.41 ICM20948_regs.h	209

Index	217
--------------	------------

Chapter 1

DevLab ICM20948 Arduino Library

Arduino driver for the module DevLab ICM-20948 9-axis IMU sensor.

This library is a DevLab adaptation derived from the 7Semi ICM20948 Arduino Library. The original MIT license notice from 7semi-solutions is preserved in `LICENSE`. See `NOTICE.md` for attribution and origin details.

The ICM-20948 integrates a 3-axis accelerometer, 3-axis gyroscope, and 3-axis magnetometer (AK09916), enabling motion tracking, orientation sensing, and navigation applications.

1.1 Features

- 9-axis motion sensing
 - 3-axis Accelerometer
 - 3-axis Gyroscope
 - 3-axis Magnetometer
- Configurable sensor ranges
 - Accel: $\pm 2g$ / $\pm 4g$ / $\pm 8g$ / $\pm 16g$
 - Gyro: ± 250 / ± 500 / ± 1000 / ± 2000 dps
- Digital Low Pass Filter configurable(DLPF)
 - Noise reduction
 - Configurable bandwidth
- Adjustable sample rates
 - Accel: up to 1125 Hz
 - Gyro: up to 1100 Hz
- Internal I2C Master (for magnetometer)
 - AK09916 integration via SLV0/SLV4
- ISupports both communication interfaces
 - I²C
 - SPI
- Temperature measurement

1.2 Connections / Wiring

The ICM-20948 supports both **I²C** and **SPI** communication.

1.3 I²C Connection

ICM-20948 Pin	MCU Pin	Notes
VCC	3.3V	5V only if board supports
GND	GND	Common ground
SDA	SDA	I ² C data
SCL	SCL	I ² C clock
AD0	GND/3V3	Address select

1.3.1 I²C Notes

- Default I²C address: 0x29
- Supported bus speeds:
 - 100 kHz
 - 400 kHz (recommended)

1.4 SPI Connection

ICM-20948 Pin	MCU Pin
VCC	3.3V
GND	GND
SCK	SCK
MOSI	MOSI
MISO	MISO
CS	GPIO

1.5 Installation

1.5.1 Arduino Library Manager

1. Open Arduino IDE
2. Go to Library Manager
3. Search for **DevLab ICM20948**
4. Click Install

1.5.2 Manual Installation

1. Download repository as ZIP
2. Arduino IDE → Sketch → Include Library → Add .ZIP Library

1.6 Library Overview

1.6.1 Initialize Sensor

- I2C

```
if (!imu.beginI2C(0x69, Wire, 400000))  
{  
  Serial.println("IMU init failed");  
  while (1);  
}
```

- SPI

```
if (!imu.beginSPI(CS_PIN, SPI, 1000000))  
{  
  Serial.println("SPI init failed");  
  while (1);  
}
```

1.6.2 Enable Sensors

```
imu.setSensors(true, true, true);
```

- accel → enable accelerometer
- gyro → enable gyroscope
- temp → enable temperatur

1.7 Reading Accelerometer

```
float ax, ay, az;  
imu.readAccel(ax, ay, az);
```

- Output in g

1.8 Reading Gyroscope

```
float gx, gy, gz;  
imu.readGyro(gx, gy, gz);
```

- Output in degrees/sec

1.9 Reading Temperature

```
float temp;  
imu.readTemperature(temp);
```

- Output in °C

1.10 Reading Magnetometer

```
imu.initMag();  
  
float mx, my, mz;  
imu.readMag(mx, my, mz);
```

- Output in microtesla (μT)

1.11 Setting Accelerometer Scale

```
imu.setAccelScale(G_4);
```

- G_2, G_4, G_8, G_16

1.11.1 Setting Gyroscope Scale

```
imu.setGyroScale(DPS_2000);
```

1.11.2 Setting Sample Rate

```
imu.setAccelSampleRate(225);  
imu.setGyroSampleRate(225);
```

1.11.3 DLPF Configuration

```
imu.setAccelDLPF(ACCEL_DLPFCFG_3, false);  
imu.setDLPF(GYRO_DLPF_3, false);
```

1.11.4 Averaging Configuration

```
imu.setAccelAveraging(AVG_8);
```

1.12 Applications

- Robotics (obstacle detection)
- Drones (altitude sensing)
- Wearables (activity tracking)
- Navigation systems (IMU fusion)
- Industrial motion sensing
- Gaming controllers

1.13 License

- MIT License
- Original work copyright: 2025 7semi-solutions
- DevLab adaptation maintains the original license notice and attribution.

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

BusIO_7Semi< Bus >	13
BusIO_7Semi< Interface_7Semi >	13
configDLPF	23
DevLab_ICM20948	24
ICM20948_IntEnableConfig	61
ICM20948_IntPinConfig	62
ICM20948_IntStatus	64
icm_plat_t	65
Interface_7Semi	66
I2C_Interface	58
SPI_Interface	68

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

BusIO_7Semi< Bus >	13
configDLPF	
Gyroscope DLPF divider and timing configuration	23
DevLab_ICM20948	24
I2C_Interface	58
ICM20948_IntEnableConfig	61
ICM20948_IntPinConfig	62
ICM20948_IntStatus	64
icm_plat_t	65
Interface_7Semi	66
SPI_Interface	68

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

examples/gyr_test/gyr_test.ino	
Gyroscope DLPF and full-scale sweep for the DevLab ICM-20948	73
examples/i2c_accel/i2c_accel.ino	82
examples/i2c_basic/i2c_basic.ino	85
examples/i2c_gyro/i2c_gyro.ino	89
examples/i2c_magneto/i2c_magneto.ino	93
examples/interrupt/interrupt.ino	
I2C interrupt example for the DevLab ICM-20948	96
examples/mag_test/mag_test.ino	
Magnetometer operation-mode sweep for the DevLab ICM-20948	101
examples/spi_accel/spi_accel.ino	106
examples/spi_basic/spi_basic.ino	110
examples/spi_gyro/spi_gyro.ino	114
examples/test/test.ino	
Accelerometer DLPF validation sketch for the 7Semi ICM-20948	117
examples/test2/test2.ino	
Extended accelerometer DLPF, full-scale, and averaging sweep	128
src/7Semi_I2C_Interface.h	138
src/7Semi_Interface.h	140
src/7Semi_SPI_Interface.h	142
src/BusIO_7Semi.h	145
src/DevLab_ICM20948.cpp	147
src/DevLab_ICM20948.h	164
src/ICM20948_platform.h	178
src/ICM20948_regs.h	179

Chapter 5

Class Documentation

5.1 BusIO_7Semi< Bus > Class Template Reference

```
#include <BusIO_7Semi.h>
```

Public Member Functions

- [BusIO_7Semi](#) (Bus &busRef)
- bool [read](#) (uint8_t reg, uint8_t &value)
- bool [read](#) (uint8_t reg, uint16_t &value)
- bool [read](#) (uint8_t reg, uint8_t *data, uint32_t len)
- bool [write](#) (uint8_t reg, uint8_t value)
- bool [write](#) (uint8_t reg, uint16_t value)
- bool [write](#) (uint8_t reg, const uint8_t *data, uint32_t len)
- bool [readBits](#) (uint8_t reg, uint8_t pos, uint8_t len, uint8_t &value)
- bool [readBits](#) (uint8_t reg, uint8_t pos, uint8_t len, uint16_t &value)
- bool [writeBits](#) (uint8_t reg, uint8_t pos, uint8_t len, uint8_t value)
- bool [writeBits](#) (uint8_t reg, uint8_t pos, uint8_t len, uint16_t value)
- bool [readBit](#) (uint8_t reg, uint8_t pos, uint8_t &value)
- bool [readBit](#) (uint8_t reg, uint8_t pos, uint16_t &value)
- bool [writeBit](#) (uint8_t reg, uint8_t pos, uint8_t value)
- bool [writeBit](#) (uint8_t reg, uint8_t pos, uint16_t value)

5.1.1 Detailed Description

```
template<typename Bus>  
class BusIO_7Semi< Bus >
```

Definition at line 5 of file [BusIO_7Semi.h](#).

5.1.2 Constructor & Destructor Documentation

5.1.2.1 BusIO_7Semi()

```
template<typename Bus >  
BusIO\_7Semi< Bus >::BusIO_7Semi (  
    Bus & busRef ) [inline]
```

Constructor

- Stores interface reference

Definition at line 14 of file [BusIO_7Semi.h](#).

5.1.3 Member Function Documentation

5.1.3.1 `read()` [1/3]

```
template<typename Bus >  
bool BusIO_7Semi< Bus >::read (  
    uint8_t reg,  
    uint16_t & value ) [inline]
```

read 16-bit

Definition at line 23 of file [BusIO_7Semi.h](#).

5.1.3.2 `read()` [2/3]

```
template<typename Bus >  
bool BusIO_7Semi< Bus >::read (  
    uint8_t reg,  
    uint8_t & value ) [inline]
```

read 8-bit

Definition at line 17 of file [BusIO_7Semi.h](#).

The diagram illustrates the state transitions for the ICM20948 sensor driver. The states are represented by rectangles, and the transitions are represented by blue arrows. The diagram is organized into several columns, showing the flow of operations from setup to data reading and back to setup.

States (Nodes):

- DevLab_ICM20948::auxReadByte**
- DevLab_ICM20948::auxRead Response**
- DevLab_ICM20948::auxRead SensorData**
- DevLab_ICM20948::auxWriteByte**
- DevLab_ICM20948::checkInt Status**
- DevLab_ICM20948::getAccelDLPF**
- DevLab_ICM20948::getAccel Offset**
- DevLab_ICM20948::getAccel SampleRate**
- DevLab_ICM20948::getClock**
- DevLab_ICM20948::getDLPF**
- DevLab_ICM20948::getGyro Offset**
- DevLab_ICM20948::getGyro SampleRate**
- DevLab_ICM20948::getAccel Scale**
- DevLab_ICM20948::getGyro Scale**
- BusIO_7Semi::readBit**
- DevLab_ICM20948::getLowPower**
- BusIO_7Semi::readBit**
- DevLab_ICM20948::getAccel Averaging**
- loop**
- runSweep**
- DevLab_ICM20948::readAccel**
- BusIO_7Semi::readBits**
- DevLab_ICM20948::readGyro**
- loop**
- DevLab_ICM20948::readTemperature**
- DevLab_ICM20948::readMag**
- firstStage**
- applySettings**
- DevLab_ICM20948::beginI2C**
- DevLab_ICM20948::readWhoAmI**
- DevLab_ICM20948::beginSPI**
- BusIO_7Semi::writeBit**
- DevLab_ICM20948::auxMaster Enable**
- DevLab_ICM20948::setDLPF**
- DevLab_ICM20948::setAccelDLPF**
- BusIO_7Semi::writeBits**
- DevLab_ICM20948::setAccel Averaging**
- BusIO_7Semi::writeBits**
- DevLab_ICM20948::setAccel Scale**
- DevLab_ICM20948::setGyro Averaging**
- DevLab_ICM20948::setGyro Scale**
- BusIO_7Semi::writeBit**
- BusIO_7Semi::read**

Transitions (Edges):

- setup** transitions to **DevLab_ICM20948::auxReadByte**, **DevLab_ICM20948::auxRead Response**, **DevLab_ICM20948::auxRead SensorData**, **DevLab_ICM20948::auxWriteByte**, **DevLab_ICM20948::checkInt Status**, **DevLab_ICM20948::getAccelDLPF**, **DevLab_ICM20948::getAccel Offset**, **DevLab_ICM20948::getAccel SampleRate**, **DevLab_ICM20948::getClock**, **DevLab_ICM20948::getDLPF**, **DevLab_ICM20948::getGyro Offset**, **DevLab_ICM20948::getGyro SampleRate**, **DevLab_ICM20948::getAccel Scale**, **DevLab_ICM20948::getGyro Scale**, **BusIO_7Semi::readBit**, **DevLab_ICM20948::getLowPower**, **BusIO_7Semi::readBit**, **DevLab_ICM20948::getAccel Averaging**, **loop**, **runSweep**, **DevLab_ICM20948::readAccel**, **BusIO_7Semi::readBits**, **DevLab_ICM20948::readGyro**, **loop**, **DevLab_ICM20948::readTemperature**, **DevLab_ICM20948::readMag**, **firstStage**, **applySettings**, **DevLab_ICM20948::beginI2C**, **DevLab_ICM20948::readWhoAmI**, **DevLab_ICM20948::beginSPI**, **BusIO_7Semi::writeBit**, **DevLab_ICM20948::auxMaster Enable**, **DevLab_ICM20948::setDLPF**, **DevLab_ICM20948::setAccelDLPF**, **BusIO_7Semi::writeBits**, **DevLab_ICM20948::setAccel Averaging**, **BusIO_7Semi::writeBits**, **DevLab_ICM20948::setAccel Scale**, **DevLab_ICM20948::setGyro Averaging**, **DevLab_ICM20948::setGyro Scale**, **BusIO_7Semi::writeBit**, and **BusIO_7Semi::read**.
- runSweep** transitions to **DevLab_ICM20948::readAccel**, **BusIO_7Semi::readBits**, **DevLab_ICM20948::readGyro**, **loop**, **DevLab_ICM20948::readTemperature**, and **DevLab_ICM20948::readMag**.
- firstStage** transitions to **applySettings**.
- applySettings** transitions to **DevLab_ICM20948::beginI2C**, **DevLab_ICM20948::readWhoAmI**, **DevLab_ICM20948::beginSPI**, **BusIO_7Semi::writeBit**, **DevLab_ICM20948::auxMaster Enable**, **DevLab_ICM20948::setDLPF**, **DevLab_ICM20948::setAccelDLPF**, **BusIO_7Semi::writeBits**, **DevLab_ICM20948::setAccel Averaging**, **BusIO_7Semi::writeBits**, **DevLab_ICM20948::setAccel Scale**, **DevLab_ICM20948::setGyro Averaging**, **DevLab_ICM20948::setGyro Scale**, **BusIO_7Semi::writeBit**, and **BusIO_7Semi::read**.
- DevLab_ICM20948::beginI2C** transitions to **DevLab_ICM20948::readWhoAmI**.
- DevLab_ICM20948::readWhoAmI** transitions to **DevLab_ICM20948::beginSPI**.
- DevLab_ICM20948::beginSPI** transitions to **BusIO_7Semi::writeBit**.
- DevLab_ICM20948::auxMaster Enable** transitions to **DevLab_ICM20948::setDLPF**.
- DevLab_ICM20948::setDLPF** transitions to **DevLab_ICM20948::setAccelDLPF**.
- DevLab_ICM20948::setAccelDLPF** transitions to **BusIO_7Semi::writeBits**.
- DevLab_ICM20948::setAccel Averaging** transitions to **BusIO_7Semi::writeBits**.
- DevLab_ICM20948::setAccel Scale** transitions to **DevLab_ICM20948::setGyro Averaging**.
- DevLab_ICM20948::setGyro Averaging** transitions to **DevLab_ICM20948::setGyro Scale**.
- DevLab_ICM20948::setGyro Scale** transitions to **BusIO_7Semi::writeBit**.
- BusIO_7Semi::writeBit** transitions to **BusIO_7Semi::read**.
- BusIO_7Semi::read** transitions to **DevLab_ICM20948::auxReadByte**.

```
template<typename Bus >
bool BusIO_7Semi< Bus >::read (
    uint8_t reg,
    uint8_t * data,
    uint32_t len ) [inline]
```

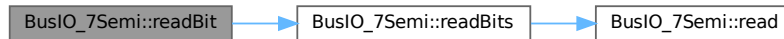
Definition at line 33 of file BusIO_7Semi.h.

```
template<typename Bus >
```

```
bool BusIO_7Semi< Bus >::readBit (
    uint8_t reg,
    uint8_t pos,
    uint16_t & value ) [inline]
```

Definition at line 142 of file [BusIO_7Semi.h](#).

Here is the call graph for this function:

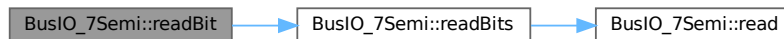


5.1.3.5 readBit() [2/2]

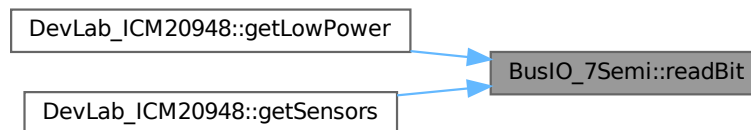
```
template<typename Bus >
bool BusIO_7Semi< Bus >::readBit (
    uint8_t reg,
    uint8_t pos,
    uint8_t & value ) [inline]
```

Definition at line 137 of file [BusIO_7Semi.h](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.1.3.6 readBits() [1/2]

```
template<typename Bus >
bool BusIO_7Semi< Bus >::readBits (
    uint8_t reg,
    uint8_t pos,
    uint8_t len,
    uint16_t & value ) [inline]
```

readBits (16-bit register)

Definition at line 78 of file [BusIO_7Semi.h](#).

Here is the call graph for this function:



5.1.3.7 readBits() [2/2]

```

template<typename Bus >
bool BusIO_7Semi< Bus >::readBits (
    uint8_t reg,
    uint8_t pos,
    uint8_t len,
    uint8_t & value ) [inline]
  
```

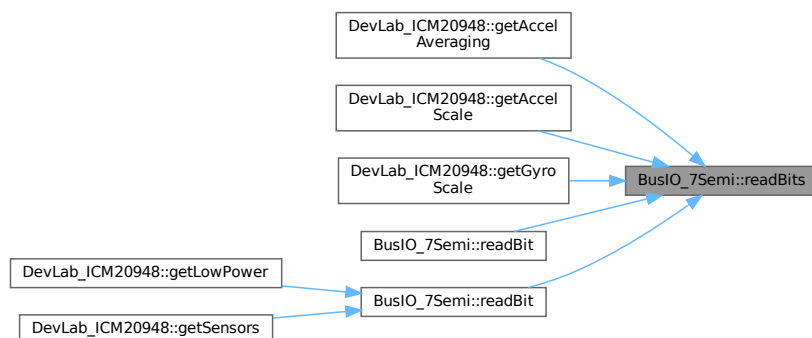
readBits (8-bit register)

Definition at line 60 of file [BusIO_7Semi.h](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.1.3.8 write() [1/3]

```

template<typename Bus >
bool BusIO_7Semi< Bus >::write (
  
```

```
uint8_t reg,  
const uint8_t * data,  
uint32_t len ) [inline]
```

write burst

Definition at line 52 of file [BusIO_7Semi.h](#).

5.1.3.9 write() [2/3]

```
template<typename Bus >  
bool BusIO_7Semi< Bus >::write (  
    uint8_t reg,  
    uint16_t value ) [inline]
```

write 16-bit

Definition at line 45 of file [BusIO_7Semi.h](#).

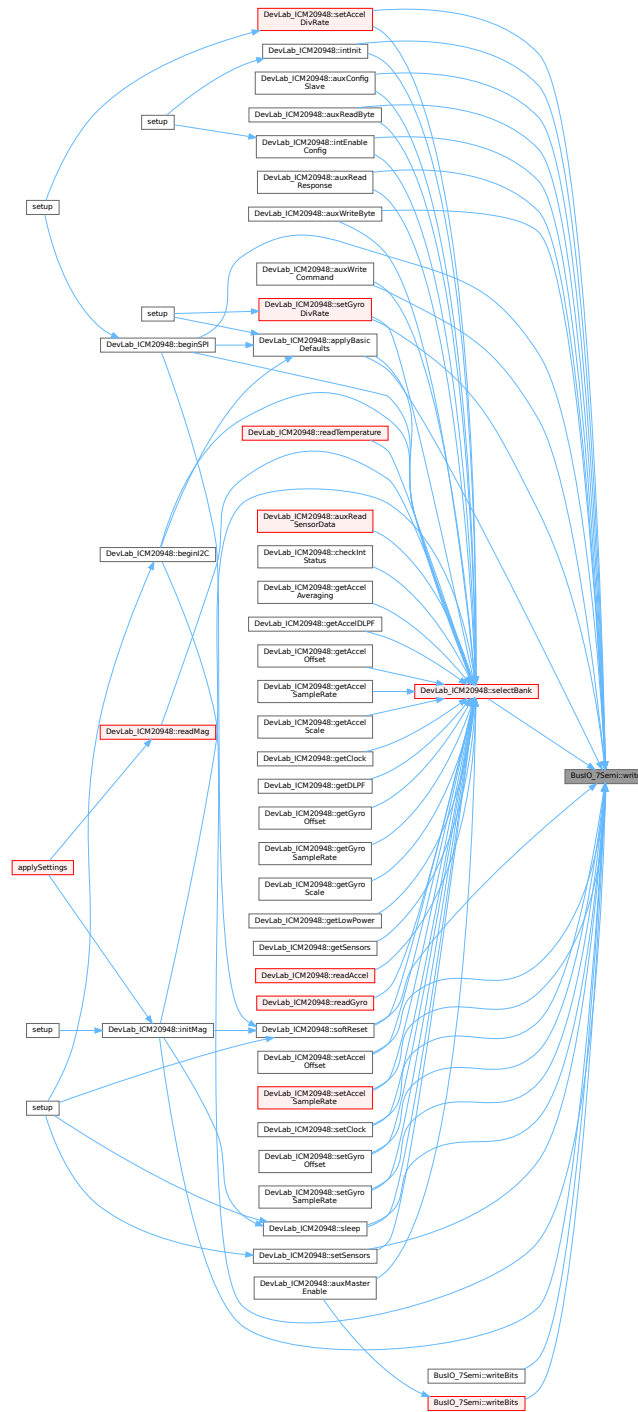
5.1.3.10 write() [3/3]

```
template<typename Bus >  
bool BusIO_7Semi< Bus >::write (  
    uint8_t reg,  
    uint8_t value ) [inline]
```

write 8-bit

Definition at line 39 of file [BusIO_7Semi.h](#).

Here is the caller graph for this function:

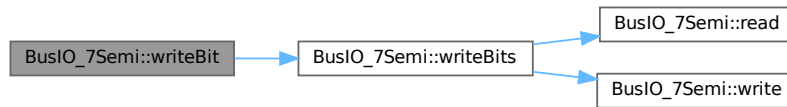


5.1.3.11 writeBit() [1/2]

```
template<typename Bus >
bool BusIO_7Semi< Bus >::writeBit (
    uint8_t reg,
    uint8_t pos,
    uint16_t value ) [inline]
```

Definition at line 151 of file [BusIO_7Semi.h](#).

Here is the call graph for this function:

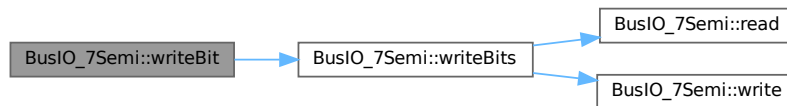


5.1.3.12 `writeBit()` [2/2]

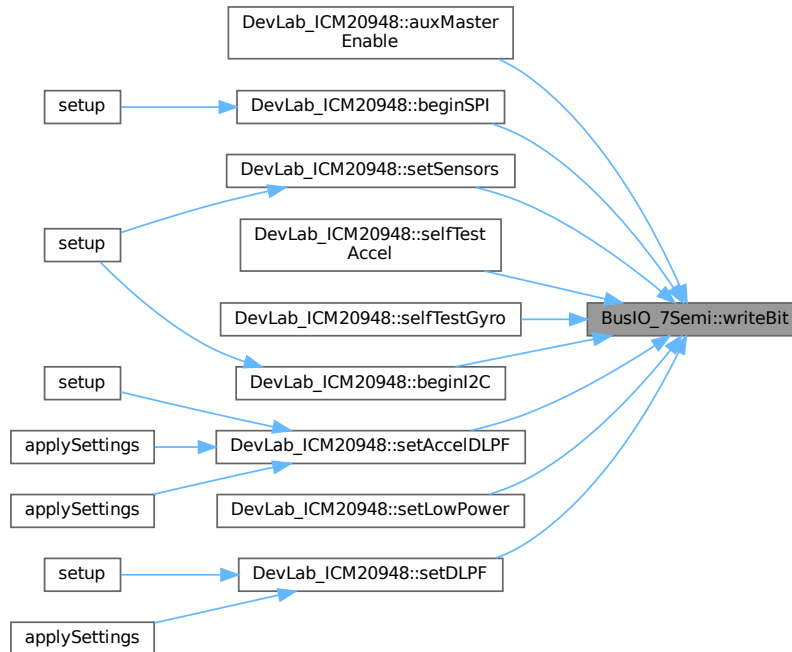
```
template<typename Bus >
bool BusIO_7Semi< Bus >::writeBit (
    uint8_t reg,
    uint8_t pos,
    uint8_t value ) [inline]
```

Definition at line 147 of file [BusIO_7Semi.h](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.1.3.13 writeBits() [1/2]

```

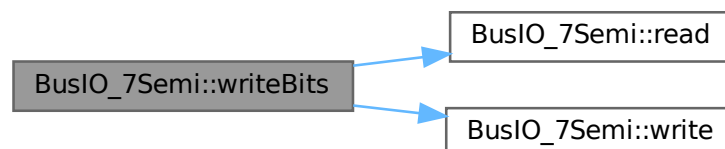
template<typename Bus >
bool BusIO_7Semi< Bus >::writeBits (
    uint8_t reg,
    uint8_t pos,
    uint8_t len,
    uint16_t value ) [inline]
writeBits (16-bit register)

```

- Modifies specific bits in 16-bit register

Definition at line 120 of file [BusIO_7Semi.h](#).

Here is the call graph for this function:

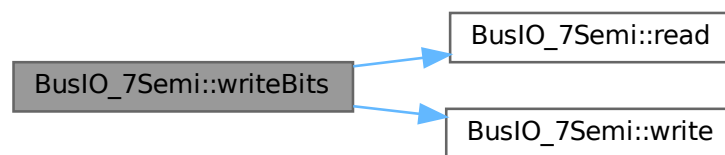


5.1.3.14 writeBits() [2/2]

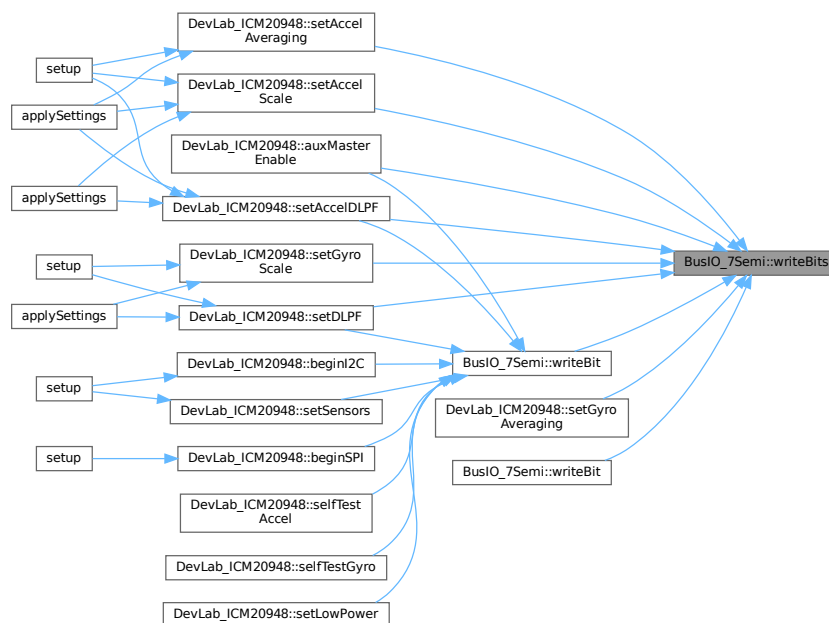
```
template<typename Bus >
bool BusIO_7Semi< Bus >::writeBits (
    uint8_t reg,
    uint8_t pos,
    uint8_t len,
    uint8_t value ) [inline]
writeBits (8-bit register)
```

- Modifies specific bits in 8-bit register

Definition at line 98 of file [BusIO_7Semi.h](#).
Here is the call graph for this function:



Here is the caller graph for this function:



The documentation for this class was generated from the following file:

- [src/BusIO_7Semi.h](#)

5.2 configDLPF Struct Reference

Gyroscope DLPF divider and timing configuration.

Public Attributes

- `uint8_t dlpf_idx`
Index into the gyroDLPF lookup table.
- `uint16_t div`
Sample-rate divider written to the IMU register.
- `float nbw_hz`
Noise bandwidth in hertz for reporting.
- `float odr_hz`
Output data rate in hertz for timing calculations.
- `const char * nombre`
Text label printed with each captured sample.

5.2.1 Detailed Description

Gyroscope DLPF divider and timing configuration.

Accelerometer DLPF divider and timing configuration.

Definition at line 27 of file [gyr_test.ino](#).

5.2.2 Member Data Documentation

5.2.2.1 div

`uint16_t configDLPF::div`

Sample-rate divider written to the IMU register.

Definition at line 31 of file [gyr_test.ino](#).

5.2.2.2 dlpf_idx

`uint8_t configDLPF::dlpf_idx`

Index into the gyroDLPF lookup table.

Index into the accelDLPF lookup table.

Definition at line 29 of file [gyr_test.ino](#).

5.2.2.3 nbw_hz

`float configDLPF::nbw_hz`

Noise bandwidth in hertz for reporting.

Definition at line 33 of file [gyr_test.ino](#).

5.2.2.4 nombre

`const char * configDLPF::nombre`

Text label printed with each captured sample.

Definition at line 37 of file [gyr_test.ino](#).

5.2.2.5 odr_hz

`float configDLPF::odr_hz`

Output data rate in hertz for timing calculations.

Definition at line 35 of file [gyr_test.ino](#).

The documentation for this struct was generated from the following files:

- [examples/gyr_test/gyr_test.ino](#)
- [examples/test/test.ino](#)
- [examples/test2/test2.ino](#)

5.3 DevLab_ICM20948 Class Reference

```
#include <DevLab_ICM20948.h>
```

Public Member Functions

- [DevLab_ICM20948](#) ()
- bool [beginI2C](#) (uint8_t address=0x69, TwoWire &i2cPort=Wire, uint32_t i2cSpeed=400000)
- bool [beginSPI](#) (uint8_t csPin, SPIClass &spiPort=SPI, uint32_t spiSpeed=1000000)
- bool [readWhoAmI](#) (uint8_t &whoAmI)
- bool [selectBank](#) (uint8_t bank)
- bool [softReset](#) ()
- bool [sleep](#) (bool en)
- bool [applyBasicDefaults](#) ()
- bool [readAccel](#) (float &x, float &y, float &z)
- bool [readGyro](#) (float &x, float &y, float &z)
- bool [readTemperature](#) (float &temperature)
- bool [readMag](#) (float &x, float &y, float &z)
- bool [initMag](#) ()
- bool [setMagOpMode](#) (ICM20948_Op_Mode opMode)
- bool [setLowPower](#) (bool enable)
- bool [getLowPower](#) (bool &enable)
- bool [setClock](#) (ICM20948_Clock_Source clock)
- bool [getClock](#) (uint8_t &clock)
- bool [setGyroSampleRate](#) (float sampleRate)
- bool [getGyroSampleRate](#) (float &sampleRate)
- bool [setGyroDivRate](#) (uint8_t divisor)
- bool [setDLPF](#) (ICM20948_Gyro_DLPF dlpf, bool bypass)
- bool [getDLPF](#) (uint8_t &dlpf, bool &bypass)
- bool [setGyroScale](#) (ICM20948_Gyro_FullScale fullScale)
- bool [setGyroAveraging](#) (ICM20948_Gyro_Average avg)
- bool [getGyroScale](#) (uint8_t &fullScale)
- bool [selfTestGyro](#) (bool x, bool y, bool z)
- bool [setAccelDivRate](#) (uint16_t divisor)
- bool [setAccelSampleRate](#) (uint16_t sampleRate)
- bool [getAccelSampleRate](#) (float &sampleRate)
- bool [selfTestAccel](#) (bool x, bool y, bool z)
- bool [setAccelScale](#) (ICM20948_Accel_FullScale fullScale)
- bool [getAccelScale](#) (uint8_t &fullScale)
- bool [setAccelDLPF](#) (uint8_t dlpf, bool bypass)
- bool [getAccelDLPF](#) (uint8_t &dlpf, bool &bypass)
- bool [setAccelAveraging](#) (ICM20948_Accel_Average avg)
- bool [getAccelAveraging](#) (uint8_t &avg)
- bool [setSensors](#) (bool accel_on, bool gyro_on, bool temp_on)
- bool [getSensors](#) (bool &accel_on, bool &gyro_on, bool &temp_on)
- bool [setGyroOffset](#) (uint16_t offsetX, uint16_t offsetY, uint16_t offsetZ)
- bool [getGyroOffset](#) (int16_t &offsetX, int16_t &offsetY, int16_t &offsetZ)
- bool [setAccelOffset](#) (int16_t offsetX, int16_t offsetY, int16_t offsetZ)
- bool [getAccelOffset](#) (int16_t &offsetX, int16_t &offsetY, int16_t &offsetZ)
- bool [intInit](#) (const ICM20948_IntPinConfig &cfg)
- bool [intEnableConfig](#) (const ICM20948_IntEnableConfig &cfg)
- bool [checkIntStatus](#) (ICM20948_IntStatus &status)
- bool [auxMasterEnable](#) (uint8_t clkFreq)
- bool [auxWriteByte](#) (uint8_t slaveAddr, uint8_t reg, uint8_t data)
- bool [auxReadByte](#) (uint8_t slaveAddr, uint8_t reg, uint8_t &data)

- bool [auxWriteCommand](#) (uint8_t slaveAddr, uint8_t cmd)
- bool [auxConfigSlave](#) (uint8_t slaveAddr, uint8_t reg, uint8_t numBytes)
- bool [auxReadSensorData](#) (uint8_t *buf, uint8_t len)
- bool [auxReadResponse](#) (uint8_t slaveAddr, uint8_t &data)
- bool [auxRead12bit](#) (uint16_t &raw)

5.3.1 Detailed Description

Class

- Manages ICM-20948 over I2C (SPI path disabled in this cut)
- Caches scale factors for LSB->physical conversions

Definition at line 172 of file [DevLab_ICM20948.h](#).

5.3.2 Constructor & Destructor Documentation

5.3.2.1 DevLab_ICM20948()

DevLab_ICM20948::DevLab_ICM20948 ()

- Construct with default I2C address; call begin() to attach bus

Definition at line 4 of file [DevLab_ICM20948.cpp](#).

5.3.3 Member Function Documentation

5.3.3.1 applyBasicDefaults()

bool DevLab_ICM20948::applyBasicDefaults ()
 applyBasicDefaults

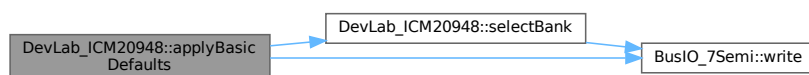
- Apply basic sensor configuration
- Configure power, filters, ranges, and sampling rates

Returns:

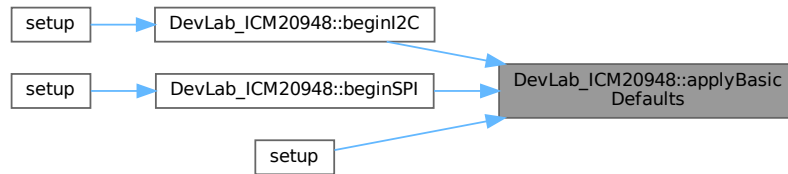
- true → Configuration successful
- false → Any register write failed

Definition at line 181 of file [DevLab_ICM20948.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.3.3.2 auxConfigSlave()

```

bool DevLab_ICM20948::auxConfigSlave (
    uint8_t slaveAddr,
    uint8_t reg,
    uint8_t numBytes )
  
```

auxConfigSlave

- Configure SLV0 for automatic periodic reads from an auxiliary I2C slave
- The ICM20948 master will read numBytes starting at reg on every ODR cycle
- Data is deposited into EXT_SLV_SENS_DATA_00 and subsequent registers
- Call once during initialization; read results with auxReadSensorData

Parameters:

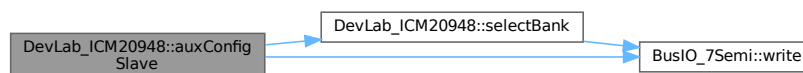
- slaveAddr: 7-bit I2C address of the auxiliary slave
- reg: starting register address to read from on the slave
- numBytes: number of bytes to read per cycle (1–15)

Returns:

- true → SLV0 configured successfully
- false → Operation failed

Definition at line 1374 of file [DevLab_ICM20948.cpp](#).

Here is the call graph for this function:



5.3.3.3 auxMasterEnable()

```

bool DevLab_ICM20948::auxMasterEnable (
    uint8_t clkFreq )
  
```

auxMasterEnable

- Enable the ICM20948 internal I2C master for auxiliary bus
- Clears BYPASS_EN so the aux bus is controlled by the ICM
- Sets I2C_MST_EN in USER_CTRL
- Configures auxiliary master clock frequency

Parameters:

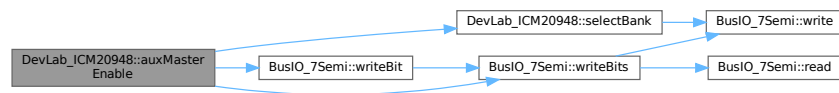
- clkFreq: clock divider value (e.g. 0x07 $\approx$ 345.6 kHz, 0x0D $\approx$ 400 kHz)

Returns:

- true $\rightarrow$ Master enabled successfully
- false $\rightarrow$ Operation failed

Definition at line 1260 of file [DevLab_ICM20948.cpp](#).

Here is the call graph for this function:



5.3.3.4 auxRead12bit()

```
bool DevLab_ICM20948::auxRead12bit (
    uint16_t & raw )
auxRead12bit
```

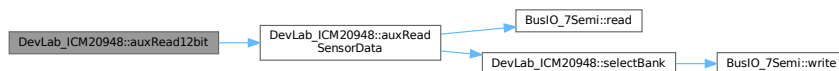
- Read a 12-bit value previously configured via `auxConfigSlave` (SLV0, numBytes = 2)
- Assumes little-endian packing: first byte = LSB, second byte = MSB
- Result is masked to 12 bits (0-4095)

Returns:

- true $\rightarrow$ Read successful
- false $\rightarrow$ Read failed

Definition at line 1424 of file [DevLab_ICM20948.cpp](#).

Here is the call graph for this function:



5.3.3.5 auxReadByte()

```
bool DevLab_ICM20948::auxReadByte (
    uint8_t slaveAddr,
    uint8_t reg,
    uint8_t & data )
auxReadByte
```

- Read a single byte from a register of an auxiliary I2C slave via SLV4
- Generates a combined write-then-read transaction (reg byte + repeated START)
- Suitable for slaves that follow standard I2C register-read protocol

Note: slaves that need separate write/read transactions (e.g. command-response firmware) should use auxWriteCommand + auxReadResponse instead.

Parameters:

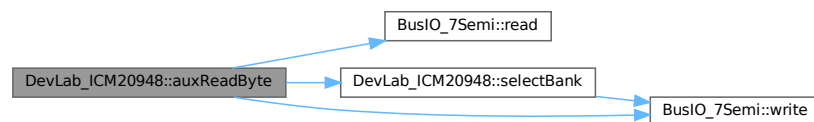
- slaveAddr: 7-bit I2C address of the auxiliary slave
- reg: register address to read from
- data: output byte received from slave

Returns:

- true → Read successful
- false → Timeout or NACK

Definition at line 1312 of file [DevLab_ICM20948.cpp](#).

Here is the call graph for this function:



5.3.3.6 auxReadResponse()

```
bool DevLab_ICM20948::auxReadResponse (
    uint8_t slaveAddr,
    uint8_t & data )
auxReadResponse
```

- Read a single response byte from an auxiliary I2C slave via SLV4
- Generates a pure read transaction: [START, addr+R, 1 byte, STOP]
- No register byte is sent (REG_DIS); slave must already be ready to transmit
- Use after auxWriteCommand to complete a command-response exchange with slaves that require separate write and read transactions (e.g. PY32F0 firmware slaves)

Parameters:

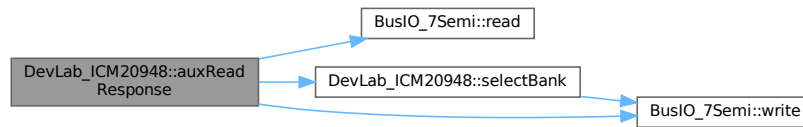
- slaveAddr: 7-bit I2C address of the auxiliary slave
- data: output byte received from slave

Returns:

- true → Response received successfully
- false → Timeout or NACK

Definition at line 1397 of file [DevLab_ICM20948.cpp](#).

Here is the call graph for this function:



5.3.3.7 auxReadSensorData()

```
bool DevLab_ICM20948::auxReadSensorData (
    uint8_t * buf,
    uint8_t len )
auxReadSensorData
```

- Read bytes from EXT_SLV_SENS_DATA registers (BANK 0)
- Returns data collected by the ICM20948 master from SLV0–SLV3 on the last cycle
- Must call `auxConfigSlave` first to set up the automatic read

Parameters:

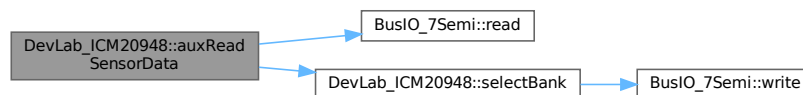
- `buf`: output buffer to store the received bytes
- `len`: number of bytes to read (must match `numBytes` configured in `auxConfigSlave`)

Returns:

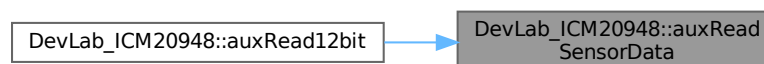
- true → Read successful
- false → Bus error

Definition at line 1391 of file [DevLab_ICM20948.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.3.3.8 auxWriteByte()

```
bool DevLab_ICM20948::auxWriteByte (
    uint8_t slaveAddr,
    uint8_t reg,
    uint8_t data )
auxWriteByte
```

- Write a single byte to a register of an auxiliary I2C slave via SLV4
- Uses one-shot SLV4 transaction: polls SLV4_DONE, checks for NACK
- Suitable for configuration writes (not continuous streaming)

Parameters:

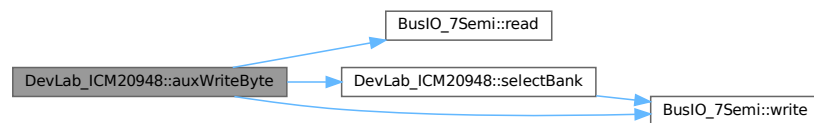
- slaveAddr: 7-bit I2C address of the auxiliary slave
- reg: target register address on the slave
- data: byte to write

Returns:

- true → Write acknowledged by slave
- false → Timeout or NACK

Definition at line 1281 of file [DevLab_ICM20948.cpp](#).

Here is the call graph for this function:



5.3.3.9 auxWriteCommand()

```
bool DevLab_ICM20948::auxWriteCommand (
    uint8_t slaveAddr,
    uint8_t cmd )
auxWriteCommand
```

- Send a single command byte to an auxiliary I2C slave via SLV4
- Uses REG_DIS: generates [START, addr+W, cmd_byte, STOP] with no register prefix
- Designed for slaves that use a command-response protocol (no register addressing)

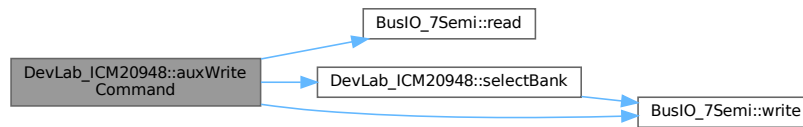
Parameters:

- slaveAddr: 7-bit I2C address of the auxiliary slave
- cmd: command byte to send

Returns:

- true → Command acknowledged by slave
- false → Timeout or NACK

Definition at line 1346 of file [DevLab_ICM20948.cpp](#).
Here is the call graph for this function:



5.3.3.10 beginI2C()

```

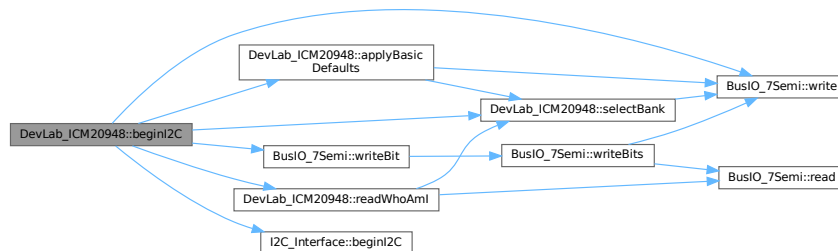
bool DevLab_ICM20948::beginI2C (
    uint8_t address = 0x69,
    TwoWire & i2cPort = Wire,
    uint32_t i2cSpeed = 400000 )
beginI2C
  
```

- Initialize ICM20948 using I2C interface
- Sets up communication layer (Interface + BusIO)
- Verifies device identity (WHO_AM_I)
- Configures basic power and clock settings

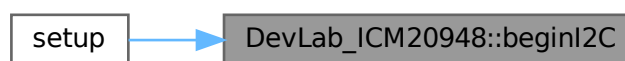
Returns:

- true → Device initialized successfully
- false → Communication or configuration failed

Definition at line 6 of file [DevLab_ICM20948.cpp](#).
Here is the call graph for this function:



Here is the caller graph for this function:

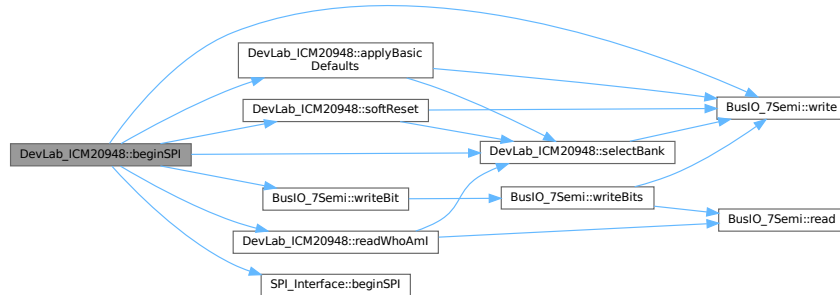


5.3.3.11 beginSPI()

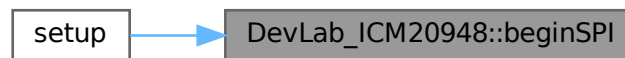
```
bool DevLab_ICM20948::beginSPI (
    uint8_t csPin,
    SPIClass & spiPort = SPI,
    uint32_t spiSpeed = 1000000 )
```

Definition at line 60 of file [DevLab_ICM20948.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.3.3.12 checkIntStatus()

```
bool DevLab_ICM20948::checkIntStatus (
    ICM20948_IntStatus & status )
```

checkIntStatus

- Read and parse all four interrupt status registers in a single burst read
- Covers INT_STATUS (0x19), INT_STATUS_1 (0x1A), INT_STATUS_2 (0x1B), INT_STATUS_3 (0x1C) from BANK 0
- Reading clears the interrupt flags when clearMode = 0 in [ICM20948_IntPinConfig](#)
- Typically called inside the INT pin ISR or polled in the main loop

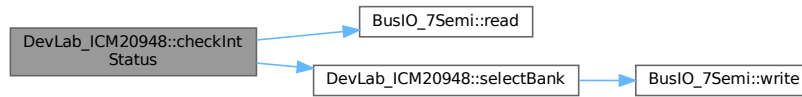
Parameters:

- status: [ICM20948_IntStatus](#) struct populated with the current interrupt flags

Returns:

- true → Status read successfully
- false → Operation failed

Definition at line 1231 of file [DevLab_ICM20948.cpp](#).
Here is the call graph for this function:



5.3.3.13 getAccelAveraging()

```
bool DevLab_ICM20948::getAccelAveraging (
    uint8_t & avg )
getAccelAveraging
```

- Read accelerometer averaging / decimation setting (DEC3)

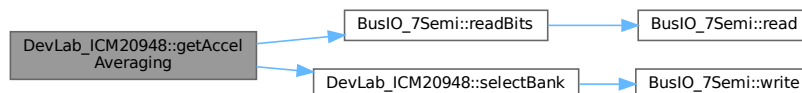
Flow:

- Validate bus pointer
- Select USER BANK 2
- Read DEC3 bits [1:0]

Returns:

- true → Read successful
- false → Operation failed

Definition at line 988 of file [DevLab_ICM20948.cpp](#).
Here is the call graph for this function:



5.3.3.14 getAccelDLPF()

```
bool DevLab_ICM20948::getAccelDLPF (
    uint8_t & dlpf,
    bool & bypass )
getAccelDLPF
```

- Read accelerometer DLPF configuration
- Returns filter setting and bypass state

Flow:

- Validate bus pointer
- Select USER BANK 2

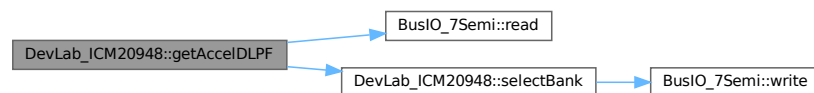
- Read ACCEL_CONFIG register
- Extract bypass and DLPF bits

Returns:

- true → Read successful
- false → Operation failed

Definition at line 926 of file [DevLab_ICM20948.cpp](#).

Here is the call graph for this function:



5.3.3.15 getAccelOffset()

```
bool DevLab_ICM20948::getAccelOffset (
    int16_t & offsetX,
    int16_t & offsetY,
    int16_t & offsetZ )
getAccelOffset
```

- Read accelerometer offset values for X, Y, Z axes
- Reads offset registers from USER BANK 1

Flow:

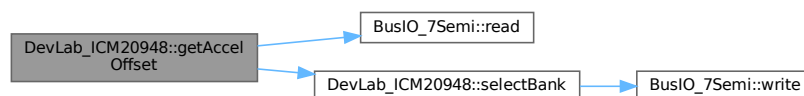
- Validate bus pointer
- Select USER BANK 1
- Read offset registers
- Convert to signed values

Returns:

- true → Read successful
- false → Operation failed

Definition at line 1151 of file [DevLab_ICM20948.cpp](#).

Here is the call graph for this function:



5.3.3.16 getAccelSampleRate()

```
bool DevLab_ICM20948::getAccelSampleRate (
    float & sampleRate )
getAccelSampleRate
```

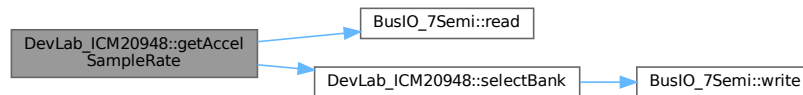
- Get accelerometer output data rate (ODR)
- Computes value from divider registers

Returns:

- true → Read successful
- false → Operation failed

Definition at line 796 of file [DevLab_ICM20948.cpp](#).

Here is the call graph for this function:

**5.3.3.17 getAccelScale()**

```
bool DevLab_ICM20948::getAccelScale (
    uint8_t & fullScale )
getAccelScale
```

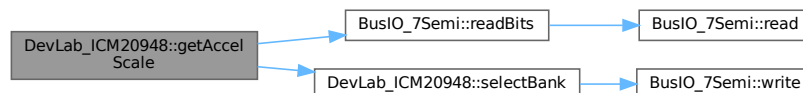
- Get current accelerometer full-scale range (FS_SEL)

Returns:

- true → Read successful
- false → Operation failed

Definition at line 878 of file [DevLab_ICM20948.cpp](#).

Here is the call graph for this function:

**5.3.3.18 getClock()**

```
bool DevLab_ICM20948::getClock (
    uint8_t & clock )
getClock
```

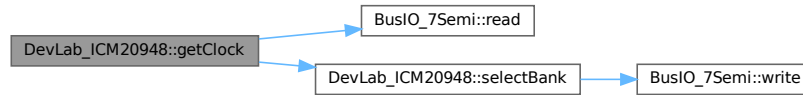
- Read current clock source (CLKSEL [2:0])

Returns:

- true → Read successful
- false → Operation failed

Definition at line 534 of file [DevLab_ICM20948.cpp](#).

Here is the call graph for this function:



5.3.3.19 getDLPF()

```
bool DevLab_ICM20948::getDLPF (
    uint8_t & dlpf,
    bool & bypass )
getDLPF
```

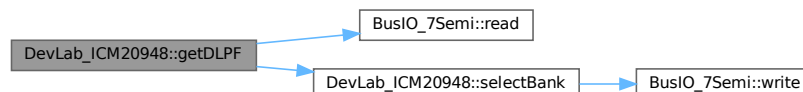
- Read gyroscope DLPF configuration
- Returns filter setting and bypass state

Returns:

- true → Read successful
- false → Operation failed

Definition at line 619 of file [DevLab_ICM20948.cpp](#).

Here is the call graph for this function:



5.3.3.20 getGyroOffset()

```
bool DevLab_ICM20948::getGyroOffset (
    int16_t & offsetX,
    int16_t & offsetY,
    int16_t & offsetZ )
getGyroOffset
```

- Read gyroscope offset values for X, Y, Z axes
- Reads offset registers from USER BANK 2

Flow:

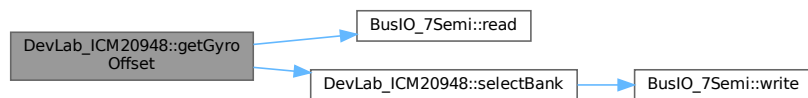
- Validate bus pointer
- Select USER BANK 2
- Read offset registers
- Convert to signed values

Returns:

- true → Read successful
- false → Operation failed

Definition at line 1099 of file [DevLab_ICM20948.cpp](#).

Here is the call graph for this function:



5.3.3.21 getGyroSampleRate()

```
bool DevLab_ICM20948::getGyroSampleRate (
    float & sampleRate )
getGyroSampleRate
```

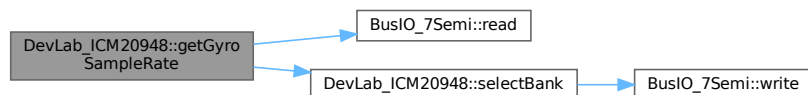
- Get current gyroscope sample rate
- Computes value from divider register

Returns:

- true → Read successful
- false → Operation failed

Definition at line 571 of file [DevLab_ICM20948.cpp](#).

Here is the call graph for this function:



5.3.3.22 getGyroScale()

```
bool DevLab_ICM20948::getGyroScale (
    uint8_t & fullScale )
getGyroScale
```

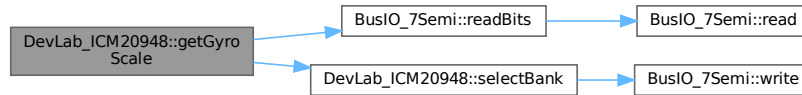
- Get current gyroscope full-scale range (FS_SEL

Returns:

- true → Read successful
- false → Operation failed

Definition at line 665 of file [DevLab_ICM20948.cpp](#).

Here is the call graph for this function:



5.3.3.23 getLowPower()

```
bool DevLab_ICM20948::getLowPower (
    bool & enable )
getLowPower
```

- Read low power mode status
- Returns state of LP_EN bit

Returns:

- true → Read successful
- false → Operation failed

Definition at line 502 of file [DevLab_ICM20948.cpp](#).

Here is the call graph for this function:



5.3.3.24 getSensors()

```
bool DevLab_ICM20948::getSensors (
    bool & accel_on,
    bool & gyro_on,
    bool & temp_on )
getSensors
```

- Read accel, gyro, and temperature enable state

Flow:

- Validate bus pointer
- Select USER BANK 0
- Read PWR_MGMT_2 register

- Read TEMP_DIS bit
- Decode enable states

Returns:

- true → Read successful
- false → Operation failed

Definition at line 1039 of file [DevLab_ICM20948.cpp](#).

Here is the call graph for this function:



5.3.3.25 initMag()

```
bool DevLab_ICM20948::initMag ( )
```

initMag

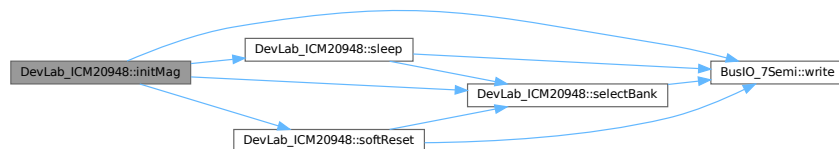
- Initialize AK09916 magnetometer using internal I2C master
- Verifies WHO_AM_I and enables continuous mode
- Configures SLV0 for automatic data read

Returns:

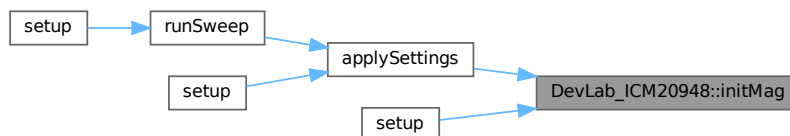
- true → Initialization successful
- false → Operation failed

Definition at line 334 of file [DevLab_ICM20948.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.3.3.26 intEnableConfig()

```
bool DevLab_ICM20948::intEnableConfig (
    const ICM20948_IntEnableConfig & cfg )
intEnableConfig
```

- Enable or disable individual interrupt sources routed to the INT pin
- Writes INT_ENABLE (0x10), INT_ENABLE_1 (0x11), INT_ENABLE_2 (0x12), and INT_ENABLE_3 (0x13) in BANK 0
- Must call [intInit\(\)](#) first to configure the pin electrical behavior

Parameters:

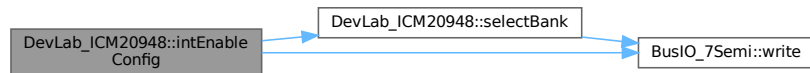
- cfg: [ICM20948_IntEnableConfig](#) struct with the desired interrupt sources enabled

Returns:

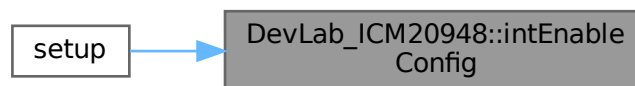
- true → All four registers written successfully
- false → Operation failed

Definition at line 1200 of file [DevLab_ICM20948.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.3.3.27 intInit()

```
bool DevLab_ICM20948::intInit (
    const ICM20948_IntPinConfig & cfg )
intInit
```

- Configure the physical behavior of the INT pin (INT_PIN_CFG register, BANK 0)
- Sets active level, drive mode (push-pull / open-drain), latch vs pulse mode, clear condition, FSYNC level, FSYNC interrupt enable, and I2C bypass

Parameters:

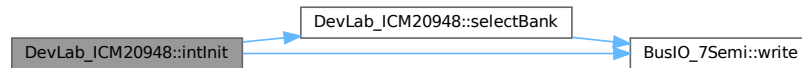
- cfg: [ICM20948_IntPinConfig](#) struct with the desired pin settings

Returns:

- true → Configuration written successfully
- false → Operation failed

Definition at line 1176 of file [DevLab_ICM20948.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.3.3.28 readAccel()

```

bool DevLab_ICM20948::readAccel (
    float & x,
    float & y,
    float & z )
  
```

readAccel

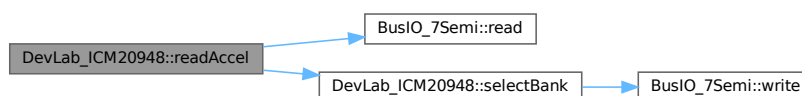
- Read accelerometer data (X, Y, Z)
- Convert raw values to g using LSB scale

Returns:

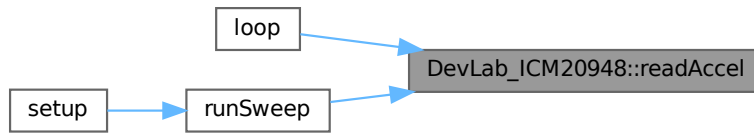
- true → Read successful
- false → Read failed

Definition at line 238 of file [DevLab_ICM20948.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.3.3.29 readGyro()

```

bool DevLab_ICM20948::readGyro (
    float & x,
    float & y,
    float & z )

```

readGyro

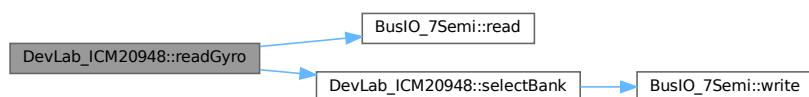
- Read gyroscope data (X, Y, Z)
- Convert raw values to dps using LSB scale

Returns:

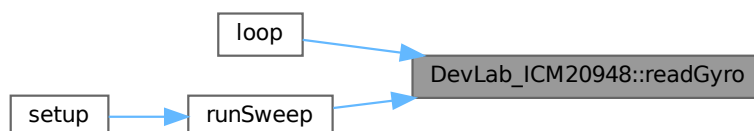
- true → Read successful
- false → Read failed

Definition at line 258 of file [DevLab_ICM20948.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.3.3.30 readMag()

```
bool DevLab_ICM20948::readMag (
    float & x,
    float & y,
    float & z )
readMag
```

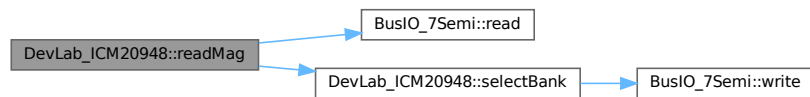
- Read magnetometer data via internal I2C master
- Data comes from EXT_SLV_SENS_DATA registers
- Convert raw values to μT

Returns:

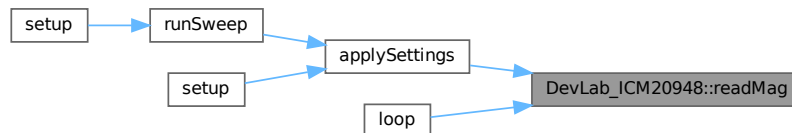
- true → Read successful
- false → Read failed

Definition at line 310 of file [DevLab_ICM20948.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:

**5.3.3.31 readTemperature()**

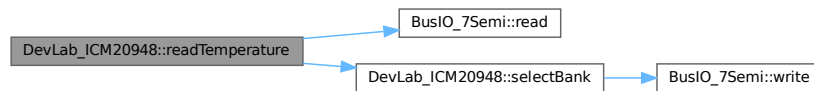
```
bool DevLab_ICM20948::readTemperature (
    float & temperature )
readTemperature
```

- Read temperature sensor
- Convert raw values to $^{\circ}\text{C}$

Returns:

- true → Read successful
- false → Read failed

Definition at line 278 of file [DevLab_ICM20948.cpp](#).
Here is the call graph for this function:



Here is the caller graph for this function:



5.3.3.32 readWhoAmI()

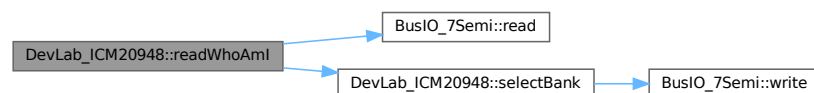
```
bool DevLab_ICM20948::readWhoAmI (
    uint8_t & whoAmI )
readWhoAmI
```

- Read WHO_AM_I register (device ID)
- Used during initialization to verify ICM20948
- Expected value: 0xEA

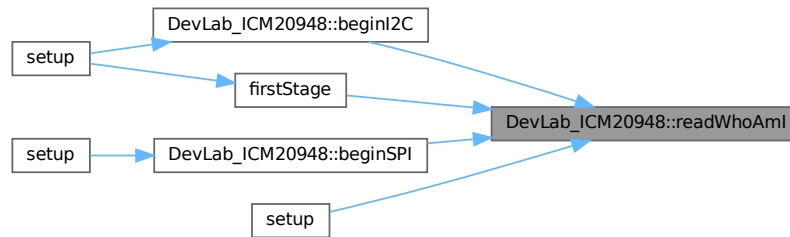
Returns:

- true → Read successful
- false → Operation failed

Definition at line 118 of file [DevLab_ICM20948.cpp](#).
Here is the call graph for this function:



Here is the caller graph for this function:



5.3.3.33 selectBank()

```
bool DevLab_ICM20948::selectBank (
    uint8_t bank )
selectBank
```

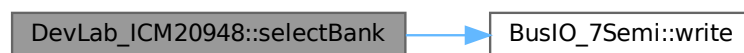
- Select USER BANK (0–3)
- Used to access different register groups inside ICM20948

Returns:

- true → Bank switch successful
- false → Write failed

Definition at line 132 of file [DevLab_ICM20948.cpp](#).

Here is the call graph for this function:



5.3.3.34 selfTestAccel()

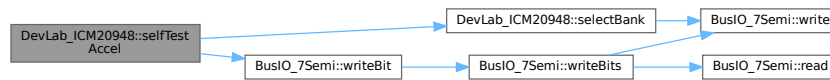
```
bool DevLab_ICM20948::selfTestAccel (
    bool x,
    bool y,
    bool z )
selfTestAccel
```

- Enable or disable accelerometer self-test per axis

Returns:

- true → Operation successful
- false → Operation failed

Definition at line 826 of file [DevLab_ICM20948.cpp](#).
Here is the call graph for this function:



5.3.3.35 selfTestGyro()

```
bool DevLab_ICM20948::selfTestGyro (
    bool x,
    bool y,
    bool z )
selfTestGyro
```

- Enable or disable gyroscope self-test per axis

Returns:

- true → Operation successful
- false → Operation failed

Definition at line 683 of file [DevLab_ICM20948.cpp](#).
Here is the call graph for this function:



5.3.3.36 setAccelAveraging()

```
bool DevLab_ICM20948::setAccelAveraging (
    ICM20948_Accel_Average avg )
setAccelAveraging
```

- Configure accelerometer averaging / decimation (DEC3)
- Controls internal averaging of accel samples

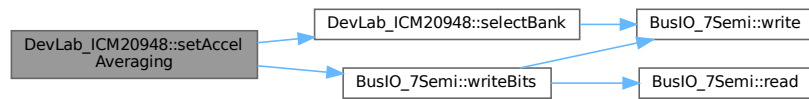
Flow:

- Validate bus pointer
- Select USER BANK 2
- Write DEC3 bits [1:0]

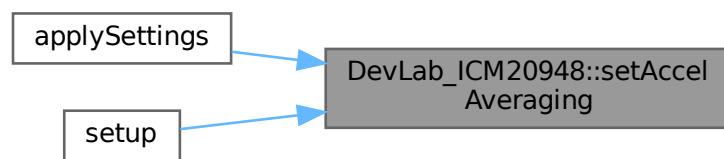
Returns:

- true → Operation successful
- false → Operation failed

Definition at line 952 of file [DevLab_ICM20948.cpp](#).
Here is the call graph for this function:



Here is the caller graph for this function:

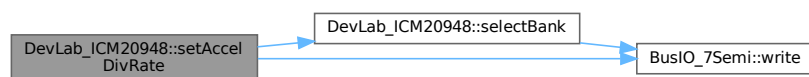


5.3.3.37 setAccelDivRate()

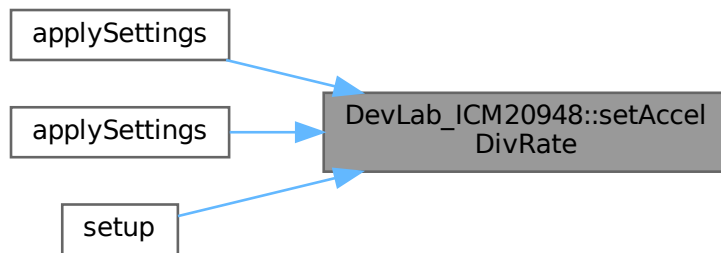
```
bool DevLab_ICM20948::setAccelDivRate (
    uint16_t divisor )
```

Definition at line 748 of file [DevLab_ICM20948.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.3.3.38 setAccelDLPF()

```
bool DevLab_ICM20948::setAccelDLPF (
    uint8_t dlpf,
    bool bypass )
setAccelDLPF
```

- Configure accelerometer digital low-pass filter (DLPF)
- Option to bypass filter (full bandwidth)

Flow:

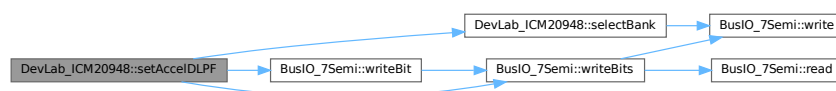
- Validate bus pointer
- Select USER BANK 2
- Set or clear bypass bit
- Configure DLPF bits if not bypassed

Returns:

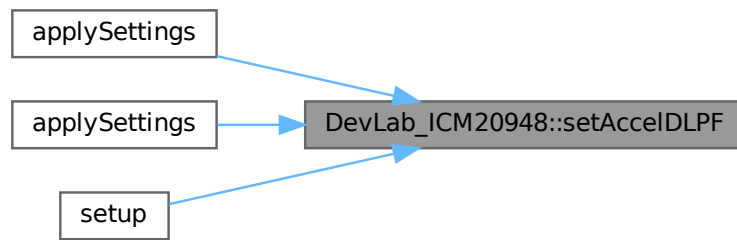
- true → Operation successful
- false → Operation failed

Definition at line 897 of file [DevLab_ICM20948.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.3.3.39 setAccelOffset()

```

bool DevLab_ICM20948::setAccelOffset (
    int16_t offsetX,
    int16_t offsetY,
    int16_t offsetZ )
setAccelOffset
  
```

- Set accelerometer offset values for X, Y, Z axes
- Writes offset registers in USER BANK 1

Flow:

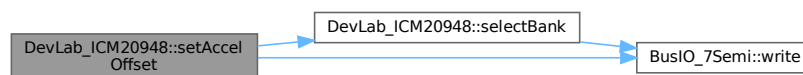
- Validate bus pointer
- Select USER BANK 1
- Pack offsets into byte array
- Write to offset registers

Returns:

- `true` → Operation successful
- `false` → Operation failed

Definition at line 1124 of file [DevLab_ICM20948.cpp](#).

Here is the call graph for this function:



5.3.3.40 setAccelSampleRate()

```
bool DevLab_ICM20948::setAccelSampleRate (
    uint16_t sampleRate )
setAccelSampleRate
```

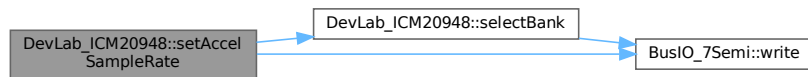
- Set accelerometer output data rate (ODR)
- Uses 1125 Hz base rate with 12-bit divider

Returns:

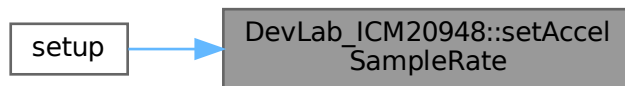
- true → Operation successful
- false → Operation failed

Definition at line 709 of file [DevLab_ICM20948.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.3.3.41 setAccelScale()

```
bool DevLab_ICM20948::setAccelScale (
    ICM20948_Accel_FullScale fullScale )
setAccelScale
```

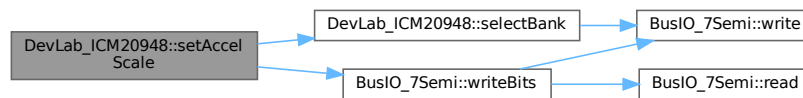
- Set accelerometer full-scale range (FS_SEL)
- Controls measurement range ($\pm 2g$, $\pm 4g$, $\pm 8g$, $\pm 16g$)

Returns:

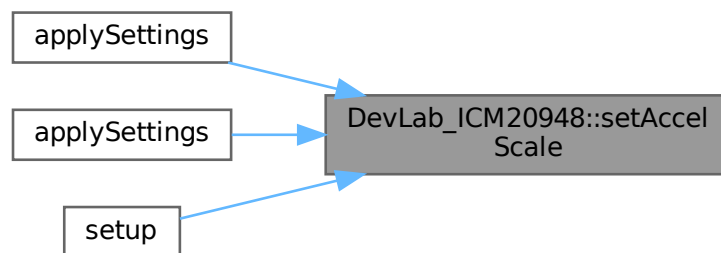
- true → Operation successful
- false → Operation failed

Definition at line 857 of file [DevLab_ICM20948.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.3.3.42 setClock()

```
bool DevLab_ICM20948::setClock (
    ICM20948_Clock_Source clock )
setClock
```

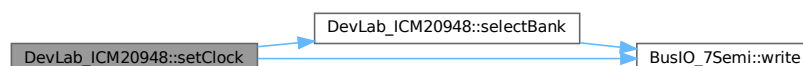
- Set clock source (CLKSEL [2:0])
- Selects internal clock for sensor operation

Returns:

- true → Operation successful
- false → Operation failed

Definition at line 520 of file `DevLab_ICM20948.cpp`.

Here is the call graph for this function:



5.3.3.43 setDLPF()

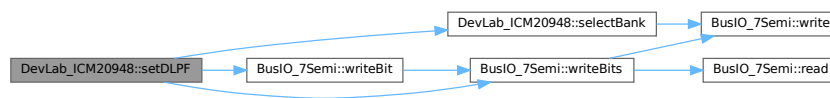
```
bool DevLab_ICM20948::setDLPF (
    ICM20948_Gyro_DLPF dlpf,
    bool bypass )
setDLPF
```

- Configure gyroscope digital low-pass filter (DLPF)
- Option to bypass filter (full bandwidth)

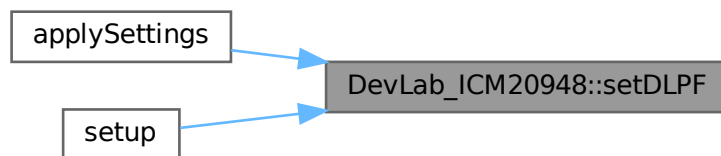
Returns:

- true → Operation successful
- false → Operation failed

Definition at line 593 of file [DevLab_ICM20948.cpp](#).
Here is the call graph for this function:



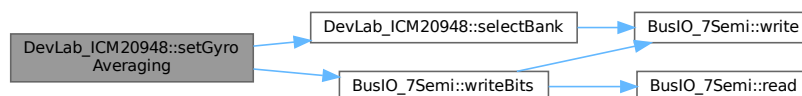
Here is the caller graph for this function:



5.3.3.44 setGyroAveraging()

```
bool DevLab_ICM20948::setGyroAveraging (
    ICM20948_Gyro_Average avg )
```

Definition at line 970 of file [DevLab_ICM20948.cpp](#).
Here is the call graph for this function:

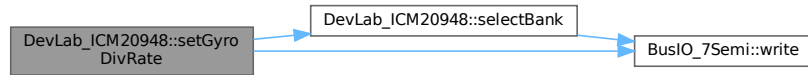


5.3.3.45 setGyroDivRate()

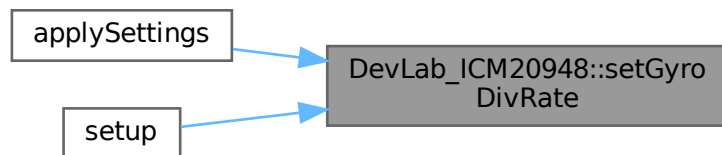
```
bool DevLab_ICM20948::setGyroDivRate (
    uint8_t divisor )
```

Definition at line 780 of file [DevLab_ICM20948.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.3.3.46 setGyroOffset()

```
bool DevLab_ICM20948::setGyroOffset (
    uint16_t offsetX,
    uint16_t offsetY,
    uint16_t offsetZ )
```

setGyroOffset

- Set gyroscope offset values for X, Y, Z axes
- Writes offset registers in USER BANK 2

Flow:

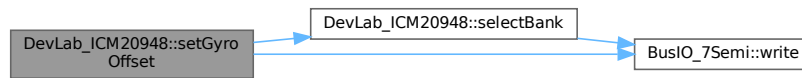
- Validate bus pointer
- Select USER BANK 2
- Pack offsets into byte array
- Write to offset registers

Returns:

- true → Operation successful
- false → Operation failed

Definition at line 1072 of file [DevLab_ICM20948.cpp](#).

Here is the call graph for this function:



5.3.3.47 `setGyroSampleRate()`

```
bool DevLab_ICM20948::setGyroSampleRate (
    float sampleRate )
setGyroSampleRate
```

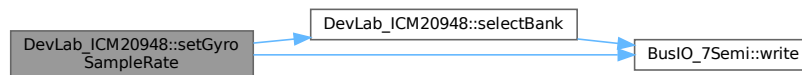
- Set gyroscope sample rate
- Uses internal divider (SRD) based on 1100 Hz base rate

Returns:

- true → Operation successful
- false → Invalid input or write failed

Definition at line 550 of file [DevLab_ICM20948.cpp](#).

Here is the call graph for this function:



5.3.3.48 `setGyroScale()`

```
bool DevLab_ICM20948::setGyroScale (
    ICM20948\_Gyro\_FullScale fullScale )
setGyroScale
```

- Set gyroscope full-scale range (FS_SEL)
- Controls sensitivity (dps range)

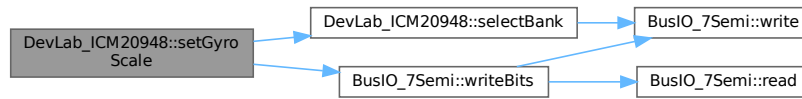
Flow:

- Validate bus pointer
- Select USER BANK 2
- Write FS_SEL bits [2:1]

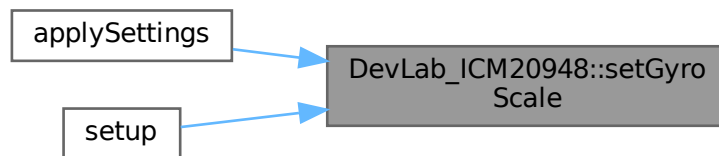
Returns:

- true → Operation successful
- false → Operation failed

Definition at line 644 of file [DevLab_ICM20948.cpp](#).
Here is the call graph for this function:



Here is the caller graph for this function:



5.3.3.49 setLowPower()

```
bool DevLab_ICM20948::setLowPower (
    bool enable )
setLowPower
```

- Enable or disable low power mode
- Controls LP_EN bit in PWR_MGMT_1

Returns:

- true → Operation successful
- false → Operation failed

Definition at line 488 of file [DevLab_ICM20948.cpp](#).
Here is the call graph for this function:



5.3.3.50 setMagOpMode()

```
bool DevLab_ICM20948::setMagOpMode (
    ICM20948_Op_Mode opMode )
setMagOpMode
```

- Set magnetometer operation mode

Returns:

- true → Mode set successfully
- false → Write failed

Definition at line 410 of file [DevLab_ICM20948.cpp](#).

Here is the caller graph for this function:



5.3.3.51 setSensors()

```

bool DevLab_ICM20948::setSensors (
    bool accel_on,
    bool gyro_on,
    bool temp_on )
setSensors
  
```

- Enable or disable accel, gyro, and temperature sensor
- Controls power gating via PWR_MGMT_2 and TEMP_DIS

Flow:

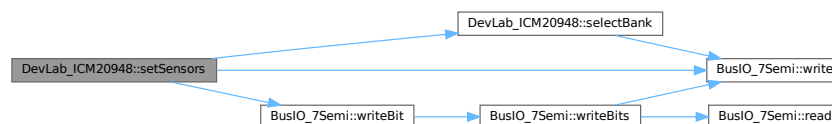
- Validate bus pointer
- Select USER BANK 0
- Build PWR_MGMT_2 mask
- Configure temperature enable/disable

Returns:

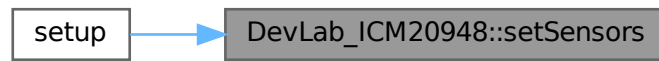
- true → Operation successful
- false → Operation failed

Definition at line 1006 of file [DevLab_ICM20948.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.3.3.52 sleep()

```
bool DevLab_ICM20948::sleep (
    bool en )
sleep
```

- Enable or disable sleep mode
- Maintains clock source (CLKSEL = 1)

Flow:

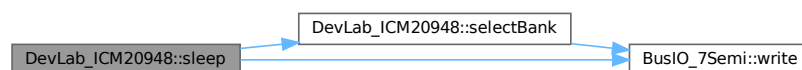
- Select USER BANK 0
- Set or clear SLEEP bit

Returns:

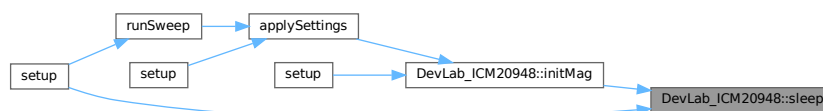
- true → Operation successful
- false → Operation failed

Definition at line 164 of file [DevLab_ICM20948.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.3.3.53 softReset()

```
bool DevLab_ICM20948::softReset ( )
softReset
```

- Perform software reset of ICM20948
- Resets all internal registers to default state

Flow:

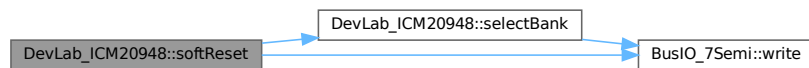
- Select USER BANK 0
- Set DEVICE_RESET bit
- Wait for device to restart

Returns:

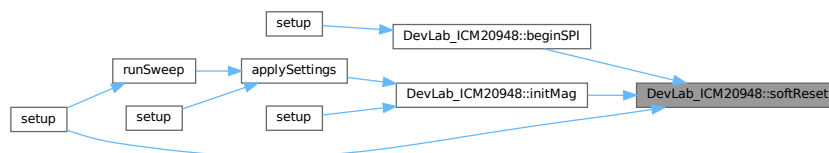
- true → Reset successful
- false → Operation failed

Definition at line 144 of file [DevLab_ICM20948.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



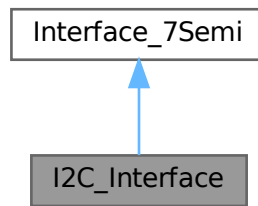
The documentation for this class was generated from the following files:

- [src/DevLab_ICM20948.h](#)
- [src/DevLab_ICM20948.cpp](#)

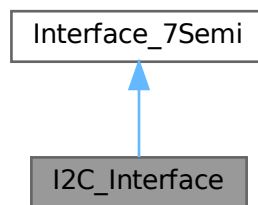
5.4 I2C_Interface Class Reference

```
#include <7Semi_I2C_Interface.h>
```

Inheritance diagram for I2C_Interface:



Collaboration diagram for I2C_Interface:



Public Member Functions

- bool [beginI2C](#) (uint8_t addr, TwoWire &wire, uint32_t speed, uint8_t sda=255, uint8_t scl=255)
- int8_t [read](#) (uint8_t reg, uint8_t *data, uint32_t len) override
- bool [beginSPI](#) (uint8_t, SPIClass &, uint32_t, uint8_t, uint8_t, uint8_t) override
- int8_t [write](#) (uint8_t reg, const uint8_t *data, uint32_t len) override

Public Member Functions inherited from [Interface_7Semi](#)

- virtual [~Interface_7Semi](#) ()

Public Attributes

- TwoWire * [i2c](#) = nullptr
- uint8_t [address](#) = 0

Additional Inherited Members

Static Protected Member Functions inherited from [Interface_7Semi](#)

- static void [delay_us](#) (uint32_t us)

5.4.1 Detailed Description

Definition at line 5 of file [7Semi_I2C_Interface.h](#).

5.4.2 Member Function Documentation

5.4.2.1 beginI2C()

```
bool I2C_Interface::beginI2C (
    uint8_t address,
    TwoWire & wire,
    uint32_t speed,
    uint8_t sda = 255,
    uint8_t scl = 255 ) [inline], [virtual]
```

[beginI2C\(\)](#)

- Initializes I2C peripheral
- Configures SDA / SCL pins if supported
- Sets communication clock

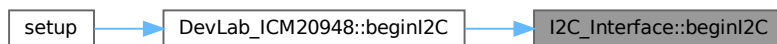
Returns:

- true → Initialization successful
- false → Initialization failed

Implements [Interface_7Semi](#).

Definition at line 11 of file [7Semi_I2C_Interface.h](#).

Here is the caller graph for this function:



5.4.2.2 beginSPI()

```
bool I2C_Interface::beginSPI (
    uint8_t ,
    SPIClass & ,
    uint32_t ,
    uint8_t ,
    uint8_t ,
    uint8_t ) [inline], [override], [virtual]
```

Implements [Interface_7Semi](#).

Definition at line 69 of file [7Semi_I2C_Interface.h](#).

5.4.2.3 read()

```
int8_t I2C_Interface::read (
    uint8_t reg,
    uint8_t * data,
    uint32_t len ) [inline], [override], [virtual]
```

[read\(\)](#)

- Reads data from device register

Parameters:

- reg : Register address

- data : Output buffer
- len : Number of bytes

Returns:

- 0 → Success
- <0 → Error

Implements [Interface_7Semi](#).

Definition at line 32 of file [7Semi_I2C_Interface.h](#).

5.4.2.4 write()

```
int8_t I2C_Interface::write (
    uint8_t reg,
    const uint8_t * data,
    uint32_t len ) [inline], [override], [virtual]
write\(\)
```

- Writes data to device register

Returns:

- 0 → Success
- <0 → Error

Implements [Interface_7Semi](#).

Definition at line 75 of file [7Semi_I2C_Interface.h](#).

5.4.3 Member Data Documentation

5.4.3.1 address

```
uint8_t I2C_Interface::address = 0
```

Definition at line 9 of file [7Semi_I2C_Interface.h](#).

5.4.3.2 i2c

```
TwoWire* I2C_Interface::i2c = nullptr
```

Definition at line 8 of file [7Semi_I2C_Interface.h](#).

The documentation for this class was generated from the following file:

- [src/7Semi_I2C_Interface.h](#)

5.5 ICM20948_IntEnableConfig Struct Reference

```
#include <DevLab_ICM20948.h>
```

Public Attributes

- uint8_t [wofEn](#): 1
- uint8_t [womIntEn](#): 1
- uint8_t [pllRdyEn](#): 1
- uint8_t [dmpInt1En](#): 1
- uint8_t [i2cMstIntEn](#): 1
- uint8_t [rawDataRdyEn](#): 1
- uint8_t [fifoOvfEn](#) [5]
- uint8_t [fifoWmEn](#) [5]

5.5.1 Detailed Description

ICM20948_IntEnableConfig

- Selects which interrupt sources are routed to the INT pin
- Covers INT_ENABLE (0x10), INT_ENABLE_1 (0x11), INT_ENABLE_2 (0x12), INT_ENABLE_3 (0x13)
- FIFO overflow and watermark fields are per-channel arrays [0]–[4]
- Pass to intEnableConfig() to write all four registers in one call

Definition at line 122 of file [DevLab_ICM20948.h](#).

5.5.2 Member Data Documentation

5.5.2.1 dmpInt1En

uint8_t ICM20948_IntEnableConfig::dmpInt1En

Definition at line 128 of file [DevLab_ICM20948.h](#).

5.5.2.2 fifoOvfEn

uint8_t ICM20948_IntEnableConfig::fifoOvfEn[5]

Definition at line 135 of file [DevLab_ICM20948.h](#).

5.5.2.3 fifoWmEn

uint8_t ICM20948_IntEnableConfig::fifoWmEn[5]

Definition at line 138 of file [DevLab_ICM20948.h](#).

5.5.2.4 i2cMstIntEn

uint8_t ICM20948_IntEnableConfig::i2cMstIntEn

Definition at line 129 of file [DevLab_ICM20948.h](#).

5.5.2.5 pllRdyEn

uint8_t ICM20948_IntEnableConfig::pllRdyEn

Definition at line 127 of file [DevLab_ICM20948.h](#).

5.5.2.6 rawDataRdyEn

uint8_t ICM20948_IntEnableConfig::rawDataRdyEn

Definition at line 132 of file [DevLab_ICM20948.h](#).

5.5.2.7 wofEn

uint8_t ICM20948_IntEnableConfig::wofEn

Definition at line 125 of file [DevLab_ICM20948.h](#).

5.5.2.8 womIntEn

uint8_t ICM20948_IntEnableConfig::womIntEn

Definition at line 126 of file [DevLab_ICM20948.h](#).

The documentation for this struct was generated from the following file:

- [src/DevLab_ICM20948.h](#)

5.6 ICM20948_IntPinConfig Struct Reference

```
#include <DevLab_ICM20948.h>
```

Public Attributes

- uint8_t [activeLevel](#): 1
- uint8_t [driveMode](#): 1
- uint8_t [latchMode](#): 1
- uint8_t [clearMode](#): 1
- uint8_t [fsyncActLevel](#): 1
- uint8_t [fsyncIntEn](#): 1
- uint8_t [bypassEn](#): 1

5.6.1 Detailed Description

[ICM20948_IntPinConfig](#)

- Physical configuration of the INT pin (INT_PIN_CFG register, BANK 0)
- Controls electrical behavior and clear condition of the interrupt line
- Pass to `intInit()` to apply settings

Definition at line 103 of file [DevLab_ICM20948.h](#).

5.6.2 Member Data Documentation

5.6.2.1 [activeLevel](#)

uint8_t ICM20948_IntPinConfig::activeLevel
Definition at line 105 of file [DevLab_ICM20948.h](#).

5.6.2.2 [bypassEn](#)

uint8_t ICM20948_IntPinConfig::bypassEn
Definition at line 111 of file [DevLab_ICM20948.h](#).

5.6.2.3 [clearMode](#)

uint8_t ICM20948_IntPinConfig::clearMode
Definition at line 108 of file [DevLab_ICM20948.h](#).

5.6.2.4 [driveMode](#)

uint8_t ICM20948_IntPinConfig::driveMode
Definition at line 106 of file [DevLab_ICM20948.h](#).

5.6.2.5 [fsyncActLevel](#)

uint8_t ICM20948_IntPinConfig::fsyncActLevel
Definition at line 109 of file [DevLab_ICM20948.h](#).

5.6.2.6 [fsyncIntEn](#)

uint8_t ICM20948_IntPinConfig::fsyncIntEn
Definition at line 110 of file [DevLab_ICM20948.h](#).

5.6.2.7 [latchMode](#)

uint8_t ICM20948_IntPinConfig::latchMode
Definition at line 107 of file [DevLab_ICM20948.h](#).

The documentation for this struct was generated from the following file:

- [src/DevLab_ICM20948.h](#)

5.7 ICM20948_IntStatus Struct Reference

```
#include <DevLab_ICM20948.h>
```

Public Attributes

- uint8_t [womInt](#): 1
- uint8_t [pllRdyInt](#): 1
- uint8_t [dmpInt1](#): 1
- uint8_t [i2cMstInt](#): 1
- uint8_t [rawDataRdy](#): 1
- uint8_t [fifoOvf](#) [5]
- uint8_t [fifoWm](#) [5]

5.7.1 Detailed Description

[ICM20948_IntStatus](#)

- Holds the parsed state of the four interrupt status registers
- Covers INT_STATUS (0x19), INT_STATUS_1 (0x1A), INT_STATUS_2 (0x1B), INT_STATUS_3 (0x1C)
- Populated by [checkIntStatus\(\)](#) via a single 4-byte burst read
- Reading these registers clears the interrupt flags (when `clearMode = 0` in [IntPinConfig](#))

Definition at line 149 of file [DevLab_ICM20948.h](#).

5.7.2 Member Data Documentation

5.7.2.1 [dmpInt1](#)

uint8_t ICM20948_IntStatus::dmpInt1

Definition at line 154 of file [DevLab_ICM20948.h](#).

5.7.2.2 [fifoOvf](#)

uint8_t ICM20948_IntStatus::fifoOvf[5]

Definition at line 161 of file [DevLab_ICM20948.h](#).

5.7.2.3 [fifoWm](#)

uint8_t ICM20948_IntStatus::fifoWm[5]

Definition at line 164 of file [DevLab_ICM20948.h](#).

5.7.2.4 [i2cMstInt](#)

uint8_t ICM20948_IntStatus::i2cMstInt

Definition at line 155 of file [DevLab_ICM20948.h](#).

5.7.2.5 [pllRdyInt](#)

uint8_t ICM20948_IntStatus::pllRdyInt

Definition at line 153 of file [DevLab_ICM20948.h](#).

5.7.2.6 [rawDataRdy](#)

uint8_t ICM20948_IntStatus::rawDataRdy

Definition at line 158 of file [DevLab_ICM20948.h](#).

5.7.2.7 womInt

uint8_t ICM20948_IntStatus::womInt

Definition at line 152 of file [DevLab_ICM20948.h](#).

The documentation for this struct was generated from the following file:

- [src/DevLab_ICM20948.h](#)

5.8 icm_plat_t Struct Reference

```
#include <ICM20948_platform.h>
```

Public Attributes

- int(* [read](#))(uint8_t reg, uint8_t *buf, size_t len, void *user)
- int(* [write](#))(uint8_t reg, const uint8_t *buf, size_t len, void *user)
- uint8_t(* [spi_reg_rw_bits](#))(uint8_t reg, int is_write)
- void(* [delay_ms](#))(uint32_t ms)
- void * [user](#)
- void(* [spi_cs](#))(int level, void *user)
- uint8_t [i2c_addr](#)

5.8.1 Detailed Description

7Semi comment style

- Platform glue for bus IO, delays, and timing
- Implement these for your system (I2C or SPI)

Definition at line 13 of file [ICM20948_platform.h](#).

5.8.2 Member Data Documentation

5.8.2.1 delay_ms

void(* [icm_plat_t::delay_ms](#))(uint32_t ms)

- Millisecond delay

Definition at line 32 of file [ICM20948_platform.h](#).

5.8.2.2 i2c_addr

uint8_t [icm_plat_t::i2c_addr](#)

- Device address (I2C) or ignored for SPI if you wrap inside user

Definition at line 45 of file [ICM20948_platform.h](#).

5.8.2.3 read

int(* [icm_plat_t::read](#))(uint8_t reg, uint8_t *buf, size_t len, void *user)

- Bus read: read len bytes from reg into buf
- Return 0 on success, nonzero on error

Definition at line 18 of file [ICM20948_platform.h](#).

5.8.2.4 spi_cs

```
void(* icm_plat_t::spi_cs) (int level, void *user)
```

- If using SPI: set chip select active (1) or inactive (0)
- For I2C: leave as NULL

Definition at line 41 of file [ICM20948_platform.h](#).

5.8.2.5 spi_reg_rw_bits

```
uint8_t(* icm_plat_t::spi_reg_rw_bits) (uint8_t reg, int is_write)
```

- Optional: SPI transfers may need register R/W bit mangling
- For I2C, leave as NULL

Definition at line 28 of file [ICM20948_platform.h](#).

5.8.2.6 user

```
void* icm_plat_t::user
```

- User context pointer passed to read/write

Definition at line 36 of file [ICM20948_platform.h](#).

5.8.2.7 write

```
int(* icm_plat_t::write) (uint8_t reg, const uint8_t *buf, size_t len, void *user)
```

- Bus write: write len bytes from buf to reg
- Return 0 on success, nonzero on error

Definition at line 23 of file [ICM20948_platform.h](#).

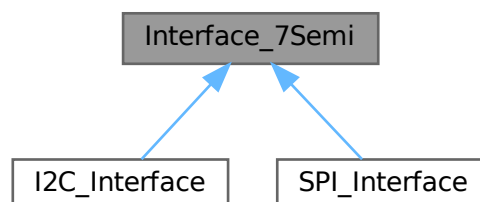
The documentation for this struct was generated from the following file:

- src/[ICM20948_platform.h](#)

5.9 Interface_7Semi Class Reference

```
#include <7Semi_Interface.h>
```

Inheritance diagram for Interface_7Semi:



Public Member Functions

- virtual [~Interface_7Semi](#) ()
- virtual bool [beginI2C](#) (uint8_t address, TwoWire &wire, uint32_t speed, uint8_t sda=255, uint8_t scl=255)=0
- virtual bool [beginSPI](#) (uint8_t cs_pin, SPIClass &wire, uint32_t speed, uint8_t csk=255, uint8_t miso=255, uint8_t mosi=255)=0
- virtual int8_t [read](#) (uint8_t reg, uint8_t *data, uint32_t len)=0
- virtual int8_t [write](#) (uint8_t reg, const uint8_t *data, uint32_t len)=0

Static Protected Member Functions

- static void [delay_us](#) (uint32_t us)

5.9.1 Detailed Description

7Semi Universal Interface Layer

- Abstract communication layer for I2C / SPI
- Ensures sensor drivers remain hardware independent

Definition at line 15 of file [7Semi_Interface.h](#).

5.9.2 Constructor & Destructor Documentation**5.9.2.1 ~Interface_7Semi()**

```
virtual Interface_7Semi::~~Interface_7Semi ( ) [inline], [virtual]
```

Definition at line 18 of file [7Semi_Interface.h](#).

5.9.3 Member Function Documentation**5.9.3.1 beginI2C()**

```
virtual bool Interface_7Semi::beginI2C (
    uint8_t address,
    TwoWire & wire,
    uint32_t speed,
    uint8_t sda = 255,
    uint8_t scl = 255 ) [pure virtual]
```

[beginI2C\(\)](#)

- Initializes I2C peripheral
- Configures SDA / SCL pins if supported
- Sets communication clock

Returns:

- true → Initialization successful
- false → Initialization failed

Implemented in [I2C_Interface](#), and [SPI_Interface](#).

5.9.3.2 beginSPI()

```
virtual bool Interface_7Semi::beginSPI (
    uint8_t cs_pin,
    SPIClass & wire,
    uint32_t speed,
    uint8_t csk = 255,
    uint8_t miso = 255,
    uint8_t mosi = 255 ) [pure virtual]
```

Implemented in [SPI_Interface](#), and [I2C_Interface](#).

5.9.3.3 delay_us()

```
static void Interface_7Semi::delay_us (
    uint32_t us ) [inline], [static], [protected]
```

Delay wrapper

Definition at line 82 of file [7Semi_Interface.h](#).

5.9.3.4 read()

```
virtual int8_t Interface_7Semi::read (
    uint8_t reg,
    uint8_t * data,
    uint32_t len ) [pure virtual]
```

[read\(\)](#)

- Reads data from device register

Parameters:

- reg : Register address
- data : Output buffer
- len : Number of bytes

Returns:

- 0 → Success
- <0 → Error

Implemented in [I2C_Interface](#), and [SPI_Interface](#).

5.9.3.5 write()

```
virtual int8_t Interface_7Semi::write (
    uint8_t reg,
    const uint8_t * data,
    uint32_t len ) [pure virtual]
```

[write\(\)](#)

- Writes data to device register

Returns:

- 0 → Success
- <0 → Error

Implemented in [I2C_Interface](#), and [SPI_Interface](#).

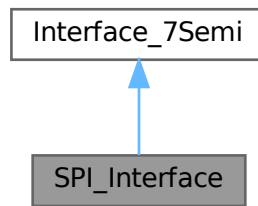
The documentation for this class was generated from the following file:

- src/[7Semi_Interface.h](#)

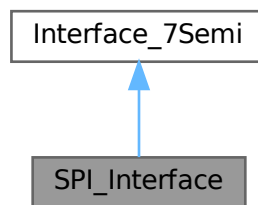
5.10 SPI_Interface Class Reference

```
#include <7Semi_SPI_Interface.h>
```

Inheritance diagram for SPI_Interface:



Collaboration diagram for SPI_Interface:



Public Member Functions

- bool [beginSPI](#) (uint8_t csPin, SPIClass &spiPort, uint32_t spiSpeed, uint8_t sck=255, uint8_t miso=255, uint8_t mosi=255) override
- bool [beginI2C](#) (uint8_t, TwoWire &, uint32_t, uint8_t, uint8_t) override
- int8_t [read](#) (uint8_t reg, uint8_t *data, uint32_t len) override
- int8_t [write](#) (uint8_t reg, const uint8_t *data, uint32_t len) override

Public Member Functions inherited from [Interface_7Semi](#)

- virtual [~Interface_7Semi](#) ()

Public Attributes

- SPIClass * [spi](#) = nullptr
- uint8_t [cs](#) = 255
- uint32_t [speed](#) = 1000000

Additional Inherited Members

Static Protected Member Functions inherited from [Interface_7Semi](#)

- static void [delay_us](#) (uint32_t us)

5.10.1 Detailed Description

7Semi SPI Interface Implementation

Definition at line 7 of file [7Semi_SPI_Interface.h](#).

5.10.2 Member Function Documentation

5.10.2.1 beginI2C()

```
bool SPI_Interface::beginI2C (
    uint8_t address,
    TwoWire & wire,
    uint32_t speed,
    uint8_t sda,
    uint8_t scl ) [inline], [override], [virtual]
```

[beginI2C\(\)](#)

- Initializes I2C peripheral
- Configures SDA / SCL pins if supported
- Sets communication clock

Returns:

- true → Initialization successful
- false → Initialization failed

Implements [Interface_7Semi](#).

Definition at line 51 of file [7Semi_SPI_Interface.h](#).

5.10.2.2 beginSPI()

```
bool SPI_Interface::beginSPI (
    uint8_t csPin,
    SPIClass & spiPort,
    uint32_t spiSpeed,
    uint8_t sck = 255,
    uint8_t miso = 255,
    uint8_t mosi = 255 ) [inline], [override], [virtual]
```

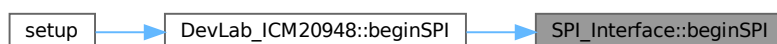
[beginSPI\(\)](#)

- Initialize SPI interface
- Configure CS and SPI bus

Implements [Interface_7Semi](#).

Definition at line 21 of file [7Semi_SPI_Interface.h](#).

Here is the caller graph for this function:



5.10.2.3 read()

```
int8_t SPI_Interface::read (
    uint8_t reg,
    uint8_t * data,
    uint32_t len ) [inline], [override], [virtual]
```

[read\(\)](#)

- Read multiple bytes from register

Send register address (read)

Read data

Implements [Interface_7Semi](#).

Definition at line 62 of file [7Semi_SPI_Interface.h](#).

5.10.2.4 write()

```
int8_t SPI_Interface::write (
    uint8_t reg,
    const uint8_t * data,
    uint32_t len ) [inline], [override], [virtual]
```

[write\(\)](#)

- Write multiple bytes to register

Send register address (write)

Write data

Implements [Interface_7Semi](#).

Definition at line 108 of file [7Semi_SPI_Interface.h](#).

5.10.3 Member Data Documentation

5.10.3.1 cs

```
uint8_t SPI_Interface::cs = 255
```

Definition at line 12 of file [7Semi_SPI_Interface.h](#).

5.10.3.2 speed

```
uint32_t SPI_Interface::speed = 1000000
```

Definition at line 13 of file [7Semi_SPI_Interface.h](#).

5.10.3.3 spi

```
SPIClass* SPI_Interface::spi = nullptr
```

Definition at line 11 of file [7Semi_SPI_Interface.h](#).

The documentation for this class was generated from the following file:

- [src/7Semi_SPI_Interface.h](#)

Chapter 6

File Documentation

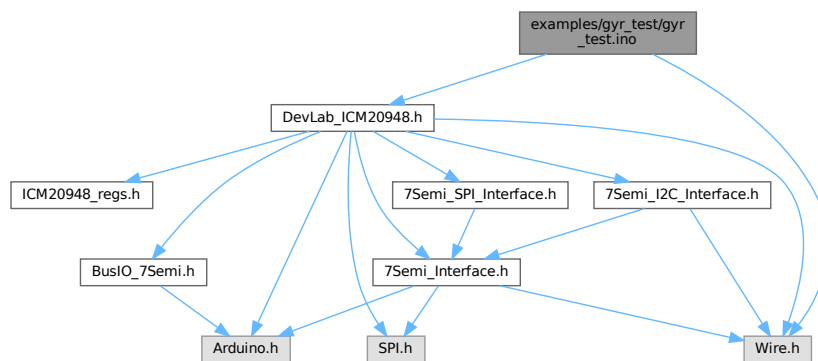
6.1 examples/gyr_test/gyr_test.ino File Reference

Gyroscope DLPF and full-scale sweep for the DevLab ICM-20948.

```
#include <Wire.h>
```

```
#include <DevLab_ICM20948.h>
```

Include dependency graph for gyr_test.ino:



Classes

- struct `configDLPF`
Gyroscope DLPF divider and timing configuration.

Macros

- #define `SDA_PIN` 6
I2C SDA pin used by the example board.
- #define `SCL_PIN` 7
I2C SCL pin used by the example board.
- #define `I2C_FREQ` 400000UL
I2C bus speed in hertz.
- #define `ICM_ADDR` 0x69
ICM-20948 I2C address selected by the AD0 pin.

Functions

- void `printLinea()`

- *Print a separator line to Serial.*
- void [printDobleLinea \(\)](#)
Print a double separator line to Serial.
- void [firstStage \(\)](#)
Verify the IMU identity register.
- bool [applySettings](#) (uint8_t fs_idx, const [configDLPF](#) &cfg)
Apply one gyroscope full-scale and DLPF configuration.
- void [runSweep \(\)](#)
Run the gyroscope validation sweep and print each captured sample.
- void [setup \(\)](#)
Initialize I2C, reset/wake the IMU, enable gyro, and start the sweep.
- void [loop \(\)](#)
Empty loop; this sketch runs the gyroscope sweep once in [setup\(\)](#).

Variables

- const [configDLPF](#) [configsDLPF](#) []
Number of gyroscope DLPF configurations.
- const uint8_t [N_DLPF](#) = sizeof([configsDLPF](#)) / sizeof([configsDLPF](#)[0])
Number of gyroscope DLPF configurations.
- const [ICM20948_Gyro_FullScale](#) [gyroFSCfg](#) []
Gyroscope full-scale ranges exercised by the sweep.
- const char * [etiquetasGyroFSCfg](#) [] = {"250", "500", "1000", "2000"}
Text labels matching gyroFSCfg.
- const uint8_t [N_FS](#) = 4
Number of gyroscope full-scale ranges.
- const [ICM20948_Gyro_DLPF](#) [gyroDLPF](#) []
Gyroscope DLPF enum values matching configsDLPF.
- const [ICM20948_Gyro_Average](#) [gyroAVGCfg](#) []
Gyroscope averaging settings available for experiments.
- const char * [etiquetasGyroAVGCfg](#) [] = {"1", "2", "4", "8", "16", "32", "64", "128"}
Text labels matching gyroAVGCfg.
- const uint8_t [N_AVG](#) = 8
Number of gyroscope averaging settings.
- [DevLab_ICM20948](#) [imu](#)
Global ICM-20948 driver instance.

6.1.1 Detailed Description

Gyroscope DLPF and full-scale sweep for the DevLab ICM-20948.

Author

Jonathan Mejorado Lopez

Applies each gyroscope filter/full-scale configuration over I2C and prints repeated samples for validation and noise comparison.

Definition in file [gyr_test.ino](#).

6.1.2 Macro Definition Documentation

6.1.2.1 I2C_FREQ

```
#define I2C_FREQ 400000UL
```

I2C bus speed in hertz.

Definition at line 19 of file [gyr_test.ino](#).

6.1.2.2 ICM_ADDR

```
#define ICM_ADDR 0x69
```

ICM-20948 I2C address selected by the AD0 pin.

Definition at line 21 of file [gyr_test.ino](#).

6.1.2.3 SCL_PIN

```
#define SCL_PIN 7
```

I2C SCL pin used by the example board.

Definition at line 17 of file [gyr_test.ino](#).

6.1.2.4 SDA_PIN

```
#define SDA_PIN 6
```

I2C SDA pin used by the example board.

Definition at line 15 of file [gyr_test.ino](#).

6.1.3 Function Documentation

6.1.3.1 applySettings()

```
bool applySettings (
    uint8_t fs_idx,
    const configDLPF & cfg )
```

Apply one gyroscope full-scale and DLPF configuration.

Parameters

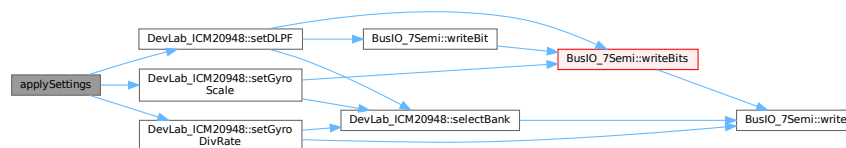
<i>fs_idx</i>	Index into gyroFSCfg.
<i>cfg</i>	DLPF timing and divider configuration.

Returns

true if all configuration writes succeeded, otherwise false.

Definition at line 124 of file [gyr_test.ino](#).

Here is the call graph for this function:



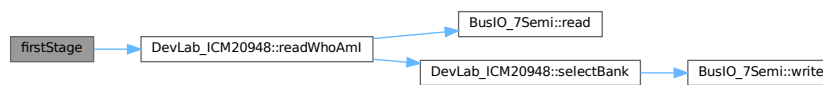
6.1.3.2 firstStage()

```
void firstStage ( )
```

Verify the IMU identity register.

Definition at line 108 of file [gyr_test.ino](#).

Here is the call graph for this function:



Here is the caller graph for this function:



6.1.3.3 loop()

```
void loop ( )
```

Empty loop; this sketch runs the gyroscope sweep once in [setup\(\)](#).

Definition at line [239](#) of file [gyr_test.ino](#).

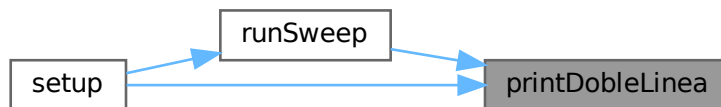
6.1.3.4 printDobleLinea()

```
void printDobleLinea ( )
```

Print a double separator line to Serial.

Definition at line [103](#) of file [gyr_test.ino](#).

Here is the caller graph for this function:



6.1.3.5 printLinea()

```
void printLinea ( )
```

Print a separator line to Serial.

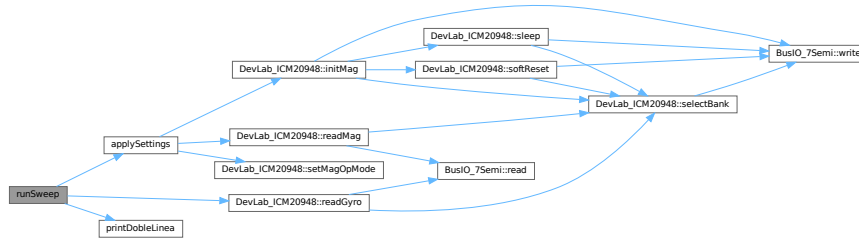
Definition at line [101](#) of file [gyr_test.ino](#).

6.1.3.6 runSweep()

```
void runSweep ( )
```

Run the gyroscope validation sweep and print each captured sample.

Definition at line 149 of file [gyr_test.ino](#).
Here is the call graph for this function:



Here is the caller graph for this function:



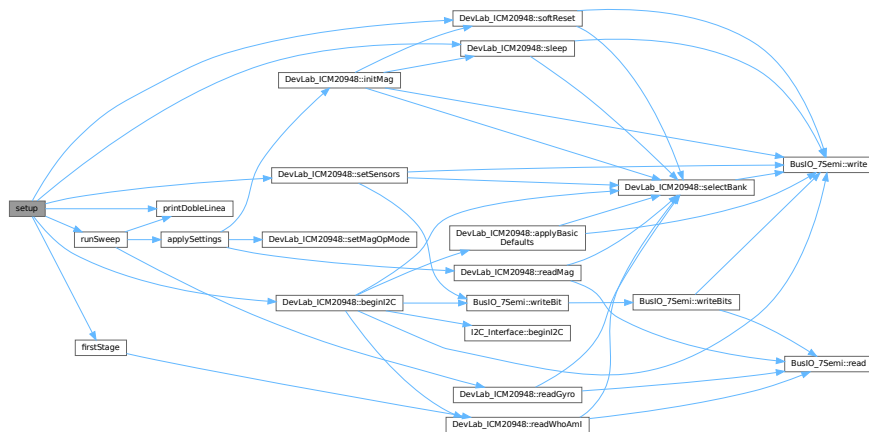
6.1.3.7 setup()

```
void setup ( )
```

Initialize I2C, reset/wake the IMU, enable gyro, and start the sweep.

Definition at line 194 of file [gyr_test.ino](#).

Here is the call graph for this function:



6.1.4 Variable Documentation

6.1.4.1 configsDLPF

```
const configDLPF configsDLPF[ ]
```

Initial value:

```
= {
```

```
{ 0, 0, 229.8f, 1125.0f, "196,6 ; 229,8 ; 1125.0 " },
{ 1, 0, 187.6f, 1125.0f, "151,8 ; 187.6 ; 1125.0 " },
{ 2, 0, 154.3f, 1125.0f, "119,5 ; 154,3 ; 1125.0 " },
{ 3, 2, 73.3f, 375.0f, "51,2 ; 73,3 ; 375 " },
{ 4, 5, 35.9f, 187.5f, "23,9 ; 35,9 ; 187.5 " },
{ 5, 14, 17.8f, 75.0f, "11,6 ; 17,8 ; 75.0 " },
{ 6, 29, 8.9f, 37.5f, "5,7 ; 8,9 ; 37.5 " },
{ 7, 0, 376.5f, 1125.0f, "361,4 ; 376,5 ; 1125 " },
}
```

Definition at line 42 of file [gyr_test.ino](#).

6.1.4.2 etiquetasGyroAVGCfg

```
const char* etiquetasGyroAVGCfg[ ] = {"1","2","4","8","16","32","64","128"}
```

Text labels matching gyroAVGCfg.

Definition at line 93 of file [gyr_test.ino](#).

6.1.4.3 etiquetasGyroFSCfg

```
const char* etiquetasGyroFSCfg[ ] = {"250", "500", "1000", "2000"}
```

Text labels matching gyroFSCfg.

Definition at line 64 of file [gyr_test.ino](#).

6.1.4.4 gyroAVGCfg

```
const ICM20948_Gyro_Average gyroAVGCfg[ ]
```

Initial value:

```
= {
    ICM20948_Gyro_Average::AVG_1,
    ICM20948_Gyro_Average::AVG_2,
    ICM20948_Gyro_Average::AVG_4,
    ICM20948_Gyro_Average::AVG_8,
    ICM20948_Gyro_Average::AVG_16,
    ICM20948_Gyro_Average::AVG_32,
    ICM20948_Gyro_Average::AVG_64,
    ICM20948_Gyro_Average::AVG_128
}
```

Gyroscope averaging settings available for experiments.

Definition at line 81 of file [gyr_test.ino](#).

6.1.4.5 gyroDLPF

```
const ICM20948_Gyro_DLPF gyroDLPF[ ]
```

Initial value:

```
= {
    ICM20948_Gyro_DLPF::DLPF_196HZ,
    ICM20948_Gyro_DLPF::DLPF_151HZ,
    ICM20948_Gyro_DLPF::DLPF_119HZ,
    ICM20948_Gyro_DLPF::DLPF_51HZ,
    ICM20948_Gyro_DLPF::DLPF_23HZ,
    ICM20948_Gyro_DLPF::DLPF_11HZ,
    ICM20948_Gyro_DLPF::DLPF_5HZ,
    ICM20948_Gyro_DLPF::DLPF_361HZ
}
```

Gyroscope DLPF enum values matching configsDLPF.

Definition at line 69 of file [gyr_test.ino](#).

6.1.4.6 gyroFSCfg

```
const ICM20948_Gyro_FullScale gyroFSCfg[ ]
```

Initial value:

```
= {
    ICM20948_Gyro_FullScale::DPS_250,
    ICM20948_Gyro_FullScale::DPS_500,
    ICM20948_Gyro_FullScale::DPS_1000,
    ICM20948_Gyro_FullScale::DPS_2000
}
```

Gyroscope full-scale ranges exercised by the sweep.

Definition at line 56 of file [gyr_test.ino](#).

6.1.4.7 imu

[DevLab_ICM20948](#) imu

Global ICM-20948 driver instance.

Definition at line 98 of file [gyr_test.ino](#).

6.1.4.8 N_AVG

const uint8_t N_AVG = 8

Number of gyroscope averaging settings.

Definition at line 95 of file [gyr_test.ino](#).

6.1.4.9 N_DLPF

const uint8_t N_DLPF = sizeof(configsDLPF) / sizeof(configsDLPF[0])

Number of gyroscope DLPF configurations.

Definition at line 53 of file [gyr_test.ino](#).

6.1.4.10 N_FS

const uint8_t N_FS = 4

Number of gyroscope full-scale ranges.

Definition at line 66 of file [gyr_test.ino](#).

6.2 gyr_test.ino

[Go to the documentation of this file.](#)

```
00001 /**
00002  * @file gyr_test.ino
00003  * @brief Gyroscope DLPF and full-scale sweep for the DevLab ICM-20948.
00004  * @author Jonathan Mejorado Lopez
00005  * @details Applies each gyroscope filter/full-scale configuration over I2C and
00006  * @prints repeated samples for validation and noise comparison.
00007  */
00008
00009 #include <Wire.h>
00010 #include <DevLab_ICM20948.h>
00011 // -----
00012 // PINES I2C – ESP32C6 NANO Unit Electronics
00013 // -----
00014 /** @brief I2C SDA pin used by the example board. */
00015 #define SDA_PIN 6
00016 /** @brief I2C SCL pin used by the example board. */
00017 #define SCL_PIN 7
00018 /** @brief I2C bus speed in hertz. */
00019 #define I2C_FREQ 400000UL
00020 /** @brief ICM-20948 I2C address selected by the AD0 pin. */
00021 #define ICM_ADDR 0x69
00022
00023 // -----
00024 // STRUCT
00025 // -----
00026 /** @brief Gyroscope DLPF divider and timing configuration. */
00027 struct configDLPF {
00028     /** @brief Index into the gyroDLPF lookup table. */
00029     uint8_t dlpf_idx;
00030     /** @brief Sample-rate divider written to the IMU register. */
00031     uint16_t div;
00032     /** @brief Noise bandwidth in hertz for reporting. */
00033     float nbw_hz;
00034     /** @brief Output data rate in hertz for timing calculations. */
00035     float odr_hz;
00036     /** @brief Text label printed with each captured sample. */
00037     const char* nombre;
00038 };
00039 //id,div,nbw_hz,odr_hz,nombre : {DLPF, NBW , ODR}
00040 /*
00041  * @brief Gyroscope DLPF configurations exercised by the sweep. */
00042 const configDLPF configsDLPF[] = {
00043     { 0, 0, 229.8f, 1125.0f, "196,6 ; 229,8 ; 1125.0 " },
00044     { 1, 0, 187.6f, 1125.0f, "151,8 ; 187.6 ; 1125.0 " },
00045     { 2, 0, 154.3f, 1125.0f, "119,5 ; 154,3 ; 1125.0 " },
00046     { 3, 2, 73.3f, 375.0f, "51,2 ; 73,3 ; 375 " },
00047     { 4, 5, 35.9f, 187.5f, "23,9 ; 35,9 ; 187.5 " },
00048     { 5, 14, 17.8f, 75.0f, "11,6 ; 17,8 ; 75.0 " },
00049 }
```

```

00049 { 6, 29, 8.9f, 37.5f, "5,7 ; 8,9 ; 37.5 " },
00050 { 7, 0, 376.5f, 1125.0f, "361,4 ; 376,5 ; 1125 " },
00051 };
00052 /** @brief Number of gyroscope DLPF configurations. */
00053 const uint8_t N_DLPF = sizeof(configsDLPF) / sizeof(configsDLPF[0]);
00054
00055 /** @brief Gyroscope full-scale ranges exercised by the sweep. */
00056 const ICM20948_Gyro_FullScale gyroFSCfg[] = {
00057     ICM20948_Gyro_FullScale::DPS_250,
00058     ICM20948_Gyro_FullScale::DPS_500,
00059     ICM20948_Gyro_FullScale::DPS_1000,
00060     ICM20948_Gyro_FullScale::DPS_2000
00061 };
00062
00063 /** @brief Text labels matching gyroFSCfg. */
00064 const char* etiquetasGyroFSCfg[] = {"250", "500", "1000", "2000"};
00065 /** @brief Number of gyroscope full-scale ranges. */
00066 const uint8_t N_FS = 4;
00067
00068 /** @brief Gyroscope DLPF enum values matching configsDLPF. */
00069 const ICM20948_Gyro_DLPF gyroDLPF[] = {
00070     ICM20948_Gyro_DLPF::DLPF_196HZ,
00071     ICM20948_Gyro_DLPF::DLPF_151HZ,
00072     ICM20948_Gyro_DLPF::DLPF_119HZ,
00073     ICM20948_Gyro_DLPF::DLPF_51HZ,
00074     ICM20948_Gyro_DLPF::DLPF_23HZ,
00075     ICM20948_Gyro_DLPF::DLPF_11HZ,
00076     ICM20948_Gyro_DLPF::DLPF_5HZ,
00077     ICM20948_Gyro_DLPF::DLPF_361HZ
00078 };
00079
00080 /** @brief Gyroscope averaging settings available for experiments. */
00081 const ICM20948_Gyro_Average gyroAVGCfg[] = {
00082     ICM20948_Gyro_Average::AVG_1,
00083     ICM20948_Gyro_Average::AVG_2,
00084     ICM20948_Gyro_Average::AVG_4,
00085     ICM20948_Gyro_Average::AVG_8,
00086     ICM20948_Gyro_Average::AVG_16,
00087     ICM20948_Gyro_Average::AVG_32,
00088     ICM20948_Gyro_Average::AVG_64,
00089     ICM20948_Gyro_Average::AVG_128
00090 };
00091
00092 /** @brief Text labels matching gyroAVGCfg. */
00093 const char* etiquetasGyroAVGCfg[] = {"1", "2", "4", "8", "16", "32", "64", "128"};
00094 /** @brief Number of gyroscope averaging settings. */
00095 const uint8_t N_AVG = 8;
00096
00097 /** @brief Global ICM-20948 driver instance. */
00098 DevLab_ICM20948 imu;
00099
00100 /** @brief Print a separator line to Serial. */
00101 void printLinea() {
00102     Serial.println(F("-----"));
00103 }
00104 /** @brief Print a double separator line to Serial. */
00105 void printDobleLinea() {
00106     Serial.println(F("====="));
00107 }
00108
00109 /**
00110  * @brief Verify the IMU identity register.
00111  */
00112 void firstStage() {
00113     uint8_t who;
00114     if (!imu.readWhoAmI(who) || who != 0xEA) {
00115         Serial.println(F("ERROR: WHO_AM_I mismatch"));
00116         while (1) delay(200);
00117     }
00118     Serial.printf("WHO_AM_I = 0x%02X OK\n", who);
00119 }
00120
00121 /**
00122  * @brief Apply one gyroscope full-scale and DLPF configuration.
00123  *
00124  * @param fs_idx Index into gyroFSCfg.
00125  * @param cfg DLPF timing and divider configuration.
00126  * @return true if all configuration writes succeeded, otherwise false.
00127  */
00128 bool applySettings(uint8_t fs_idx, const configDLPF &cfg) {
00129     ICM20948_Gyro_DLPF dlpf_val = gyroDLPF[cfg.dlpf_idx];
00130     bool ok = true;
00131
00132     ok &= imu.setGyroScale(gyroFSCfg[fs_idx]);
00133
00134     // Paso 1: escribe DLPFCFG en bits [5:3]
00135     // SetDLPF false -> Activa DLPF
00136     // SetDLPF true -> Desactiva DLPF

```

```

00134   ok &= imu.setDLPF(dlpf_val, false);
00135
00136   //ok &= imu.setGyroAveraging(gyroAVGCfg[avg]);
00137   // Paso 2: activa FCHOICE=1 (DLPF activo) sin tocar DLPFCFG
00138   //ok &= imu.setAcceLDLPF(0, true);
00139
00140   // DIV directo al registro, sin conversion interna
00141   ok &= imu.setGyroDivRate(cfg.div);
00142
00143   return ok;
00144 }
00145
00146 /**
00147  * @brief Run the gyroscope validation sweep and print each captured sample.
00148  */
00149 void runSweep(){
00150     uint16_t n = 0;
00151     printDobleLinea();
00152     Serial.println(F("   ICM-20948 | UNIT Electronics"));
00153     Serial.println(F("   MODO VERIFICACION - FS=+-2g fijo, 3 lecturas por config"));
00154     Serial.println(F("   Sensor PLANO y QUIETO | Az esperado: ~+1.000g"));
00155     printDobleLinea();
00156     for (uint8_t i = 0; i < N_FS ; i++){
00157         for(uint8_t d = 0; d < N_DLPF; d++){
00158
00159             bool cfg_ok = applySettings(i,configsDLPF[d]);
00160             if(!cfg_ok){
00161                 Serial.println(F("ERROR: applySettings fallo"));
00162             }
00163
00164             // Esperar que el filtro estabilice (10 periodos del ODR, min 50ms)
00165             uint32_t settle_ms = max((uint32_t)(10000.0f / configsDLPF[d].odr_hz),
00166                                     (uint32_t)50);
00167             delay(settle_ms);
00168             // Tomar 3 lecturas espaciadas por el periodo del ODR
00169             uint32_t periodo_ms = max((uint32_t)(1000.0f / configsDLPF[d].odr_hz),
00170                                     (uint32_t)1);
00171             float gx, gy, gz;
00172             for (uint8_t r = 0; r < 100; r++) {
00173                 n++;
00174                 delay(periodo_ms);
00175                 if (imu.readGyro(gx, gy, gz)) {
00176                     Serial.printf("%d ; %s ; %s ; %.4f ; %.4f ; %.4f\n",
00177                                   n,
00178                                   configsDLPF[d].nombre,    // "246 ; 265 ; 1125"
00179                                   etiquetasGyroFSCfg[i],        // "+-2G"
00180                                   //etiquetasGyroAVGCfg[j],      // "8"
00181                                   gx, gy, gz);
00182                 } else {
00183                     Serial.println(F("  [?] readGyr() fallo"));
00184                 }
00185             }
00186         }
00187     }
00188 }
00189 }
00190
00191 /**
00192  * @brief Initialize I2C, reset/wake the IMU, enable gyro, and start the sweep.
00193  */
00194 void setup() {
00195     // put your setup code here, to run once:
00196     Serial.begin(115200);
00197     delay(200);
00198
00199     printDobleLinea();
00200     Serial.println(F("   ICM-20948 VALIDACION ACELEROMETRO"));
00201     printDobleLinea();
00202
00203     Wire.begin(SDA_PIN, SCL_PIN);
00204
00205     if (!imu.beginI2C(ICM_ADDR, Wire, I2C_FREQ)) {
00206         Serial.println(F("ERROR: beginI2C() fallo"));
00207         while (1) delay(200);
00208     }
00209
00210     firstStage();
00211
00212     // softReset pone el chip a dormir
00213     // Despues del reset hay que despertar el dispositivo
00214     imu.softReset();
00215     delay(100);
00216     imu.sleep(false);    // limpia SLEEP bit, CLKSEL=1
00217     delay(50);
00218
00219     if (!imu.setSensors(false, true, false)) {
00220         Serial.println(F("ERROR: setSensors() fallo"));
00221     }

```

```

00221 }
00222
00223 Serial.println(F(" Sensor listo.\n"));
00224 Serial.println(F(" Sensor PLANO y QUIETO sobre la mesa."));
00225 Serial.println(F(" Iniciando en 5 segundos..."));
00226 for (int i = 5; i > 0; i--) {
00227     Serial.printf(" %d...\n", i);
00228     delay(1000);
00229 }
00230
00231 //for (int runs = 0; runs < 20 ; runs++){
00232     runSweep();
00233 //}
00234 }
00235
00236 /**
00237  * @brief Empty loop; this sketch runs the gyroscope sweep once in setup().
00238  */
00239 void loop() {
00240     // put your main code here, to run repeatedly:
00241
00242 }

```

6.3 examples/i2c_accel/i2c_accel.ino File Reference

```

#include <Wire.h>
#include <DevLab_ICM20948.h>

```

Macros

- `#define SDA_PIN 6`
I2C SDA pin used by the example board.
- `#define SCL_PIN 7`
I2C SCL pin used by the example board.
- `#define I2C_FREQ 400000UL`
I2C bus speed in hertz.
- `#define ICM_ADDR 0x69`
ICM-20948 I2C address selected by the AD0 pin.

Functions

- `void setup ()`
Configure the IMU for accelerometer-only operation over I2C.
- `void loop ()`
Read and print accelerometer data in g.

Variables

- `DevLab_ICM20948 imu`
Global ICM-20948 driver instance.

6.3.1 Macro Definition Documentation

6.3.1.1 I2C_FREQ

```

#define I2C_FREQ 400000UL
I2C bus speed in hertz.
Definition at line 41 of file i2c_accel.ino.

```

6.3.1.2 ICM_ADDR

```

#define ICM_ADDR 0x69
ICM-20948 I2C address selected by the AD0 pin.
Definition at line 43 of file i2c_accel.ino.

```

6.3.1.3 SCL_PIN

```
#define SCL_PIN 7
```

I2C SCL pin used by the example board.

Definition at line 39 of file [i2c_accel.ino](#).

6.3.1.4 SDA_PIN

```
#define SDA_PIN 6
```

I2C SDA pin used by the example board.

Definition at line 37 of file [i2c_accel.ino](#).

6.3.2 Function Documentation

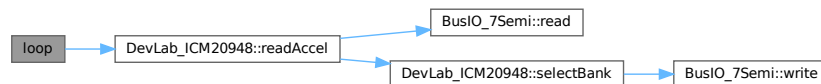
6.3.2.1 loop()

```
void loop ( )
```

Read and print accelerometer data in g.

Definition at line 122 of file [i2c_accel.ino](#).

Here is the call graph for this function:



6.3.2.2 setup()

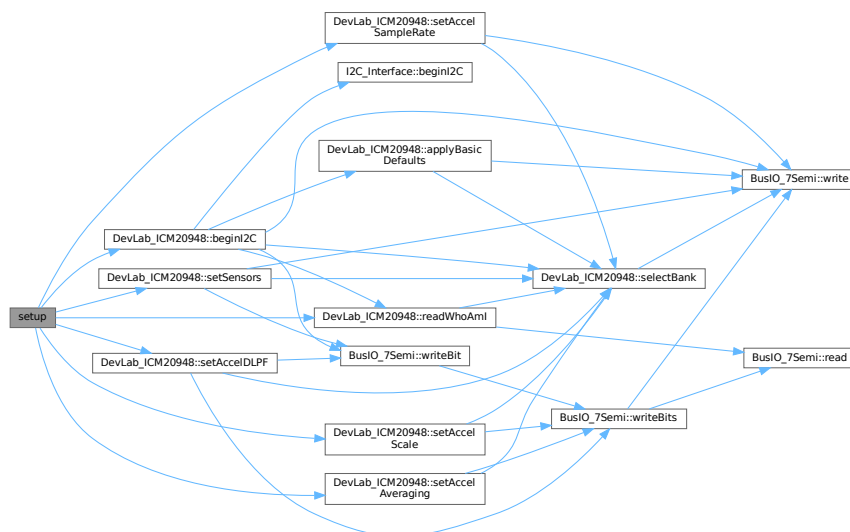
```
void setup ( )
```

Configure the IMU for accelerometer-only operation over I2C.

The function initializes the bus, verifies the device identity, enables the accelerometer, and applies range, DLPF, averaging, and sample-rate settings.

Definition at line 54 of file [i2c_accel.ino](#).

Here is the call graph for this function:



6.3.3 Variable Documentation

6.3.3.1 imu

[DevLab_ICM20948](#) imu

Global ICM-20948 driver instance.

Definition at line 46 of file [i2c_accel.ino](#).

6.4 i2c_accel.ino

[Go to the documentation of this file.](#)

```
00001 /*****
00002  * @file      I2C_Accel.ino
00003  * @author    7Semi, Jonathan Mejorado Lopez
00004  * @brief     Minimal I2C bring-up for 7Semi ICM-20948 on UNO/ESP32 +
00005  *            continuous accelerometer readout
00006  *
00007  * Features
00008  * - I2C init on default/custom pins
00009  * - IMU beginI2C() on I2C (addr 0x68/0x69)
00010  * - Manual sensor enable (setSensors)
00011  * - Accel config: scale, DLPF, averaging, sample rate
00012  * - Read accel (g) at ~2 Hz print
00013  *
00014  * Notes
00015  * - WHO_AM_I must be 0xEA
00016  * - Sensors must be explicitly enabled
00017  * - Bank switching handled internally by library
00018  *
00019  * Wiring (Arduino UNO I2C)
00020  * - SDA -> A4
00021  * - SCL -> A5
00022  * - VCC -> 3V3 (or 5V if your board supports it)
00023  * - GND -> GND
00024  *
00025  * Wiring (ESP32 default I2C)
00026  * - SDA -> GPIO21
00027  * - SCL -> GPIO22
00028  * - VCC -> 3V3
00029  * - GND -> GND
00030  *****/
00031
00032 #include <Wire.h>
00033 #include <DevLab_ICM20948.h>
00034
00035 /* ===== User Config ===== */
00036 /** @brief I2C SDA pin used by the example board. */
00037 #define SDA_PIN 6
00038 /** @brief I2C SCL pin used by the example board. */
00039 #define SCL_PIN 7
00040 /** @brief I2C bus speed in hertz. */
00041 #define I2C_FREQ 400000UL
00042 /** @brief ICM-20948 I2C address selected by the AD0 pin. */
00043 #define ICM_ADDR 0x69
00044
00045 /** @brief Global ICM-20948 driver instance. */
00046 DevLab_ICM20948 imu;
00047
00048 /**
00049  * @brief Configure the IMU for accelerometer-only operation over I2C.
00050  *
00051  * The function initializes the bus, verifies the device identity, enables the
00052  * accelerometer, and applies range, DLPF, averaging, and sample-rate settings.
00053  */
00054 void setup() {
00055     Serial.begin(115200);
00056     delay(200);
00057     Serial.println(F("ICM-20948 (I2C) - Accel Example"));
00058     Wire.begin(SDA_PIN, SCL_PIN);
00059     /* Initialize IMU using I2C. */
00060     if (!imu.beginI2C(ICM_ADDR, Wire, 400000)) {
00061         Serial.println(F("ERROR: ICM-20948 beginI2C() failed."));
00062         while (1) delay(200);
00063     }
00064
00065     Serial.println(F("ICM-20948 ready."));
00066
00067     /* Verify device identity. */
00068     uint8_t who;
00069     if (imu.readWhoAmI(who)) {
00070         Serial.print("WHO_AM_I: 0x");
00071         Serial.println(who, HEX);
00072     }
00073 }
```

```

00072 }
00073
00074 /* Enable only accelerometer
00075  * - accel = true
00076  * - gyro = false
00077  * - temp = false
00078  */
00079 if (!imu.setSensors(true, false, false)) {
00080     Serial.println(F("setSensors failed.));
00081 }
00082
00083 /* ----- ACCEL CONFIG -----
00084  * - Scale: ±2/±4/±8/±16 g
00085  * - DLPF: noise filtering
00086  * - Averaging: noise reduction
00087  * - Sample rate: output frequency
00088  */
00089
00090 /* Set accelerometer full-scale range. */
00091 if (!imu.setAccelScale(ICM20948_Accel_FullScale::G_4)) {
00092     Serial.println(F("setAccelScale failed.));
00093 }
00094
00095 /* Enable DLPF (recommended)
00096  * - bypass = false → DLPF enabled
00097  */
00098 if (!imu.setAccelDLPF(ACCEL_DLPFCFG_3, false)) {
00099     Serial.println(F("setAccelDLPF failed.));
00100 }
00101
00102 /* Set averaging (noise reduction). */
00103 if (!imu.setAccelAveraging(ICM20948_Accel_Average::AVG_8)) {
00104     Serial.println(F("setAccelAveraging failed.));
00105 }
00106
00107 /* Set output data rate (ODR)
00108  * - Base = 1125 Hz
00109  * - ODR = 1125 / (1 + divider)
00110  * - Example: 225 Hz
00111  */
00112 if (!imu.setAccelSampleRate(225)) {
00113     Serial.println(F("setAccelSampleRate failed.));
00114 }
00115
00116 delay(10);
00117 }
00118
00119 /**
00120  * @brief Read and print accelerometer data in g.
00121  */
00122 void loop() {
00123     float ax, ay, az;
00124
00125     /* Read accelerometer data
00126      * - Output in g units
00127      * - Returns true on success
00128      */
00129     if (imu.readAccel(ax, ay, az)) {
00130         Serial.print("ACCEL [g]: ");
00131         Serial.print(ax, 3);
00132         Serial.print(", ");
00133         Serial.print(ay, 3);
00134         Serial.print(", ");
00135         Serial.println(az, 3);
00136     } else {
00137         Serial.println(F("ACCEL read failed"));
00138     }
00139
00140     Serial.println(F("-----"));
00141     delay(500);
00142 }

```

6.5 examples/i2c_basic/i2c_basic.ino File Reference

```

#include <Wire.h>
#include <DevLab_ICM20948.h>

```

Macros

- #define SDA_PIN 6

- I2C SDA pin used by the example board.*
 - `#define SCL_PIN 7`
I2C SCL pin used by the example board.
 - `#define I2C_FREQ 400000UL`
I2C bus speed in hertz.
 - `#define ICM_ADDR 0x69`
ICM-20948 I2C address selected by the AD0 pin.

Functions

- `void setup ()`
Configure Serial, I2C, sensor identity check, and enabled sensors.
- `void loop ()`
Read accelerometer, gyroscope, magnetometer, and temperature data.

Variables

- `DevLab_ICM20948 imu`
Global ICM-20948 driver instance.

6.5.1 Macro Definition Documentation

6.5.1.1 I2C_FREQ

```
#define I2C_FREQ 400000UL
```

I2C bus speed in hertz.
Definition at line 37 of file [i2c_basic.ino](#).

6.5.1.2 ICM_ADDR

```
#define ICM_ADDR 0x69
```

ICM-20948 I2C address selected by the AD0 pin.
Definition at line 39 of file [i2c_basic.ino](#).

6.5.1.3 SCL_PIN

```
#define SCL_PIN 7
```

I2C SCL pin used by the example board.
Definition at line 35 of file [i2c_basic.ino](#).

6.5.1.4 SDA_PIN

```
#define SDA_PIN 6
```

I2C SDA pin used by the example board.
Definition at line 33 of file [i2c_basic.ino](#).

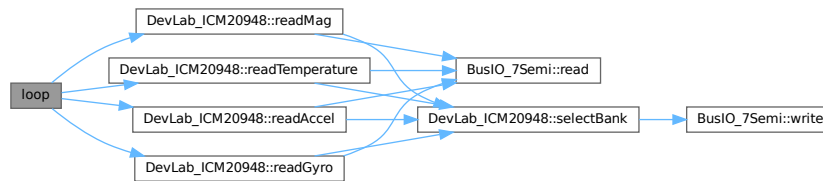
6.5.2 Function Documentation

6.5.2.1 loop()

```
void loop ( )
```

Read accelerometer, gyroscope, magnetometer, and temperature data.
Values are printed periodically to the Serial monitor in engineering units.
Definition at line 91 of file [i2c_basic.ino](#).

Here is the call graph for this function:



6.5.2.2 setup()

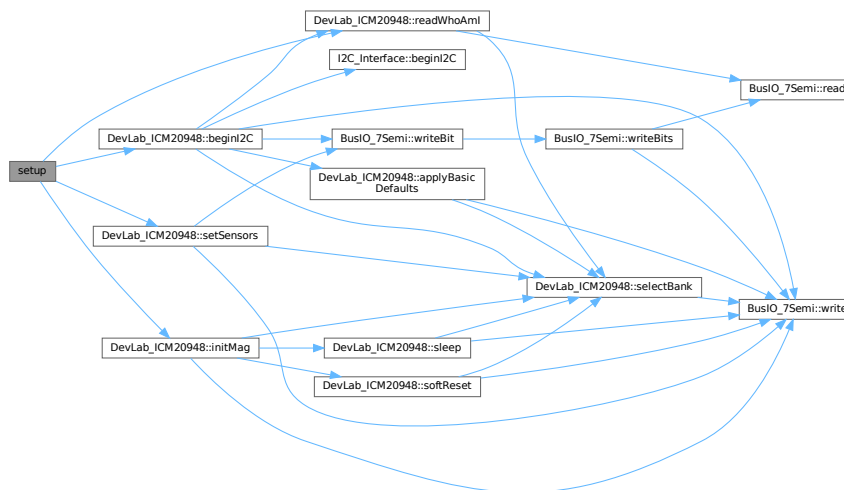
```
void setup ( )
```

Configure Serial, I2C, sensor identity check, and enabled sensors.

The sketch stops in an infinite loop if the IMU cannot be initialized or the WHO_AM_I register does not match the expected ICM-20948 value.

Definition at line 50 of file [i2c_basic.ino](#).

Here is the call graph for this function:



6.5.3 Variable Documentation

6.5.3.1 imu

[DevLab_ICM20948](#) imu

Global ICM-20948 driver instance.

Definition at line 42 of file [i2c_basic.ino](#).

6.6 i2c_basic.ino

[Go to the documentation of this file.](#)

```

00001 /*****
00002  * @file    I2C_Basic.ino
00003  * @author  7Semi,Jonathan Mejorado Lopez
00004  * @brief   Minimal I2C bring-up for 7Semi ICM-20948 +
00005  *          basic Accel/Gyro/Temp/Mag readout (updated API).
00006  *
00007  * Features
  
```

```

00008  * - I2C init (UNO / ESP32)
00009  * - beginI2C() initialization
00010  * - Manual sensor enable (setSensors)
00011  * - Read Accel / Gyro / Temp / Mag
00012  *
00013  * Notes
00014  * - WHO_AM_I must be 0xEA
00015  * - Sensors must be explicitly enabled
00016  * - Magnetometer requires initMag()
00017  *
00018  * Wiring (Arduino UNO I2C)
00019  * - SDA -> A4
00020  * - SCL -> A5
00021  * - VCC -> 3V3 (or 5V if supported)
00022  * - GND -> GND
00023  *
00024  * Wiring (ESP32 default I2C)
00025  * - SDA -> GPIO21
00026  * - SCL -> GPIO22
00027  *****/
00028 #include <Wire.h>
00029 #include <DevLab_ICM20948.h>
00030
00031 /* ===== User Config ===== */
00032 /** @brief I2C SDA pin used by the example board. */
00033 #define SDA_PIN 6
00034 /** @brief I2C SCL pin used by the example board. */
00035 #define SCL_PIN 7
00036 /** @brief I2C bus speed in hertz. */
00037 #define I2C_FREQ 400000UL
00038 /** @brief ICM-20948 I2C address selected by the AD0 pin. */
00039 #define ICM_ADDR 0x69
00040
00041 /** @brief Global ICM-20948 driver instance. */
00042 DevLab_ICM20948 imu;
00043
00044 /**
00045  * @brief Configure Serial, I2C, sensor identity check, and enabled sensors.
00046  *
00047  * The sketch stops in an infinite loop if the IMU cannot be initialized or
00048  * the WHO_AM_I register does not match the expected ICM-20948 value.
00049  */
00050 void setup() {
00051     Serial.begin(115200);
00052     delay(200);
00053     Serial.println(F("ICM-20948 - I2C Basic"));
00054     Wire.begin(SDA_PIN, SCL_PIN);
00055     /* Initialize IMU. */
00056     if (!imu.beginI2C(ICM_ADDR, Wire, 400000)) {
00057         Serial.println(F("ERROR: beginI2C() failed"));
00058         while (1) delay(200);
00059     }
00060
00061     /* WHO_AM_I check. */
00062     uint8_t who;
00063     if (!imu.readWhoAmI(who) || who != 0xEA) {
00064         Serial.println(F("ERROR: WHO_AM_I mismatch"));
00065         while (1) delay(200);
00066     }
00067
00068     Serial.print(F("WHO_AM_I = 0x"));
00069     Serial.println(who, HEX);
00070
00071     /* Enable all sensors. */
00072     if (!imu.setSensors(true, true, true)) {
00073         Serial.println(F("ERROR: setSensors failed"));
00074     }
00075
00076     /* Initialize magnetometer. */
00077     if (!imu.initMag()) {
00078         Serial.println(F("Mag init failed"));
00079     } else {
00080         Serial.println(F("Mag initialized"));
00081     }
00082     Serial.println(F("Ready.\n"));
00083 }
00084
00085 /**
00086  * @brief Read accelerometer, gyroscope, magnetometer, and temperature data.
00087  *
00088  * Values are printed periodically to the Serial monitor in engineering units.
00089  */
00090
00091 void loop() {
00092     float ax, ay, az;
00093     float gx, gy, gz;
00094     float mx, my, mz;

```

```

00095 float tC;
00096
00097 /* Read accelerometer. */
00098 if (imu.readAccel(ax, ay, az)) {
00099     Serial.print(F("ACC [g]: "));
00100     Serial.print(ax, 3); Serial.print(", ");
00101     Serial.print(ay, 3); Serial.print(", ");
00102     Serial.println(az, 3);
00103 } else {
00104     Serial.println(F("ACC read failed"));
00105 }
00106
00107 /* Read gyroscope. */
00108 if (imu.readGyro(gx, gy, gz)) {
00109     Serial.print(F("GYR [dps]: "));
00110     Serial.print(gx, 2); Serial.print(", ");
00111     Serial.print(gy, 2); Serial.print(", ");
00112     Serial.println(gz, 2);
00113 } else {
00114     Serial.println(F("GYR read failed"));
00115 }
00116
00117 /* Read magnetometer. */
00118 if (imu.readMag(mx, my, mz)) {
00119     Serial.print(F("MAG [uT]: "));
00120     Serial.print(mx, 2); Serial.print(", ");
00121     Serial.print(my, 2); Serial.print(", ");
00122     Serial.println(mz, 2);
00123 } else {
00124     Serial.println(F("MAG read failed"));
00125 }
00126
00127 /* Read temperature. */
00128 if (imu.readTemperature(tC)) {
00129     Serial.print(F("TMP [C]: "));
00130     Serial.println(tC, 2);
00131 } else {
00132     Serial.println(F("TMP read failed"));
00133 }
00134
00135 Serial.println(F("-----"));
00136 delay(500);
00137 }
00138
00139 /* Note on magnetometer:
00140 * - Uses internal I2C master (AK09916 via ICM20948)
00141 * - initMag() must be called before readMag()
00142 * - If values are zero:
00143 *   → check initMag()
00144 *   → verify power + pull-ups
00145 */

```

6.7 examples/i2c_gyro/i2c_gyro.ino File Reference

```

#include <Wire.h>
#include <DevLab_ICM20948.h>

```

Macros

- #define `SDA_PIN` 6
I2C SDA pin used by the example board.
- #define `SCL_PIN` 7
I2C SCL pin used by the example board.
- #define `I2C_FREQ` 400000UL
I2C bus speed in hertz.
- #define `ICM_ADDR` 0x69
ICM-20948 I2C address selected by the AD0 pin.

Functions

- void `setup()`
Configure the IMU for gyroscope-only operation over I2C.

- `void loop()`
Read and print gyroscope data in degrees per second.

Variables

- `DevLab_ICM20948 imu`
Global ICM-20948 driver instance.

6.7.1 Macro Definition Documentation

6.7.1.1 I2C_FREQ

`#define I2C_FREQ 400000UL`
I2C bus speed in hertz.
Definition at line 46 of file [i2c_gyro.ino](#).

6.7.1.2 ICM_ADDR

`#define ICM_ADDR 0x69`
ICM-20948 I2C address selected by the AD0 pin.
Definition at line 48 of file [i2c_gyro.ino](#).

6.7.1.3 SCL_PIN

`#define SCL_PIN 7`
I2C SCL pin used by the example board.
Definition at line 44 of file [i2c_gyro.ino](#).

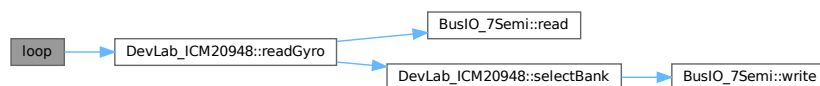
6.7.1.4 SDA_PIN

`#define SDA_PIN 6`
I2C SDA pin used by the example board.
Definition at line 42 of file [i2c_gyro.ino](#).

6.7.2 Function Documentation

6.7.2.1 loop()

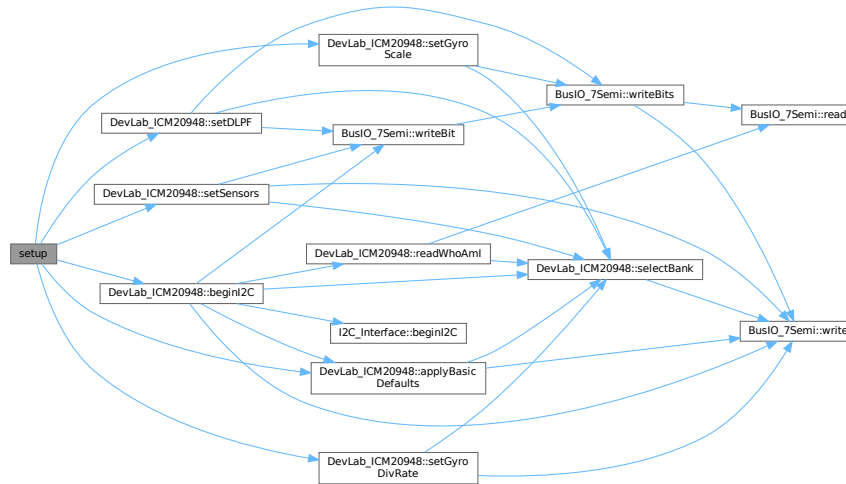
`void loop ( )`
Read and print gyroscope data in degrees per second.
Definition at line 125 of file [i2c_gyro.ino](#).
Here is the call graph for this function:



6.7.2.2 setup()

`void setup ( )`
Configure the IMU for gyroscope-only operation over I2C.
The function initializes the I2C interface, applies safe defaults, enables the gyroscope, and configures DLFP, full-scale range, and output data rate.
Definition at line 59 of file [i2c_gyro.ino](#).

Here is the call graph for this function:



6.7.3 Variable Documentation

6.7.3.1 imu

`DevLab_ICM20948` imu

Global ICM-20948 driver instance.

Definition at line 51 of file `i2c_gyro.ino`.

6.8 i2c_gyro.ino

Go to the documentation of this file.

```
00001 /*****
00002  * @file      I2C_Gyro.ino
00003  * @author    7Semi, Jonathan Mejorado Lopez
00004  * @brief     Minimal I2C bring-up for 7Semi ICM-20948 on ESP32/UNO +
00005  *           continuous gyroscope readout.
00006  *
00007  * Features
00008  * - I2C init on default/custom pins
00009  * - IMU begin() on I2C (addr 0x68/0x69)
00010  * - Safe defaults (applyBasicDefaults)
00011  * - Optional sensor gating: gyro only
00012  * - Gyro config: DLPF, full-scale, ODR, AVG
00013  * - Read gyro (dps) at ~2 Hz print
00014  *
00015  * Wiring (Arduino UNO, I2C)
00016  * - SDA -> A4
00017  * - SCL -> A5
00018  * - VCC -> 3V3 (or 5V if your board supports it)
00019  * - GND -> GND
00020  *
00021  * Wiring (ESP32, I2C)
00022  * - SDA -> GPIO21 (default)
00023  * - SCL -> GPIO22 (default)
00024  * - VCC -> 3V3
00025  * - GND -> GND
00026  *
00027  * Address
00028  * - Default ICM-20948 I2C address is 0x68 (AD0=LOW). If AD0=HIGH, use 0x69.
00029  *****/
00030
00031 #include <Wire.h>
00032 #include <DevLab_ICM20948.h>
00033
00034 /* ===== User Config =====
00035  * - Edit pins/addr if needed
00036  * - UNO uses fixed SDA=A4, SCL=A5 (no need to call Wire.begin with pins)
00037  * - ESP32 can use default Wire (SDA=21, SCL=22) or custom pins via Wire.begin(sda,scl)
00038  */
```

```

00039
00040 /* ===== User Config ===== */
00041 /** @brief I2C SDA pin used by the example board. */
00042 #define SDA_PIN 6
00043 /** @brief I2C SCL pin used by the example board. */
00044 #define SCL_PIN 7
00045 /** @brief I2C bus speed in hertz. */
00046 #define I2C_FREQ 400000UL
00047 /** @brief ICM-20948 I2C address selected by the AD0 pin. */
00048 #define ICM_ADDR 0x69
00049
00050 /** @brief Global ICM-20948 driver instance. */
00051 DevLab_ICM20948 imu;
00052
00053 /**
00054  * @brief Configure the IMU for gyroscope-only operation over I2C.
00055  *
00056  * The function initializes the I2C interface, applies safe defaults, enables
00057  * the gyroscope, and configures DLPF, full-scale range, and output data rate.
00058  */
00059 void setup() {
00060     Serial.begin(115200);
00061     delay(200);
00062     Serial.println(F("ICM-20948 - I2C Basic"));
00063     Wire.begin(SDA_PIN, SCL_PIN);
00064     /* Initialize IMU. */
00065     if (!imu.beginI2C(ICM_ADDR, Wire, 400000)) {
00066         Serial.println(F("ERROR: beginI2C() failed"));
00067         while (1) delay(200);
00068     }
00069
00070     /* Apply safe defaults. */
00071     if (!imu.applyBasicDefaults()) {
00072         Serial.println(F("ERROR: applyBasicDefaults() failed.));
00073         while (1) delay(200);
00074     }
00075
00076     Serial.println(F("ICM-20948 ready.));
00077
00078     /* Optional: gate sensors
00079      * - setSensors(accel_on, gyro_on, temp_on)
00080      * - Here: enable only gyro (accel/temp off)
00081      */
00082     if (!imu.setSensors(/*accel*/ false, /*gyro*/ true, /*temp*/ false)) {
00083         Serial.println(F("setSensors failed.));
00084     }
00085
00086     /* ----- GYRO DLPF Config (reference) -----
00087      * GyroConfig(DLPFCFG, FS_SEL, FCHOICE, XGYRO, YGYRO, ZGYRO, AVGCFG)
00088      * - DLPF : GYRO_DLPFCFG_0..7 (use 3 or 4 for balanced settings)
00089      * - FS_SEL : dps250 / dps500 / dps1000 / dps2000
00090      * - FCHOICE : false → DLPF path (recommended), true → bypass (very wide BW)
00091      * - XGYRO/YGYRO/ZGYRO : per-axis enable (true = enable axis)
00092      * - AVGCFG : 0..7 internal averaging (higher = more smoothing, more delay)
00093      * FCHOICE=0 (DLPF on):
00094      * - GYRO_DLPFCFG_1 : 3dB ≈ 196.6 Hz, NBW ≈ 229.8 Hz
00095      * - GYRO_DLPFCFG_2 : 3dB ≈ 151.8 Hz, NBW ≈ 187.6 Hz
00096      * - GYRO_DLPFCFG_3 : 3dB ≈ 119.5 Hz, NBW ≈ 154.3 Hz
00097      * - GYRO_DLPFCFG_4 : 3dB ≈ 51.2 Hz, NBW ≈ 73.3 Hz
00098      * - GYRO_DLPFCFG_5 : 3dB ≈ 23.9 Hz, NBW ≈ 35.9 Hz ← good default
00099      * - GYRO_DLPFCFG_6 : 3dB ≈ 5.7 Hz, NBW ≈ 8.3 Hz
00100      * - GYRO_DLPFCFG_7 : 3dB ≈ 473 Hz, NBW ≈ 499 Hz
00101      * FCHOICE=1 (Bypass) : very wide (use with care; highest noise)
00102      */
00103
00104     if (!imu.setDLPF(ICM20948_Gyro_DLPF::DLPF_23HZ, false)) {
00105         Serial.println(F("setDLPF failed.));
00106     }
00107     /* Set gyroscope full-scale range. */
00108     if (!imu.setGyroScale(ICM20948_Gyro_FullScale::DPS_2000)) {
00109         Serial.println(F("setGyroScale failed.));
00110     }
00111
00112     /* Set gyroscope output data rate (ODR)
00113      * - Gyro_DIV_RATE = 1100 / (1 + DIV_RATE)
00114      * - DIV_RATE = 0..255
00115      * - Example: DIV_RATE=5 → ODR ≈ 183.33 Hz
00116      */
00117     if (!imu.setGyroDivRate(5)) { // 1100 / (1 + 5) = 183.33 Hz
00118         Serial.println(F("setGyroDivRate failed.));
00119     }
00120 }
00121
00122 /**
00123  * @brief Read and print gyroscope data in degrees per second.
00124  */
00125 void loop() {

```

```

00126 float gx, gy, gz; // gyro X/Y/Z in dps
00127
00128 /* - readGyro(x,y,z)
00129 * - Returns:
00130 * - true on success;
00131 * - Output:
00132 * - gx/gy/gz in dps
00133 */
00134 if (imu.readGyro(gx, gy, gz)) {
00135     Serial.print("GYRO [dps]: ");
00136     Serial.print(gx, 3);
00137     Serial.print(", ");
00138     Serial.print(gy, 3);
00139     Serial.print(", ");
00140     Serial.println(gz, 3);
00141 } else {
00142     Serial.println(F("GYRO read failed"));
00143 }
00144
00145 Serial.println(F("-----"));
00146 delay(500);
00147 }

```

6.9 examples/i2c_magneto/i2c_magneto.ino File Reference

```

#include <Wire.h>
#include <DevLab_ICM20948.h>

```

Macros

- `#define SDA_PIN 6`
I2C SDA pin used by the example board.
- `#define SCL_PIN 7`
I2C SCL pin used by the example board.
- `#define I2C_FREQ 400000UL`
I2C bus speed in hertz.
- `#define ICM_ADDR 0x69`
ICM-20948 I2C address selected by the AD0 pin.

Functions

- `void setup ()`
Configure I2C, verify the IMU, and initialize the magnetometer.
- `void loop ()`
Read and print magnetometer data in microtesla.

Variables

- `DevLab_ICM20948 imu`
Global ICM-20948 driver instance.

6.9.1 Macro Definition Documentation

6.9.1.1 I2C_FREQ

```

#define I2C_FREQ 400000UL
I2C bus speed in hertz.
Definition at line 43 of file i2c_magneto.ino.

```

6.9.1.2 ICM_ADDR

```

#define ICM_ADDR 0x69
ICM-20948 I2C address selected by the AD0 pin.
Definition at line 45 of file i2c_magneto.ino.

```

6.9.1.3 SCL_PIN

```
#define SCL_PIN 7
```

I2C SCL pin used by the example board.

Definition at line 41 of file [i2c_magneto.ino](#).

6.9.1.4 SDA_PIN

```
#define SDA_PIN 6
```

I2C SDA pin used by the example board.

Definition at line 39 of file [i2c_magneto.ino](#).

6.9.2 Function Documentation

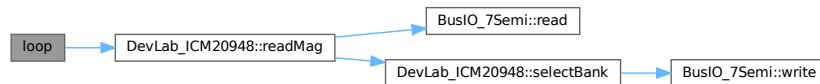
6.9.2.1 loop()

```
void loop ( )
```

Read and print magnetometer data in microtesla.

Definition at line 98 of file [i2c_magneto.ino](#).

Here is the call graph for this function:



6.9.2.2 setup()

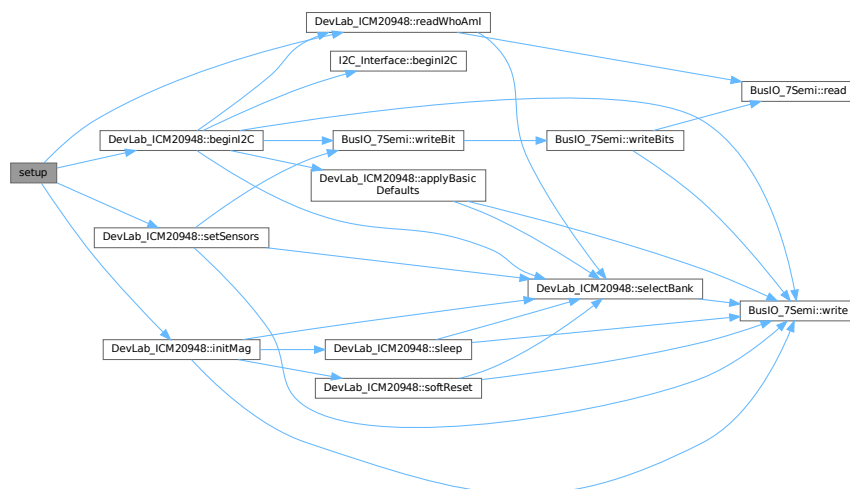
```
void setup ( )
```

Configure I2C, verify the IMU, and initialize the magnetometer.

The magnetometer is accessed through the ICM-20948 internal I2C master, so `initMag()` must succeed before `loop()` can read magnetic field data.

Definition at line 56 of file [i2c_magneto.ino](#).

Here is the call graph for this function:



6.9.3 Variable Documentation

6.9.3.1 imu

[DevLab_ICM20948](#) imu

Global ICM-20948 driver instance.

Definition at line 48 of file [i2c_magneto.ino](#).

6.10 i2c_magneto.ino

[Go to the documentation of this file.](#)

```
00001 /*****
00002  * @file      I2C_Magneto.ino
00003  * @author    7Semi, Jonathan Mejorado Lopez
00004  * @brief     Minimal I2C bring-up for 7Semi ICM-20948 +
00005  *           AK09916 magnetometer readout (updated API).
00006  *
00007  * Features
00008  * - I2C init on default/custom pins
00009  * - IMU beginI2C() initialization
00010  * - Manual sensor enable
00011  * - Magnetometer initialization (initMag)
00012  * - Read magnetometer (uT) at ~2 Hz
00013  *
00014  * Wiring (Arduino UNO, I2C)
00015  * - SDA -> A4
00016  * - SCL -> A5
00017  * - VCC -> 3V3 (or 5V if board supports it)
00018  * - GND -> GND
00019  *
00020  * Wiring (ESP32, I2C default)
00021  * - SDA -> GPIO21
00022  * - SCL -> GPIO22
00023  * - VCC -> 3V3
00024  * - GND -> GND
00025  *
00026  * Notes
00027  * - WHO_AM_I must be 0xEA
00028  * - initMag() must be called before readMag()
00029  * - Magnetometer runs via internal I2C master
00030  *
00031  * Author : 7Semi
00032  * License : MIT
00033  *****/
00034 #include <Wire.h>
00035 #include <DevLab_ICM20948.h>
00036
00037 /* ===== User Config ===== */
00038 /** @brief I2C SDA pin used by the example board. */
00039 #define SDA_PIN 6
00040 /** @brief I2C SCL pin used by the example board. */
00041 #define SCL_PIN 7
00042 /** @brief I2C bus speed in hertz. */
00043 #define I2C_FREQ 400000UL
00044 /** @brief ICM-20948 I2C address selected by the AD0 pin. */
00045 #define ICM_ADDR 0x69
00046
00047 /** @brief Global ICM-20948 driver instance. */
00048 DevLab_ICM20948 imu;
00049
00050 /**
00051  * @brief Configure I2C, verify the IMU, and initialize the magnetometer.
00052  *
00053  * The magnetometer is accessed through the ICM-20948 internal I2C master, so
00054  * initMag() must succeed before loop() can read magnetic field data.
00055  */
00056 void setup() {
00057     Serial.begin(115200);
00058     delay(200);
00059     Serial.println(F("ICM-20948 (I2C) - Magnetometer Example"));
00060     Wire.begin(SDA_PIN, SCL_PIN);
00061     /* Initialize IMU. */
00062     if (!imu.beginI2C(ICM_ADDR, Wire, 400000)) {
00063         Serial.println(F("ERROR: beginI2C() failed."));
00064         while (1) delay(200);
00065     }
00066
00067     Serial.println(F("ICM-20948 ready (I2C)."));
00068
00069     /* Verify device. */
00070     uint8_t who;
00071     if (!imu.readWhoAmI(who) || who != 0xEA) {
```

```

00072     Serial.println(F("ERROR: WHO_AM_I mismatch"));
00073     while (1) delay(200);
00074 }
00075
00076 Serial.print(F("WHO_AM_I = 0x"));
00077 Serial.println(who, HEX);
00078
00079 /* Enable required sensors (mag uses internal I2C master)
00080  * - accel/gyro not strictly required but safe to keep ON
00081  */
00082 if (!imu.setSensors(true, true, false)) {
00083     Serial.println(F("setSensors failed.));
00084 }
00085
00086 /* Initialize magnetometer. */
00087 if (!imu.initMag()) {
00088     Serial.println(F("ERROR: initMag() failed"));
00089     while (1) delay(200);
00090 }
00091
00092 Serial.println(F("Magnetometer ready"));
00093 }
00094
00095 /**
00096  * @brief Read and print magnetometer data in microtesla.
00097  */
00098 void loop() {
00099     float mx, my, mz;
00100
00101     /* Read magnetometer data
00102      * - Output in microtesla (uT)
00103      * - Returns true on success
00104      */
00105     if (imu.readMag(mx, my, mz)) {
00106         Serial.print(F("MAG [uT]: "));
00107         Serial.print(mx, 2); Serial.print(F(", "));
00108         Serial.print(my, 2); Serial.print(F(", "));
00109         Serial.println(mz, 2);
00110     } else {
00111         Serial.println(F("Mag read failed"));
00112     }
00113
00114     Serial.println(F("-----"));
00115     delay(500);
00116 }

```

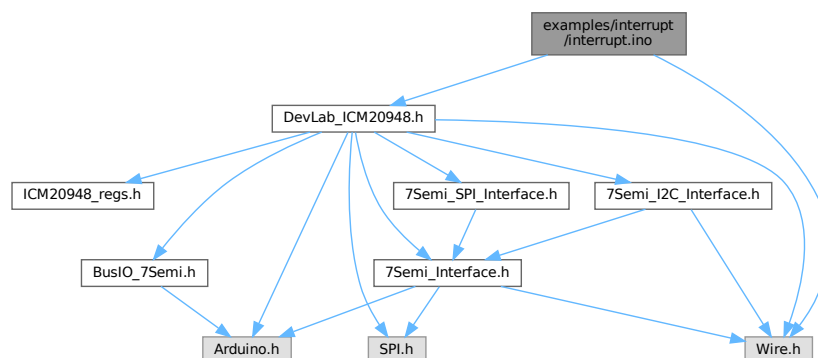
6.11 examples/interrupt/interrupt.ino File Reference

I2C interrupt example for the DevLab ICM-20948.

```
#include <Wire.h>
```

```
#include <DevLab_ICM20948.h>
```

Include dependency graph for interrupt.ino:



Macros

- `#define SDA_PIN 6`

- I2C SDA pin used by the example board.*
 - `#define SCL_PIN 7`
- I2C SCL pin used by the example board.*
 - `#define I2C_FREQ 400000UL`
- I2C bus speed in hertz.*
 - `#define ICM_ADDR 0x69`
- ICM-20948 I2C address selected by the AD0 pin.*
 - `#define PIN_INT D7`
- MCU pin connected to the IMU interrupt output.*

Functions

- void `printLinea ()`
Print a separator line to Serial.
- void `printDobleLinea ()`
Print a double separator line to Serial.
- void `IRAM_ATTR isrHandler ()`
Interrupt service routine for the IMU data-ready signal.
- void `setup ()`
Initialize I2C, verify the IMU, configure interrupts, and attach ISR.
- void `loop ()`
Report interrupt events when the ISR count changes.

Variables

- `DevLab_ICM20948 imu`
Global ICM-20948 driver instance.
- `ICM20948_IntPinConfig pinCfg`
Interrupt pin electrical and latch configuration.
- `ICM20948_IntEnableConfig intEn`
Interrupt enable flags used by this example.
- volatile uint32_t `isrCount = 0`
Number of interrupt events observed by the ISR.

6.11.1 Detailed Description

I2C interrupt example for the DevLab ICM-20948.

Author

Jonathan Mejorado Lopez

Configures the IMU raw-data-ready interrupt pin and counts ISR events on an Arduino-compatible board.
Definition in file `interrupt.ino`.

6.11.2 Macro Definition Documentation

6.11.2.1 I2C_FREQ

`#define I2C_FREQ 400000UL`
I2C bus speed in hertz.
Definition at line 17 of file `interrupt.ino`.

6.11.2.2 ICM_ADDR

`#define ICM_ADDR 0x69`
ICM-20948 I2C address selected by the AD0 pin.
Definition at line 19 of file `interrupt.ino`.

6.11.2.3 PIN_INT

```
#define PIN_INT D7
```

MCU pin connected to the IMU interrupt output.

Definition at line 21 of file [interrupt.ino](#).

6.11.2.4 SCL_PIN

```
#define SCL_PIN 7
```

I2C SCL pin used by the example board.

Definition at line 15 of file [interrupt.ino](#).

6.11.2.5 SDA_PIN

```
#define SDA_PIN 6
```

I2C SDA pin used by the example board.

Definition at line 13 of file [interrupt.ino](#).

6.11.3 Function Documentation

6.11.3.1 isrHandler()

```
void IRAM_ATTR isrHandler ( )
```

Interrupt service routine for the IMU data-ready signal.

Definition at line 53 of file [interrupt.ino](#).

Here is the caller graph for this function:



6.11.3.2 loop()

```
void loop ( )
```

Report interrupt events when the ISR count changes.

Definition at line 95 of file [interrupt.ino](#).

6.11.3.3 printDobleLinea()

```
void printDobleLinea ( )
```

Print a double separator line to Serial.

Definition at line 29 of file [interrupt.ino](#).

6.11.3.4 printLinea()

```
void printLinea ( )
```

Print a separator line to Serial.

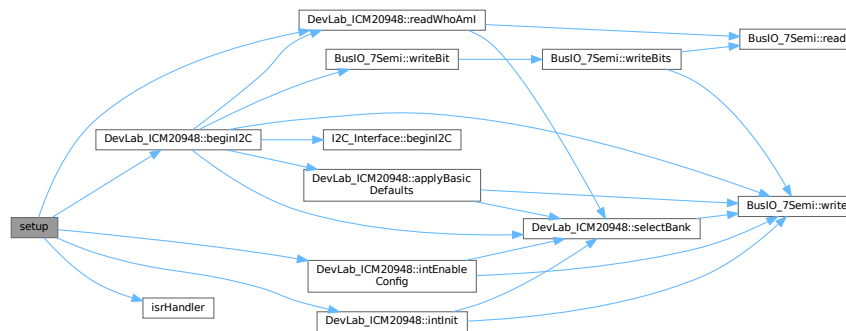
Definition at line 27 of file [interrupt.ino](#).

6.11.3.5 setup()

```
void setup ( )
```

Initialize I2C, verify the IMU, configure interrupts, and attach ISR.

Definition at line 60 of file [interrupt.ino](#).
Here is the call graph for this function:



6.11.4 Variable Documentation

6.11.4.1 imu

[DevLab_ICM20948](#) imu
Global ICM-20948 driver instance.
Definition at line 24 of file [interrupt.ino](#).

6.11.4.2 intEn

[ICM20948_IntEnableConfig](#) intEn

Initial value:

```
= {
    .rawDataRdyEn = 1
}
```

Interrupt enable flags used by this example.
Definition at line 43 of file [interrupt.ino](#).

6.11.4.3 isrCount

`volatile uint32_t isrCount = 0`
Number of interrupt events observed by the ISR.
Definition at line 48 of file [interrupt.ino](#).

6.11.4.4 pinCfg

[ICM20948_IntPinConfig](#) pinCfg

Initial value:

```
= {
    .activeLevel = 1,
    .driveMode = 0,
    .latchMode = 0,
    .clearMode = 1,
    .fsyncActLevel = 0,
    .fsyncIntEn = 0,
    .bypassEn = 0
}
```

Interrupt pin electrical and latch configuration.
Definition at line 32 of file [interrupt.ino](#).

6.12 interrupt.ino

[Go to the documentation of this file.](#)

```
00001 /**
00002  * @file interrupt.ino
00003  * @brief I2C interrupt example for the DevLab ICM-20948.
```

```

00004  * @author Jonathan Mejorado Lopez
00005  * @details Configures the IMU raw-data-ready interrupt pin and counts ISR
00006  * events on an Arduino-compatible board.
00007  */
00008
00009 #include <Wire.h>
00010 #include <DevLab_ICM20948.h>
00011
00012 /** @brief I2C SDA pin used by the example board. */
00013 #define SDA_PIN 6
00014 /** @brief I2C SCL pin used by the example board. */
00015 #define SCL_PIN 7
00016 /** @brief I2C bus speed in hertz. */
00017 #define I2C_FREQ 400000UL
00018 /** @brief ICM-20948 I2C address selected by the AD0 pin. */
00019 #define ICM_ADDR 0x69
00020 /** @brief MCU pin connected to the IMU interrupt output. */
00021 #define PIN_INT D7 // ajusta a tu pin real
00022
00023 /** @brief Global ICM-20948 driver instance. */
00024 DevLab_ICM20948 imu;
00025
00026 /** @brief Print a separator line to Serial. */
00027 void printLinea() {
00028     Serial.println(F("-----"));
00029 }
00030 /** @brief Print a double separator line to Serial. */
00031 void printDobleLinea() {
00032     Serial.println(F("====="));
00033 }
00034
00035 /** @brief Interrupt pin electrical and latch configuration. */
00036 ICM20948_IntPinConfig pinCfg = {
00037     .activeLevel = 1,
00038     .driveMode = 0,
00039     .latchMode = 0,
00040     .clearMode = 1,
00041     .fsyncActLevel = 0,
00042     .fsyncIntEn = 0,
00043     .bypassEn = 0
00044 };
00045
00046 /** @brief Interrupt enable flags used by this example. */
00047 ICM20948_IntEnableConfig intEn = {
00048     .rawDataRdyEn = 1 // inicializacion directa
00049 };
00050
00051 /** @brief Number of interrupt events observed by the ISR. */
00052 volatile uint32_t isrCount = 0;
00053
00054 /**
00055  * @brief Interrupt service routine for the IMU data-ready signal.
00056  */
00057 void IRAM_ATTR isrHandler() {
00058     isrCount++;
00059 }
00060
00061 /**
00062  * @brief Initialize I2C, verify the IMU, configure interrupts, and attach ISR.
00063  */
00064 void setup() {
00065     Serial.begin(115200);
00066     delay(200);
00067
00068     Wire.begin(SDA_PIN, SCL_PIN);
00069
00070     if (!imu.beginI2C(ICM_ADDR, Wire, I2C_FREQ)) {
00071         Serial.println(F("ERROR: beginI2C() failed."));
00072         while (1) delay(200);
00073     }
00074     Serial.println(F("ICM-20948 ready."));
00075
00076     uint8_t who;
00077     if (!imu.readWhoAmI(who) || who != 0xEA) {
00078         Serial.println(F("ERROR: WHO_AM_I mismatch"));
00079         while (1) delay(200);
00080     }
00081     Serial.printf("WHO_AM_I = 0x%02X\n", who);
00082
00083     if (!imu.intInit(pinCfg)) {
00084         Serial.println(F("ERROR: Initialization interruption failed"));
00085         while (1) delay(200);
00086     }
00087     if (!imu.intEnableConfig(intEn)) {
00088         Serial.println(F("ERROR: Initialization interruption failed"));
00089         while (1) delay(200);
00090     }
00091
00092     attachInterrupt(digitalPinToInterrupt(PIN_INT), isrHandler, FALLING);

```

```

00089 Serial.println(F("Interrupt configured. Waiting..."));
00090 }
00091
00092 /**
00093  * @brief Report interrupt events when the ISR count changes.
00094  */
00095 void loop() {
00096     static uint32_t lastCount = 0;
00097
00098     if (isrCount != lastCount) {
00099         lastCount = isrCount;
00100         Serial.printf(">>> ISR disparada! Total: %lu\n", isrCount);
00101     }
00102 }

```

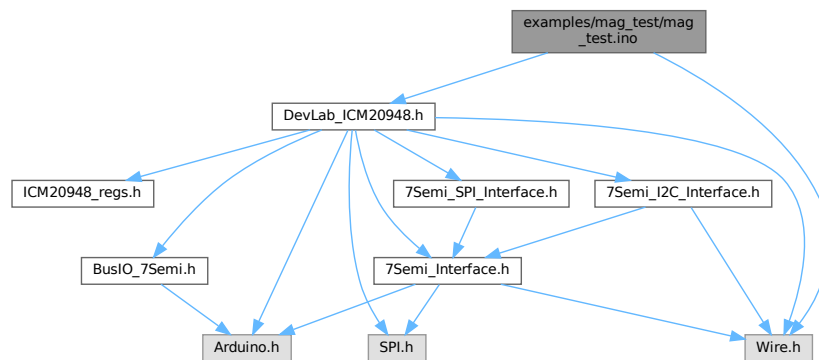
6.13 examples/mag_test/mag_test.ino File Reference

Magnetometer operation-mode sweep for the DevLab ICM-20948.

```
#include <Wire.h>
```

```
#include <DevLab_ICM20948.h>
```

Include dependency graph for mag_test.ino:



Macros

- `#define SDA_PIN 6`
I2C SDA pin used by the example board.
- `#define SCL_PIN 7`
I2C SCL pin used by the example board.
- `#define I2C_FREQ 400000UL`
I2C bus speed in hertz.
- `#define ICM_ADDR 0x69`
ICM-20948 I2C address selected by the AD0 pin.

Functions

- `void printLinea ()`
Print a separator line to Serial.
- `void printDobleLinea ()`
Print a double separator line to Serial.
- `void applySettings ()`
Initialize the magnetometer and collect samples for each mode.
- `void setup ()`
Initialize I2C, verify the IMU, enable sensors, and start the sweep.

- void `loop()`

Empty loop; this sketch runs the magnetometer sweep once in `setup()`.

Variables

- `DevLab_ICM20948 imu`

Global ICM-20948 driver instance.

- const `ICM20948_Op_Mode opMode []`

Magnetometer operating modes tested by the sweep.

- const char * `etiquetasOpModeCfg []`

Text labels matching the operating modes printed in the sweep.

- const uint8_t `N_OPM = 6`

Number of printable magnetometer operation modes.

6.13.1 Detailed Description

Magnetometer operation-mode sweep for the DevLab ICM-20948.

Author

Jonathan Mejorado Lopez

Initializes the AK09916 magnetometer through the ICM-20948 internal I2C master, iterates through continuous measurement modes, and prints CSV-like magnetic field samples to the Serial monitor.

Definition in file [mag_test.ino](#).

6.13.2 Macro Definition Documentation

6.13.2.1 I2C_FREQ

```
#define I2C_FREQ 400000UL
```

I2C bus speed in hertz.

Definition at line 20 of file [mag_test.ino](#).

6.13.2.2 ICM_ADDR

```
#define ICM_ADDR 0x69
```

ICM-20948 I2C address selected by the AD0 pin.

Definition at line 22 of file [mag_test.ino](#).

6.13.2.3 SCL_PIN

```
#define SCL_PIN 7
```

I2C SCL pin used by the example board.

Definition at line 18 of file [mag_test.ino](#).

6.13.2.4 SDA_PIN

```
#define SDA_PIN 6
```

I2C SDA pin used by the example board.

Definition at line 16 of file [mag_test.ino](#).

6.13.3 Function Documentation

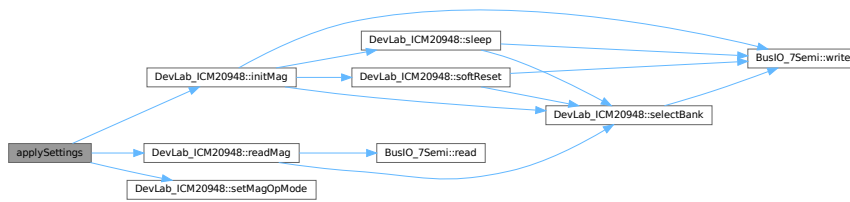
6.13.3.1 applySettings()

```
void applySettings ( )
```

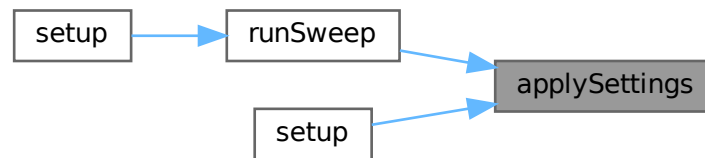
Initialize the magnetometer and collect samples for each mode.

Definition at line 53 of file [mag_test.ino](#).

Here is the call graph for this function:



Here is the caller graph for this function:



6.13.3.2 loop()

```
void loop ( )
```

Empty loop; this sketch runs the magnetometer sweep once in [setup\(\)](#).

Definition at line 138 of file [mag_test.ino](#).

6.13.3.3 printDobleLinea()

```
void printDobleLinea ( )
```

Print a double separator line to Serial.

Definition at line 30 of file [mag_test.ino](#).

6.13.3.4 printLinea()

```
void printLinea ( )
```

Print a separator line to Serial.

Definition at line 28 of file [mag_test.ino](#).

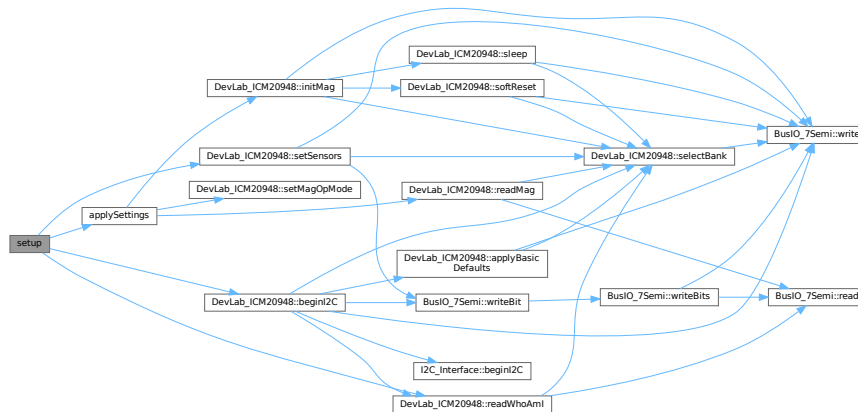
6.13.3.5 setup()

```
void setup ( )
```

Initialize I2C, verify the IMU, enable sensors, and start the sweep.

Definition at line 91 of file [mag_test.ino](#).

Here is the call graph for this function:



6.13.4 Variable Documentation

6.13.4.1 etiquetasOpModeCfg

```
const char* etiquetasOpModeCfg[ ]
```

Initial value:

```
= {
    "PowerDown", "SingleMeasure", "Continuo1", "Continuo2", "Continuo3", "Continuo4"
}
```

Text labels matching the operating modes printed in the sweep.

Definition at line 44 of file [mag_test.ino](#).

6.13.4.2 imu

```
DevLab_ICM20948 imu
```

Global ICM-20948 driver instance.

Definition at line 25 of file [mag_test.ino](#).

6.13.4.3 N_OPM

```
const uint8_t N_OPM = 6
```

Number of printable magnetometer operation modes.

Definition at line 48 of file [mag_test.ino](#).

6.13.4.4 opMode

```
const ICM20948_Op_Mode opMode[ ]
```

Initial value:

```
= {
    ICM20948_Op_Mode::MODE_POWER_DOWN,
    ICM20948_Op_Mode::MODE_SINGLE_MEASUREMENT,
    ICM20948_Op_Mode::MODE_CONTINUOUS_1,
    ICM20948_Op_Mode::MODE_CONTINUOUS_2,
    ICM20948_Op_Mode::MODE_CONTINUOUS_3,
    ICM20948_Op_Mode::MODE_CONTINUOUS_4,
    ICM20948_Op_Mode::MODE_SELF_TEST
}
```

Magnetometer operating modes tested by the sweep.

Definition at line 33 of file [mag_test.ino](#).

6.14 mag_test.ino

[Go to the documentation of this file.](#)

00001 /**

```

00002  * @file mag_test.ino
00003  * @brief Magnetometer operation-mode sweep for the DevLab ICM-20948.
00004  * @author Jonathan Mejorado Lopez
00005  * @details Initializes the AK09916 magnetometer through the ICM-20948 internal
00006  * I2C master, iterates through continuous measurement modes, and prints CSV-like
00007  * magnetic field samples to the Serial monitor.
00008  */
00009
00010 #include <Wire.h>
00011 #include <DevLab_ICM20948.h>
00012 // -----
00013 // PINES I2C - ESP32C6 NANO Unit Electronics
00014 // -----
00015 /** @brief I2C SDA pin used by the example board. */
00016 #define SDA_PIN 6
00017 /** @brief I2C SCL pin used by the example board. */
00018 #define SCL_PIN 7
00019 /** @brief I2C bus speed in hertz. */
00020 #define I2C_FREQ 400000UL
00021 /** @brief ICM-20948 I2C address selected by the AD0 pin. */
00022 #define ICM_ADDR 0x69
00023
00024 /** @brief Global ICM-20948 driver instance. */
00025 DevLab_ICM20948 imu;
00026
00027 /** @brief Print a separator line to Serial. */
00028 void printLinea() {
00029     Serial.println(F("-----"));
00030 }
00031 /** @brief Print a double separator line to Serial. */
00032 void printDobleLinea() {
00033     Serial.println(F("====="));
00034 }
00035
00036 /** @brief Magnetometer operating modes tested by the sweep. */
00037 const ICM20948_Op_Mode opMode[] = {
00038     ICM20948_Op_Mode::MODE_POWER_DOWN,
00039     ICM20948_Op_Mode::MODE_SINGLE_MEASUREMENT,
00040     ICM20948_Op_Mode::MODE_CONTINUOUS_1,
00041     ICM20948_Op_Mode::MODE_CONTINUOUS_2,
00042     ICM20948_Op_Mode::MODE_CONTINUOUS_3,
00043     ICM20948_Op_Mode::MODE_CONTINUOUS_4,
00044     ICM20948_Op_Mode::MODE_SELF_TEST
00045 };
00046
00047 /** @brief Text labels matching the operating modes printed in the sweep. */
00048 const char* etiquetasOpModeCfg[] = {
00049     "PowerDown", "SingleMeasure", "Continuo1", "Continuo2", "Continuo3", "Continuo4"
00050 };
00051
00052 /** @brief Number of printable magnetometer operation modes. */
00053 const uint8_t N_OPM = 6;
00054
00055 /**
00056  * @brief Initialize the magnetometer and collect samples for each mode.
00057  */
00058 void applySettings(){
00059     /** Initialize magnetometer. */
00060     if (!imu.initMag()) {
00061         Serial.println(F("ERROR: initMag() failed"));
00062         while (1) delay(200);
00063     }
00064     for(int i = 2; i < N_OPM; i++){
00065         imu.setMagOpMode(opMode[i]);
00066         delay(100);
00067         Serial.println(F("Magnetometer ready"));
00068         float mx, my, mz;
00069
00070         /** Read magnetometer data
00071          * - Output in microtesla (uT)
00072          * - Returns true on success
00073          */
00074         for(int r = 0; r < 500 ; r++){
00075             if (imu.readMag(mx, my, mz)) {
00076                 Serial.printf("[%d]", r);
00077                 Serial.print(",");
00078                 Serial.print(etiquetasOpModeCfg[i]);
00079                 Serial.print(",");
00080                 Serial.print(mx, 2); Serial.print(F(" ");
00081                 Serial.print(my, 2); Serial.print(F(" ");
00082                 Serial.println(mz, 2);
00083             } else {
00084                 Serial.println(F("Mag read failed"));
00085             }
00086         }
00087         Serial.println(F("-----"));
00088         delay(500);
00089     }
00090 }
00091
00092 }

```

```

00087
00088 /**
00089  * @brief Initialize I2C, verify the IMU, enable sensors, and start the sweep.
00090  */
00091 void setup() {
00092     // put your setup code here, to run once:
00093     Serial.begin(115200);
00094     delay(200);
00095     Serial.println(F("ICM-20948 (I2C) - Magnetometer Example"));
00096
00097     Wire.begin(SDA_PIN, SCL_PIN);
00098     /* Initialize IMU. */
00099     if (!imu.beginI2C(ICM_ADDR, Wire, 400000)) {
00100         Serial.println(F("ERROR: beginI2C() failed."));
00101         while (1) delay(200);
00102     }
00103
00104     Serial.println(F("ICM-20948 ready (I2C)."));
00105
00106     /* Verify device. */
00107     uint8_t who;
00108     if (!imu.readWhoAmI(who) || who != 0xEA) {
00109         Serial.println(F("ERROR: WHO_AM_I mismatch"));
00110         while (1) delay(200);
00111     }
00112
00113     Serial.print(F("WHO_AM_I = 0x"));
00114     Serial.println(who, HEX);
00115
00116     /* Enable required sensors (mag uses internal I2C master)
00117      * - accel/gyro not strictly required but safe to keep ON
00118      */
00119     if (!imu.setSensors(true, true, false)) {
00120         Serial.println(F("setSensors failed."));
00121     }
00122
00123     Serial.println(F("  Sensor listo.\n"));
00124     Serial.println(F("  Sensor PLANO y QUIETO sobre la mesa."));
00125     Serial.println(F("  Iniciando en 5 segundos..."));
00126     for (int i = 5; i > 0; i--) {
00127         Serial.printf("  %d...\n", i);
00128         delay(1000);
00129     }
00130
00131     applySettings();
00132 }
00133
00134 /**
00135  * @brief Empty loop; this sketch runs the magnetometer sweep once in setup().
00136  */
00137
00138 void loop() {
00139     // put your main code here, to run repeatedly:
00140
00141 }

```

6.15 examples/spi_accel/spi_accel.ino File Reference

```
#include <DevLab_ICM20948.h>
```

Macros

- #define `SPI_FAST_SPEED` 1000000
SPI clock used during sensor initialization and reads.
- #define `CS_PIN` D10

Functions

- SPIClass `spi_bus` (SPI)
SPI bus object used by the IMU driver.
- void `setup` ()
Configure the IMU for accelerometer-only operation over SPI.
- void `loop` ()
Read and print accelerometer data in g.

Variables

- [DevLab_ICM20948 imu](#)

Global ICM-20948 driver instance.

6.15.1 Macro Definition Documentation

6.15.1.1 CS_PIN

```
#define CS_PIN D10
```

Definition at line 36 of file [spi_accel.ino](#).

6.15.1.2 SPI_FAST_SPEED

```
#define SPI_FAST_SPEED 1000000
```

SPI clock used during sensor initialization and reads.

Definition at line 34 of file [spi_accel.ino](#).

6.15.2 Function Documentation

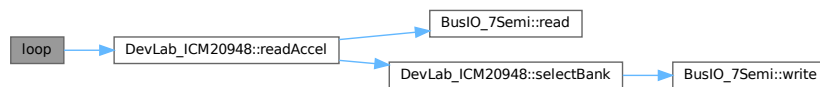
6.15.2.1 loop()

```
void loop ( )
```

Read and print accelerometer data in g.

Definition at line 122 of file [spi_accel.ino](#).

Here is the call graph for this function:



6.15.2.2 setup()

```
void setup ( )
```

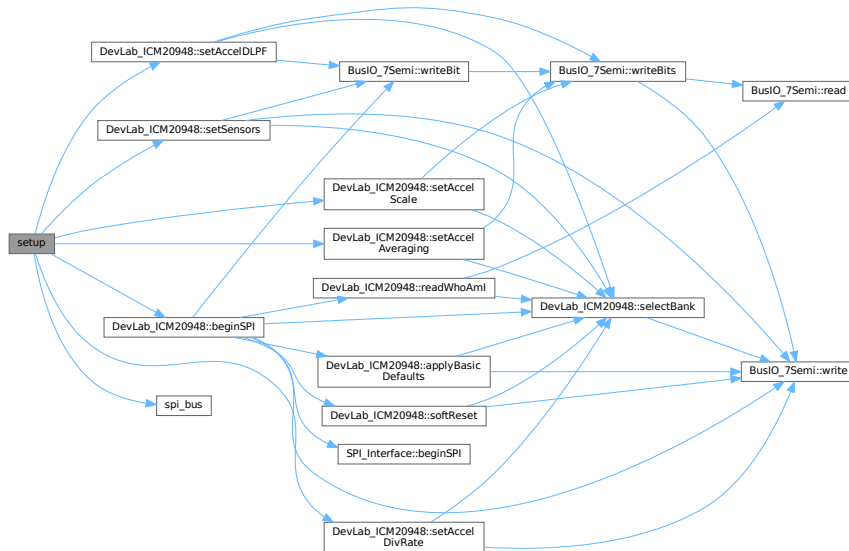
Configure the IMU for accelerometer-only operation over SPI.

Note

The sketch expects a board-level CS_PIN definition to select the chip-select pin used by beginSPI().

Definition at line 49 of file [spi_accel.ino](#).

Here is the call graph for this function:



6.15.2.3 spi_bus()

```
SPIClass spi_bus (
    SPI )
```

SPI bus object used by the IMU driver.

Here is the caller graph for this function:



6.15.3 Variable Documentation

6.15.3.1 imu

[DevLab_ICM20948](#) imu

Global ICM-20948 driver instance.

Definition at line 38 of file [spi_accel.ino](#).

6.16 spi_accel.ino

[Go to the documentation of this file.](#)

```
00001 /*****
00002  * @file    SPI_Accel.ino
```

```

00003 * @author 7Semi, Jonathan Mejorado Lopez
00004 * @brief Minimal SPI bring-up for 7Semi ICM-20948 on ESP32 +
00005 * continuous accelerometer readout.
00006 *
00007 * Features
00008 * - SPI init on custom pins (ESP32 VSPI)
00009 * - IMU begin() on SPI (CS pin user-defined)
00010 * - Safe defaults (applyBasicDefaults)
00011 * - Optional sensor gating: accel only
00012 * - Accel config: DLPF, full-scale, ODR, DEC3
00013 * - Read accel (g) at ~2 Hz print
00014 *
00015 * Wiring (Arduino UNO)
00016 * - SCK -> GPIO13
00017 * - MOSI(SDA) -> GPIO12
00018 * - MISO(SDO) -> GPIO11
00019 * - CS -> GPIO10 (changeable)
00020 * - VCC -> 3V3
00021 * - GND -> GND
00022 * Wiring (ESP32 VSPI default)
00023 * - SCK -> GPIO18
00024 * - MOSI -> GPIO23
00025 * - MISO -> GPIO19
00026 * - CS -> GPIO5 (changeable)
00027 * - VCC -> 3V3
00028 * - GND -> GND
00029 *****/
00030
00031 #include <DevLab_ICM20948.h>
00032
00033 /** @brief SPI clock used during sensor initialization and reads. */
00034 #define SPI_FAST_SPEED 1000000
00035
00036 #define CS_PIN D10 // Chip-select pin for SPI (change as needed)
00037 /** @brief Global ICM-20948 driver instance. */
00038 DevLab_ICM20948 imu;
00039
00040 /** @brief SPI bus object used by the IMU driver. */
00041 SPIClass spi_bus(SPI); // use el bus SPI por defecto del Arduino
00042
00043 /**
00044 * @brief Configure the IMU for accelerometer-only operation over SPI.
00045 *
00046 * @note The sketch expects a board-level CS_PIN definition to select the
00047 * chip-select pin used by beginSPI().
00048 */
00049 void setup() {
00050     Serial.begin(115200);
00051     delay(200);
00052
00053
00054     if (!imu.beginSPI(CS_PIN, spi_bus, 1000000)) {
00055         Serial.println("ERROR: beginSPI() failed");
00056         while (1) delay(200);
00057     }
00058
00059     Serial.println(F("ICM-20948 ready.));
00060
00061     /* Optional: gate sensors
00062     * - setSensors(accel_on, gyro_on, temp_on)
00063     * - Here: enable only accel (gyro/temp off)
00064     */
00065     if (!imu.setSensors(/*accel*/ true, /*gyro*/ false, /*temp*/ false)) {
00066         Serial.println(F("setSensors failed.));
00067     }
00068
00069     /* ----- ACCEL DLPF Config (reference) -----
00070     * ACCEL_DLPFCFG (0..7) - 3dB & Noise Bandwidth (datasheet)
00071     * DLPF path ODR = 1125 / (1 + ACCEL_SMPLRT_DIV), DIV: 0..4095
00072     *
00073     * FCHOICE=0 (Bypass) : 3dB ≈ 1209 Hz, NBW ≈ 1248 Hz, Rate ≈ 4500 Hz
00074     * FCHOICE=1 (DLPF on):
00075     * - ACCEL_DLPFCFG_0 : 3dB ≈ 246.0 Hz, NBW ≈ 265.0 Hz
00076     * - ACCEL_DLPFCFG_1 : 3dB ≈ 246.0 Hz, NBW ≈ 265.0 Hz
00077     * - ACCEL_DLPFCFG_2 : 3dB ≈ 111.4 Hz, NBW ≈ 136.0 Hz
00078     * - ACCEL_DLPFCFG_3 : 3dB ≈ 50.4 Hz, NBW ≈ 68.8 Hz ← good default
00079     * - ACCEL_DLPFCFG_4 : 3dB ≈ 23.9 Hz, NBW ≈ 34.4 Hz
00080     * - ACCEL_DLPFCFG_5 : 3dB ≈ 11.5 Hz, NBW ≈ 17.0 Hz
00081     * - ACCEL_DLPFCFG_6 : 3dB ≈ 5.7 Hz, NBW ≈ 8.3 Hz
00082     * - ACCEL_DLPFCFG_7 : 3dB ≈ 473 Hz, NBW ≈ 499 Hz
00083     */
00084
00085     /* Configure accelerometer
00086     * AccelConfigure(DLPF, FS_SEL, dlpf_enable, dec3, stX, stY, stZ)
00087     * - DLPF : ACCEL_DLPFCFG_0..7 (use 3 for balanced setting)
00088     * - FS_SEL : 0..3 => ±2/±4/±8/±16 g (g2=0, g4=1, g8=2, g16=3)
00089     * - dlpf_enable: true to use DLPF path (recommended)

```

```

00090  * - dec3      : 0..3 (DEC3_CFG averaging)
00091  *              0 = ACCEL_DEC3_AVG_1_OR_4 (depends on FCHOICE)
00092  *              1 = ACCEL_DEC3_AVG_8
00093  *              2 = ACCEL_DEC3_AVG_16
00094  *              3 = ACCEL_DEC3_AVG_32
00095  * - stX/Y/Z   : enable self-test (false here)
00096  */
00097  if(!imu.setAccelDLPF(ACCEL_DLPFCFG_3,false)) {
00098      Serial.println(F("setAccelDLPF failed."));
00099  }
00100
00101  if(!imu.setAccelScale(ICM20948_Accel_FullScale::G_4)) {
00102      Serial.println(F("setAccelFS failed."));
00103  }
00104  if(!imu.setAccelAveraging(ICM20948_Accel_Average::AVG_8)) {
00105      Serial.println(F("setAccelAveraging failed."));
00106  }
00107
00108  /* Set accelerometer output data rate (ODR)
00109  * - Accel_SMPLRT(rate_hz)
00110  * - Base (DLPF path) = 1125 Hz
00111  * - Valid range: 1-1125 Hz
00112  * - Example: 225 Hz
00113  */
00114  if (!imu.setAccelDivRate(4)) { // 1125 / (1 + 4) = 225 Hz
00115      Serial.println(F("Accel_SMPLRT failed."));
00116  }
00117 }
00118
00119 /**
00120 * @brief Read and print accelerometer data in g.
00121 */
00122 void loop() {
00123     float ax, ay, az; // accel X/Y/Z in g
00124
00125     /* - readAccel(x,y,z)
00126     * - Returns:
00127     *   - true on success;
00128     * - Output:
00129     *   - ax/ay/az in g
00130     */
00131     if (imu.readAccel(ax, ay, az)) {
00132         Serial.print("ACCEL [g]: ");
00133         Serial.print(ax, 3);
00134         Serial.print(", ");
00135         Serial.print(ay, 3);
00136         Serial.print(", ");
00137         Serial.println(az, 3);
00138     } else {
00139         Serial.println(F("ACCEL read failed"));
00140     }
00141
00142     Serial.println(F("-----"));
00143     delay(500);
00144 }

```

6.17 examples/spi_basic/spi_basic.ino File Reference

```
#include <DevLab_ICM20948.h>
```

Macros

- #define `CS_PIN` D10
Chip-select pin for SPI.
- #define `SPI_FAST_SPEED` 1000000
SPI clock used during sensor initialization and reads.

Functions

- SPIClass `spi_bus` (SPI)
SPI bus object used by the IMU driver.
- void `setup` ()
Initialize Serial and start the ICM-20948 over SPI.
- void `loop` ()

Read and print accelerometer, gyroscope, and temperature data.

Variables

- [DevLab_ICM20948 imu](#)

Global ICM-20948 driver instance.

6.17.1 Macro Definition Documentation

6.17.1.1 CS_PIN

```
#define CS_PIN D10
```

Chip-select pin for SPI.

Definition at line 35 of file [spi_basic.ino](#).

6.17.1.2 SPI_FAST_SPEED

```
#define SPI_FAST_SPEED 1000000
```

SPI clock used during sensor initialization and reads.

Definition at line 38 of file [spi_basic.ino](#).

6.17.2 Function Documentation

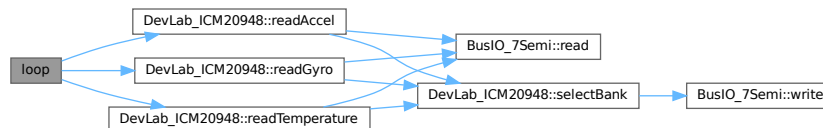
6.17.2.1 loop()

```
void loop ( )
```

Read and print accelerometer, gyroscope, and temperature data.

Definition at line 69 of file [spi_basic.ino](#).

Here is the call graph for this function:



6.17.2.2 setup()

```
void setup ( )
```

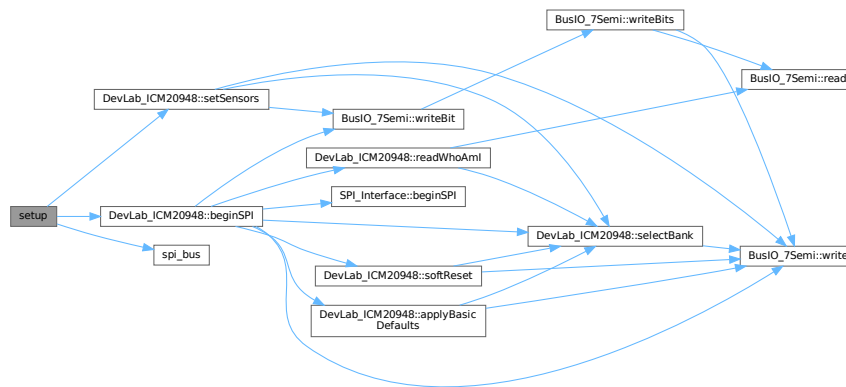
Initialize Serial and start the ICM-20948 over SPI.

Note

The sketch expects a board-level CS_PIN definition to select the chip-select pin used by beginSPI().

Definition at line 52 of file [spi_basic.ino](#).

Here is the call graph for this function:



6.17.2.3 spi_bus()

```
SPIClass spi_bus (
    SPI )
```

SPI bus object used by the IMU driver.

Here is the caller graph for this function:



6.17.3 Variable Documentation

6.17.3.1 imu

[DevLab_ICM20948](#) imu

Global ICM-20948 driver instance.

Definition at line 41 of file [spi_basic.ino](#).

6.18 spi_basic.ino

[Go to the documentation of this file.](#)

```
00001 /*****
00002  * @file      SPI_Basic.ino
00003  * @author    7Semi, Jonathan Mejorado Lopez
00004  * @brief     Minimal SPI bring-up for 7Semi ICM-20948 +
00005  *           basic Accel/Gyro/Temp readout (updated API).
00006  *
00007  * Features
00008  * - SPI init (UNO / ESP32)
00009  * - beginSPI() initialization
00010  * - Manual sensor enable
```

```

00011 * - Read Accel / Gyro / Temp
00012 *
00013 * Notes
00014 * - WHO_AM_I must be 0xEA
00015 * - Sensors must be explicitly enabled
00016 * - SPI speed ~1MHz recommended during init
00017 *
00018 * Wiring (Arduino UNO SPI)
00019 * - SCK(SCL) -> D13
00020 * - MOSI(SDA) -> D11
00021 * - MISO(SD0/AD0) -> D12
00022 * - CS(nCS) -> D10 (changeable; update CS_PIN)
00023 * - VCC -> 3V3
00024 * - GND -> GND
00025 *
00026 * Wiring (ESP32 VSPI default)
00027 * - SCK -> D13
00028 * - MOSI -> D11
00029 * - MISO -> D12
00030 * - CS -> D10
00031 *****/
00032 #include <DevLab_ICM20948.h>
00033 /** @brief Chip-select pin for SPI. */
00034
00035 #define CS_PIN D10
00036 /** @brief SPI clock used during sensor initialization and reads. */
00037
00038 #define SPI_FAST_SPEED 1000000
00039
00040 /** @brief Global ICM-20948 driver instance. */
00041 DevLab_ICM20948 imu;
00042
00043 /** @brief SPI bus object used by the IMU driver. */
00044 SPIClass spi_bus(SPI);
00045
00046 /**
00047  * @brief Initialize Serial and start the ICM-20948 over SPI.
00048  *
00049  * @note The sketch expects a board-level CS_PIN definition to select the
00050  * chip-select pin used by beginSPI().
00051  */
00052 void setup() {
00053     Serial.begin(115200);
00054     delay(200);
00055
00056     if (!imu.beginSPI(CS_PIN, spi_bus, SPI_FAST_SPEED)) {
00057         Serial.println("ERROR: beginSPI() failed");
00058         while (1) delay(200);
00059     }
00060
00061     imu.setSensors(true, true, true);
00062     Serial.println("Ready.");
00063 }
00064
00065 /**
00066  * @brief Read and print accelerometer, gyroscope, and temperature data.
00067  */
00068 void loop() {
00069     float ax, ay, az;
00070     float gx, gy, gz;
00071     float tC;
00072
00073     /* Read accelerometer. */
00074     if (imu.readAccel(ax, ay, az)) {
00075         Serial.print(F("ACC [g]: "));
00076         Serial.print(ax, 3); Serial.print(", ");
00077         Serial.print(ay, 3); Serial.print(", ");
00078         Serial.println(az, 3);
00079     } else {
00080         Serial.println(F("ACC read failed"));
00081     }
00082
00083     /* Read gyroscope. */
00084     if (imu.readGyro(gx, gy, gz)) {
00085         Serial.print(F("GYR [dps]: "));
00086         Serial.print(gx, 2); Serial.print(", ");
00087         Serial.print(gy, 2); Serial.print(", ");
00088         Serial.println(gz, 2);
00089     } else {
00090         Serial.println(F("GYR read failed"));
00091     }
00092
00093     /* Read temperature. */
00094     if (imu.readTemperature(tC)) {
00095         Serial.print(F("TMP [C]: "));
00096         Serial.println(tC, 2);
00097     }
00098 }

```

```

00098 } else {
00099     Serial.println(F("TMP read failed"));
00100 }
00101
00102 Serial.println(F("-----"));
00103 delay(500);
00104 }

```

6.19 examples/spi_gyro/spi_gyro.ino File Reference

```
#include <DevLab_ICM20948.h>
```

Macros

- `#define CS_PIN D10`
- `#define SPI_FAST_SPEED 1000000`
SPI clock used during sensor initialization and reads.

Functions

- SPIClass `spi_bus` (SPI)
SPI bus object used by the IMU driver.
- void `setup` ()
Configure the IMU for gyroscope-only operation over SPI.
- void `loop` ()
Read and print gyroscope data in degrees per second.

Variables

- `DevLab_ICM20948 imu`
Global ICM-20948 driver instance.

6.19.1 Macro Definition Documentation

6.19.1.1 CS_PIN

```
#define CS_PIN D10
```

Definition at line 33 of file [spi_gyro.ino](#).

6.19.1.2 SPI_FAST_SPEED

```
#define SPI_FAST_SPEED 1000000
```

SPI clock used during sensor initialization and reads.

Definition at line 36 of file [spi_gyro.ino](#).

6.19.2 Function Documentation

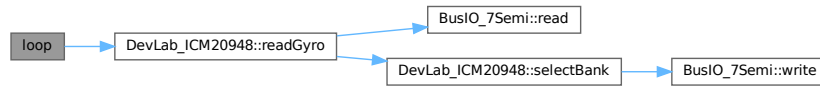
6.19.2.1 loop()

```
void loop ( )
```

Read and print gyroscope data in degrees per second.

Definition at line 121 of file [spi_gyro.ino](#).

Here is the call graph for this function:



6.19.2.2 setup()

```
void setup ( )
```

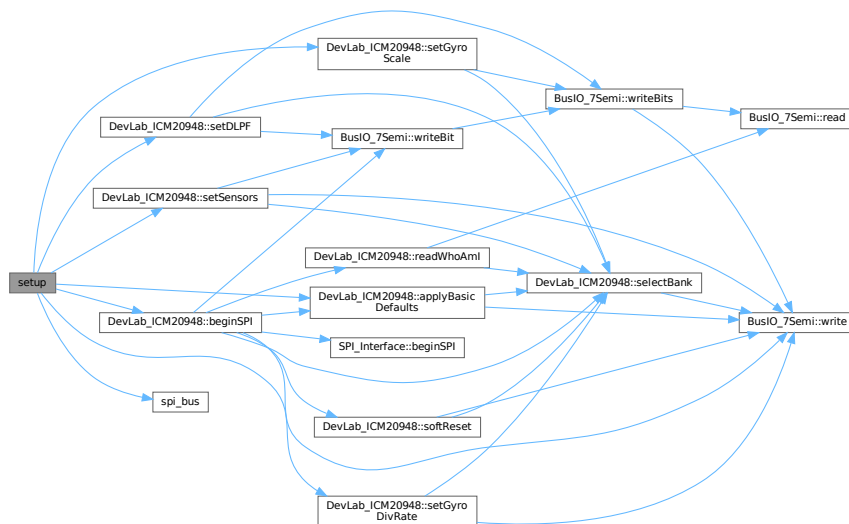
Configure the IMU for gyroscope-only operation over SPI.

Note

The sketch expects a board-level CS_PIN definition to select the chip-select pin used by beginSPI().

Definition at line 50 of file [spi_gyro.ino](#).

Here is the call graph for this function:



6.19.2.3 spi_bus()

```
SPIclass spi_bus (
    SPI )
```

SPI bus object used by the IMU driver.

Here is the caller graph for this function:



6.19.3 Variable Documentation

6.19.3.1 imu

[DevLab_ICM20948](#) imu

Global ICM-20948 driver instance.

Definition at line 39 of file [spi_gyro.ino](#).

6.20 spi_gyro.ino

[Go to the documentation of this file.](#)

```
00001 /*****
00002  * @file      ICM20948_SPI_Gyro.ino
00003  * @brief     Minimal SPI bring-up for 7Semi ICM-20948 on ESP32/UNO +
00004  *           continuous gyroscope readout.
00005  * @author    7Semi, Jonathan Mejorado Lopez
00006  *
00007  * Features
00008  * - SPI init on custom pins
00009  * - IMU begin() on SPI (CS pin user-defined)
00010  * - Safe defaults (applyBasicDefaults)
00011  * - Optional sensor gating: gyro only
00012  * - Gyro config: DLPF, full-scale, ODR, AVG
00013  * - Read gyro (dps) at ~2 Hz print
00014  *
00015  * Wiring (Arduino UNO)
00016  * - SCK -> GPIO13
00017  * - MOSI(SDA) -> GPIO11
00018  * - MISO(SDO) -> GPIO12
00019  * - CS -> GPIO10 (changeable)
00020  * - VCC -> 3V3
00021  * - GND -> GND
00022  * Wiring (ESP32 VSPI default)
00023  * - SCK -> GPIO18
00024  * - MOSI -> GPIO23
00025  * - MISO -> GPIO19
00026  * - CS -> GPIO5 (changeable)
00027  * - VCC -> 3V3
00028  * - GND -> GND
00029  *****/
00030
00031 #include <DevLab_ICM20948.h>
00032
00033 #define CS_PIN D10 // Chip-select pin for SPI (change as needed)
00034
00035 /** @brief SPI clock used during sensor initialization and reads. */
00036 #define SPI_FAST_SPEED 1000000
00037
00038 /** @brief Global ICM-20948 driver instance. */
00039 DevLab_ICM20948 imu;
00040
00041 /** @brief SPI bus object used by the IMU driver. */
00042 SPIClass spi_bus(SPI);
00043
00044 /**
00045  * @brief Configure the IMU for gyroscope-only operation over SPI.
00046  *
00047  * @note The sketch expects a board-level CS_PIN definition to select the
00048  * chip-select pin used by beginSPI().
00049  */
00050 void setup() {
00051     Serial.begin(115200);
00052     delay(200);
00053     Serial.println(F("ICM-20948 (SPI) - Gyro Example"));
00054
00055     /* Init SPI bus on your pins. */
00056     // SPI.begin(SCK_PIN, MISO_PIN, MOSI_PIN, CS_PIN);
00057
00058     if (!imu.beginSPI(CS_PIN, spi_bus, SPI_FAST_SPEED)) {
00059         Serial.println("ERROR: beginSPI() failed");
00060         while (1) delay(200);
00061     }
00062
00063     /* Apply safe defaults. */
00064     if (!imu.applyBasicDefaults()) {
00065         Serial.println(F("ERROR: applyBasicDefaults() failed."));
00066         while (1) delay(200);
00067     }
00068
00069     Serial.println(F("ICM-20948 ready."));
00070
00071     /* Optional: gate sensors
```

```

00072     * - setSensors(accel_on, gyro_on, temp_on)
00073     * - Here: enable only gyro (accel/temp off)
00074     */
00075     if (!imu.setSensors(/*accel*/ false, /*gyro*/ true, /*temp*/ false)) {
00076         Serial.println(F("setSensors failed."));
00077     }
00078
00079     /* ----- GYRO DLPF Config (reference) -----
00080     * GyroConfig(DLPFCFG, FS_SEL, FCHOICE, XGYRO, YGYRO, ZGYRO, AVGCFG)
00081     * - DLPF      : GYRO_DLPFCFG_0..7 (use 3 for balanced setting)
00082     * - FS_SEL    : dps250 / dps500 / dps1000 / dps2000
00083     * - FCHOICE   : false → DLPF path (recommended), true → bypass (very wide BW)
00084     * - XGYRO/YGYRO/ZGYRO : per-axis enable (true = enable axis)
00085     * - AVGCFG    : 0..7 internal averaging (higher = more smoothing, more delay)
00086     * FCHOICE=0 (DLPF on):
00087     * - GYRO_DLPFCFG_0 : 3dB ≈ 12106 Hz, NBW ≈ 12316 Hz (very wide)
00088     * - GYRO_DLPFCFG_1 : 3dB ≈ 196.6 Hz, NBW ≈ 229.8 Hz
00089     * - GYRO_DLPFCFG_2 : 3dB ≈ 151.8 Hz, NBW ≈ 187.6 Hz
00090     * - GYRO_DLPFCFG_3 : 3dB ≈ 119.5 Hz, NBW ≈ 154.3 Hz ← good default
00091     * - GYRO_DLPFCFG_4 : 3dB ≈ 51.2 Hz, NBW ≈ 73.3 Hz
00092     * - GYRO_DLPFCFG_5 : 3dB ≈ 23.9 Hz, NBW ≈ 35.9 Hz
00093     * - GYRO_DLPFCFG_6 : 3dB ≈ 5.7 Hz, NBW ≈ 8.3 Hz
00094     * - GYRO_DLPFCFG_7 : 3dB ≈ 473 Hz, NBW ≈ 499 Hz
00095     *
00096     * FCHOICE=1 (Bypass) : very wide (use with care; highest noise)
00097     */
00098
00099     if (!imu.setDLPF(ICM20948_Gyro_DLPF::DLPF_51HZ, false)) {
00100         Serial.println(F("setGyroDLPF failed."));
00101     }
00102     if (!imu.setGyroScale(ICM20948_Gyro_FullScale::DPS_2000)) {
00103         Serial.println(F("setGyroScale failed."));
00104     }
00105
00106     /* Set gyroscope output data rate (ODR)
00107     * - Gyro_SMPLRT(rate_hz)
00108     * - Base (DLPF path) = 1100 Hz
00109     * - Valid range: ~4.3-1100 Hz
00110     * - Example: 1000 Hz
00111     */
00112     if (!imu.setGyroDivRate(2)) { // 1100 / (1 + 1) = 550 Hz
00113         Serial.println(F("setGyroDivRate failed."));
00114     }
00115 }
00116
00117 /**
00118  * @brief Read and print gyroscope data in degrees per second.
00119  */
00120 void loop() {
00121     float gx, gy, gz; // gyro X/Y/Z in dps
00122
00123     /* - readGyro(x,y,z)
00124     * - Returns:
00125     *   - true on success;
00126     *   - Output:
00127     *     - gx/gy/gz in dps
00128     */
00129     if (imu.readGyro(gx, gy, gz)) {
00130         Serial.print("GYRO [dps]: ");
00131         Serial.print(gx, 3);
00132         Serial.print(", ");
00133         Serial.print(gy, 3);
00134         Serial.print(", ");
00135         Serial.print(gz, 3);
00136         Serial.println();
00137     } else {
00138         Serial.println(F("GYRO read failed"));
00139     }
00140
00141     Serial.println(F("-----"));
00142     delay(500);
00143 }

```

6.21 examples/test/test.ino File Reference

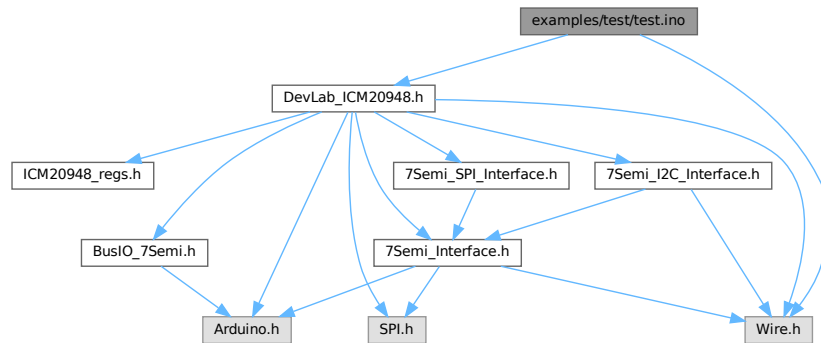
Accelerometer DLPF validation sketch for the 7Semi ICM-20948.

```

#include <Wire.h>
#include <DevLab_ICM20948.h>

```

Include dependency graph for test.ino:



Classes

- struct [configDLPF](#)
Gyroscope DLPF divider and timing configuration.

Macros

- `#define` [SDA_PIN](#) 6
I2C SDA pin used by the example board.
- `#define` [SCL_PIN](#) 7
I2C SCL pin used by the example board.
- `#define` [I2C_FREQ](#) 400000UL
I2C bus speed in hertz.
- `#define` [ICM_ADDR](#) 0x69
ICM-20948 I2C address selected by the AD0 pin.

Functions

- void [printLinea](#) ()
Print a separator line to Serial.
- void [printDobleLinea](#) ()
Print a double separator line to Serial.
- bool [applySettings](#) (uint8_t fs_idx, const [configDLPF](#) &cfg)
Apply one accelerometer full-scale and DLPF configuration.
- void [firstStage](#) ()
Verify the IMU identity register.
- void [runSweep](#) ()
Run the accelerometer DLPF validation sweep and print samples.
- void [setup](#) ()
Initialize I2C, reset/wake the IMU, enable sensors, and start sweep.
- void [loop](#) ()
Optional post-sweep loop for manual accelerometer reads.

Variables

- const [configDLPF configsDLPF](#) []
Accelerometer DLPF configurations exercised by this validation.
- const uint8_t [N_DLPF](#) = sizeof([configsDLPF](#)) / sizeof([configsDLPF](#)[0])
Number of accelerometer DLPF configurations.
- const [ICM20948_Accel_FullScale accelCfg](#) [] = { [ICM20948_Accel_FullScale::G_2](#), [ICM20948_Accel_FullScale::G_4](#), [ICM20948_Accel_FullScale::G_8](#), [ICM20948_Accel_FullScale::G_16](#) }
Accelerometer full-scale ranges available for validation.
- const char * [etiquetasAccelCfg](#) [] = { "+-2G", "+-4G", "+-8G", "+-16G" }
Text labels matching accelCfg.
- const uint8_t [N_FS](#) = 4
Number of accelerometer full-scale ranges.
- const [ICM20948_Accel_DLPFCFG accelDLPF](#) [] = { [ICM20948_Accel_DLPFCFG::DLPF0_246HZ](#), [ICM20948_Accel_DLPFCFG::DLPF_246HZ](#), [ICM20948_Accel_DLPFCFG::DLPF_111HZ](#), [ICM20948_Accel_DLPFCFG::DLPF_24HZ](#), [ICM20948_Accel_DLPFCFG::DLPF_12HZ](#), [ICM20948_Accel_DLPFCFG::DLPF_6HZ](#) }
Accelerometer DLPF enum values matching configsDLPF.
- const char * [etiquetasAccelDLPFCFG](#) [] = { "246/265", "246/265", "111/136", "50.4/68.8", "23.9/34.4", "11.5/17.0", "5.7/8.3", "473/499" }
Text labels describing accelDLPF bandwidth pairs.
- const [ICM20948_Accel_Average accelAVG](#) [] = { [ICM20948_Accel_Average::AVG_1_OR_4](#), [ICM20948_Accel_Average::AVG_16](#), [ICM20948_Accel_Average::AVG_32](#) }
Accelerometer averaging settings available for validation.
- const char * [etiquetasAccelAVG](#) [] = { "1 OR 4", "8", "16", "32" }
Text labels matching accelAVG.
- const uint16_t [accelSR](#) [] = { 0, 1, 3, 5, 7, 10, 15, 22, 31, 63, 127, 255, 513, 1022, 2044, 4095 }
Raw accelerometer sample-rate divider values for experiments.
- const char * [etiquetasSR](#) [] = { "1125.0", "562.5", "281.3", "187.5", "140.6", "102.3", "70.3", "48.9", "35.2", "17.6", "8.8", "4.4", "2.2", "1.1", "0.55", "0.27" }
Text labels matching accelSR output data rates.
- [DevLab_ICM20948 imu](#)
Global ICM-20948 driver instance.
- uint8_t [total_pass](#) = 0
Count of validation passes reserved for future checks.
- uint8_t [total_fail](#) = 0
Count of validation failures reserved for future checks.
- uint8_t [total_warn](#) = 0
Count of validation warnings reserved for future checks.

6.21.1 Detailed Description

Accelerometer DLPF validation sketch for the 7Semi ICM-20948.

Author

Jonathan Mejorado Lopez

Runs an I2C accelerometer sweep across DLPF settings, prints sample data to Serial, and is intended for bench validation with the sensor flat and stationary.

Definition in file [test.ino](#).

6.21.2 Macro Definition Documentation

6.21.2.1 I2C_FREQ

```
#define I2C_FREQ 400000UL
```

I2C bus speed in hertz.
Definition at line 20 of file [test.ino](#).

6.21.2.2 ICM_ADDR

```
#define ICM_ADDR 0x69
```

ICM-20948 I2C address selected by the AD0 pin.
Definition at line 26 of file [test.ino](#).

6.21.2.3 SCL_PIN

```
#define SCL_PIN 7
```

I2C SCL pin used by the example board.
Definition at line 18 of file [test.ino](#).

6.21.2.4 SDA_PIN

```
#define SDA_PIN 6
```

I2C SDA pin used by the example board.
Definition at line 16 of file [test.ino](#).

6.21.3 Function Documentation

6.21.3.1 applySettings()

```
bool applySettings (
    uint8_t fs_idx,
    const configDLPF & cfg )
```

Apply one accelerometer full-scale and DLPF configuration.

Parameters

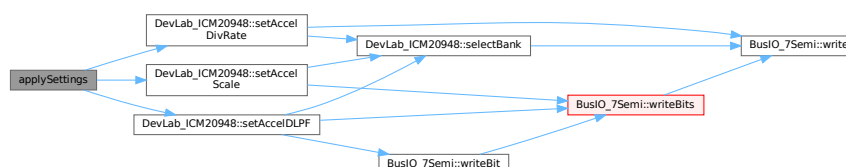
<i>fs_idx</i>	Index into accelCfg.
<i>cfg</i>	DLPF timing and divider configuration.

Returns

true if all configuration writes succeeded, otherwise false.

Definition at line 115 of file [test.ino](#).

Here is the call graph for this function:



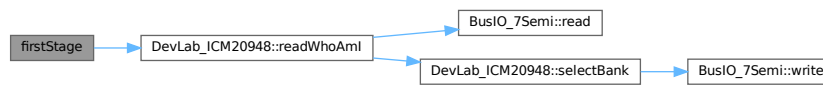
6.21.3.2 firstStage()

```
void firstStage ( )
```

Verify the IMU identity register.

Definition at line 132 of file [test.ino](#).

Here is the call graph for this function:



Here is the caller graph for this function:



6.21.3.3 loop()

```
void loop ( )
```

Optional post-sweep loop for manual accelerometer reads.

Definition at line 226 of file [test.ino](#).

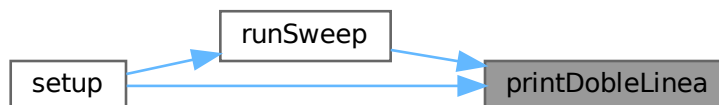
6.21.3.4 printDobleLinea()

```
void printDobleLinea ( )
```

Print a double separator line to Serial.

Definition at line 104 of file [test.ino](#).

Here is the caller graph for this function:



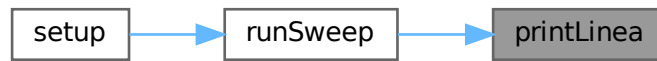
6.21.3.5 printLinea()

```
void printLinea ( )
```

Print a separator line to Serial.

Definition at line 99 of file [test.ino](#).

Here is the caller graph for this function:



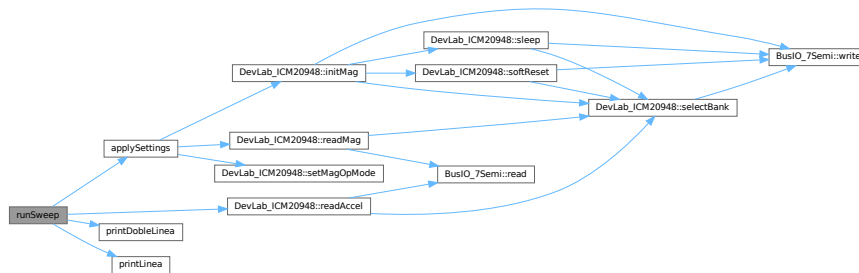
6.21.3.6 runSweep()

```
void runSweep ( )
```

Run the accelerometer DLPF validation sweep and print samples.

Definition at line 145 of file [test.ino](#).

Here is the call graph for this function:



Here is the caller graph for this function:



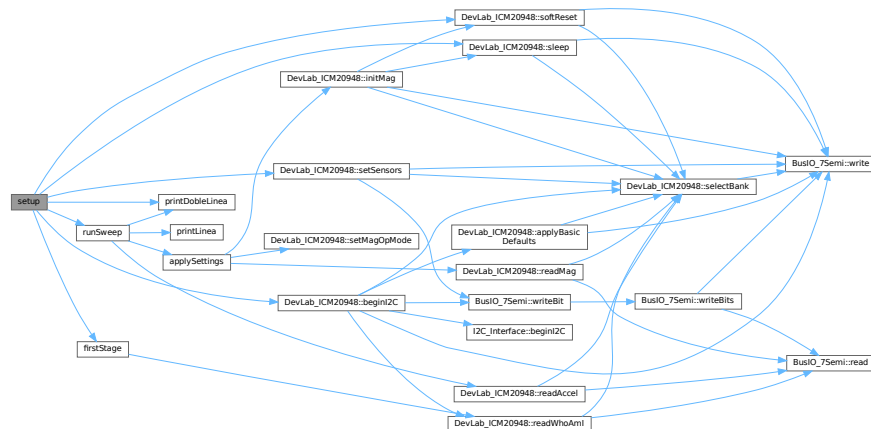
6.21.3.7 setup()

```
void setup ( )
```

Initialize I2C, reset/wake the IMU, enable sensors, and start sweep.

Definition at line 187 of file [test.ino](#).

Here is the call graph for this function:



6.21.4 Variable Documentation

6.21.4.1 accelAVG

```
const ICM20948_Accel_Average accelAVG[] = { ICM20948_Accel_Average::AVG_1_OR_4 , ICM20948_Accel_Average::AVG_8,
ICM20948_Accel_Average::AVG_16, ICM20948_Accel_Average::AVG_32}
```

Accelerometer averaging settings available for validation.

Definition at line 73 of file [test.ino](#).

6.21.4.2 accelCfg

```
const ICM20948_Accel_FullScale accelCfg[] = { ICM20948_Accel_FullScale::G_2, ICM20948_Accel_FullScale::G_4,
ICM20948_Accel_FullScale::G_8, ICM20948_Accel_FullScale::G_16}
```

Accelerometer full-scale ranges available for validation.

Definition at line 61 of file [test.ino](#).

6.21.4.3 accelDLPF

```
const ICM20948_Accel_DLPFCFG accelDLPF[] = { ICM20948_Accel_DLPFCFG::DLPF0_246HZ, ICM20948_Accel_DLPFCFG::DLPF1_111HZ,
ICM20948_Accel_DLPFCFG::DLPF2_50HZ, ICM20948_Accel_DLPFCFG::DLPF3_24HZ,
ICM20948_Accel_DLPFCFG::DLPF4_12HZ, ICM20948_Accel_DLPFCFG::DLPF5_6HZ, ICM20948_Accel_DLPFCFG::DLPF6_473HZ
}
```

Accelerometer DLPF enum values matching configsDLPF.

Definition at line 68 of file [test.ino](#).

6.21.4.4 accelSR

```
const uint16_t accelSR[] = {0, 1, 3, 5, 7, 10, 15, 22, 31, 63, 127, 255, 513, 1022, 2044,
4095}
```

Raw accelerometer sample-rate divider values for experiments.

Definition at line 78 of file [test.ino](#).

6.21.4.5 configsDLPF

```
const configDLPF configsDLPF[]
```

Initial value:

```
= {
```

```
{ 0, 0, 265.0f, 1125.0f, "DLPF0_246 | NBW=265Hz | ODR=1125Hz (4.2x)" },
{ 1, 0, 265.0f, 1125.0f, "DLPF_246 | NBW=265Hz | ODR=1125Hz (4.2x)" },
{ 2, 1, 136.0f, 562.5f, "DLPF_111 | NBW=136Hz | ODR= 562Hz (4.1x)" },
{ 3, 3, 68.8f, 281.3f, "DLPF_50 | NBW= 69Hz | ODR= 281Hz (4.1x)" },
```

```
{ 4, 7, 34.4f, 140.6f, "DLPF_24 | NBW= 34Hz | ODR= 141Hz (4.1x)" },
{ 5, 15, 17.0f, 70.3f, "DLPF_12 | NBW= 17Hz | ODR= 70Hz (4.1x)" },
{ 6, 31, 8.3f, 35.2f, "DLPF_6 | NBW= 8Hz | ODR= 35Hz (4.2x)" },
{ 7, 0, 499.0f, 1125.0f, "DLPF_473 | NBW=499Hz | ODR=1125Hz (2.3x)" },
}
```

Accelerometer DLPF configurations exercised by this validation.

Definition at line 43 of file [test.ino](#).

6.21.4.6 etiquetasAccelAVG

```
const char* etiquetasAccelAVG[] = {"1 OR 4", "8", "16", "32"}
```

Text labels matching accelAVG.

Definition at line 75 of file [test.ino](#).

6.21.4.7 etiquetasAccelCfg

```
const char* etiquetasAccelCfg[] = {"+-2G", "+-4G", "+-8G", "+-16G"}
```

Text labels matching accelCfg.

Definition at line 63 of file [test.ino](#).

6.21.4.8 etiquetasAccelDLPFCFG

```
const char* etiquetasAccelDLPFCFG[] = { "246/265", "246/265", "111/136", "50.4/68.8", "23.↵
9/34.4", "11.5/17.0", "5.7/8.3", "473/499"}
```

Text labels describing accelDLPF bandwidth pairs.

Definition at line 70 of file [test.ino](#).

6.21.4.9 etiquetasSR

```
const char* etiquetasSR[] = { "1125.0", "562.5", "281.3", "187.5", "140.6", "102.3", "70.3",
"48.9", "35.2", "17.6", "8.8", "4.4", "2.2", "1.1", "0.55", "0.27"}
```

Text labels matching accelSR output data rates.

Definition at line 80 of file [test.ino](#).

6.21.4.10 imu

[DevLab_ICM20948](#) imu

Global ICM-20948 driver instance.

Definition at line 85 of file [test.ino](#).

6.21.4.11 N_DLPF

```
const uint8_t N_DLPF = sizeof(configsDLPF) / sizeof(configsDLPF[0])
```

Number of accelerometer DLPF configurations.

Definition at line 58 of file [test.ino](#).

6.21.4.12 N_FS

```
const uint8_t N_FS = 4
```

Number of accelerometer full-scale ranges.

Definition at line 65 of file [test.ino](#).

6.21.4.13 total_fail

```
uint8_t total_fail = 0
```

Count of validation failures reserved for future checks.

Definition at line 90 of file [test.ino](#).

6.21.4.14 total_pass

uint8_t total_pass = 0

Count of validation passes reserved for future checks.

Definition at line 88 of file test.ino.

6.21.4.15 total_warn

uint8_t total_warn = 0

Count of validation warnings reserved for future checks.

Definition at line 92 of file test.ino.

6.22 test.ino

[Go to the documentation of this file.](#)

```
00001 /**
00002  * @file test.ino
00003  * @author Jonathan Mejorado Lopez
00004  * @brief Accelerometer DLPF validation sketch for the 7Semi ICM-20948.
00005  * @details Runs an I2C accelerometer sweep across DLPF settings, prints sample
00006  * data to Serial, and is intended for bench validation with the sensor flat and
00007  * stationary.
00008  */
00009
00010 #include <Wire.h>
00011 #include <DevLab_ICM20948.h>
00012 // -----
00013 // PINES I2C – ESP32C6 NANO Unit Electronics
00014 // -----
00015 /** @brief I2C SDA pin used by the example board. */
00016 #define SDA_PIN 6
00017 /** @brief I2C SCL pin used by the example board. */
00018 #define SCL_PIN 7
00019 /** @brief I2C bus speed in hertz. */
00020 #define I2C_FREQ 400000UL // Fast Mode 400 kHz
00021
00022 // -----
00023 // DIRECCIONES I2C
00024 // -----
00025 /** @brief ICM-20948 I2C address selected by the AD0 pin. */
00026 #define ICM_ADDR 0x69 // AD0 = LOW (GND)
00027
00028 /** @brief Accelerometer DLPF divider and timing configuration. */
00029 struct configDLPF {
00030     /** @brief Index into the accelDLPF lookup table. */
00031     uint8_t dlpf_idx;
00032     /** @brief Sample-rate divider written to the IMU register. */
00033     uint16_t div; // el valor REAL del registro, 0-4095, sin ambigüedad
00034     /** @brief Noise bandwidth in hertz for reporting. */
00035     float nbw_hz;
00036     /** @brief Output data rate in hertz for timing calculations. */
00037     float odr_hz; // = 1125.0f / (1 + div), para calcular tiempos
00038     /** @brief Text label printed with each captured sample. */
00039     const char* nombre;
00040 };
00041
00042 /** @brief Accelerometer DLPF configurations exercised by this validation. */
00043 const configDLPF configsDLPF[] = {
00044     // idx div NBW(Hz) ODR(Hz) nombre
00045     { 0, 0, 265.0f, 1125.0f, "DLPF0_246 | NBW=265Hz | ODR=1125Hz (4.2x)" },
00046     { 1, 0, 265.0f, 1125.0f, "DLPF_246 | NBW=265Hz | ODR=1125Hz (4.2x)" },
00047     { 2, 1, 136.0f, 562.5f, "DLPF_111 | NBW=136Hz | ODR= 562Hz (4.1x)" },
00048     { 3, 3, 68.8f, 281.3f, "DLPF_50 | NBW= 69Hz | ODR= 281Hz (4.1x)" },
00049     { 4, 7, 34.4f, 140.6f, "DLPF_24 | NBW= 34Hz | ODR= 141Hz (4.1x)" },
00050     { 5, 15, 17.0f, 70.3f, "DLPF_12 | NBW= 17Hz | ODR= 70Hz (4.1x)" },
00051     { 6, 31, 8.3f, 35.2f, "DLPF_6 | NBW= 8Hz | ODR= 35Hz (4.2x)" },
00052     { 7, 0, 499.0f, 1125.0f, "DLPF_473 | NBW=499Hz | ODR=1125Hz (2.3x)" },
00053 };
00054 //-----
00055 // ARRAYS DE CONFIGURACIÓN
00056 //-----
00057 /** @brief Number of accelerometer DLPF configurations. */
00058 const uint8_t N_DLPF = sizeof(configsDLPF) / sizeof(configsDLPF[0]);
00059
00060 /** @brief Accelerometer full-scale ranges available for validation. */
00061 const ICM20948_Accel_FullScale accelCfg[] = { ICM20948_Accel_FullScale::G_2,
00062     ICM20948_Accel_FullScale::G_4, ICM20948_Accel_FullScale::G_8, ICM20948_Accel_FullScale::G_16 };
00063 /** @brief Text labels matching accelCfg. */
00064 const char* etiquetasAccelCfg[] = {"+-2G", "+-4G", "+-8G", "+-16G"};
```

```

00065 const uint8_t N_FS = 4;
00066
00067 /** @brief Accelerometer DLPF enum values matching configsDLPF. */
00068 const ICM20948_Accel_DLPFCFG accelDLPF[] = { ICM20948_Accel_DLPFCFG::DLPF0_246HZ,
ICM20948_Accel_DLPFCFG::DLPF_246HZ, ICM20948_Accel_DLPFCFG::DLPF_111HZ,
ICM20948_Accel_DLPFCFG::DLPF_50HZ, ICM20948_Accel_DLPFCFG::DLPF_24HZ,
ICM20948_Accel_DLPFCFG::DLPF_12HZ, ICM20948_Accel_DLPFCFG::DLPF_6HZ, ICM20948_Accel_DLPFCFG::DLPF_473HZ
};
00069 /** @brief Text labels describing accelDLPF bandwidth pairs. */
00070 const char* etiquetasAccelDLPFCFG[] = { "246/265", "246/265", "111/136", "50.4/68.8", "23.9/34.4",
"11.5/17.0", "5.7/8.3", "473/499"};
00071
00072 /** @brief Accelerometer averaging settings available for validation. */
00073 const ICM20948_Accel_Average accelAVG[] = {ICM20948_Accel_Average::AVG_1_OR_4
,ICM20948_Accel_Average::AVG_8, ICM20948_Accel_Average::AVG_16, ICM20948_Accel_Average::AVG_32};
00074 /** @brief Text labels matching accelAVG. */
00075 const char* etiquetasAccelAVG[] = {"1 OR 4", "8", "16", "32"};
00076
00077 /** @brief Raw accelerometer sample-rate divider values for experiments. */
00078 const uint16_t accelSR[] = {0, 1, 3, 5, 7, 10, 15, 22, 31, 63, 127, 255, 513, 1022, 2044, 4095};
00079 /** @brief Text labels matching accelSR output data rates. */
00080 const char* etiquetasSR[] = { "1125.0", "562.5", "281.3", "187.5", "140.6", "102.3", "70.3", "48.9",
"35.2", "17.6", "8.8", "4.4", "2.2", "1.1", "0.55", "0.27"};
00081 // -----
00082 // INSTANCIA DEL SENSOR
00083 // -----
00084 /** @brief Global ICM-20948 driver instance. */
00085 DevLab_ICM20948 imu;
00086
00087 /** @brief Count of validation passes reserved for future checks. */
00088 uint8_t total_pass = 0;
00089 /** @brief Count of validation failures reserved for future checks. */
00090 uint8_t total_fail = 0;
00091 /** @brief Count of validation warnings reserved for future checks. */
00092 uint8_t total_warn = 0;
00093
00094 // =====
00095 // UTILIDADES DE IMPRESIÓN
00096 // =====
00097
00098 /** @brief Print a separator line to Serial. */
00099 void printLinea() {
00100     Serial.println(F("-----"));
00101 }
00102
00103 /** @brief Print a double separator line to Serial. */
00104 void printDobleLinea() {
00105     Serial.println(F("====="));
00106 }
00107
00108 /**
00109  * @brief Apply one accelerometer full-scale and DLPF configuration.
00110  *
00111  * @param fs_idx Index into accelCfg.
00112  * @param cfg DLPF timing and divider configuration.
00113  * @return true if all configuration writes succeeded, otherwise false.
00114  */
00115 bool applySettings(uint8_t fs_idx, const configDLPF &cfg){
00116     uint8_t dlpf_val = (uint8_t)accelDLPF[cfg.dlpf_idx];
00117     bool ok = true;
00118
00119     ok &= imu.setAccelScale(accelCfg[fs_idx]);
00120
00121     ok &= imu.setAccelDLPF(dlpf_val, false);
00122
00123     ok &= imu.setAccelDivRate(cfg.div);
00124
00125     return ok;
00126 }
00127
00128 /**
00129  * @brief Verify the IMU identity register.
00130  */
00131 void firstStage(){
00132     uint8_t who;
00133     if (!imu.readWhoAmI(who) || who != 0xEA) {
00134         Serial.println(F("ERROR: WHO_AM_I mismatch"));
00135         while (1) delay(200);
00136     }
00137     Serial.print(F("WHO_AM_I = 0x"));
00138     Serial.println(who, HEX);
00139 }
00140
00141 /**
00142  * @brief Run the accelerometer DLPF validation sweep and print samples.
00143  */

```

```

00145 void runSweep(){
00146     printDobleLinea();
00147     Serial.println("Acelerometro ICM-200948 | UNIT Electronics");
00148     Serial.printf("%d DLPFS x %d FS = %d combinaciones\n",N_DLPF,N_FS,N_DLPF * N_FS);
00149     Serial.println(F("Coloque el sensor plano y quieto mirando hacia arriba"));
00150     printDobleLinea();
00151     for(uint8_t d = 0; d < N_DLPF; d++){
00152         Serial.printf(" BLOQUE %d/%d: %s\n", d+1, N_DLPF,configsDLPF[d].nombre);
00153         Serial.printf(" ODR = %.1fHz | Delay = %lums | Periodo=
%.1fms\n",configsDLPF[d].odr_hz,max((uint32_t)(10000.0f/configsDLPF[d].odr_hz),(uint32_t)50),
1000.0f/configsDLPF[d].odr_hz);
00154         printDobleLinea();
00155         bool cfg_ok = applySettings(configsDLPF[d].dlpf_idx, configsDLPF[d]);
00156         if (!cfg_ok) {
00157             Serial.println(F(" ERROR: applySettings() fallo"));
00158             continue;
00159         }
00160         // Esperar que el filtro estabilice (10 periodos del ODR, min 50ms)
00161         uint32_t settle_ms = max((uint32_t)(10000.0f / configsDLPF[d].odr_hz),
00162             (uint32_t)50);
00163         delay(settle_ms);
00164
00165         // Tomar 3 lecturas espaciadas por el periodo del ODR
00166         uint32_t periodo_ms = max((uint32_t)(1000.0f / configsDLPF[d].odr_hz),
00167             (uint32_t)1);
00168         float ax, ay, az;
00169         for (uint8_t r = 0; r < 3; r++) {
00170             delay(periodo_ms);
00171             if (imu.readAccel(ax, ay, az)) {
00172                 Serial.printf(" [%d] Ax=+.3f Ay=+.3f Az=+.3f g\n",
00173                     r+1, ax, ay, az);
00174             } else {
00175                 Serial.println(F(" [?] readAccel() fallo"));
00176             }
00177         }
00178         printLinea();
00179     }
00180 }
00181 }
00182 }
00183 /**
00184  * @brief Initialize I2C, reset/wake the IMU, enable sensors, and start sweep.
00185  */
00186 void setup() {
00187     Serial.begin(115200);
00188     delay(200);
00189     Serial.println(F("ICM-20948 - I2C Basic"));
00190     printDobleLinea();
00191     Serial.println(F(" ICM-20948 VALIDACION ACELEROMETRO"));
00192     printDobleLinea();
00193     // I2C init (UNO/ESP32 defaults). For ESP32 custom pins: Wire.begin(SDA,SCL);
00194     Wire.begin(SDA_PIN,SCL_PIN);
00195     // Attach IMU
00196     if (!imu.beginI2C(ICM_ADDR,Wire,I2C_FREQ)) {
00197         Serial.println(F("ERROR: begin(I2C) failed"));
00198         while (1) delay(200);
00199     }
00200     firstStage();
00201     imu.softReset();
00202     imu.sleep(false);
00203     delay(50);
00204     /* Enable all sensors. */
00205     if (!imu.setSensors(true, true, true)) {
00206         Serial.println(F("ERROR: setSensors failed"));
00207     }
00208     Serial.println(F(" Sensor listo.\n"));
00209     Serial.println(F(" Sensor PLANO y QUIETO sobre la mesa."));
00210     Serial.println(F(" Iniciando en 5 segundos..."));
00211     for (int i = 5; i > 0; i--) {
00212         Serial.printf(" %d...\n", i);
00213         delay(1000);
00214     }
00215     runSweep();
00216 }
00217 }
00218 /**
00219  * @brief Optional post-sweep loop for manual accelerometer reads.
00220  */
00221 void loop() {
00222     /*float ax, ay, az;
00223     if (imu.readAccel(ax, ay, az)) {
00224         Serial.printf("X:%.3f Y:%.3f Z:%.3f\n", ax, ay, az);
00225     }
00226     */
00227 }

```

```

00230     }
00231     delay(200); */
00232 }

```

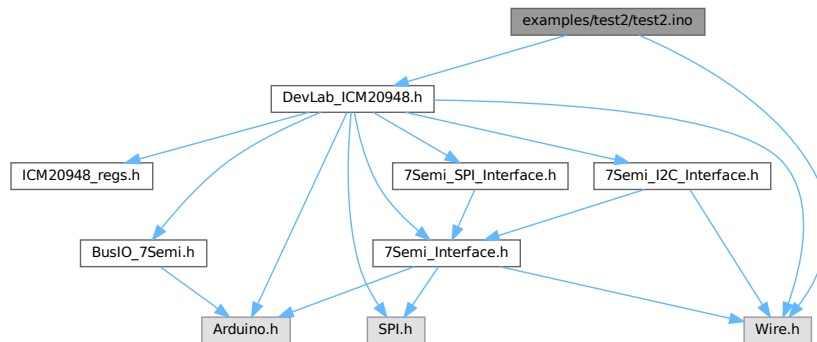
6.23 examples/test2/test2.ino File Reference

Extended accelerometer DLPF, full-scale, and averaging sweep.

```
#include <Wire.h>
```

```
#include <DevLab_ICM20948.h>
```

Include dependency graph for test2.ino:



Classes

- struct [configDLPF](#)
Gyroscope DLPF divider and timing configuration.

Macros

- #define [SDA_PIN](#) 6
I2C SDA pin used by the example board.
- #define [SCL_PIN](#) 7
I2C SCL pin used by the example board.
- #define [I2C_FREQ](#) 400000UL
I2C bus speed in hertz.
- #define [ICM_ADDR](#) 0x69
ICM-20948 I2C address selected by the AD0 pin.

Functions

- void [printLinea](#) ()
Print a separator line to Serial.
- void [printDobleLinea](#) ()
Print a double separator line to Serial.
- void [firstStage](#) ()
Verify the IMU identity register.
- bool [applySettings](#) (uint8_t fs_idx, const [configDLPF](#) &cfg, uint8_t avg)
Apply one accelerometer full-scale, DLPF, and averaging configuration.
- void [runSweep](#) ()
Run the full accelerometer validation matrix and print samples.
- void [setup](#) ()

Initialize I2C, reset/wake the IMU, enable sensors, and start sweep.

- void `loop` ()

Optional post-sweep loop for manual accelerometer reads.

Variables

- const `configDLPF configsDLPF` []
Accelerometer DLPF configurations exercised by this sweep.
- const uint8_t `N_DLPF` = sizeof(`configsDLPF`) / sizeof(`configsDLPF`[0])
Number of accelerometer DLPF configurations.
- const `ICM20948_Accel_FullScale accelCfg` []
Accelerometer full-scale ranges exercised by the sweep.
- const char * `etiquetasAccelCfg` [] = { "+-2G", "+-4G", "+-8G", "+-16G" }
Text labels matching accelCfg.
- const uint8_t `N_FS` = 4
Number of accelerometer full-scale ranges.
- const `ICM20948_Accel_DLPFCFG accelDLPF` []
Accelerometer DLPF enum values matching configsDLPF.
- const `ICM20948_Accel_Average accelAVG` []
Accelerometer averaging settings exercised by the sweep.
- const char * `etiquetasAccelAVG` [] = { "14", "8", "16", "32" }
Text labels matching accelAVG.
- const uint8_t `N_AVG` = 4
Number of accelerometer averaging settings.
- const uint16_t `accelSR` []
Raw accelerometer sample-rate divider values for experiments.
- const char * `etiquetasSR` []
Text labels matching accelSR output data rates.
- `DevLab_ICM20948 imu`
Global ICM-20948 driver instance.
- uint8_t `total_pass` = 0
Count of validation passes reserved for future checks.
- uint8_t `total_fail` = 0
Count of validation failures reserved for future checks.
- uint8_t `total_warn` = 0
Count of validation warnings reserved for future checks.

6.23.1 Detailed Description

Extended accelerometer DLPF, full-scale, and averaging sweep.

Author

Jonathan Mejorado Lopez

Runs a larger I2C validation matrix for the DevLab ICM-20948 accelerometer and prints CSV-like samples for later analysis.

Definition in file [test2.ino](#).

6.23.2 Macro Definition Documentation

6.23.2.1 I2C_FREQ

```
#define I2C_FREQ 400000UL
```

I2C bus speed in hertz.

Definition at line 20 of file [test2.ino](#).

6.23.2.2 ICM_ADDR

```
#define ICM_ADDR 0x69
```

ICM-20948 I2C address selected by the AD0 pin.

Definition at line 22 of file [test2.ino](#).

6.23.2.3 SCL_PIN

```
#define SCL_PIN 7
```

I2C SCL pin used by the example board.

Definition at line 18 of file [test2.ino](#).

6.23.2.4 SDA_PIN

```
#define SDA_PIN 6
```

I2C SDA pin used by the example board.

Definition at line 16 of file [test2.ino](#).

6.23.3 Function Documentation

6.23.3.1 applySettings()

```
bool applySettings (
    uint8_t fs_idx,
    const configDLPF & cfg,
    uint8_t avg )
```

Apply one accelerometer full-scale, DLPF, and averaging configuration.

Parameters

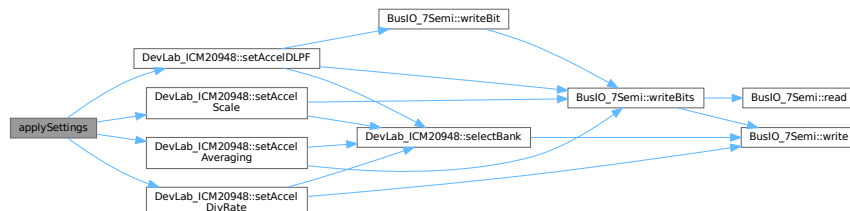
<i>fs_idx</i>	Index into accelCfg.
<i>cfg</i>	DLPF timing and divider configuration.
<i>avg</i>	Index into accelAVG.

Returns

true if all configuration writes succeeded, otherwise false.

Definition at line 150 of file [test2.ino](#).

Here is the call graph for this function:



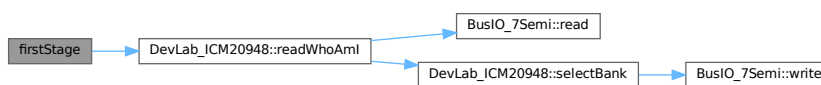
6.23.3.2 firstStage()

```
void firstStage ( )
```

Verify the IMU identity register.

Definition at line 124 of file [test2.ino](#).

Here is the call graph for this function:



Here is the caller graph for this function:



6.23.3.3 loop()

```
void loop ( )
```

Optional post-sweep loop for manual accelerometer reads.

Definition at line 299 of file [test2.ino](#).

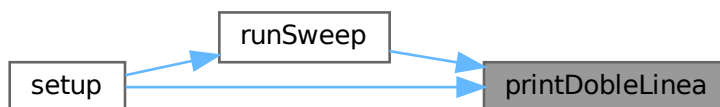
6.23.3.4 printDobleLinea()

```
void printDobleLinea ( )
```

Print a double separator line to Serial.

Definition at line 117 of file [test2.ino](#).

Here is the caller graph for this function:



6.23.3.5 printLinea()

```
void printLinea ( )
```

Print a separator line to Serial.

Definition at line 115 of file [test2.ino](#).

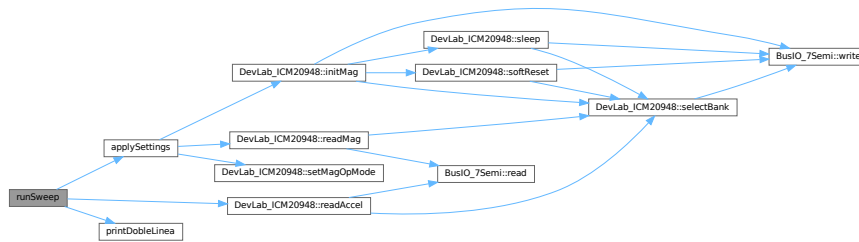
6.23.3.6 runSweep()

```
void runSweep ( )
```

Run the full accelerometer validation matrix and print samples.

Definition at line 184 of file test2.ino.

Here is the call graph for this function:



Here is the caller graph for this function:



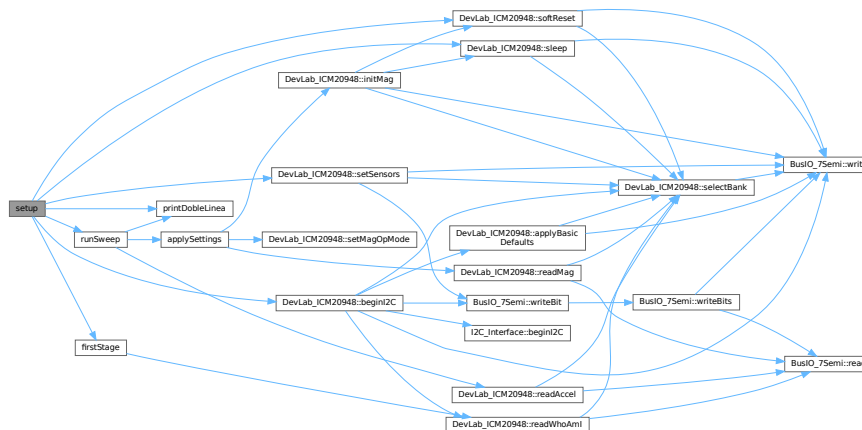
6.23.3.7 `setup()`

```
void setup ( )
```

Initialize I2C, reset/wake the IMU, enable sensors, and start sweep.

Definition at line 251 of file test2.ino.

Here is the call graph for this function:



6.23.4 Variable Documentation

6.23.4.1 accelAVG

```
const ICM20948_Accel_Average accelAVG[ ]
```

Initial value:

$$= \{$$

```

    ICM20948_Accel_Average::AVG_1_OR_4,
    ICM20948_Accel_Average::AVG_8,
    ICM20948_Accel_Average::AVG_16,
    ICM20948_Accel_Average::AVG_32
}

```

Accelerometer averaging settings exercised by the sweep.
Definition at line 83 of file [test2.ino](#).

6.23.4.2 accelCfg

```
const ICM20948_Accel_FullScale accelCfg[ ]
```

Initial value:

```

= {
    ICM20948_Accel_FullScale::G_2,
    ICM20948_Accel_FullScale::G_4,
    ICM20948_Accel_FullScale::G_8,
    ICM20948_Accel_FullScale::G_16
}

```

Accelerometer full-scale ranges exercised by the sweep.
Definition at line 59 of file [test2.ino](#).

6.23.4.3 accelDLPF

```
const ICM20948_Accel_DLPFCFG accelDLPF[ ]
```

Initial value:

```

= {
    ICM20948_Accel_DLPFCFG::DLPF0_246HZ,
    ICM20948_Accel_DLPFCFG::DLPF_246HZ,
    ICM20948_Accel_DLPFCFG::DLPF_111HZ,
    ICM20948_Accel_DLPFCFG::DLPF_50HZ,
    ICM20948_Accel_DLPFCFG::DLPF_24HZ,
    ICM20948_Accel_DLPFCFG::DLPF_12HZ,
    ICM20948_Accel_DLPFCFG::DLPF_6HZ,
    ICM20948_Accel_DLPFCFG::DLPF_473HZ
}

```

Accelerometer DLPF enum values matching configsDLPF.
Definition at line 71 of file [test2.ino](#).

6.23.4.4 accelSR

```
const uint16_t accelSR[ ]
```

Initial value:

```

= {
    0, 1, 3, 5, 7, 10, 15, 22, 31, 63, 127, 255, 513, 1022, 2044, 4095
}

```

Raw accelerometer sample-rate divider values for experiments.
Definition at line 94 of file [test2.ino](#).

6.23.4.5 configsDLPF

```
const configDLPF configsDLPF[ ]
```

Initial value:

```

= {
    { 0, 0, 265.0f, 1125.0f, "246 ; 265 ; 1125 " },
    { 1, 0, 265.0f, 1125.0f, "246 ; 265 ; 1125 " },
    { 2, 1, 136.0f, 562.5f, "111 ; 136 ; 562 " },
    { 3, 2, 68.8f, 375.0f, "50 ; 69 ; 375 " },
    { 4, 7, 34.4f, 140.6f, "24 ; 34 ; 141 " },
    { 5, 15, 17.0f, 70.3f, "12 ; 17 ; 70 " },
    { 6, 29, 8.3f, 37.5f, "6 ; 8 ; 35 " },
    { 7, 0, 499.0f, 1125.0f, "473 ; 499 ; 1125 " },
}

```

Accelerometer DLPF configurations exercised by this sweep.
Definition at line 42 of file [test2.ino](#).

6.23.4.6 etiquetasAccelAVG

```
const char* etiquetasAccelAVG[ ] = { "14", "8", "16", "32" }
```

Text labels matching accelAVG.

Definition at line 90 of file [test2.ino](#).

6.23.4.7 etiquetasAccelCfg

```
const char* etiquetasAccelCfg[] = { "+-2G", "+-4G", "+-8G", "+-16G" }
```

Text labels matching accelCfg.

Definition at line 66 of file [test2.ino](#).

6.23.4.8 etiquetasSR

```
const char* etiquetasSR[]
```

Initial value:

```
= {  
    "1125.0", "562.5", "281.3", "187.5", "140.6", "102.3", "70.3",  
    "48.9", "35.2", "17.6", "8.8", "4.4", "2.2", "1.1", "0.55", "0.27"  
}
```

Text labels matching accelSR output data rates.

Definition at line 98 of file [test2.ino](#).

6.23.4.9 imu

```
DevLab_ICM20948 imu
```

Global ICM-20948 driver instance.

Definition at line 105 of file [test2.ino](#).

6.23.4.10 N_AVG

```
const uint8_t N_AVG = 4
```

Number of accelerometer averaging settings.

Definition at line 92 of file [test2.ino](#).

6.23.4.11 N_DLPF

```
const uint8_t N_DLPF = sizeof(configsDLPF) / sizeof(configsDLPF[0])
```

Number of accelerometer DLPF configurations.

Definition at line 53 of file [test2.ino](#).

6.23.4.12 N_FS

```
const uint8_t N_FS = 4
```

Number of accelerometer full-scale ranges.

Definition at line 68 of file [test2.ino](#).

6.23.4.13 total_fail

```
uint8_t total_fail = 0
```

Count of validation failures reserved for future checks.

Definition at line 109 of file [test2.ino](#).

6.23.4.14 total_pass

```
uint8_t total_pass = 0
```

Count of validation passes reserved for future checks.

Definition at line 107 of file [test2.ino](#).

6.23.4.15 total_warn

```
uint8_t total_warn = 0
```

Count of validation warnings reserved for future checks.

Definition at line 111 of file [test2.ino](#).

6.24 test2.ino

[Go to the documentation of this file.](#)

```
00001 /**
00002  * @file test2.ino
00003  * @brief Extended accelerometer DLPF, full-scale, and averaging sweep.
00004  * @author Jonathan Mejorado Lopez
00005  * @details Runs a larger I2C validation matrix for the DevLab ICM-20948
00006  * accelerometer and prints CSV-like samples for later analysis.
00007  */
00008
00009 #include <Wire.h>
00010 #include <DevLab_ICM20948.h>
00011
00012 // -----
00013 // PINES I2C - ESP32C6 NANO Unit Electronics
00014 // -----
00015 /** @brief I2C SDA pin used by the example board. */
00016 #define SDA_PIN 6
00017 /** @brief I2C SCL pin used by the example board. */
00018 #define SCL_PIN 7
00019 /** @brief I2C bus speed in hertz. */
00020 #define I2C_FREQ 400000UL
00021 /** @brief ICM-20948 I2C address selected by the AD0 pin. */
00022 #define ICM_ADDR 0x69
00023
00024 // -----
00025 // STRUCT
00026 // -----
00027 /** @brief Accelerometer DLPF divider and timing configuration. */
00028 struct configDLPF {
00029     /** @brief Index into the accelDLPF lookup table. */
00030     uint8_t dlpf_idx;
00031     /** @brief Sample-rate divider written to the IMU register. */
00032     uint16_t div;
00033     /** @brief Noise bandwidth in hertz for reporting. */
00034     float nbw_hz;
00035     /** @brief Output data rate in hertz for timing calculations. */
00036     float odr_hz;
00037     /** @brief Text label printed with each captured sample. */
00038     const char* nombre;
00039 };
00040 //id,div,nbw_hz,odr_hz,nombre : {DLPF, NBW , ODR}
00041 /** @brief Accelerometer DLPF configurations exercised by this sweep. */
00042 const configDLPF configsDLPF[] = {
00043     { 0, 0, 265.0f, 1125.0f, "246 ; 265 ; 1125 " },
00044     { 1, 0, 265.0f, 1125.0f, "246 ; 265 ; 1125 " },
00045     { 2, 1, 136.0f, 562.5f, "111 ; 136 ; 562 " },
00046     { 3, 2, 68.8f, 375.0f, "50 ; 69 ; 375 " },
00047     { 4, 7, 34.4f, 140.6f, "24 ; 34 ; 141 " },
00048     { 5, 15, 17.0f, 70.3f, "12 ; 17 ; 70 " },
00049     { 6, 29, 8.3f, 37.5f, "6 ; 8 ; 35 " },
00050     { 7, 0, 499.0f, 1125.0f, "473 ; 499 ; 1125 " },
00051 };
00052 /** @brief Number of accelerometer DLPF configurations. */
00053 const uint8_t N_DLPF = sizeof(configsDLPF) / sizeof(configsDLPF[0]);
00054
00055 // -----
00056 // ARRAYS ORIGINALES
00057 // -----
00058 /** @brief Accelerometer full-scale ranges exercised by the sweep. */
00059 const ICM20948_Accel_FullScale accelCfg[] = {
00060     ICM20948_Accel_FullScale::G_2,
00061     ICM20948_Accel_FullScale::G_4,
00062     ICM20948_Accel_FullScale::G_8,
00063     ICM20948_Accel_FullScale::G_16
00064 };
00065 /** @brief Text labels matching accelCfg. */
00066 const char* etiquetasAccelCfg[] = { "+-2G", "+-4G", "+-8G", "+-16G" };
00067 /** @brief Number of accelerometer full-scale ranges. */
00068 const uint8_t N_FS = 4;
00069
00070 /** @brief Accelerometer DLPF enum values matching configsDLPF. */
00071 const ICM20948_Accel_DLPFCFG accelDLPF[] = {
00072     ICM20948_Accel_DLPFCFG::DLPF0_246HZ,
00073     ICM20948_Accel_DLPFCFG::DLPF_246HZ,
00074     ICM20948_Accel_DLPFCFG::DLPF_111HZ,
00075     ICM20948_Accel_DLPFCFG::DLPF_50HZ,
00076     ICM20948_Accel_DLPFCFG::DLPF_24HZ,
00077     ICM20948_Accel_DLPFCFG::DLPF_12HZ,
00078     ICM20948_Accel_DLPFCFG::DLPF_6HZ,
00079     ICM20948_Accel_DLPFCFG::DLPF_473HZ
00080 };
00081
00082 /** @brief Accelerometer averaging settings exercised by the sweep. */
00083 const ICM20948_Accel_Average accelAVG[] = {
```

```

00084 ICM20948_Accel_Average::AVG_1_OR_4,
00085 ICM20948_Accel_Average::AVG_8,
00086 ICM20948_Accel_Average::AVG_16,
00087 ICM20948_Accel_Average::AVG_32
00088 };
00089 /** @brief Text labels matching accelAVG. */
00090 const char* etiquetasAccelAVG[] = { "14", "8", "16", "32" };
00091 /** @brief Number of accelerometer averaging settings. */
00092 const uint8_t N_AVG = 4;
00093 /** @brief Raw accelerometer sample-rate divider values for experiments. */
00094 const uint16_t accelSR[] = {
00095     0, 1, 3, 5, 7, 10, 15, 22, 31, 63, 127, 255, 513, 1022, 2044, 4095
00096 };
00097 /** @brief Text labels matching accelSR output data rates. */
00098 const char* etiquetasSR[] = {
00099     "1125.0", "562.5", "281.3", "187.5", "140.6", "102.3", "70.3",
00100     "48.9", "35.2", "17.6", "8.8", "4.4", "2.2", "1.1", "0.55", "0.27"
00101 };
00102
00103 // -----
00104 /** @brief Global ICM-20948 driver instance. */
00105 DevLab_ICM20948 imu;
00106 /** @brief Count of validation passes reserved for future checks. */
00107 uint8_t total_pass = 0;
00108 /** @brief Count of validation failures reserved for future checks. */
00109 uint8_t total_fail = 0;
00110 /** @brief Count of validation warnings reserved for future checks. */
00111 uint8_t total_warn = 0;
00112
00113
00114 /** @brief Print a separator line to Serial. */
00115 void printLinea() {
00116     Serial.println(F("-----"));
00117 }
00118 /** @brief Print a double separator line to Serial. */
00119 void printDobleLinea() {
00120     Serial.println(F("====="));
00121 }
00122 // -----
00123 // WHO_AM_I
00124 // -----
00125 /**
00126  * @brief Verify the IMU identity register.
00127  */
00128 void firstStage() {
00129     uint8_t who;
00130     if (!imu.readWhoAmI(who) || who != 0xEA) {
00131         Serial.println(F("ERROR: WHO_AM_I mismatch"));
00132         while (1) delay(200);
00133     }
00134     Serial.printf("WHO_AM_I = 0x%02X OK\n", who);
00135 }
00136
00137 // -----
00138 // APLICAR CONFIGURACION
00139 //
00140 // FIX 1: Se agregan ambas llamadas a setAccelDLPF:
00141 // Llamada 1 (bypass=false): escribe bits DLPFCFG, deja FCHOICE=0
00142 // Llamada 2 (bypass=true): activa FCHOICE=1 sin tocar DLPFCFG
00143 //
00144 // FIX 2: setAccelDivRate escribe el DIV directo al registro
00145 // -----
00146 /**
00147  * @brief Apply one accelerometer full-scale, DLPF, and averaging configuration.
00148  *
00149  * @param fs_idx Index into accelCfg.
00150  * @param cfg DLPF timing and divider configuration.
00151  * @param avg Index into accelAVG.
00152  * @return true if all configuration writes succeeded, otherwise false.
00153  */
00154 bool applySettings(uint8_t fs_idx, const configDLPF &cfg, uint8_t avg) {
00155     uint8_t dlpf_val = (uint8_t)accelDLPF[cfg.dlpf_idx];
00156     bool ok = true;
00157
00158     ok &= imu.setAccelScale(accelCfg[fs_idx]);
00159
00160     // Paso 1: escribe DLPFCFG en bits [5:3]
00161     ok &= imu.setAccelDLPF(dlpf_val, false);
00162
00163     ok &= imu.setAccelAveraging(accelAVG[avg]);
00164     // Paso 2: activa FCHOICE=1 (DLPF activo) sin tocar DLPFCFG
00165     ok &= imu.setAccelDLPF(0, true);
00166
00167     // DIV directo al registro, sin conversion interna
00168     ok &= imu.setAccelDivRate(cfg.div);
00169
00170     return ok;
00171 }

```

```

00169
00170 // -----
00171 // SWEEP - MODO VERIFICACION
00172 //
00173 // Por cada configuracion DLPF:
00174 // 1. Aplica la config (FS fijo en +-2g para verificacion)
00175 // 2. Espera settle del filtro
00176 // 3. Imprime 3 lecturas reales
00177 //
00178 // Cuando las lecturas muestren Az ~1g en reposo,
00179 // el sensor esta funcionando y se puede activar el sweep completo.
00180 // -----
00181 /**
00182  * @brief Run the full accelerometer validation matrix and print samples.
00183  */
00184 void runSweep() {
00185
00186     uint16_t n = 0;
00187     printDobleLinea();
00188     Serial.println(F(" ICM-20948 | UNIT Electronics"));
00189     Serial.println(F(" MODO VERIFICACION - FS=+-2g fijo, 3 lecturas por config"));
00190     Serial.println(F(" Sensor PLANO y QUIETO | Az esperado: ~+1.000g"));
00191     printDobleLinea();
00192     for(uint8_t i = 0; i < N_AVG; i++){
00193         for(uint8_t j = 0; j < N_FS; j++){
00194             for (uint8_t d = 0; d < N_DLPF; d++) {
00195
00196                 //Serial.println();
00197                 //printLinea();
00198                 /*Serial.printf(" Config %d/%d: %s\n", d+1, N_DLPF, configsDLPF[d].nombre);
00199                 Serial.printf(" ODR=%.1fHz | Periodo=%.1fms\n",
00200                     configsDLPF[d].odr_hz,
00201                     1000.0f / configsDLPF[d].odr_hz);*/
00202
00203                 // --- FIX: aplicar la configuracion antes de leer ---
00204                 bool cfg_ok = applySettings(j, configsDLPF[d], i); // FS=0 → +-2g
00205                 if (!cfg_ok) {
00206                     Serial.println(F(" ERROR: applySettings() fallo"));
00207                     continue;
00208                 }
00209
00210                 // Esperar que el filtro estabilice (10 periodos del ODR, min 50ms)
00211                 uint32_t settle_ms = max((uint32_t)(10000.0f / configsDLPF[d].odr_hz),
00212                     (uint32_t)50);
00213                 delay(settle_ms);
00214
00215                 // Tomar 3 lecturas espaciadas por el periodo del ODR
00216                 uint32_t periodo_ms = max((uint32_t)(1000.0f / configsDLPF[d].odr_hz),
00217                     (uint32_t)1);
00218                 float ax, ay, az;
00219                 for (uint8_t r = 0; r < 100; r++) {
00220                     n++;
00221                     delay(periodo_ms);
00222                     if (imu.readAccel(ax, ay, az)) {
00223                         Serial.printf("%d ; %s ; %s ; %s ; %.4f ; %.4f ; %.4f\n",
00224                             n,
00225                             configsDLPF[d].nombre, // "246 ; 265 ; 1125"
00226                             etiquetasAccelCfg[i], // "+-2g"
00227                             etiquetasAccelAVG[j], // "8"
00228                             ax, ay, az);
00229                     } else {
00230                         Serial.println(F(" [?] readAccel() fallo"));
00231                     }
00232                 }
00233             }
00234         }
00235     }
00236     Serial.println();
00237     printDobleLinea();
00238     Serial.println(F(" VERIFICACION COMPLETA"));
00239     Serial.println(F(" Si Az ~ +1.0g en todos los bloques:"));
00240     Serial.println(F(" el sensor funciona correctamente.));
00241     Serial.println(F(" Activar sweep completo cuando este listo.));
00242     printDobleLinea();
00243 }
00244
00245 // -----
00246 // SETUP
00247 // -----
00248 /**
00249  * @brief Initialize I2C, reset/wake the IMU, enable sensors, and start sweep.
00250  */
00251 void setup() {
00252     Serial.begin(115200);
00253     delay(200);
00254     Serial.println(F("ICM-20948 - Verificacion"));
00255

```

```

00256     printDobleLinea();
00257     Serial.println(F("   ICM-20948 VALIDACION ACELEROMETRO"));
00258     printDobleLinea();
00259
00260     Wire.begin(SDA_PIN, SCL_PIN);
00261
00262     if (!imu.beginI2C(ICM_ADDR, Wire, I2C_FREQ)) {
00263         Serial.println(F("ERROR: beginI2C() fallo"));
00264         while (1) delay(200);
00265     }
00266
00267     firstStage();
00268
00269     // softReset pone el chip a dormir
00270     // Despues del reset hay que despertar el dispositivo
00271     imu.softReset();
00272     delay(100);
00273     imu.sleep(false);    // limpia SLEEP bit, CLKSEL=1
00274     delay(50);
00275
00276     if (!imu.setSensors(true, true, true)) {
00277         Serial.println(F("ERROR: setSensors() fallo"));
00278     }
00279
00280     Serial.println(F("   Sensor listo.\n"));
00281     Serial.println(F("   Sensor PLANO y QUIETO sobre la mesa.));
00282     Serial.println(F("   Iniciando en 5 segundos..."));
00283     for (int i = 5; i > 0; i--) {
00284         Serial.printf("   %d...\n", i);
00285         delay(1000);
00286     }
00287
00288     //for (int runs = 0; runs < 20 ; runs++){
00289         runSweep();
00290     //}
00291 }
00292
00293 // -----
00294 // LOOP – lectura simple post-sweep
00295 // -----
00296 /**
00297  * @brief Optional post-sweep loop for manual accelerometer reads.
00298  */
00299 void loop() {
00300     /*float ax, ay, az;
00301     if (imu.readAccel(ax, ay, az)) {
00302         Serial.printf("X:%.3f Y:%.3f Z:%.3f\n", ax, ay, az);
00303     }
00304     delay(500);*/
00305 }

```

6.25 README.md File Reference

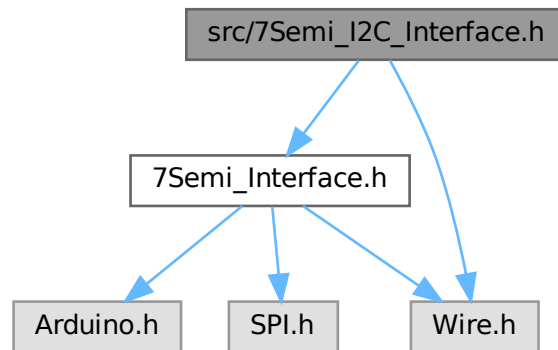
6.26 src/7Semi_I2C_Interface.h File Reference

```

#include "7Semi_Interface.h"
#include <Wire.h>

```

Include dependency graph for 7Semi_I2C_Interface.h:



This graph shows which files directly or indirectly include this file:



Classes

- class [I2C_Interface](#)

6.27 7Semi_I2C_Interface.h

[Go to the documentation of this file.](#)

```

00001 #pragma once
00002 #include "7Semi_Interface.h"
00003 #include <Wire.h>
00004
00005 class I2C_Interface : public Interface_7Semi
00006 {
00007 public:
00008     TwoWire *i2c = nullptr;
00009     uint8_t address = 0;
00010
00011     bool beginI2C(uint8_t addr, TwoWire &wire, uint32_t speed,
00012                 uint8_t sda = 255, uint8_t scl = 255)
00013     {
00014         address = addr;
00015         i2c = &wire;
00016
00017         #if defined(ESP32)
00018             if (sda != 255 && scl != 255)
00019                 i2c->begin(sda, scl);
00020             else
00021                 i2c->begin();
00022         #else
00023             i2c->begin();
00024         #endif
00025         // i2c->setClock(speed);
00026
00027         // Probe device
00028         i2c->beginTransmission(address);
00029         return (i2c->endTransmission() == 0);
00030     }
00031
00032     int8_t read(uint8_t reg, uint8_t *data, uint32_t len) override
00033     {

```

```

00034         if (!i2c)
00035             return -1;
00036         i2c->beginTransaction(address);
00037         i2c->write(reg );
00038
00039         if (i2c->endTransmission(false) != 0)
00040             return -1;
00041
00042         delay(1);
00043
00044         if (len > 255)
00045             return -3;
00046         uint8_t received = i2c->requestFrom(address, (uint8_t)len);
00047         if (received != len)
00048             return -2;
00049
00050         for (uint32_t i = 0; i < len; i++)
00051         {
00052             if (!i2c->available())
00053                 return -4;
00054             data[i] = i2c->read();
00055         }
00056         // Serial.print("R-Reg: ");
00057         // Serial.print(reg, HEX);
00058         // Serial.print(" |Data: ");
00059         // for (int i = 0; i < len; i++)
00060         // {
00061         //     Serial.print(" ");
00062         //     Serial.print(data[i], HEX);
00063         // }
00064         // Serial.println();
00065         return 0;
00066     }
00067 }
00068
00069 bool beginSPI(uint8_t, SPIClass &, uint32_t,
00070              uint8_t, uint8_t, uint8_t) override
00071 {
00072     return false;
00073 }
00074
00075 int8_t write(uint8_t reg, const uint8_t *data, uint32_t len) override
00076 {
00077     if (!i2c)
00078         return -1;
00079
00080     i2c->beginTransaction(address);
00081     i2c->write(reg);
00082     for (uint32_t i = 0; i < len; i++)
00083         i2c->write(data[i]);
00084
00085     uint8_t status = i2c->endTransmission();
00086
00087     // Serial.print("W-Reg: ");
00088     // Serial.print(reg, HEX);
00089     // Serial.print(" |Data: ");
00090     // for (int i = 0; i < len; i++)
00091     // {
00092     //     Serial.print(" ");
00093     //     Serial.print(data[i], HEX);
00094     // }
00095     // Serial.println();
00096     return (status == 0) ? 0 : -status;
00097 }
00098 };

```

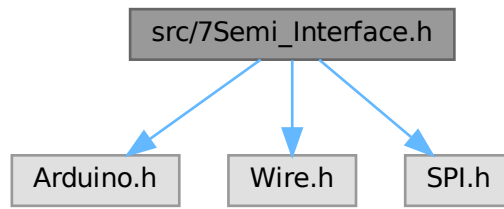
6.28 src/7Semi_Interface.h File Reference

```

#include <Arduino.h>
#include <Wire.h>
#include <SPI.h>

```

Include dependency graph for 7Semi_Interface.h:



This graph shows which files directly or indirectly include this file:



Classes

- class [Interface_7Semi](#)

6.29 7Semi_Interface.h

Go to the documentation of this file.

```

00001 // #pragma once
00002 #ifndef INTERFACE_7SEMI_H
00003 #define INTERFACE_7SEMI_H
00004
00005 #include <Arduino.h>
00006 #include <Wire.h>
00007 #include <SPI.h>
00008
00009 /**
00010  * 7Semi Universal Interface Layer
00011  *
00012  * - Abstract communication layer for I2C / SPI
00013  * - Ensures sensor drivers remain hardware independent
00014  */
00015 class Interface_7Semi {
00016 public:
00017     virtual ~Interface_7Semi() {}
00018
00019     /**
00020      * beginI2C()
00021      *
00022      * - Initializes I2C peripheral
00023      * - Configures SDA / SCL pins if supported
00024      * - Sets communication clock
00025      *
00026      * Returns:
00027      * - true → Initialization successful
00028      * - false → Initialization failed
00029      */
00030     virtual bool beginI2C(
00031         uint8_t address,
00032         TwoWire &wire,
00033         uint32_t speed,
00034         uint8_t sda = 255,
00035         uint8_t scl = 255) = 0;
00036
00037     virtual bool beginSPI(
00038         uint8_t cs_pin,
  
```

```

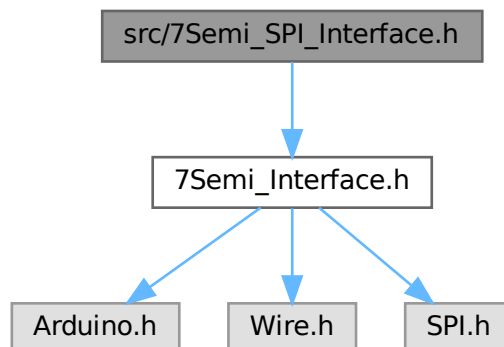
00040     SPIClass &wire,
00041     uint32_t speed,
00042     uint8_t csk = 255,
00043     uint8_t miso = 255,
00044     uint8_t mosi = 255) = 0;
00045
00046
00047     /**
00048     * read()
00049     *
00050     * - Reads data from device register
00051     *
00052     * Parameters:
00053     * - reg : Register address
00054     * - data : Output buffer
00055     * - len : Number of bytes
00056     *
00057     * Returns:
00058     * - 0 → Success
00059     * - <0 → Error
00060     */
00061     virtual int8_t read(
00062         uint8_t reg,
00063         uint8_t *data,
00064         uint32_t len) = 0;
00065
00066     /**
00067     * write()
00068     *
00069     * - Writes data to device register
00070     *
00071     * Returns:
00072     * - 0 → Success
00073     * - <0 → Error
00074     */
00075     virtual int8_t write(
00076         uint8_t reg,
00077         const uint8_t *data,
00078         uint32_t len) = 0;
00079
00080 protected:
00081     /** Delay wrapper */
00082     static void delay_us(uint32_t us)
00083     {
00084         delayMicroseconds(us);
00085     }
00086 };
00087 #endif

```

6.30 src/7Semi_SPI_Interface.h File Reference

#include "7Semi_Interface.h"

Include dependency graph for 7Semi_SPI_Interface.h:



- class `SPI_Interface`

[Go to the documentation of this file.](#)

Generated on Wed Jun 17 2026 22:32:52 for DevLab_ICM20948 by Doxygen

```

00066
00067     SPISettings settings(speed, MSBFIRST, SPI_MODE0);
00068
00069     spi->beginTransaction(settings);
00070     digitalWrite(cs, LOW);
00071     delayMicroseconds(10);
00072
00073     /**
00074      * Send register address (read)
00075      */
00076     spi->transfer(reg | 0x80);
00077
00078     delayMicroseconds(10);
00079
00080     /**
00081      * Read data
00082      */
00083     for (uint32_t i = 0; i < len; i++)
00084         data[i] = spi->transfer(0x00);
00085
00086     digitalWrite(cs, HIGH);
00087     delayMicroseconds(10);
00088     spi->endTransaction();
00089
00090     // Serial.print("R-Reg: ");
00091     // Serial.print(reg, HEX);
00092     // Serial.print(" |Data: ");
00093     // for (int i = 0; i < len; i++)
00094     // {
00095     //     Serial.print(" ");
00096     //     Serial.print(data[i], HEX);
00097     // }
00098     // Serial.println();
00099
00100     return 0;
00101 }
00102
00103 /**
00104  * write()
00105  *
00106  * - Write multiple bytes to register
00107  */
00108 int8_t write(uint8_t reg, const uint8_t *data, uint32_t len) override
00109 {
00110     if (!spi || !data || len == 0)
00111         return -1;
00112
00113     SPISettings settings(speed, MSBFIRST, SPI_MODE0);
00114
00115     spi->beginTransaction(settings);
00116     digitalWrite(cs, LOW);
00117     delayMicroseconds(10);
00118
00119     /**
00120      * Send register address (write)
00121      */
00122     spi->transfer(reg & 0x7F);
00123
00124     /**
00125      * Write data
00126      */
00127     for (uint32_t i = 0; i < len; i++)
00128         spi->transfer(data[i]);
00129
00130     digitalWrite(cs, HIGH);
00131     delayMicroseconds(10);
00132     spi->endTransaction();
00133
00134     // Serial.print("W-Reg: ");
00135     // Serial.print(reg, HEX);
00136     // Serial.print(" |Data: ");
00137     // for (int i = 0; i < len; i++)
00138     // {
00139     //     Serial.print(" ");
00140     //     Serial.print(data[i], HEX);
00141     // }
00142     // Serial.println();
00143
00144     return 0;
00145 }
00146 };

```



```

00037
00038 /** write 8-bit */
00039 inline bool write(uint8_t reg, uint8_t value)
00040 {
00041     return (bus.write(reg, &value, 1) == 0);
00042 }
00043
00044 /** write 16-bit */
00045 inline bool write(uint8_t reg, uint16_t value)
00046 {
00047     uint8_t data[2] = {(uint8_t)(value >> 8), (uint8_t)(value & 0xFF)};
00048     return (bus.write(reg, data, 2) == 0);
00049 }
00050
00051 /** write burst */
00052 inline bool write(uint8_t reg, const uint8_t *data, uint32_t len)
00053 {
00054     return (bus.write(reg, data, len) == 0);
00055 }
00056
00057 /**
00058  * readBits (8-bit register)
00059  */
00060 bool readBits(uint8_t reg, uint8_t pos, uint8_t len, uint8_t &value)
00061 {
00062     if (len == 0 || len > 8 || (pos + len) > 8)
00063         return false;
00064
00065     uint8_t reg_val;
00066     if (!read(reg, reg_val))
00067         return false;
00068
00069     uint8_t mask = ((uint8_t)1 << len) - 1;
00070
00071     value = (reg_val >> pos) & mask;
00072     return true;
00073 }
00074
00075 /**
00076  * readBits (16-bit register)
00077  */
00078 bool readBits(uint8_t reg, uint8_t pos, uint8_t len, uint16_t &value)
00079 {
00080     if (len == 0 || len > 16 || (pos + len) > 16)
00081         return false;
00082
00083     uint8_t reg_val;
00084     if (!read(reg, reg_val))
00085         return false;
00086
00087     uint16_t mask = ((uint16_t)1 << len) - 1;
00088
00089     value = (reg_val >> pos) & mask;
00090     return true;
00091 }
00092
00093 /**
00094  * writeBits (8-bit register)
00095  *
00096  * - Modifies specific bits in 8-bit register
00097  */
00098 bool writeBits(uint8_t reg, uint8_t pos, uint8_t len, uint8_t value)
00099 {
00100     if (len == 0 || len > 8 || (pos + len) > 8)
00101         return false;
00102
00103     uint8_t reg_val;
00104     if (!read(reg, reg_val))
00105         return false;
00106
00107     uint8_t mask = ((uint8_t)1 << len) - 1;
00108
00109     reg_val &= ~(mask << pos);
00110     reg_val |= (value & mask) << pos;
00111
00112     return write(reg, reg_val);
00113 }
00114
00115 /**
00116  * writeBits (16-bit register)
00117  *
00118  * - Modifies specific bits in 16-bit register
00119  */
00120 bool writeBits(uint8_t reg, uint8_t pos, uint8_t len, uint16_t value)
00121 {
00122     if (len == 0 || len > 16 || (pos + len) > 16)
00123         return false;

```

```

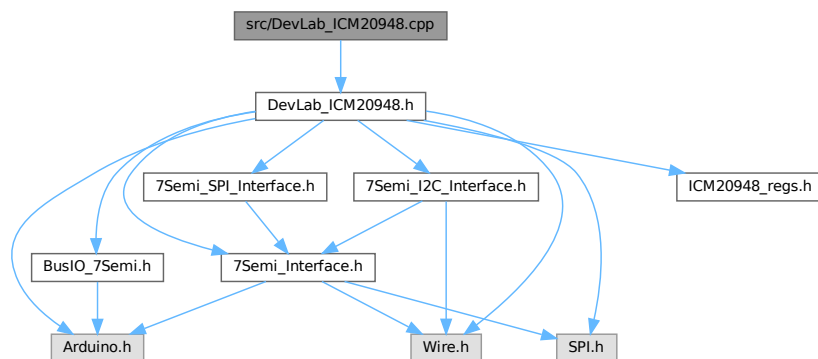
00124
00125     uint8_t reg_val;
00126     if (!read(reg, reg_val))
00127         return false;
00128
00129     uint16_t mask = ((uint16_t)1 << len) - 1;
00130
00131     reg_val &= ~(mask << pos);
00132     reg_val |= (value & mask) << pos;
00133
00134     return write(reg, reg_val);
00135 }
00136
00137 bool readBit(uint8_t reg, uint8_t pos, uint8_t &value)
00138 {
00139     return readBits(reg, pos, 1, value);
00140 }
00141
00142 bool readBit(uint8_t reg, uint8_t pos, uint16_t &value)
00143 {
00144     return readBits(reg, pos, 1, value);
00145 }
00146
00147 bool writeBit(uint8_t reg, uint8_t pos, uint8_t value)
00148 {
00149     return writeBits(reg, pos, 1, (uint8_t)value );
00150 }
00151 bool writeBit(uint8_t reg, uint8_t pos, uint16_t value)
00152 {
00153     return writeBits(reg, pos, 1, value);
00154 }
00155
00156 private:
00157     Bus &bus;
00158 };

```

6.34 src/DevLab_ICM20948.cpp File Reference

```
#include "DevLab_ICM20948.h"
```

Include dependency graph for DevLab_ICM20948.cpp:



6.35 DevLab_ICM20948.cpp

[Go to the documentation of this file.](#)

```

00001 #include "DevLab_ICM20948.h"
00002
00003 /** - Construct with default I2C address; call begin() to attach bus */
00004 DevLab_ICM20948::DevLab_ICM20948() {}
00005
00006 bool DevLab_ICM20948::beginI2C(uint8_t address, TwoWire &i2cPort, uint32_t i2cSpeed)
00007 {
00008     // Free previous BusIO instance
00009     if (bus)
00010     {
00011         delete bus;
00012         bus = nullptr;

```

```

00013     }
00014
00015     // Assign I2C interface implementation
00016     iface = &i2c;
00017
00018     // Initialize I2C (400kHz) and probe device
00019     if (!i2c.beginI2C(address, i2cPort, i2cSpeed))
00020         return false;
00021
00022     // Create BusIO abstraction layer
00023     bus = new BusIO_7Semi<Interface_7Semi>(*iface);
00024     if (!bus)
00025         return false;
00026
00027     // Allow device to stabilize after power-up
00028     delay(100);
00029
00030     // Read WHO_AM_I register
00031     uint8_t who_am_i;
00032     if (!readWhoAmI(who_am_i))
00033         return false;
00034
00035     // Validate device ID (expected 0xEA)
00036     if (who_am_i != WHO_AM_I_VAL)
00037         return false;
00038
00039     if (!selectBank(2))
00040         return false;
00041
00042     if (!bus->write(ODR_ALIGN_EN, (uint8_t)0x01))
00043         return false;
00044
00045     // Enable I2C interface (I2C_IF_DIS = 0)
00046     if (!bus->writeBit(USER_CTRL, 4, (uint8_t)0x00))
00047         return false;
00048
00049     // Set clock source to PLL (auto select best clock)
00050     if (!bus->write(PWR_MGMT_1, (uint8_t)0x01))
00051         return false;
00052
00053     if(!applyBasicDefaults())
00054         return false;
00055
00056     // Initialization successful
00057     return true;
00058 }
00059
00060 bool DevLab_ICM20948::beginSPI(uint8_t csPin, SPIClass &spiPort, uint32_t spiSpeed)
00061 {
00062     // Free previous BusIO instance
00063     if (bus)
00064     {
00065         delete bus;
00066         bus = nullptr;
00067     }
00068
00069     // Assign SPI interface implementation
00070     iface = &spi;
00071
00072     // Initialize SPI (1MHz) and probe device
00073     if (!spi.beginSPI(csPin, spiPort, spiSpeed))
00074         return false;
00075
00076     // Create BusIO abstraction layer
00077     bus = new BusIO_7Semi<Interface_7Semi>(*iface);
00078     if (!bus)
00079         return false;
00080
00081     softReset();
00082
00083     // Allow device to stabilize after power-up
00084     delay(100);
00085
00086     // Read WHO_AM_I register
00087     uint8_t who_am_i;
00088
00089     if (!readWhoAmI(who_am_i))
00090         return false;
00091
00092     // Validate device ID (expected 0xEA)
00093     if (who_am_i != WHO_AM_I_VAL)
00094         return false;
00095
00096     if (!selectBank(2))
00097         return false;
00098
00099

```

```

00100     if (!bus->write(ODR_ALIGN_EN, (uint8_t)0x01))
00101         return false;
00102
00103     // Enable I2C interface (I2C_IF_DIS = 0)
00104     if (!bus->writeBit(USER_CTRL, 4, (uint8_t)0x01))
00105         return false;
00106
00107     // Wake device from sleep mode (SLEEP = 0)
00108     if (!bus->write(PWR_MGMT_1, (uint8_t)0x01))
00109         return false;
00110
00111     if(!applyBasicDefaults())
00112         return false;
00113
00114     // Initialization successful
00115     return true;
00116 }
00117
00118 bool DevLab_ICM20948::readWhoAmI(uint8_t &whoAmI)
00119 {
00120     // Select USER BANK 0
00121     if (!selectBank(0))
00122         return false;
00123
00124     // Read WHO_AM_I register
00125     if (!bus->read(WHO_AM_I, whoAmI))
00126         return false;
00127
00128     // Success
00129     return true;
00130 }
00131
00132 bool DevLab_ICM20948::selectBank(uint8_t bank)
00133 {
00134     if (bank > 3)
00135         return false;
00136
00137     // Mask bank value to valid range (0-3) and shift to bits [5:4]
00138     uint8_t v = (bank & 0x03) << 4;
00139
00140     // Write bank selection to REG_BANK_SEL register
00141     return bus->write(REG_BANK_SEL, v);
00142 }
00143
00144 bool DevLab_ICM20948::softReset()
00145 {
00146     // Select USER BANK 0
00147     if (!selectBank(0))
00148         return false;
00149
00150     // Set DEVICE_RESET bit (bit 7)
00151     if (!bus->write(PWR_MGMT_1, (uint8_t)0x80))
00152         return false;
00153
00154     // Allow device to reset
00155     delay(100);
00156
00157     selectBank(2);
00158     bus->write(ODR_ALIGN_EN, (uint8_t)0x01);
00159
00160     // Success
00161     return true;
00162 }
00163
00164 bool DevLab_ICM20948::sleep(bool en)
00165 {
00166     // Select USER BANK 0
00167     if (!selectBank(0))
00168         return false;
00169     uint8_t v = 1;
00170     if(en)
00171         v|= 1<<6;
00172
00173     // Set or clear SLEEP bit (bit 6)
00174     if (!bus->write(PWR_MGMT_1, v))
00175         return false;
00176
00177     // Success
00178     return true;
00179 }
00180
00181 bool DevLab_ICM20948::applyBasicDefaults()
00182 {
00183     // Select USER BANK 0
00184     if (!selectBank(0))
00185         return false;
00186

```

```

00187 // Wake device and set clock source (CLKSEL=1, SLEEP=0)
00188 if (!bus->write(PWR_MGMT_1, (uint8_t)0x01))
00189     return false;
00190
00191 if (!bus->write(PWR_MGMT_2, (uint8_t)0x00))
00192     return false;
00193
00194 // Allow device to stabilize
00195 delay(10);
00196
00197 // Disable duty-cycling (LP mode off)
00198 if (!bus->write(LP_CONFIG, (uint8_t)0x00))
00199     return false;
00200
00201 // Select USER BANK 2
00202 if (!selectBank(2))
00203     return false;
00204
00205 // Configure gyro (DLPF enabled, FS = 2000 dps)
00206 if (!bus->write(GYRO_CONFIG_1, (uint8_t)0x1F))
00207     return false;
00208
00209 // Set gyro sample rate divider (SRD = 0 → max rate)
00210 if (!bus->write(GYRO_SMPLRT_DIV, (uint8_t)0x00))
00211     return false;
00212
00213 // Configure accel (DLPF enabled, FS = 16g)
00214 if (!bus->write(ACCEL_CONFIG, (uint8_t)0x1F))
00215     return false;
00216
00217 // Set accel sample rate divider high byte
00218 if (!bus->write(ACCEL_SMPLRT_DIV_1, (uint8_t)0x00))
00219     return false;
00220
00221 // Set accel sample rate divider low byte
00222 if (!bus->write(ACCEL_SMPLRT_DIV_2, (uint8_t)0x00))
00223     return false;
00224
00225 // Select USER BANK 0
00226 if (!selectBank(0))
00227     return false;
00228
00229 // Configure interrupt pin (open-drain, active-low, latch)
00230 if (!bus->write(INT_PIN_CFG, (uint8_t)0x30))
00231     return false;
00232
00233 // Configuration successful
00234 return true;
00235 }
00236 }
00237
00238 bool DevLab_ICM20948::readAccel(float &x, float &y, float &z)
00239 {
00240     // Select USER BANK 0
00241     if (!selectBank(0))
00242         return false;
00243
00244     // Read 6 bytes (X, Y, Z)
00245     uint8_t raw[6];
00246     if (!bus->read(ACCEL_XOUT_H, raw, 6))
00247         return false;
00248
00249     // Convert raw data to g
00250     x = (int16_t)((raw[0] << 8) | raw[1]) / mg_per_lsb;
00251     y = (int16_t)((raw[2] << 8) | raw[3]) / mg_per_lsb;
00252     z = (int16_t)((raw[4] << 8) | raw[5]) / mg_per_lsb;
00253
00254     // Success
00255     return true;
00256 }
00257
00258 bool DevLab_ICM20948::readGyro(float &x, float &y, float &z)
00259 {
00260     // Select USER BANK 0
00261     if (!selectBank(0))
00262         return false;
00263
00264     // Read 6 bytes (X, Y, Z)
00265     uint8_t raw[6];
00266     if (!bus->read(GYRO_XOUT_H, raw, 6))
00267         return false;
00268
00269     // Convert raw data to dps
00270     x = (int16_t)((raw[0] << 8) | raw[1]) / degree_per_second;
00271     y = (int16_t)((raw[2] << 8) | raw[3]) / degree_per_second;
00272     z = (int16_t)((raw[4] << 8) | raw[5]) / degree_per_second;
00273

```

```

00274 // Success
00275 return true;
00276 }
00277
00278 bool DevLab_ICM20948::readTemperature(float &temperature)
00279 {
00280 // Select USER BANK 0
00281 if (!selectBank(0))
00282 return false;
00283
00284 // Initialize accumulator
00285 temperature = 0;
00286
00287 // Read and average 5 samples
00288 for (int i = 0; i < 5; i++)
00289 {
00290 // Read temperature registers
00291 uint8_t raw[2];
00292 if (!bus->read(TEMP_OUT_H, raw, 2))
00293 return false;
00294
00295 // Convert raw to signed value
00296 int16_t temp = (int16_t)((raw[0] << 8) | raw[1]);
00297
00298 // Convert to °C and accumulate
00299 temperature += temp / 333.87f + 21.0f;
00300 }
00301
00302 // Compute average temperature
00303 temperature /= 5.0f;
00304
00305 // Success
00306 return true;
00307 }
00308
00309
00310 bool DevLab_ICM20948::readMag(float &x, float &y, float &z)
00311 {
00312 // Select USER BANK 0
00313 if (!selectBank(0))
00314 return false;
00315
00316 // Read 8 bytes (ST1 + XYZ + ST2)
00317 uint8_t buf[8];
00318 if (!bus->read(EXT_SLV_SENS_DATA_00, buf, 8))
00319 return false;
00320
00321 int16_t mx = (int16_t)(buf[1] | (buf[2] << 8));
00322 int16_t my = (int16_t)(buf[3] | (buf[4] << 8));
00323 int16_t mz = (int16_t)(buf[5] | (buf[6] << 8));
00324
00325 // Convert to µT (AK09916 sensitivity)
00326 x = mx * 0.15f;
00327 y = my * 0.15f;
00328 z = mz * 0.15f;
00329
00330 // Success
00331 return true;
00332 }
00333
00334 bool DevLab_ICM20948::initMag()
00335 {
00336 if (!bus)
00337 return false;
00338
00339 softReset();
00340
00341 sleep(false);
00342
00343 if (!selectBank(0))
00344 return false;
00345 // Enable I2C master mode and disable I2C slave mode
00346 if (!bus->write(USER_CTRL, (uint8_t)USER_CTRL_I2C_MST_EN))
00347 return false;
00348
00349 if (!selectBank(3))
00350 return false;
00351
00352 if (!bus->write(I2C_MST_CTRL, (uint8_t)0x07))
00353 return false;
00354
00355 delay(10);
00356
00357 //Soft Reset
00358 writeSlave4(AK_CNTL3, 0x01);
00359
00360 delay(100);

```

```

00361
00362     uint8_t i, who;
00363     for (i = 0; i < 10; i++)
00364     {
00365         readSlave4(AK_WIA2, who);
00366         if (who == AK_WIA2_VAL)
00367         {
00368             //Mode 3 (continuous measurement 100Hz)
00369             writeSlave4(AK_CNTL2, 0x08);
00370
00371             delay(10);
00372
00373             // ---- Setup auto-read (SLV0) ----
00374             if (!selectBank(3))
00375                 return false;
00376
00377             if (!bus->write(I2C_SLV0_ADDR, (uint8_t)(AK09916_I2C_ADDR | 0x80)))
00378                 return false;
00379
00380             if (!bus->write(I2C_SLV0_REG, (uint8_t)AK_ST1))
00381                 return false;
00382
00383             if (!bus->write(I2C_SLV0_CTRL, (uint8_t)(I2C_SLVx_EN | 8)))
00384                 return false;
00385
00386             return true;
00387         }
00388
00389         writeSlave4(AK_CNTL2, 0x08);
00390
00391         delay(10);
00392
00393         // ---- Setup auto-read (SLV0) ----
00394         if (!selectBank(3))
00395             return false;
00396
00397         if (!bus->write(I2C_SLV0_ADDR, (uint8_t)(AK09916_I2C_ADDR | 0x80)))
00398             return false;
00399
00400         if (!bus->write(I2C_SLV0_REG, (uint8_t)AK_ST1))
00401             return false;
00402
00403         if (!bus->write(I2C_SLV0_CTRL, (uint8_t)(I2C_SLVx_EN | 8)))
00404             return false;
00405
00406         return false;
00407     }
00408 }
00409
00410 bool DevLab_ICM20948::setMagOpMode(ICM20948_Op_Mode opMode)
00411 {
00412     // Write operation mode to magnetometer
00413     writeSlave4(AK_CNTL2, (uint8_t)opMode);
00414
00415     return true;
00416 }
00417
00418 bool DevLab_ICM20948::writeSlave4(uint8_t reg, uint8_t value)
00419 {
00420     // Select USER BANK 3
00421     if (!selectBank(3))
00422         return false;
00423
00424     // 7 - 1:r , 0:w
00425     //[6:0]: I2C slave address (AK09916_I2C_ADDR)
00426     // Set slave address (write)
00427     if (!bus->write(I2C_SLV4_ADDR, (uint8_t)AK09916_I2C_ADDR))
00428         return false;
00429
00430     // Set data to write to slave
00431     if (!bus->write(I2C_SLV4_DO, value))
00432         return false;
00433
00434     // Set target register
00435     if (!bus->write(I2C_SLV4_REG, reg))
00436         return false;
00437
00438     // Start transaction (EN bit)
00439     if (!bus->write(I2C_SLV4_CTRL, (uint8_t)128))
00440         return false;
00441
00442     // Wait for completion (with timeout)
00443     uint32_t start = millis();
00444     uint8_t status;
00445
00446     while (millis() - start < 10)
00447     {
00448         if (bus->read(I2C_SLV4_CTRL, status))

```

```

00448         if (!(status & 0x80))
00449             return true;
00450     }
00451
00452     // Timeout
00453     return false;
00454 }
00455
00456 bool DevLab_ICM20948::readSlave4(uint8_t reg, uint8_t &value)
00457 {
00458     // Select USER BANK 3
00459     if (!selectBank(3))
00460         return false;
00461
00462     // Set slave address (read)
00463     if (!bus->write(I2C_SLV4_ADDR, (uint8_t)(AK09916_I2C_ADDR | 0x80)))
00464         return false;
00465
00466     // Set target register
00467     if (!bus->write(I2C_SLV4_REG, reg))
00468         return false;
00469
00470     // Start transaction
00471     if (!bus->write(I2C_SLV4_CTRL, (uint8_t)128))
00472         return false;
00473     // Wait for completion
00474     uint32_t start = millis();
00475     uint8_t status;
00476     while (millis() - start < 10)
00477     {
00478         if (bus->read(I2C_SLV4_CTRL, status))
00479             if (!(status & 0x80))
00480             { if (!bus->read(I2C_SLV4_DI, value))
00481                 return false;
00482                 return true;
00483             }
00484     }
00485     return false;
00486 }
00487
00488 bool DevLab_ICM20948::setLowPower(bool enable)
00489 {
00490     // Select USER BANK 0
00491     if (!selectBank(0))
00492         return false;
00493
00494     // Set or clear LP_EN bit (bit 5)
00495     if (!bus->writeBit(PWR_MGMT_1, 5, (uint8_t)enable))
00496         return false;
00497
00498     // Success
00499     return true;
00500 }
00501
00502 bool DevLab_ICM20948::getLowPower(bool &enable)
00503 {
00504     // Select USER BANK 0
00505     if (!selectBank(0))
00506         return false;
00507
00508     // Read LP_EN bit (bit 5)
00509     uint8_t v = 0;
00510     if (!bus->readBit(PWR_MGMT_1, 5, v))
00511         return false;
00512
00513     // Convert bit value to boolean
00514     enable = (v == 1);
00515
00516     // Success
00517     return true;
00518 }
00519
00520 bool DevLab_ICM20948::setClock(ICM20948_Clock_Source clock)
00521 {
00522     // Select USER BANK 0
00523     if (!selectBank(0))
00524         return false;
00525
00526     // Set CLKSEL bits [2:0]
00527     if (!bus->write(PWR_MGMT_1, (uint8_t)clock))
00528         return false;
00529
00530     // Success
00531     return true;
00532 }
00533
00534 bool DevLab_ICM20948::getClock(uint8_t &clock)

```

```

00535 {
00536     // Select USER BANK 0
00537     if (!selectBank(0))
00538         return false;
00539
00540     // Read CLKSEL bits [2:0]
00541     if (!bus->read(PWR_MGMT_1, clock))
00542         return false;
00543
00544     clock &= 0x07;
00545
00546     // Success
00547     return true;
00548 }
00549
00550 bool DevLab_ICM20948::setGyroSampleRate(float sampleRate)
00551 {
00552     // Validate sample rate range (approx 4.3 Hz to 1100 Hz)
00553     if ((sampleRate < 4.3f) || (sampleRate > 1100.0f))
00554         return false;
00555
00556     // Compute divider value
00557     uint8_t v = (uint8_t)((1100.0f / sampleRate) - 1.0f);
00558
00559     // Select USER BANK 2
00560     if (!selectBank(2))
00561         return false;
00562
00563     // Write sample rate divider
00564     if (!bus->write(GYRO_SMPLRT_DIV, v))
00565         return false;
00566
00567     // Success
00568     return true;
00569 }
00570
00571 bool DevLab_ICM20948::getGyroSampleRate(float &sampleRate)
00572 {
00573     // Validate bus pointer
00574     if (!bus)
00575         return false;
00576
00577     // Select USER BANK 2
00578     if (!selectBank(2))
00579         return false;
00580
00581     // Read sample rate divider
00582     uint8_t v = 0;
00583     if (!bus->read(GYRO_SMPLRT_DIV, v))
00584         return false;
00585
00586     // Compute sample rate
00587     sampleRate = 1100.0f / (1.0f + v);
00588
00589     // Success
00590     return true;
00591 }
00592
00593 bool DevLab_ICM20948::setDLPF(ICM20948_Gyro_DLPF dlpf, bool bypass)
00594 {
00595     // Validate bus pointer
00596     if (!bus)
00597         return false;
00598
00599     // Select USER BANK 2
00600     if (!selectBank(2))
00601         return false;
00602
00603     // Enable or disable DLPF bypass (bit 0)
00604     if (!bus->writeBit(GYRO_CONFIG_1, 0, (uint8_t)bypass))
00605         return false;
00606
00607     // If bypass enabled, skip DLPF configuration
00608     if (bypass)
00609         return true;
00610
00611     // Set DLPF configuration bits [5:3]
00612     if (!bus->writeBits(GYRO_CONFIG_1, 3, 3, (uint8_t)dlpf))
00613         return false;
00614
00615     // Success
00616     return true;
00617 }
00618
00619 bool DevLab_ICM20948::getDLPF(uint8_t &dlpf, bool &bypass)
00620 {
00621     // Validate bus pointer

```

```

00622     if (!bus)
00623         return false;
00624
00625     // Select USER BANK 2
00626     if (!selectBank(2))
00627         return false;
00628
00629     // Read GYRO_CONFIG_1 register
00630     uint8_t v = 0;
00631     if (!bus->read(GYRO_CONFIG_1, v))
00632         return false;
00633
00634     // Extract bypass bit (bit 0)
00635     bypass = (v & 0x01);
00636
00637     // Extract DLPF bits [5:3]
00638     dlpf = (v >> 3) & 0x07;
00639
00640     // Success
00641     return true;
00642 }
00643
00644 bool DevLab_ICM20948::setGyroScale(ICM20948_Gyro_FullScale fullScale)
00645 {
00646     // Validate bus pointer
00647     if (!bus)
00648         return false;
00649
00650     // Select USER BANK 2
00651     if (!selectBank(2))
00652         return false;
00653
00654     // Set full-scale range bits [2:1]
00655     if (!bus->writeBits(GYRO_CONFIG_1, 1, 2, (uint8_t)fullScale))
00656         return false;
00657
00658     // Update internal scale (LSB per g)
00659     degree_per_second = 32768.0f / (250.0f * (1 << ((uint8_t)fullScale)));
00660
00661     // Success
00662     return true;
00663 }
00664
00665 bool DevLab_ICM20948::getGyroScale(uint8_t &fullScale)
00666 {
00667     // Validate bus pointer
00668     if (!bus)
00669         return false;
00670
00671     // Select USER BANK 2
00672     if (!selectBank(2))
00673         return false;
00674
00675     // Read full-scale range bits [2:1]
00676     if (!bus->readBits(GYRO_CONFIG_1, 1, 2, fullScale))
00677         return false;
00678
00679     // Success
00680     return true;
00681 }
00682
00683 bool DevLab_ICM20948::selfTestGyro(bool x, bool y, bool z)
00684 {
00685     // Validate bus pointer
00686     if (!bus)
00687         return false;
00688
00689     // Select USER BANK 2
00690     if (!selectBank(2))
00691         return false;
00692
00693     // Set Z-axis self-test (bit 3)
00694     if (!bus->writeBit(GYRO_CONFIG_2, 3, (uint8_t)z))
00695         return false;
00696
00697     // Set Y-axis self-test (bit 4)
00698     if (!bus->writeBit(GYRO_CONFIG_2, 4, (uint8_t)y))
00699         return false;
00700
00701     // Set X-axis self-test (bit 5)
00702     if (!bus->writeBit(GYRO_CONFIG_2, 5, (uint8_t)x))
00703         return false;
00704
00705     // Success
00706     return true;
00707 }
00708

```

```

00709 bool DevLab_ICM20948::setAccelSampleRate(uint16_t sampleRate)
00710 {
00711     // Validate bus pointer
00712     if (!bus)
00713         return false;
00714
00715     // Select USER BANK 2
00716     if (!selectBank(2))
00717         return false;
00718
00719     // Validate input
00720     if (sampleRate == 0)
00721         return false;
00722
00723     // Clamp maximum rate
00724     if (sampleRate >= 1125)
00725         return false;
00726
00727     // Compute divider (rounded)
00728     uint16_t div = (1125.0f / sampleRate) - 1;
00729
00730     if (div > 4095u)
00731         div = 4095u;
00732
00733     // Split divider into high and low bytes
00734     uint8_t msb = (uint8_t)((div >> 8) & 0xFF);
00735     uint8_t lsb = (uint8_t)(div & 0xFF);
00736
00737     // Write divider high byte
00738     if (!bus->write(ACCEL_SMPLRT_DIV_1, msb))
00739         return false;
00740
00741     // Write divider low byte
00742     if (!bus->write(ACCEL_SMPLRT_DIV_2, lsb))
00743         return false;
00744
00745     // Success
00746     return true;
00747 }
00748 bool DevLab_ICM20948::setAccelDivRate(uint16_t divisor)
00749 {
00750     if (!bus)
00751         return false;
00752
00753     // Select USER BANK 2
00754     if (!selectBank(2))
00755         return false;
00756
00757     // Validate input
00758     if (divisor == 0)
00759         return false;
00760
00761     if (divisor > 4095u)
00762         divisor = 4095u;
00763
00764     // Split divider into high and low bytes
00765     uint8_t msb = (uint8_t)((divisor >> 8) & 0xFF);
00766     uint8_t lsb = (uint8_t)(divisor & 0xFF);
00767
00768     // Write divider high byte
00769     if (!bus->write(ACCEL_SMPLRT_DIV_1, msb))
00770         return false;
00771
00772     // Write divider low byte
00773     if (!bus->write(ACCEL_SMPLRT_DIV_2, lsb))
00774         return false;
00775
00776     // Success
00777     return true;
00778 }
00779
00780 bool DevLab_ICM20948::setGyroDivRate(uint8_t divisor)
00781 {
00782     if (!bus)
00783         return false;
00784
00785     // Select USER BANK 2
00786     if (!selectBank(2))
00787         return false;
00788
00789     // Write sample rate divider
00790     if (!bus->write(GYRO_SMPLRT_DIV, divisor))
00791         return false;
00792
00793     // Success
00794     return true;
00795 }

```

```

00796 bool DevLab_ICM20948::getAccelSampleRate(float &sampleRate)
00797 {
00798     // Validate bus pointer
00799     if (!bus)
00800         return false;
00801
00802     // Select USER BANK 2
00803     if (!selectBank(2))
00804         return false;
00805
00806     // Read divider high byte
00807     uint8_t msb = 0;
00808     if (!bus->read(ACCEL_SMPLRT_DIV_1, msb))
00809         return false;
00810
00811     // Read divider low byte
00812     uint8_t lsb = 0;
00813     if (!bus->read(ACCEL_SMPLRT_DIV_2, lsb))
00814         return false;
00815
00816     // Combine 12-bit divider
00817     uint16_t div = ((msb & 0x0F) << 8) | lsb;
00818
00819     // Compute sample rate
00820     sampleRate = 1125.0f / (1.0f + div);
00821
00822     // Success
00823     return true;
00824 }
00825
00826 bool DevLab_ICM20948::selfTestAccel(bool x, bool y, bool z)
00827 {
00828     // Validate bus pointer
00829     if (!bus)
00830         return false;
00831
00832     // Select USER BANK 2
00833     if (!selectBank(2))
00834         return false;
00835
00836     if (x)
00837     { // Set X-axis self-test (bit 7)
00838         if (!bus->writeBit(ACCEL_CONFIG_2, 7, (uint8_t)x))
00839             return false;
00840     }
00841
00842     if (y)
00843     { // Set Y-axis self-test (bit 6)
00844         if (!bus->writeBit(ACCEL_CONFIG_2, 6, (uint8_t)y))
00845             return false;
00846     }
00847
00848     if (z)
00849     { // Set Z-axis self-test (bit 5)
00850         if (!bus->writeBit(ACCEL_CONFIG_2, 5, (uint8_t)z))
00851             return false;
00852     }
00853
00854     // Success
00855     return true;
00856 }
00857 bool DevLab_ICM20948::setAccelScale(ICM20948_Accel_FullScale fullScale)
00858 {
00859     // Validate bus pointer
00860     if (!bus)
00861         return false;
00862
00863     // Select USER BANK 2
00864     if (!selectBank(2))
00865         return false;
00866
00867     // Set full-scale bits [2:1]
00868     if (!bus->writeBits(ACCEL_CONFIG, 1, 2, (uint8_t)fullScale))
00869         return false;
00870
00871     // Update internal scale (LSB per g)
00872     mg_per_lsb = 16384.0f / (1 << (uint8_t)fullScale);
00873
00874     // Success
00875     return true;
00876 }
00877
00878 bool DevLab_ICM20948::getAccelScale(uint8_t &fullScale)
00879 {
00880     // Validate bus pointer
00881     if (!bus)
00882         return false;

```

```

00883
00884 // Select USER BANK 2
00885 if (!selectBank(2))
00886     return false;
00887
00888 // Read full-scale bits [2:1]
00889 if (!bus->readBits(ACCEL_CONFIG, 1, 2, fullScale))
00890     return false;
00891
00892 // Success
00893 return true;
00894 }
00895
00896
00897 bool DevLab_ICM20948::setAccelDLPF(uint8_t dlpf, bool bypass)
00898 {
00899     // Validate bus pointer
00900     if (!bus)
00901         return false;
00902
00903     // Select USER BANK 2
00904     if (!selectBank(2))
00905         return false;
00906
00907     // Set or clear DLPF bypass (bit 0)
00908     if (!bus->writeBit(ACCEL_CONFIG, 0, (uint8_t)bypass))
00909         return false;
00910
00911     // If bypass enabled, skip DLPF configuration
00912     if (bypass)
00913         return true;
00914
00915     // Limit DLPF value to valid range
00916     dlpf &= 0x07;
00917
00918     // Set DLPF configuration bits [5:3]
00919     if (!bus->writeBits(ACCEL_CONFIG, 3, 3, dlpf))
00920         return false;
00921
00922     // Success
00923     return true;
00924 }
00925
00926 bool DevLab_ICM20948::getAccelDLPF(uint8_t &dlpf, bool &bypass)
00927 {
00928     // Validate bus pointer
00929     if (!bus)
00930         return false;
00931
00932     // Select USER BANK 2
00933     if (!selectBank(2))
00934         return false;
00935
00936     // Read ACCEL_CONFIG register
00937     uint8_t v = 0;
00938     if (!bus->read(ACCEL_CONFIG, v))
00939         return false;
00940
00941     // Extract bypass bit (bit 0)
00942     bypass = (v & 0x01);
00943
00944     // Extract DLPF bits [5:3]
00945     dlpf = (v >> 3) & 0x07;
00946
00947     // Success
00948     return true;
00949 }
00950
00951
00952 bool DevLab_ICM20948::setAccelAveraging(ICM20948_Accel_Average avg)
00953 {
00954     // Validate bus pointer
00955     if (!bus)
00956         return false;
00957
00958     // Select USER BANK 2
00959     if (!selectBank(2))
00960         return false;
00961
00962     // Set DEC3 averaging bits [1:0]
00963     if (!bus->writeBits(ACCEL_CONFIG_2, 0, 2, (uint8_t)avg))
00964         return false;
00965
00966     // Success
00967     return true;
00968 }
00969

```

```

00970 bool DevLab_ICM20948::setGyroAveraging(ICM20948_Gyro_Average avg)
00971 {
00972     // Validate bus pointer
00973     if (!bus)
00974         return false;
00975
00976     // Select USER BANK 2
00977     if (!selectBank(2))
00978         return false;
00979
00980     // Set DEC3 averaging bits [1:0]
00981     if (!bus->writeBits(GYRO_CONFIG_2, 0, 2, (uint8_t)avg))
00982         return false;
00983
00984     // Success
00985     return true;
00986 }
00987
00988 bool DevLab_ICM20948::getAccelAveraging(uint8_t &avg)
00989 {
00990     // Validate bus pointer
00991     if (!bus)
00992         return false;
00993
00994     // Select USER BANK 2
00995     if (!selectBank(2))
00996         return false;
00997
00998     // Read DEC3 averaging bits [1:0]
00999     if (!bus->readBits(ACCEL_CONFIG_2, 0, 2, avg))
01000         return false;
01001
01002     // Success
01003     return true;
01004 }
01005
01006 bool DevLab_ICM20948::setSensors(bool accel_on, bool gyro_on, bool temp_on)
01007 {
01008     // Validate bus pointer
01009     if (!bus)
01010         return false;
01011
01012     // Select USER BANK 0
01013     if (!selectBank(0))
01014         return false;
01015
01016     // Build PWR_MGMT_2 mask
01017     uint8_t v = 0;
01018
01019     // Disable accel axes if not enabled
01020     if (!accel_on)
01021         v |= 0x38; // bits [5:3]
01022
01023     // Disable gyro axes if not enabled
01024     if (!gyro_on)
01025         v |= 0x07; // bits [2:0]
01026
01027     // Write sensor enable/disable mask
01028     if (!bus->write(PWR_MGMT_2, v))
01029         return false;
01030
01031     // Set or clear TEMP_DIS bit (bit 3)
01032     if (!bus->writeBit(PWR_MGMT_1, 3, (uint8_t)!temp_on))
01033         return false;
01034
01035     // Success
01036     return true;
01037 }
01038
01039 bool DevLab_ICM20948::getSensors(bool &accel_on, bool &gyro_on, bool &temp_on)
01040 {
01041     // Validate bus pointer
01042     if (!bus)
01043         return false;
01044
01045     // Select USER BANK 0
01046     if (!selectBank(0))
01047         return false;
01048
01049     // Read PWR_MGMT_2 register
01050     uint8_t v = 0;
01051     if (!bus->read(PWR_MGMT_2, v))
01052         return false;
01053
01054     // Decode gyro state (bits [2:0])
01055     gyro_on = ((v & 0x07) == 0);
01056 }

```

```

01057 // Decode accel state (bits [5:3])
01058 accel_on = ((v & 0x38) == 0);
01059
01060 // Read TEMP_DIS bit (bit 3)
01061 uint8_t t = 0;
01062 if (!bus->readBit(PWR_MGMT_1, 3, t))
01063     return false;
01064
01065 // Decode temperature state
01066 temp_on = (t == 0);
01067
01068 // Success
01069 return true;
01070 }
01071
01072 bool DevLab_ICM20948::setGyroOffset(uint16_t offsetX, uint16_t offsetY, uint16_t offsetZ)
01073 {
01074     // Validate bus pointer
01075     if (!bus)
01076         return false;
01077
01078     // Select USER BANK 2
01079     if (!selectBank(2))
01080         return false;
01081
01082     // Pack offset values into byte array
01083     uint8_t data[6] = {
01084         (uint8_t)(offsetX >> 8),
01085         (uint8_t)(offsetX & 0xFF),
01086         (uint8_t)(offsetY >> 8),
01087         (uint8_t)(offsetY & 0xFF),
01088         (uint8_t)(offsetZ >> 8),
01089         (uint8_t)(offsetZ & 0xFF)};
01090
01091     // Write offset registers
01092     if (!bus->write(XG_OFFS_USRH, data, 6))
01093         return false;
01094
01095     // Success
01096     return true;
01097 }
01098
01099 bool DevLab_ICM20948::getGyroOffset(int16_t &offsetX, int16_t &offsetY, int16_t &offsetZ)
01100 {
01101     // Validate bus pointer
01102     if (!bus)
01103         return false;
01104
01105     // Select USER BANK 2
01106     if (!selectBank(2))
01107         return false;
01108
01109     // Read offset registers
01110     uint8_t data[6];
01111     if (!bus->read(XG_OFFS_USRH, data, 6))
01112         return false;
01113
01114     // Convert to signed values
01115     offsetX = (int16_t)((data[0] << 8) | data[1]);
01116     offsetY = (int16_t)((data[2] << 8) | data[3]);
01117     offsetZ = (int16_t)((data[4] << 8) | data[5]);
01118
01119     // Success
01120     return true;
01121 }
01122
01123
01124 bool DevLab_ICM20948::setAccelOffset(int16_t offsetX, int16_t offsetY, int16_t offsetZ)
01125 {
01126     // Validate bus pointer
01127     if (!bus)
01128         return false;
01129
01130     // Select USER BANK 1 (accel offsets are here)
01131     if (!selectBank(1))
01132         return false;
01133
01134     // Pack offset values into byte array
01135     uint8_t data[6] = {
01136         (uint8_t)(offsetX >> 8),
01137         (uint8_t)(offsetX & 0xFF),
01138         (uint8_t)(offsetY >> 8),
01139         (uint8_t)(offsetY & 0xFF),
01140         (uint8_t)(offsetZ >> 8),
01141         (uint8_t)(offsetZ & 0xFF)};
01142
01143     // Write offset registers

```

```

01144     if (!bus->write(XA_OFFS_H, data, 6))
01145         return false;
01146
01147     // Success
01148     return true;
01149 }
01150
01151 bool DevLab_ICM20948::getAccelOffset(int16_t &offsetX, int16_t &offsetY, int16_t &offsetZ)
01152 {
01153     // Validate bus pointer
01154     if (!bus)
01155         return false;
01156
01157     // Select USER BANK 1
01158     if (!selectBank(1))
01159         return false;
01160
01161     // Read offset registers
01162     uint8_t data[6];
01163     if (!bus->read(XA_OFFS_H, data, 6))
01164         return false;
01165
01166     // Convert to signed values
01167     offsetX = (int16_t)((data[0] << 8) | data[1]);
01168     offsetY = (int16_t)((data[2] << 8) | data[3]);
01169     offsetZ = (int16_t)((data[4] << 8) | data[5]);
01170
01171     // Success
01172     return true;
01173 }
01174
01175
01176 bool DevLab_ICM20948::intInit(const ICM20948_IntPinConfig &cfg){
01177     // Validate bus pointer
01178     if (!bus)
01179         return false;
01180
01181     if(!selectBank(0))
01182         return false;
01183
01184     uint8_t reg = (cfg.activeLevel << 7) |
01185                  (cfg.driveMode << 6) |
01186                  (cfg.latchMode << 5) |
01187                  (cfg.clearMode << 4) |
01188                  (cfg.fsyncActLevel << 3) |
01189                  (cfg.fsyncIntEn << 2) |
01190                  (cfg.bypassEn << 1);
01191
01192     // Initialize INT1
01193     // Configure interrupt pin (open-drain, active-low, latch)
01194     if (!bus->write(INT_PIN_CFG, reg))
01195         return false;
01196
01197     // Configuration successful
01198     return true;
01199 }
01200
01201 bool DevLab_ICM20948::intEnableConfig(const ICM20948_IntEnableConfig &cfg)
01202 {
01203     if (!bus) return false;
01204     if (!selectBank(0)) return false;
01205
01206     // INT_ENABLE
01207     uint8_t reg0 = (cfg.wofEn << 7) |
01208                  (cfg.womIntEn << 3) |
01209                  (cfg.pllRdyEn << 2) |
01210                  (cfg.dmpInt1En << 1) |
01211                  (cfg.i2cMstIntEn << 0);
01212
01213     if (!bus->write(INT_ENABLE, reg0)) return false;
01214
01215     // INT_ENABLE_1
01216     if (!bus->write(INT_ENABLE_1, (uint8_t)(cfg.rawDataRdyEn & 0x01))) return false;
01217
01218     // INT_ENABLE_2 – construye máscara desde el array
01219     uint8_t ovfMask = 0;
01220     for (uint8_t i = 0; i < 5; i++)
01221         ovfMask |= (cfg.fifoOvfEn[i] ? BIT(i) : 0);
01222     if (!bus->write(INT_ENABLE_2, ovfMask)) return false;
01223
01224     // INT_ENABLE_3 – igual
01225     uint8_t wmMask = 0;
01226     for (uint8_t i = 0; i < 5; i++)
01227         wmMask |= (cfg.fifoWmEn[i] ? BIT(i) : 0);
01228     if (!bus->write(INT_ENABLE_3, wmMask)) return false;
01229
01230     return true;
01231 }
01232
01233

```

```

01231 bool DevLab_ICM20948::checkIntStatus(ICM20948_IntStatus &status)
01232 {
01233     if (!bus)                return false;
01234     if (!selectBank(0)) return false;
01235
01236     // Leer los 4 registros de status en una sola operación de burst
01237     uint8_t raw[4];
01238     if (!bus->read(INT_STATUS, raw, 4)) return false;
01239
01240     // INT_STATUS (0x19)
01241     status.womInt    = (raw[0] & STS_WOM_INT)    ? 1 : 0;
01242     status.pllRdyInt = (raw[0] & STS_PLL_RDY_INT) ? 1 : 0;
01243     status.dmpInt1   = (raw[0] & STS_DMP_INT1)   ? 1 : 0;
01244     status.i2cMstInt = (raw[0] & STS_I2C_MST_INT) ? 1 : 0;
01245
01246     // INT_STATUS_1 (0x1A)
01247     status.rawDataRdy = (raw[1] & STS_RAW_DATA_0_RDY_INT) ? 1 : 0;
01248
01249     // INT_STATUS_2 (0x1B) – FIFO overflow canal por canal
01250     for (uint8_t i = 0; i < 5; i++)
01251         status.fifoOvf[i] = (raw[2] & BIT(i)) ? 1 : 0;
01252
01253     // INT_STATUS_3 (0x1C) – FIFO watermark canal por canal
01254     for (uint8_t i = 0; i < 5; i++)
01255         status.fifoWm[i] = (raw[3] & BIT(i)) ? 1 : 0;
01256
01257     return true;
01258 }
01259
01260 bool DevLab_ICM20948::auxMasterEnable(uint8_t clkFreq)
01261 {
01262     // BANK 0: asegurarse que BYPASS_EN=0 e I2C_MST_EN=1
01263     if (!selectBank(0)) return false;
01264
01265     // Limpiar BYPASS_EN (bit 1 de INT_PIN_CFG)
01266     if (!bus->writeBit(INT_PIN_CFG, 1, (uint8_t)0)) return false;
01267
01268     // Activar I2C_MST_EN (bit 5 de USER_CTRL)
01269     if (!bus->writeBit(USER_CTRL, 5, (uint8_t)1)) return false;
01270
01271     // BANK 3: configurar clock del master auxiliar
01272     if (!selectBank(3)) return false;
01273
01274     // clkFreq típico: 0x07 = ~345.6 kHz | 0x0D = ~400 kHz
01275     if (!bus->writeBits(I2C_MST_CTRL, 0, 4, clkFreq)) return false;
01276
01277     // Volver a BANK 0
01278     return selectBank(0);
01279 }
01280
01281 bool DevLab_ICM20948::auxWriteByte(uint8_t slaveAddr, uint8_t reg, uint8_t data)
01282 {
01283     if (!selectBank(3)) return false;
01284
01285     // Dirección del slave, bit 7=0 para escritura
01286     if (!bus->write(I2C_SLV4_ADDR, (uint8_t)(slaveAddr & 0x7F))) return false;
01287
01288     // Registro destino que queremos escribir en el slave
01289     if (!bus->write(I2C_SLV4_REG, reg)) return false;
01290
01291     // Dato a escribir
01292     if (!bus->write(I2C_SLV4_DO, data)) return false;
01293
01294     // Disparar transacción (I2C_SLV4_EN, se auto-limpia al terminar)
01295     if (!bus->write(I2C_SLV4_CTRL, (uint8_t)I2C_SLVx_EN)) return false;
01296
01297     // Esperar a que complete – verificar SLV4_DONE en BANK 0
01298     if (!selectBank(0)) return false;
01299
01300     uint8_t status = 0;
01301     uint8_t timeout = 50;
01302     do {
01303         delay(1);
01304         if (!bus->read(I2C_MST_STATUS, status)) return false;
01305         if (--timeout == 0) return false; // timeout
01306     } while (!(status & MST_SLV4_DONE));
01307
01308     // Verificar que no hubo NACK
01309     return !(status & MST_SLV4_NACK);
01310 }
01311
01312 bool DevLab_ICM20948::auxReadByte(uint8_t slaveAddr, uint8_t reg, uint8_t &data)
01313 {
01314     if (!selectBank(3)) return false;
01315
01316     // Dirección del slave, bit 7=1 para lectura
01317     if (!bus->write(I2C_SLV4_ADDR, (uint8_t)((slaveAddr & 0x7F) | I2C_SLVx_RNW))) return false;

```

```

01318
01319 // Registro origen que queremos leer del slave
01320 if (!bus->write(I2C_SLV4_REG, reg)) return false;
01321
01322 // Disparar transacción (I2C_SLV4_EN, se auto-limpia al terminar)
01323 if (!bus->write(I2C_SLV4_CTRL, (uint8_t)I2C_SLVx_EN)) return false;
01324
01325 // Esperar a que complete – verificar SLV4_DONE en BANK 0
01326 if (!selectBank(0)) return false;
01327
01328 uint8_t status = 0;
01329 uint8_t timeout = 50;
01330 do {
01331     delay(1);
01332     if (!bus->read(I2C_MST_STATUS, status)) return false;
01333     if (--timeout == 0) return false; // timeout
01334 } while (!(status & MST_SLV4_DONE));
01335
01336 // Verificar que no hubo NACK
01337 if (status & MST_SLV4_NACK) return false;
01338
01339 // Leer el dato recibido del slave (BANK 3)
01340 if (!selectBank(3)) return false;
01341 if (!bus->read(I2C_SLV4_DI, data)) return false;
01342
01343 return selectBank(0);
01344 }
01345
01346 bool DevLab_ICM20948::auxWriteCommand(uint8_t slaveAddr, uint8_t cmd)
01347 {
01348     if (!selectBank(3)) return false;
01349
01350     // Dirección del slave, bit 7=0 para escritura
01351     if (!bus->write(I2C_SLV4_ADDR, (uint8_t)(slaveAddr & 0x7F))) return false;
01352
01353     // Byte de comando a enviar (único byte de la transacción)
01354     if (!bus->write(I2C_SLV4_DO, cmd)) return false;
01355
01356     // EN + REG_DIS: el SLV4 envía solo I2C_SLV4_DO, sin byte de registro
01357     if (!bus->write(I2C_SLV4_CTRL, (uint8_t)(I2C_SLVx_EN | I2C_SLVx_REG_DIS))) return false;
01358
01359     // Esperar a que complete – verificar SLV4_DONE en BANK 0
01360     if (!selectBank(0)) return false;
01361
01362     uint8_t status = 0;
01363     uint8_t timeout = 50;
01364     do {
01365         delay(1);
01366         if (!bus->read(I2C_MST_STATUS, status)) return false;
01367         if (--timeout == 0) return false; // timeout
01368     } while (!(status & MST_SLV4_DONE));
01369
01370     // Verificar que no hubo NACK
01371     return !(status & MST_SLV4_NACK);
01372 }
01373
01374 bool DevLab_ICM20948::auxConfigSlave(uint8_t slaveAddr, uint8_t reg, uint8_t numBytes)
01375 {
01376     if (!selectBank(3)) return false;
01377
01378     // Slave 0: dirección + bit READ
01379     if (!bus->write(I2C_SLV0_ADDR, (uint8_t)(slaveAddr | I2C_SLVx_RNW))) return false;
01380
01381     // Registro de inicio
01382     if (!bus->write(I2C_SLV0_REG, reg)) return false;
01383
01384     // Habilitar + número de bytes
01385     if (!bus->write(I2C_SLV0_CTRL, (uint8_t)(I2C_SLVx_EN | (numBytes & 0x0F)))) return false;
01386
01387     return selectBank(0);
01388 }
01389
01390 // Para leer los datos que el ICM leyó automáticamente:
01391 bool DevLab_ICM20948::auxReadSensorData(uint8_t *buf, uint8_t len)
01392 {
01393     if (!selectBank(0)) return false;
01394     return bus->read(EXT_SLV_SENS_DATA_00, buf, len);
01395 }
01396
01397 bool DevLab_ICM20948::auxReadResponse(uint8_t slaveAddr, uint8_t &data)
01398 {
01399     if (!selectBank(3)) return false;
01400
01401     // Slave address con bit READ, sin byte de registro (REG_DIS)
01402     // Genera: [START, addr+R, lee 1 byte, STOP]
01403     if (!bus->write(I2C_SLV4_ADDR, (uint8_t)((slaveAddr & 0x7F) | I2C_SLVx_RNW))) return false;
01404     if (!bus->write(I2C_SLV4_CTRL, (uint8_t)(I2C_SLVx_EN | I2C_SLVx_REG_DIS))) return false;

```

```

01405
01406     if (!selectBank(0)) return false;
01407
01408     uint8_t status = 0;
01409     uint8_t timeout = 50;
01410     do {
01411         delay(1);
01412         if (!bus->read(I2C_MST_STATUS, status)) return false;
01413         if (--timeout == 0) return false;
01414     } while (!(status & MST_SLV4_DONE));
01415
01416     if (status & MST_SLV4_NACK) return false;
01417
01418     if (!selectBank(3)) return false;
01419     if (!bus->read(I2C_SLV4_DI, data)) return false;
01420
01421     return selectBank(0);
01422 }
01423
01424 bool DevLab_ICM20948::auxRead12bit(uint16_t &raw)
01425 {
01426     uint8_t buf[2];
01427     if (!auxReadSensorData(buf, 2)) return false;
01428
01429     // LSB primero (little-endian): buf[0] = byte bajo, buf[1] = byte alto
01430     raw = (uint16_t)(buf[0] | (buf[1] << 8)) & 0xFFFF;
01431
01432     return true;
01433 }

```

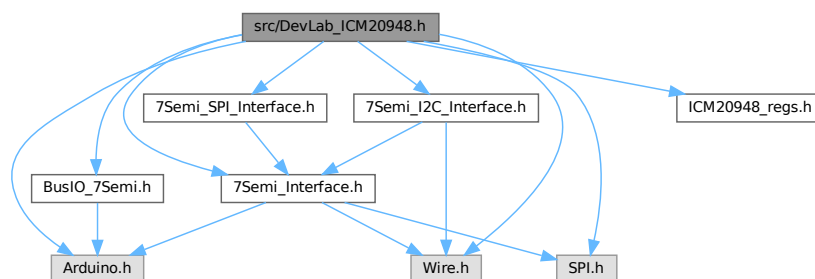
6.36 src/DevLab_ICM20948.h File Reference

```

#include <Arduino.h>
#include <Wire.h>
#include <SPI.h>
#include "ICM20948_regs.h"
#include "7Semi_Interface.h"
#include "7Semi_I2C_Interface.h"
#include "7Semi_SPI_Interface.h"
#include "BusIO_7Semi.h"

```

Include dependency graph for DevLab_ICM20948.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct [ICM20948_IntPinConfig](#)
- struct [ICM20948_IntEnableConfig](#)

- struct [ICM20948_IntStatus](#)
- class [DevLab_ICM20948](#)

Enumerations

- enum class [ICM20948_Op_Mode](#) : uint8_t {
[MODE_POWER_DOWN](#) = 0x00, [MODE_SINGLE_MEASUREMENT](#) = 0x01, [MODE_CONTINUOUS_1](#) = 0x02, [MODE_CONTINUOUS_2](#) = 0x04,
[MODE_CONTINUOUS_3](#) = 0x06, [MODE_CONTINUOUS_4](#) = 0x08, [MODE_SELF_TEST](#) = 0x10 }
- enum class [ICM20948_Clock_Source](#) : uint8_t {
[INTERNAL_20MHZ_0](#) = 0x00, [AUTO_PLL_1](#) = 0x01, [AUTO_PLL_2](#) = 0x02, [AUTO_PLL_3](#) = 0x03,
[AUTO_PLL_4](#) = 0x04, [AUTO_PLL_5](#) = 0x05, [INTERNAL_20MHZ_6](#) = 0x06, [STOP_CLOCK](#) = 0x07 }
- enum class [ICM20948_Gyro_DLPF](#) : uint8_t {
[DLPF_196HZ](#) = 0x00, [DLPF_151HZ](#) = 0x01, [DLPF_119HZ](#) = 0x02, [DLPF_51HZ](#) = 0x03,
[DLPF_23HZ](#) = 0x04, [DLPF_11HZ](#) = 0x05, [DLPF_5HZ](#) = 0x06, [DLPF_361HZ](#) = 0x07 }
- enum class [ICM20948_Gyro_Average](#) : uint8_t {
[AVG_1](#) = 0x00, [AVG_2](#) = 0x01, [AVG_4](#) = 0x02, [AVG_8](#) = 0x03,
[AVG_16](#) = 0x04, [AVG_32](#) = 0x05, [AVG_64](#) = 0x06, [AVG_128](#) = 0x07 }
- enum class [ICM20948_Gyro_FullScale](#) : uint8_t { [DPS_250](#) = 0x00, [DPS_500](#) = 0x01, [DPS_1000](#) = 0x02,
[DPS_2000](#) = 0x03 }
- enum class [ICM20948_Accel_FullScale](#) : uint8_t { [G_2](#) = 0x00, [G_4](#) = 0x01, [G_8](#) = 0x02, [G_16](#) = 0x03 }
- enum class [ICM20948_Accel_Average](#) : uint8_t { [AVG_1_OR_4](#) = 0x00, [AVG_8](#) = 0x01, [AVG_16](#) = 0x02,
[AVG_32](#) = 0x03 }
- enum class [ICM20948_Accel_DLPFCFG](#) : uint8_t {
[DLPF0_246HZ](#) = 0x00, [DLPF_246HZ](#) = 0x01, [DLPF_111HZ](#) = 0x02, [DLPF_50HZ](#) = 0x03,
[DLPF_24HZ](#) = 0x04, [DLPF_12HZ](#) = 0x05, [DLPF_6HZ](#) = 0x06, [DLPF_473HZ](#) = 0x07 }

6.36.1 Enumeration Type Documentation

6.36.1.1 ICM20948_Accel_Average

enum class [ICM20948_Accel_Average](#) : uint8_t [strong]

Enumerator

AVG_1_OR_4	
AVG_8	
AVG_16	
AVG_32	

Definition at line 76 of file [DevLab_ICM20948.h](#).

6.36.1.2 ICM20948_Accel_DLPFCFG

enum class [ICM20948_Accel_DLPFCFG](#) : uint8_t [strong]

Enumerator

DLPF0_246HZ	
DLPF_246HZ	
DLPF_111HZ	
DLPF_50HZ	
DLPF_24HZ	
DLPF_12HZ	
DLPF_6HZ	
DLPF_473HZ	

Definition at line 84 of file [DevLab_ICM20948.h](#).

6.36.1.3 ICM20948_Accel_FullScale

```
enum class ICM20948_Accel_FullScale : uint8_t [strong]
```

Enumerator

G_2	
G_4	
G_8	
G_16	

Definition at line 68 of file [DevLab_ICM20948.h](#).

6.36.1.4 ICM20948_Clock_Source

```
enum class ICM20948_Clock_Source : uint8_t [strong]
```

Enumerator

INTERNAL_20MHZ↔ _0	
AUTO_PLL_1	
AUTO_PLL_2	
AUTO_PLL_3	
AUTO_PLL_4	
AUTO_PLL_5	
INTERNAL_20MHZ↔ _6	
STOP_CLOCK	

Definition at line 25 of file [DevLab_ICM20948.h](#).

6.36.1.5 ICM20948_Gyro_Average

```
enum class ICM20948_Gyro_Average : uint8_t [strong]
```

Enumerator

AVG_1	
AVG_2	
AVG_4	
AVG_8	
AVG_16	
AVG_32	
AVG_64	
AVG_128	

Definition at line 49 of file [DevLab_ICM20948.h](#).

6.36.1.6 ICM20948_Gyro_DLPF

```
enum class ICM20948_Gyro_DLPF : uint8_t [strong]
```

Enumerator

DLPF_196HZ	
DLPF_151HZ	
DLPF_119HZ	
DLPF_51HZ	
DLPF_23HZ	
DLPF_11HZ	
DLPF_5HZ	
DLPF_361HZ	

Definition at line 37 of file [DevLab_ICM20948.h](#).

6.36.1.7 ICM20948_Gyro_FullScale

```
enum class ICM20948_Gyro_FullScale : uint8_t [strong]
```

Enumerator

DPS_250	
DPS_500	
DPS_1000	
DPS_2000	

Definition at line 60 of file [DevLab_ICM20948.h](#).

6.36.1.8 ICM20948_Op_Mode

```
enum class ICM20948_Op_Mode : uint8_t [strong]
```

Enumerator

MODE_POWER_DOWN	
MODE_SINGLE_MEASUREMENT	
MODE_CONTINUOUS_1	
MODE_CONTINUOUS_2	
MODE_CONTINUOUS_3	
MODE_CONTINUOUS_4	
MODE_SELF_TEST	

Definition at line 14 of file [DevLab_ICM20948.h](#).

6.37 DevLab_ICM20948.h

[Go to the documentation of this file.](#)

```
00001 #ifndef ICM20948_DEVLAB_H
00002 #define ICM20948_DEVLAB_H
00003
00004 #include <Arduino.h>
00005 #include <Wire.h>
00006 #include <SPI.h>
00007 #include "ICM20948_regs.h"
00008
00009 #include "7Semi_Interface.h"
00010 #include "7Semi_I2C_Interface.h"
00011 #include "7Semi_SPI_Interface.h"
00012 #include "BusIO_7Semi.h"
00013
00014 enum class ICM20948_Op_Mode : uint8_t
00015 {
```

```

00016  MODE_POWER_DOWN = 0x00,
00017  MODE_SINGLE_MEASUREMENT = 0x01,
00018  MODE_CONTINUOUS_1 = 0x02,
00019  MODE_CONTINUOUS_2 = 0x04,
00020  MODE_CONTINUOUS_3 = 0x06,
00021  MODE_CONTINUOUS_4 = 0x08,
00022  MODE_SELF_TEST = 0x10
00023 };
00024
00025 enum class ICM20948_Clock_Source : uint8_t
00026 {
00027     INTERNAL_20MHZ_0 = 0x00, // Internal oscillator
00028     AUTO_PLL_1 = 0x01, // Auto select (PLL if ready)
00029     AUTO_PLL_2 = 0x02,
00030     AUTO_PLL_3 = 0x03,
00031     AUTO_PLL_4 = 0x04,
00032     AUTO_PLL_5 = 0x05,
00033     INTERNAL_20MHZ_6 = 0x06, // Internal oscillator
00034     STOP_CLOCK = 0x07 // Stops clock, timing generator reset
00035 };
00036
00037 enum class ICM20948_Gyro_DLPF : uint8_t
00038 {
00039     DLPF_196HZ = 0x00, // BW ≈ 196.6 Hz, NBW ≈ 229.8 Hz
00040     DLPF_151HZ = 0x01, // BW ≈ 151.8 Hz, NBW ≈ 187.6 Hz
00041     DLPF_119HZ = 0x02, // BW ≈ 119.5 Hz, NBW ≈ 154.3 Hz
00042     DLPF_51HZ = 0x03, // BW ≈ 51.2 Hz, NBW ≈ 73.3 Hz
00043     DLPF_23HZ = 0x04, // BW ≈ 23.9 Hz, NBW ≈ 35.9 Hz
00044     DLPF_11HZ = 0x05, // BW ≈ 11.6 Hz, NBW ≈ 17.8 Hz
00045     DLPF_5HZ = 0x06, // BW ≈ 5.7 Hz, NBW ≈ 8.9 Hz
00046     DLPF_361HZ = 0x07 // BW ≈ 361.4 Hz, NBW ≈ 376.5 Hz
00047 };
00048
00049 enum class ICM20948_Gyro_Average : uint8_t
00050 {
00051     AVG_1 = 0x00, // 1x Average (no averaging)
00052     AVG_2 = 0x01, // 2x Average
00053     AVG_4 = 0x02, // 4x Average
00054     AVG_8 = 0x03, // 8x Average
00055     AVG_16 = 0x04, // Average 16 samples
00056     AVG_32 = 0x05, // Average 32 samples
00057     AVG_64 = 0x06, // Average 64 samples
00058     AVG_128 = 0x07 // Average 128 samples
00059 };
00060 enum class ICM20948_Gyro_FullScale : uint8_t
00061 {
00062     DPS_250 = 0x00, // ±250 dps
00063     DPS_500 = 0x01, // ±500 dps
00064     DPS_1000 = 0x02, // ±1000 dps
00065     DPS_2000 = 0x03 // ±2000 dps
00066 };
00067
00068 enum class ICM20948_Accel_FullScale : uint8_t
00069 {
00070     G_2 = 0x00, // ±2g
00071     G_4 = 0x01, // ±4g
00072     G_8 = 0x02, // ±8g
00073     G_16 = 0x03 // ±16g
00074 };
00075
00076 enum class ICM20948_Accel_Average : uint8_t
00077 {
00078     AVG_1_OR_4 = 0x00, // Depends on ACCEL_FCHOICE
00079     AVG_8 = 0x01, // Average 8 samples
00080     AVG_16 = 0x02, // Average 16 samples
00081     AVG_32 = 0x03 // Average 32 samples
00082 };
00083
00084 enum class ICM20948_Accel_DLPFCFG : uint8_t
00085 {
00086     DLPF0_246HZ = 0x00, // BW ≈ 246 Hz, NBW ≈ 265 Hz
00087     DLPF_246HZ = 0x01, // BW ≈ 246 Hz, NBW ≈ 265 Hz
00088     DLPF_111HZ = 0x02, // BW ≈ 111 Hz, NBW ≈ 136 Hz
00089     DLPF_50HZ = 0x03, // BW ≈ 50 Hz, NBW ≈ 68.8 Hz
00090     DLPF_24HZ = 0x04, // BW ≈ 24 Hz, NBW ≈ 34.4 Hz
00091     DLPF_12HZ = 0x05, // BW ≈ 12 Hz, NBW ≈ 17 Hz
00092     DLPF_6HZ = 0x06, // BW ≈ 6 Hz, NBW ≈ 8.3 Hz
00093     DLPF_473HZ = 0x07, // BW ≈ 473 Hz, NBW ≈ 499 Hz
00094 };
00095
00096 /**
00097  * ICM20948_IntPinConfig
00098  *
00099  * - Physical configuration of the INT pin (INT_PIN_CFG register, BANK 0)
00100  * - Controls electrical behavior and clear condition of the interrupt line
00101  * - Pass to intInit() to apply settings
00102  */

```

```

00103 struct ICM20948_IntPinConfig
00104 {
00105     uint8_t activeLevel : 1; // bit 7 | 0 = active high,      1 = active low
00106     uint8_t driveMode   : 1; // bit 6 | 0 = push-pull,       1 = open-drain
00107     uint8_t latchMode   : 1; // bit 5 | 0 = pulse 50µs,      1 = latch held
00108     uint8_t clearMode   : 1; // bit 4 | 0 = read INT_STATUS, 1 = any read
00109     uint8_t fsyncActLevel : 1; // bit 3 | 0 = FSYNC active high, 1 = FSYNC active low
00110     uint8_t fsyncIntEn   : 1; // bit 2 | 0 = FSYNC disabled,  1 = FSYNC as interrupt
00111     uint8_t bypassEn     : 1; // bit 1 | 0 = normal,        1 = I2C bypass mode
00112 };
00113
00114 /**
00115  * ICM20948_IntEnableConfig
00116  *
00117  * - Selects which interrupt sources are routed to the INT pin
00118  * - Covers INT_ENABLE (0x10), INT_ENABLE_1 (0x11), INT_ENABLE_2 (0x12), INT_ENABLE_3 (0x13)
00119  * - FIFO overflow and watermark fields are per-channel arrays [0]-[4]
00120  * - Pass to intEnableConfig() to write all four registers in one call
00121  */
00122 struct ICM20948_IntEnableConfig
00123 {
00124     // --- INT_ENABLE (0x10) ---
00125     uint8_t wofEn : 1; // Wake on FSYNC
00126     uint8_t womIntEn : 1; // Wake on motion
00127     uint8_t pllRdyEn : 1; // PLL ready
00128     uint8_t dmpInt1En : 1; // DMP interrupt
00129     uint8_t i2cMstIntEn : 1; // I2C master interrupt
00130
00131     // --- INT_ENABLE_1 (0x11) ---
00132     uint8_t rawDataRdyEn : 1; // Raw data ready
00133
00134     // --- INT_ENABLE_2 (0x12) --- canal por canal
00135     uint8_t fifoOvfEn[5]; // [0]-[4]: 1 = overflow interrupt ON para ese canal
00136
00137     // --- INT_ENABLE_3 (0x13) --- canal por canal
00138     uint8_t fifoWmEn[5]; // [0]-[4]: 1 = watermark interrupt ON para ese canal
00139 };
00140
00141 /**
00142  * ICM20948_IntStatus
00143  *
00144  * - Holds the parsed state of the four interrupt status registers
00145  * - Covers INT_STATUS (0x19), INT_STATUS_1 (0x1A), INT_STATUS_2 (0x1B), INT_STATUS_3 (0x1C)
00146  * - Populated by checkIntStatus() via a single 4-byte burst read
00147  * - Reading these registers clears the interrupt flags (when clearMode = 0 in IntPinConfig)
00148  */
00149 struct ICM20948_IntStatus
00150 {
00151     // --- INT_STATUS (0x19) ---
00152     uint8_t womInt : 1; // Wake on motion ocurrió
00153     uint8_t pllRdyInt : 1; // PLL listo
00154     uint8_t dmpInt1 : 1; // DMP generó interrupción
00155     uint8_t i2cMstInt : 1; // I2C master generó interrupción
00156
00157     // --- INT_STATUS_1 (0x1A) ---
00158     uint8_t rawDataRdy : 1; // Datos de sensores listos para leer
00159
00160     // --- INT_STATUS_2 (0x1B) ---
00161     uint8_t fifoOvf[5]; // [0]-[4]: FIFO overflow por canal
00162
00163     // --- INT_STATUS_3 (0x1C) ---
00164     uint8_t fifoWm[5]; // [0]-[4]: FIFO watermark por canal
00165 };
00166
00167 /**
00168  * Class
00169  * - Manages ICM-20948 over I2C (SPI path disabled in this cut)
00170  * - Caches scale factors for LSB->physical conversions
00171  */
00172 class DevLab_ICM20948
00173 {
00174 public:
00175     DevLab_ICM20948();
00176
00177     /**
00178      * beginI2C
00179      *
00180      * - Initialize ICM20948 using I2C interface
00181      * - Sets up communication layer (Interface + BusIO)
00182      * - Verifies device identity (WHO_AM_I)
00183      * - Configures basic power and clock settings
00184      *
00185      * Returns:
00186      * - true → Device initialized successfully
00187      * - false → Communication or configuration failed
00188      */
00189

```

```

00190 bool beginI2C(uint8_t address = 0x69, TwoWire &i2cPort = Wire, uint32_t i2cSpeed = 400000);
00191
00192 bool beginSPI(uint8_t csPin, SPIClass &spiPort = SPI, uint32_t spiSpeed = 1000000);
00193
00194 /**
00195  * readWhoAmI
00196  *
00197  * - Read WHO_AM_I register (device ID)
00198  * - Used during initialization to verify ICM20948
00199  * - Expected value: 0xEA
00200  *
00201  * Returns:
00202  * - true → Read successful
00203  * - false → Operation failed
00204  */
00205 bool readWhoAmI(uint8_t &whoAmI);
00206
00207 /**
00208  * selectBank
00209  *
00210  * - Select USER BANK (0-3)
00211  * - Used to access different register groups inside ICM20948
00212  *
00213  * Returns:
00214  * - true → Bank switch successful
00215  * - false → Write failed
00216  */
00217 bool selectBank(uint8_t bank);
00218
00219 /**
00220  * softReset
00221  *
00222  * - Perform software reset of ICM20948
00223  * - Resets all internal registers to default state
00224  *
00225  * Flow:
00226  * - Select USER BANK 0
00227  * - Set DEVICE_RESET bit
00228  * - Wait for device to restart
00229  *
00230  * Returns:
00231  * - true → Reset successful
00232  * - false → Operation failed
00233  */
00234 bool softReset();
00235
00236 /**
00237  * sleep
00238  *
00239  * - Enable or disable sleep mode
00240  * - Maintains clock source (CLKSEL = 1)
00241  *
00242  * Flow:
00243  * - Select USER BANK 0
00244  * - Set or clear SLEEP bit
00245  *
00246  * Returns:
00247  * - true → Operation successful
00248  * - false → Operation failed
00249  */
00250 bool sleep(bool en);
00251
00252 /**
00253  * applyBasicDefaults
00254  *
00255  * - Apply basic sensor configuration
00256  * - Configure power, filters, ranges, and sampling rates
00257  *
00258  * Returns:
00259  * - true → Configuration successful
00260  * - false → Any register write failed
00261  */
00262 bool applyBasicDefaults();
00263
00264 /**
00265  * readAccel
00266  *
00267  * - Read accelerometer data (X, Y, Z)
00268  * - Convert raw values to g using LSB scale
00269  *
00270  * Returns:
00271  * - true → Read successful
00272  * - false → Read failed
00273  */
00274 bool readAccel(float &x, float &y, float &z);
00275
00276 /**

```

```

00277     * readGyro
00278     *
00279     * - Read gyroscope data (X, Y, Z)
00280     * - Convert raw values to dps using LSB scale
00281     *
00282     * Returns:
00283     * - true → Read successful
00284     * - false → Read failed
00285     */
00286     bool readGyro(float &x, float &y, float &z);
00287
00288     /**
00289     * readTemperature
00290     *
00291     * - Read temperature sensor
00292     * - Convert raw values to °C
00293     *
00294     * Returns:
00295     * - true → Read successful
00296     * - false → Read failed
00297     */
00298     bool readTemperature(float &temperature);
00299
00300     /**
00301     * readMag
00302     *
00303     * - Read magnetometer data via internal I2C master
00304     * - Data comes from EXT_SLV_SENS_DATA registers
00305     * - Convert raw values to µT
00306     *
00307     * Returns:
00308     * - true → Read successful
00309     * - false → Read failed
00310     */
00311     bool readMag(float &x, float &y, float &z);
00312
00313     /**
00314     * initMag
00315     *
00316     * - Initialize AK09916 magnetometer using internal I2C master
00317     * - Verifies WHO_AM_I and enables continuous mode
00318     * - Configures SLV0 for automatic data read
00319     *
00320     * Returns:
00321     * - true → Initialization successful
00322     * - false → Operation failed
00323     */
00324     bool initMag();
00325
00326     /**
00327     * setMagOpMode
00328     *
00329     * - Set magnetometer operation mode
00330     *
00331     * Returns:
00332     * - true → Mode set successfully
00333     * - false → Write failed
00334     */
00335     bool setMagOpMode(ICM20948_Op_Mode opMode);
00336
00337     /**
00338     * setLowPower
00339     *
00340     * - Enable or disable low power mode
00341     * - Controls LP_EN bit in PWR_MGMT_1
00342     *
00343     * Returns:
00344     * - true → Operation successful
00345     * - false → Operation failed
00346     */
00347     bool setLowPower(bool enable);
00348
00349     /**
00350     * getLowPower
00351     *
00352     * - Read low power mode status
00353     * - Returns state of LP_EN bit
00354     *
00355     * Returns:
00356     * - true → Read successful
00357     * - false → Operation failed
00358     */
00359     bool getLowPower(bool &enable);
00360
00361     /**
00362     * setClock
00363     *

```

```

00364     * - Set clock source (CLKSEL [2:0])
00365     * - Selects internal clock for sensor operation
00366     *
00367     * Returns:
00368     * - true → Operation successful
00369     * - false → Operation failed
00370     */
00371     bool setClock(ICM20948_Clock_Source clock);
00372
00373     /**
00374     * getClock
00375     *
00376     * - Read current clock source (CLKSEL [2:0])
00377     *
00378     * Returns:
00379     * - true → Read successful
00380     * - false → Operation failed
00381     */
00382     bool getClock(uint8_t &clock);
00383
00384     /**
00385     * setGyroSampleRate
00386     *
00387     * - Set gyroscope sample rate
00388     * - Uses internal divider (SRD) based on 1100 Hz base rate
00389     *
00390     * Returns:
00391     * - true → Operation successful
00392     * - false → Invalid input or write failed
00393     */
00394     bool setGyroSampleRate(float sampleRate);
00395
00396     /**
00397     * getGyroSampleRate
00398     *
00399     * - Get current gyroscope sample rate
00400     * - Computes value from divider register
00401     *
00402     * Returns:
00403     * - true → Read successful
00404     * - false → Operation failed
00405     */
00406     bool getGyroSampleRate(float &sampleRate);
00407     bool setGyroDivRate(uint8_t divisor);
00408     /**
00409     * setDLPF
00410     *
00411     * - Configure gyroscope digital low-pass filter (DLPF)
00412     * - Option to bypass filter (full bandwidth)
00413     *
00414     * Returns:
00415     * - true → Operation successful
00416     * - false → Operation failed
00417     */
00418     bool setDLPF(ICM20948_Gyro_DLPF dlpf, bool bypass);
00419
00420     /**
00421     * getDLPF
00422     *
00423     * - Read gyroscope DLPF configuration
00424     * - Returns filter setting and bypass state
00425     *
00426     * Returns:
00427     * - true → Read successful
00428     * - false → Operation failed
00429     */
00430     bool getDLPF(uint8_t &dlpf, bool &bypass);
00431
00432     /**
00433     * setGyroScale
00434     *
00435     * - Set gyroscope full-scale range (FS_SEL)
00436     * - Controls sensitivity (dps range)
00437     *
00438     * Flow:
00439     * - Validate bus pointer
00440     * - Select USER BANK 2
00441     * - Write FS_SEL bits [2:1]
00442     *
00443     * Returns:
00444     * - true → Operation successful
00445     * - false → Operation failed
00446     */
00447     bool setGyroScale(ICM20948_Gyro_FullScale fullScale);
00448     bool setGyroAveraging(ICM20948_Gyro_Average avg);
00449     /**
00450     * getGyroScale

```

```

00451     *
00452     * - Get current gyroscope full-scale range (FS_SEL
00453     *
00454     * Returns:
00455     * - true → Read successful
00456     * - false → Operation failed
00457     */
00458     bool getGyroScale(uint8_t &fullScale);
00459
00460     /**
00461     * selfTestGyro
00462     *
00463     * - Enable or disable gyroscope self-test per axis
00464     *
00465     * Returns:
00466     * - true → Operation successful
00467     * - false → Operation failed
00468     */
00469     bool selfTestGyro(bool x, bool y, bool z);
00470
00471     bool setAccelDivRate(uint16_t divisor);
00472     /**
00473     * setAccelSampleRate
00474     *
00475     * - Set accelerometer output data rate (ODR)
00476     * - Uses 1125 Hz base rate with 12-bit divider
00477     *
00478     * Returns:
00479     * - true → Operation successful
00480     * - false → Operation failed
00481     */
00482     bool setAccelSampleRate(uint16_t sampleRate);
00483
00484     /**
00485     * getAccelSampleRate
00486     *
00487     * - Get accelerometer output data rate (ODR)
00488     * - Computes value from divider registers
00489     *
00490     * Returns:
00491     * - true → Read successful
00492     * - false → Operation failed
00493     */
00494     bool getAccelSampleRate(float &sampleRate);
00495
00496     /**
00497     * selfTestAccel
00498     *
00499     * - Enable or disable accelerometer self-test per axis
00500     *
00501     * Returns:
00502     * - true → Operation successful
00503     * - false → Operation failed
00504     */
00505     bool selfTestAccel(bool x, bool y, bool z);
00506
00507     /**
00508     * setAccelScale
00509     *
00510     * - Set accelerometer full-scale range (FS_SEL)
00511     * - Controls measurement range (±2g, ±4g, ±8g, ±16g)
00512     *
00513     * Returns:
00514     * - true → Operation successful
00515     * - false → Operation failed
00516     */
00517     bool setAccelScale(ICM20948_Accel_FullScale fullScale);
00518
00519     /**
00520     * getAccelScale
00521     *
00522     * - Get current accelerometer full-scale range (FS_SEL)
00523     *
00524     * Returns:
00525     * - true → Read successful
00526     * - false → Operation failed
00527     */
00528     bool getAccelScale(uint8_t &fullScale);
00529
00530     /**
00531     * setAccelDLPF
00532     *
00533     * - Configure accelerometer digital low-pass filter (DLPF)
00534     * - Option to bypass filter (full bandwidth)
00535     *
00536     * Flow:
00537     * - Validate bus pointer

```

```

00538     * - Select USER BANK 2
00539     * - Set or clear bypass bit
00540     * - Configure DLPF bits if not bypassed
00541     *
00542     * Returns:
00543     * - true → Operation successful
00544     * - false → Operation failed
00545     */
00546 bool setAccelDLPF(uint8_t dlpf, bool bypass);
00547
00548 /**
00549     * getAccelDLPF
00550     *
00551     * - Read accelerometer DLPF configuration
00552     * - Returns filter setting and bypass state
00553     *
00554     * Flow:
00555     * - Validate bus pointer
00556     * - Select USER BANK 2
00557     * - Read ACCEL_CONFIG register
00558     * - Extract bypass and DLPF bits
00559     *
00560     * Returns:
00561     * - true → Read successful
00562     * - false → Operation failed
00563     */
00564 bool getAccelDLPF(uint8_t &dlpf, bool &bypass);
00565
00566 /**
00567     * setAccelAveraging
00568     *
00569     * - Configure accelerometer averaging / decimation (DEC3)
00570     * - Controls internal averaging of accel samples
00571     *
00572     * Flow:
00573     * - Validate bus pointer
00574     * - Select USER BANK 2
00575     * - Write DEC3 bits [1:0]
00576     *
00577     * Returns:
00578     * - true → Operation successful
00579     * - false → Operation failed
00580     */
00581 bool setAccelAveraging(ICM20948_Accel_Average avg);
00582
00583 /**
00584     * getAccelAveraging
00585     *
00586     * - Read accelerometer averaging / decimation setting (DEC3)
00587     *
00588     * Flow:
00589     * - Validate bus pointer
00590     * - Select USER BANK 2
00591     * - Read DEC3 bits [1:0]
00592     *
00593     * Returns:
00594     * - true → Read successful
00595     * - false → Operation failed
00596     */
00597 bool getAccelAveraging(uint8_t &avg);
00598
00599 /**
00600     * setSensors
00601     *
00602     * - Enable or disable accel, gyro, and temperature sensor
00603     * - Controls power gating via PWR_MGMT_2 and TEMP_DIS
00604     *
00605     * Flow:
00606     * - Validate bus pointer
00607     * - Select USER BANK 0
00608     * - Build PWR_MGMT_2 mask
00609     * - Configure temperature enable/disable
00610     *
00611     * Returns:
00612     * - true → Operation successful
00613     * - false → Operation failed
00614     */
00615 bool setSensors(bool accel_on, bool gyro_on, bool temp_on);
00616
00617 /**
00618     * getSensors
00619     *
00620     * - Read accel, gyro, and temperature enable state
00621     *
00622     * Flow:
00623     * - Validate bus pointer
00624     * - Select USER BANK 0

```

```

00625     * - Read PWR_MGMT_2 register
00626     * - Read TEMP_DIS bit
00627     * - Decode enable states
00628     *
00629     * Returns:
00630     * - true → Read successful
00631     * - false → Operation failed
00632     */
00633 bool getSensors(bool &accel_on, bool &gyro_on, bool &temp_on);
00634
00635 /**
00636     * setGyroOffset
00637     *
00638     * - Set gyroscope offset values for X, Y, Z axes
00639     * - Writes offset registers in USER BANK 2
00640     *
00641     * Flow:
00642     * - Validate bus pointer
00643     * - Select USER BANK 2
00644     * - Pack offsets into byte array
00645     * - Write to offset registers
00646     *
00647     * Returns:
00648     * - true → Operation successful
00649     * - false → Operation failed
00650     */
00651 bool setGyroOffset(uint16_t offsetX, uint16_t offsetY, uint16_t offsetZ);
00652
00653 /**
00654     * getGyroOffset
00655     *
00656     * - Read gyroscope offset values for X, Y, Z axes
00657     * - Reads offset registers from USER BANK 2
00658     *
00659     * Flow:
00660     * - Validate bus pointer
00661     * - Select USER BANK 2
00662     * - Read offset registers
00663     * - Convert to signed values
00664     *
00665     * Returns:
00666     * - true → Read successful
00667     * - false → Operation failed
00668     */
00669 bool getGyroOffset(int16_t &offsetX, int16_t &offsetY, int16_t &offsetZ);
00670
00671 /**
00672     * setAccelOffset
00673     *
00674     * - Set accelerometer offset values for X, Y, Z axes
00675     * - Writes offset registers in USER BANK 1
00676     *
00677     * Flow:
00678     * - Validate bus pointer
00679     * - Select USER BANK 1
00680     * - Pack offsets into byte array
00681     * - Write to offset registers
00682     *
00683     * Returns:
00684     * - true → Operation successful
00685     * - false → Operation failed
00686     */
00687 bool setAccelOffset(int16_t offsetX, int16_t offsetY, int16_t offsetZ);
00688
00689 /**
00690     * getAccelOffset
00691     *
00692     * - Read accelerometer offset values for X, Y, Z axes
00693     * - Reads offset registers from USER BANK 1
00694     *
00695     * Flow:
00696     * - Validate bus pointer
00697     * - Select USER BANK 1
00698     * - Read offset registers
00699     * - Convert to signed values
00700     *
00701     * Returns:
00702     * - true → Read successful
00703     * - false → Operation failed
00704     */
00705 bool getAccelOffset(int16_t &offsetX, int16_t &offsetY, int16_t &offsetZ);
00706
00707 /**
00708     * intInit
00709     *
00710     * - Configure the physical behavior of the INT pin (INT_PIN_CFG register, BANK 0)
00711     * - Sets active level, drive mode (push-pull / open-drain), latch vs pulse mode,

```

```

00712     *   clear condition, FSYNC level, FSYNC interrupt enable, and I2C bypass
00713     *
00714     * Parameters:
00715     * - cfg: ICM20948_IntPinConfig struct with the desired pin settings
00716     *
00717     * Returns:
00718     * - true → Configuration written successfully
00719     * - false → Operation failed
00720     */
00721 bool intInit(const ICM20948_IntPinConfig &cfg);
00722
00723 /**
00724  * intEnableConfig
00725  *
00726  * - Enable or disable individual interrupt sources routed to the INT pin
00727  * - Writes INT_ENABLE (0x10), INT_ENABLE_1 (0x11), INT_ENABLE_2 (0x12),
00728  *   and INT_ENABLE_3 (0x13) in BANK 0
00729  * - Must call intInit() first to configure the pin electrical behavior
00730  *
00731  * Parameters:
00732  * - cfg: ICM20948_IntEnableConfig struct with the desired interrupt sources enabled
00733  *
00734  * Returns:
00735  * - true → All four registers written successfully
00736  * - false → Operation failed
00737  */
00738 bool intEnableConfig(const ICM20948_IntEnableConfig &cfg);
00739
00740 /**
00741  * checkIntStatus
00742  *
00743  * - Read and parse all four interrupt status registers in a single burst read
00744  * - Covers INT_STATUS (0x19), INT_STATUS_1 (0x1A), INT_STATUS_2 (0x1B),
00745  *   INT_STATUS_3 (0x1C) from BANK 0
00746  * - Reading clears the interrupt flags when clearMode = 0 in ICM20948_IntPinConfig
00747  * - Typically called inside the INT pin ISR or polled in the main loop
00748  *
00749  * Parameters:
00750  * - status: ICM20948_IntStatus struct populated with the current interrupt flags
00751  *
00752  * Returns:
00753  * - true → Status read successfully
00754  * - false → Operation failed
00755  */
00756 bool checkIntStatus(ICM20948_IntStatus &status);
00757
00758 /**
00759  * auxMasterEnable
00760  *
00761  * - Enable the ICM20948 internal I2C master for auxiliary bus
00762  * - Clears BYPASS_EN so the aux bus is controlled by the ICM
00763  * - Sets I2C_MST_EN in USER_CTRL
00764  * - Configures auxiliary master clock frequency
00765  *
00766  * Parameters:
00767  * - clkFreq: clock divider value (e.g. 0x07 ≈ 345.6 kHz, 0x0D ≈ 400 kHz)
00768  *
00769  * Returns:
00770  * - true → Master enabled successfully
00771  * - false → Operation failed
00772  */
00773 bool auxMasterEnable(uint8_t clkFreq);
00774
00775 /**
00776  * auxWriteByte
00777  *
00778  * - Write a single byte to a register of an auxiliary I2C slave via SLV4
00779  * - Uses one-shot SLV4 transaction: polls SLV4_DONE, checks for NACK
00780  * - Suitable for configuration writes (not continuous streaming)
00781  *
00782  * Parameters:
00783  * - slaveAddr: 7-bit I2C address of the auxiliary slave
00784  * - reg:       target register address on the slave
00785  * - data:      byte to write
00786  *
00787  * Returns:
00788  * - true → Write acknowledged by slave
00789  * - false → Timeout or NACK
00790  */
00791 bool auxWriteByte(uint8_t slaveAddr, uint8_t reg, uint8_t data);
00792
00793 /**
00794  * auxReadByte
00795  *
00796  * - Read a single byte from a register of an auxiliary I2C slave via SLV4
00797  * - Generates a combined write-then-read transaction (reg byte + repeated START)
00798  * - Suitable for slaves that follow standard I2C register-read protocol

```

```

00799  *
00800  * Note: slaves that need separate write/read transactions (e.g. command-response
00801  * firmware) should use auxWriteCommand + auxReadResponse instead.
00802  *
00803  * Parameters:
00804  * - slaveAddr: 7-bit I2C address of the auxiliary slave
00805  * - reg:       register address to read from
00806  * - data:      output byte received from slave
00807  *
00808  * Returns:
00809  * - true  → Read successful
00810  * - false → Timeout or NACK
00811  */
00812 bool auxReadByte(uint8_t slaveAddr, uint8_t reg, uint8_t &data);
00813
00814 /**
00815  * auxWriteCommand
00816  *
00817  * - Send a single command byte to an auxiliary I2C slave via SLV4
00818  * - Uses REG_DIS: generates [START, addr+W, cmd_byte, STOP] with no register prefix
00819  * - Designed for slaves that use a command-response protocol (no register addressing)
00820  *
00821  * Parameters:
00822  * - slaveAddr: 7-bit I2C address of the auxiliary slave
00823  * - cmd:       command byte to send
00824  *
00825  * Returns:
00826  * - true  → Command acknowledged by slave
00827  * - false → Timeout or NACK
00828  */
00829 bool auxWriteCommand(uint8_t slaveAddr, uint8_t cmd);
00830
00831 /**
00832  * auxConfigSlave
00833  *
00834  * - Configure SLV0 for automatic periodic reads from an auxiliary I2C slave
00835  * - The ICM20948 master will read numBytes starting at reg on every ODR cycle
00836  * - Data is deposited into EXT_SLV_SENS_DATA_00 and subsequent registers
00837  * - Call once during initialization; read results with auxReadSensorData
00838  *
00839  * Parameters:
00840  * - slaveAddr: 7-bit I2C address of the auxiliary slave
00841  * - reg:       starting register address to read from on the slave
00842  * - numBytes:  number of bytes to read per cycle (1-15)
00843  *
00844  * Returns:
00845  * - true  → SLV0 configured successfully
00846  * - false → Operation failed
00847  */
00848 bool auxConfigSlave(uint8_t slaveAddr, uint8_t reg, uint8_t numBytes);
00849
00850 /**
00851  * auxReadSensorData
00852  *
00853  * - Read bytes from EXT_SLV_SENS_DATA registers (BANK 0)
00854  * - Returns data collected by the ICM20948 master from SLV0-SLV3 on the last cycle
00855  * - Must call auxConfigSlave first to set up the automatic read
00856  *
00857  * Parameters:
00858  * - buf: output buffer to store the received bytes
00859  * - len: number of bytes to read (must match numBytes configured in auxConfigSlave)
00860  *
00861  * Returns:
00862  * - true  → Read successful
00863  * - false → Bus error
00864  */
00865 bool auxReadSensorData(uint8_t *buf, uint8_t len);
00866
00867 /**
00868  * auxReadResponse
00869  *
00870  * - Read a single response byte from an auxiliary I2C slave via SLV4
00871  * - Generates a pure read transaction: [START, addr+R, 1 byte, STOP]
00872  * - No register byte is sent (REG_DIS); slave must already be ready to transmit
00873  * - Use after auxWriteCommand to complete a command-response exchange with slaves
00874  *   that require separate write and read transactions (e.g. PY32F0 firmware slaves)
00875  *
00876  * Parameters:
00877  * - slaveAddr: 7-bit I2C address of the auxiliary slave
00878  * - data:      output byte received from slave
00879  *
00880  * Returns:
00881  * - true  → Response received successfully
00882  * - false → Timeout or NACK
00883  */
00884 bool auxReadResponse(uint8_t slaveAddr, uint8_t &data);
00885

```

```

00886  /**
00887   * auxRead12bit
00888   *
00889   * - Read a 12-bit value previously configured via auxConfigSlave (SLV0, numBytes = 2)
00890   * - Assumes little-endian packing: first byte = LSB, second byte = MSB
00891   * - Result is masked to 12 bits (0-4095)
00892   *
00893   * Returns:
00894   * - true → Read successful
00895   * - false → Read failed
00896   */
00897  bool auxRead12bit(uint16_t &raw);
00898 private:
00899  I2C_Interface i2c;
00900  SPI_Interface spi;
00901
00902  Interface_7Semi *iface = nullptr;
00903
00904  BusIO_7Semi<Interface_7Semi> *bus = nullptr;
00905
00906  float mg_per_lsb = 16384.0f; // LSB/g at ±16g
00907  float degree_per_second = 131.072f; // LSB/dps at ±2000 dps
00908
00909  bool writeSlave4(uint8_t reg, uint8_t value);
00910  bool readSlave4(uint8_t reg, uint8_t &value);
00911 };
00912
00913
00914 #endif /* ICM20948_DEVLAB_H */

```

6.38 src/ICM20948_platform.h File Reference

Classes

- struct [icm_plat_t](#)

6.39 ICM20948_platform.h

[Go to the documentation of this file.](#)

```

00001 #ifndef ICM20948_PLATFORM_H
00002 #define ICM20948_PLATFORM_H
00003
00004 // #include <stdint.h>
00005 // #include <stddef.h>
00006
00007 /**
00008  * 7Semi comment style
00009  * - Platform glue for bus IO, delays, and timing
00010  * - Implement these for your system (I2C or SPI)
00011  */
00012
00013 typedef struct {
00014  /**
00015   * - Bus read: read `len` bytes from `reg` into `buf`
00016   * - Return 0 on success, nonzero on error
00017   */
00018  int (*read)(uint8_t reg, uint8_t *buf, size_t len, void *user);
00019  /**
00020   * - Bus write: write `len` bytes from `buf` to `reg`
00021   * - Return 0 on success, nonzero on error
00022   */
00023  int (*write)(uint8_t reg, const uint8_t *buf, size_t len, void *user);
00024  /**
00025   * - Optional: SPI transfers may need register R/W bit mangling
00026   * - For I2C, leave as NULL
00027   */
00028  uint8_t (*spi_reg_rw_bits)(uint8_t reg, int is_write);
00029  /**
00030   * - Millisecond delay
00031   */
00032  void (*delay_ms)(uint32_t ms);
00033  /**
00034   * - User context pointer passed to read/write
00035   */
00036  void *user;
00037  /**
00038   * - If using SPI: set chip select active (1) or inactive (0)
00039   * - For I2C: leave as NULL
00040   */
00041  void (*spi_cs)(int level, void *user);
00042  /**

```

6.40 src/ICM20948_regs.h File Reference

[illegible]

```

• #define WHO_AM_I 0x00 /* WHO_AM_I[7:0] */
• #define WHO_AM_I_VAL 0xEA
• #define USER_CTRL 0x03 /* DMP_EN FIFO_EN I2C_MST_EN I2C_IF_DIS DMP_RST SRAM_RST I2C_MST_RST - */
• #define USER_CTRL_DMP_EN BIT(7)
• #define USER_CTRL_FIFO_EN BIT(6)
• #define USER_CTRL_I2C_MST_EN BIT(5)
• #define USER_CTRL_I2C_IF_DIS BIT(4)
• #define USER_CTRL_DMP_RST BIT(3)
• #define USER_CTRL_SRAM_RST BIT(2)
• #define USER_CTRL_I2C_MST_RST BIT(1)
• #define LP_CONFIG 0x05 /* I2C_MST_CYCLE ACCEL_CYCLE GYRO_CYCLE - */
• #define LP_I2C_MST_CYCLE BIT(6)
• #define LP_ACCEL_CYCLE BIT(5)
• #define LP_GYRO_CYCLE BIT(4)
• #define PWR_MGMT_1 0x06 /* DEVICE_RESET SLEEP LP_EN - TEMP_DIS CLKSEL[2:0] */
• #define PWR_DEVICE_RESET BIT(7)
• #define PWR_SLEEP BIT(6)
• #define PWR_LP_EN BIT(5)
• #define PWR_TEMP_DIS BIT(3)
• #define PWR_CLKSEL_MASK 0x07
• #define PWR_CLKSEL_INT_20MHZ 0x01
• #define PWR_CLKSEL_AUTO 0x01 /* typical: auto selects best source */
• #define PWR_MGMT_2 0x07 /* - DISABLE_ACCEL DISABLE_GYRO */
• #define PWR_DISABLE_ACCEL BIT(3)
• #define PWR_DISABLE_GYRO BIT(0)
• #define INT_PIN_CFG 0x0F /* INT1_ACTL INT1_OPEN INT1_LATCH_INT_EN INT_ANYRD_2CLEAR ACTL_FSYNC FSYNC_INT_MODE_EN BYPASS_EN - */
• #define INT1_ACTL BIT(7) /* active low */
• #define INT1_OPEN BIT(6) /* open-drain */
• #define INT1_LATCH_INT_EN BIT(5) /* latch until status read */
• #define INT_ANYRD_2CLEAR BIT(4)
• #define INT_ACTL_FSYNC BIT(3)
• #define FSYNC_INT_MODE_EN BIT(2)
• #define BYPASS_EN BIT(1) /* I2C bypass to aux devices */
• #define INT_ENABLE 0x10 /* REG_WOF_EN - WOM_INT_EN PLL_RDY_EN DMP_INT1_EN I2C_MST_INT_EN */
• #define INT_REG_WOF_EN BIT(7)
• #define INT_WOM_INT_EN BIT(3)

```

- #define INT_PLL_RDY_EN BIT(2)
- #define INT_DMP_INT1_EN BIT(1)
- #define INT_I2C_MST_INT_EN BIT(0)
- #define INT_ENABLE_1 0x11 /* RAW_DATA_0_RDY_EN */
- #define INT_RAW_DATA_0_RDY_EN BIT(0)
- #define INT_ENABLE_2 0x12 /* FIFO_OVERFLOW_EN[4:0] */
- #define INT_ENABLE_3 0x13 /* FIFO_WM_EN[4:0] */
- #define I2C_MST_STATUS 0x17 /* PASS_THROUGH, *_DONE, *_NACK, LOST_ARB */
- #define MST_PASS_THROUGH BIT(7)
- #define MST_SLV4_DONE BIT(6)
- #define MST_LOST_ARB BIT(5)
- #define MST_SLV4_NACK BIT(4)
- #define MST_SLV3_NACK BIT(3)
- #define MST_SLV2_NACK BIT(2)
- #define MST_SLV1_NACK BIT(1)
- #define MST_SLV0_NACK BIT(0)
- #define INT_STATUS 0x19 /* WOM_INT PLL_RDY_INT DMP_INT1 I2C_MST_INT */
- #define STS_WOM_INT BIT(3)
- #define STS_PLL_RDY_INT BIT(2)
- #define STS_DMP_INT1 BIT(1)
- #define STS_I2C_MST_INT BIT(0)
- #define INT_STATUS_1 0x1A /* RAW_DATA_0_RDY_INT */
- #define STS_RAW_DATA_0_RDY_INT BIT(0)
- #define INT_STATUS_2 0x1B /* FIFO_OVERFLOW_INT[4:0] */
- #define INT_STATUS_3 0x1C /* FIFO_WM_INT[4:0] */
- #define DELAY_TIMEH 0x28
- #define DELAY_TIMEL 0x29
- #define ACCEL_XOUT_H 0x2D
- #define ACCEL_XOUT_L 0x2E
- #define ACCEL_YOUT_H 0x2F
- #define ACCEL_YOUT_L 0x30
- #define ACCEL_ZOUT_H 0x31
- #define ACCEL_ZOUT_L 0x32
- #define GYRO_XOUT_H 0x33
- #define GYRO_XOUT_L 0x34
- #define GYRO_YOUT_H 0x35
- #define GYRO_YOUT_L 0x36
- #define GYRO_ZOUT_H 0x37
- #define GYRO_ZOUT_L 0x38
- #define TEMP_OUT_H 0x39
- #define TEMP_OUT_L 0x3A
- #define EXT_SLV_SENS_DATA_00 0x3B
- #define EXT_SLV_SENS_DATA_23 0x52 /* contiguous range 0x3B..0x52 */
- #define FIFO_EN_1 0x66 /* SLV3..SLV0 FIFO_EN */
- #define FIFO_EN_2 0x67 /* ACCEL GYRO_Z/Y/X TEMP FIFO_EN */
- #define FIFO_RST 0x68 /* FIFO_RESET[4:0] */
- #define FIFO_MODE 0x69 /* FIFO_MODE[4:0] */
- #define FIFO_COUNTH 0x70
- #define FIFO_COUNTL 0x71
- #define FIFO_R_W 0x72
- #define FIFO_CFG 0x76
- #define REG_BANK_SEL 0x7F /* USER_BANK[1:0] */
- #define SELF_TEST_X_GYRO 0x02 /* XG_ST_DATA[7:0] */
- #define SELF_TEST_Y_GYRO 0x03 /* YG_ST_DATA[7:0] */
- #define SELF_TEST_Z_GYRO 0x04 /* ZG_ST_DATA[7:0] */

- #define [SELF_TEST_X_ACCEL](#) 0x0E /* XA_ST_DATA[7:0] */
- #define [SELF_TEST_Y_ACCEL](#) 0x0F /* YA_ST_DATA[7:0] */
- #define [SELF_TEST_Z_ACCEL](#) 0x10 /* ZA_ST_DATA[7:0] */
- #define [XA_OFFSETS_H](#) 0x14 /* XA_OFFSETS[14:7] */
- #define [XA_OFFSETS_L](#) 0x15 /* XA_OFFSETS[6:0] */
- #define [YA_OFFSETS_H](#) 0x17 /* YA_OFFSETS[14:7] */
- #define [YA_OFFSETS_L](#) 0x18 /* YA_OFFSETS[6:0] */
- #define [ZA_OFFSETS_H](#) 0x1A /* ZA_OFFSETS[14:7] */
- #define [ZA_OFFSETS_L](#) 0x1B /* ZA_OFFSETS[6:0] */
- #define [TIMEBASE_CORRECTION_PLL](#) 0x28 /* TBC_PLL[7:0] */
- #define [BANK1_REG_BANK_SEL](#) 0x7F /* mirror of [REG_BANK_SEL](#) */
- #define [GYRO_SMPLRT_DIV](#) 0x00 /* GYRO_SMPLRT_DIV[7:0] */
- #define [GYRO_CONFIG_1](#) 0x01 /* GYRO_DLPFCFG[2:0] GYRO_FS_SEL[1:0] [GYRO_FCHOICE](#) */
- #define [GYRO_FCHOICE](#) BIT(0) /* when 0: DLPF on, when 1: off (per datasheet) */
- #define [GYRO_FS_SEL_SHIFT](#) 1
- #define [GYRO_FS_SEL_MASK](#) (0x3u << [GYRO_FS_SEL_SHIFT](#))
- #define [GYRO_FS_250DPS](#) (0u << [GYRO_FS_SEL_SHIFT](#))
- #define [GYRO_FS_500DPS](#) (1u << [GYRO_FS_SEL_SHIFT](#))
- #define [GYRO_FS_1000DPS](#) (2u << [GYRO_FS_SEL_SHIFT](#))
- #define [GYRO_FS_2000DPS](#) (3u << [GYRO_FS_SEL_SHIFT](#))
- #define [GYRO_DLPFCFG_SHIFT](#) 5
- #define [GYRO_DLPFCFG_MASK](#) (0x7u << [GYRO_DLPFCFG_SHIFT](#))
- #define [GYRO_CONFIG_2](#) 0x02 /* [XGYRO_CTEN](#) [YGYRO_CTEN](#) [ZGYRO_CTEN](#) [GYRO_AVGCFG](#)[2:0] */
- #define [XGYRO_CTEN](#) BIT(5)
- #define [YGYRO_CTEN](#) BIT(4)
- #define [ZGYRO_CTEN](#) BIT(3)
- #define [GYRO_AVGCFG_SHIFT](#) 0
- #define [GYRO_AVGCFG_MASK](#) (0x7u << [GYRO_AVGCFG_SHIFT](#))
- #define [XG_OFFSETS_USRH](#) 0x03
- #define [XG_OFFSETS_USRL](#) 0x04
- #define [YG_OFFSETS_USRH](#) 0x05
- #define [YG_OFFSETS_USRL](#) 0x06
- #define [ZG_OFFSETS_USRH](#) 0x07
- #define [ZG_OFFSETS_USRL](#) 0x08
- #define [ODR_ALIGN_EN](#) 0x09 /* ODR_ALIGN_EN */
- #define [ODR_ALIGN_EN_BIT](#) BIT(0)
- #define [ACCEL_SMPLRT_DIV_1](#) 0x10 /* ACCEL_SMPLRT_DIV[11:8] */
- #define [ACCEL_SMPLRT_DIV_2](#) 0x11 /* ACCEL_SMPLRT_DIV[7:0] */
- #define [ACCEL_INTEL_CTRL](#) 0x12 /* [ACCEL_INTEL_EN](#) [ACCEL_INTEL_MODE_INT](#) */
- #define [ACCEL_INTEL_EN](#) BIT(7)
- #define [ACCEL_INTEL_MODE_INT](#) BIT(6)
- #define [ACCEL_WOM_THR](#) 0x13 /* WOM_THRESHOLD[7:0] */
- #define [ACCEL_CONFIG](#) 0x14 /* ACCEL_DLPFCFG[2:0] ACCEL_FS_SEL[1:0] [ACCEL_FCHOICE](#) */
- #define [ACCEL_FCHOICE](#) BIT(0)
- #define [ACCEL_FS_SEL_SHIFT](#) 1
- #define [ACCEL_FS_SEL_MASK](#) (0x3u << [ACCEL_FS_SEL_SHIFT](#))
- #define [ACCEL_FS_2G](#) (0u << [ACCEL_FS_SEL_SHIFT](#))
- #define [ACCEL_FS_4G](#) (1u << [ACCEL_FS_SEL_SHIFT](#))
- #define [ACCEL_FS_8G](#) (2u << [ACCEL_FS_SEL_SHIFT](#))
- #define [ACCEL_FS_16G](#) (3u << [ACCEL_FS_SEL_SHIFT](#))
- #define [ACCEL_DLPFCFG_SHIFT](#) 5
- #define [ACCEL_DLPFCFG_MASK](#) (0x7u << [ACCEL_DLPFCFG_SHIFT](#))
- #define [ACCEL_CONFIG_2](#) 0x15 /* [AX_ST_EN_REG](#) [AY_ST_EN_REG](#) [AZ_ST_EN_REG](#) [DEC3_CFG](#)[1:0] */

- #define `AX_ST_EN_REG` BIT(7)
- #define `AY_ST_EN_REG` BIT(6)
- #define `AZ_ST_EN_REG` BIT(5)
- #define `DEC3_CFG_SHIFT` 0
- #define `DEC3_CFG_MASK` (0x3u << `DEC3_CFG_SHIFT`)
- #define `FSYNC_CONFIG` 0x52 /* `DELAY_TIME_EN` `WOF DEGLITCH_EN` `WOF_EDGE_INT` `EXT_SYNC_SET[3:0]` */
- #define `DELAY_TIME_EN` BIT(7)
- #define `WOF DEGLITCH_EN` BIT(6)
- #define `WOF_EDGE_INT` BIT(5)
- #define `EXT_SYNC_SET_MASK` 0x0F
- #define `TEMP_CONFIG` 0x53 /* `TEMP_DLPFCFG[2:0]` */
- #define `TEMP_DLPFCFG_SHIFT` 5
- #define `TEMP_DLPFCFG_MASK` (0x7u << `TEMP_DLPFCFG_SHIFT`)
- #define `MOD_CTRL_USR` 0x54 /* `REG_LP_DMP_EN` */
- #define `REG_LP_DMP_EN` BIT(7)
- #define `BANK2_REG_BANK_SEL` 0x7F /* mirror of `REG_BANK_SEL` */
- #define `I2C_MST_ODR_CONFIG` 0x00 /* `I2C_MST_ODR_CONFIG[3:0]` */
- #define `MST_ODR_CFG_MASK` 0x0F
- #define `I2C_MST_CTRL` 0x01 /* `MULT_MST_EN` - `I2C_MST_P_NSR` `I2C_MST_CLK[3:0]` */
- #define `MULT_MST_EN` BIT(7)
- #define `I2C_MST_P_NSR` BIT(4)
- #define `I2C_MST_CLK_MASK` 0x0F
- #define `I2C_MST_DELAY_CTRL` 0x02 /* `DELAY_ES_SHADOW` + `I2C_SLVx_DELAY_EN` bits */
- #define `DELAY_ES_SHADOW` BIT(7)
- #define `I2C_SLV4_DELAY_EN` BIT(4)
- #define `I2C_SLV3_DELAY_EN` BIT(3)
- #define `I2C_SLV2_DELAY_EN` BIT(2)
- #define `I2C_SLV1_DELAY_EN` BIT(1)
- #define `I2C_SLV0_DELAY_EN` BIT(0)
- #define `I2C_SLV0_ADDR` 0x03 /* `RNW` + `ID[6:0]` */
- #define `I2C_SLVx_RNW` BIT(7)
- #define `I2C_SLV0_REG` 0x04
- #define `I2C_SLV0_CTRL` 0x05 /* `EN BYTE_SW` `REG_DIS` `GRP LENG[3:0]` */
- #define `I2C_SLVx_EN` BIT(7)
- #define `I2C_SLVx_BYTE_SW` BIT(6)
- #define `I2C_SLVx_REG_DIS` BIT(5)
- #define `I2C_SLVx_GRP` BIT(4)
- #define `I2C_SLVx_LENG_MASK` 0x0F
- #define `I2C_SLV0_DO` 0x06
- #define `I2C_SLV1_ADDR` 0x07
- #define `I2C_SLV1_REG` 0x08
- #define `I2C_SLV1_CTRL` 0x09
- #define `I2C_SLV1_DO` 0x0A
- #define `I2C_SLV2_ADDR` 0x0B
- #define `I2C_SLV2_REG` 0x0C
- #define `I2C_SLV2_CTRL` 0x0D
- #define `I2C_SLV2_DO` 0x0E
- #define `I2C_SLV3_ADDR` 0x0F
- #define `I2C_SLV3_REG` 0x10
- #define `I2C_SLV3_CTRL` 0x11
- #define `I2C_SLV3_DO` 0x12
- #define `I2C_SLV4_ADDR` 0x13
- #define `I2C_SLV4_REG` 0x14
- #define `I2C_SLV4_CTRL` 0x15 /* `EN BYTE_SW` `REG_DIS` `DLY[4:0]` */

- #define I2C_SLV4_DLY_MASK 0x1F
- #define I2C_SLV4_DO 0x16
- #define I2C_SLV4_DI 0x17
- #define BANK3_REG_BANK_SEL 0x7F /* mirror of REG_BANK_SEL */
- #define REG_BANK_SEL_USER_BANK_SHIFT 4
- #define REG_BANK_SEL_USER_BANK_MASK (0x3u << REG_BANK_SEL_USER_BANK_SHIFT)
- #define USER_BANK_0 BANK0
- #define USER_BANK_1 BANK1
- #define USER_BANK_2 BANK2
- #define USER_BANK_3 BANK3
- #define AK09916_I2C_ADDR 0x0C
- #define AK_WIA2 0x01
- #define AK_ST1 0x10
- #define AK_HXL 0x11
- #define AK_ST2 0x18
- #define AK_CNTL2 0x31
- #define AK_CNTL3 0x32
- #define AK_WIA2_VAL 0x09 /* AK09916C WIA2 expected */
- #define INTERNAL_20MHZ 0x00 /* 0: Internal 20 MHz RC */
- #define AUTO_SEL 0x01 /* 1–5: Auto/PLL preferred (use 1 as default) */
- #define CLK_STOP 0x07 /* 7: Stop clock / timing gen reset */
- #define g2 0
- #define g4 1
- #define g8 2
- #define g16 3
- #define ACCEL_FCHOICE_BYPASS 0
- #define ACCEL_FCHOICE_DLPF 1
- #define ACCEL_DLPFCFG_0 0
- #define ACCEL_DLPFCFG_1 1
- #define ACCEL_DLPFCFG_2 2
- #define ACCEL_DLPFCFG_3 3
- #define ACCEL_DLPFCFG_4 4
- #define ACCEL_DLPFCFG_5 5
- #define ACCEL_DLPFCFG_6 6
- #define ACCEL_DLPFCFG_7 7
- #define ACCEL_DEC3_AVG_4 0u /* averages 1 or 4 (depends on FCHOICE) */
- #define ACCEL_DEC3_AVG_8 1u
- #define ACCEL_DEC3_AVG_16 2u
- #define ACCEL_DEC3_AVG_32 3u
- #define ACCEL_SMPLRT_DIV_MIN 0u
- #define ACCEL_SMPLRT_DIV_MAX 4095u
- #define ACCEL_DLPF_BASE_HZ 1125.0f
- #define ACCEL_DLPF_ODR_HZ(div) (ACCEL_DLPF_BASE_HZ / (1.0f + (float)(div)))
- #define ACCEL_BYPASS_3DB_BW_HZ 1209.0f
- #define ACCEL_BYPASS_NBW_HZ 1248.0f
- #define ACCEL_BYPASS_RATE_HZ 4500.0f
- #define ACCEL_DLPF0_3DB_BW_HZ 246.0f
- #define ACCEL_DLPF0_NBW_HZ 265.0f
- #define ACCEL_DLPF1_3DB_BW_HZ 246.0f
- #define ACCEL_DLPF1_NBW_HZ 265.0f
- #define ACCEL_DLPF2_3DB_BW_HZ 111.4f
- #define ACCEL_DLPF2_NBW_HZ 136.0f
- #define ACCEL_DLPF3_3DB_BW_HZ 50.4f
- #define ACCEL_DLPF3_NBW_HZ 68.8f
- #define ACCEL_DLPF4_3DB_BW_HZ 23.9f

- #define [ACCEL_DLPF4_NBW_HZ](#) 34.4f
- #define [ACCEL_DLPF5_3DB_BW_HZ](#) 11.5f
- #define [ACCEL_DLPF5_NBW_HZ](#) 17.0f
- #define [ACCEL_DLPF6_3DB_BW_HZ](#) 5.7f
- #define [ACCEL_DLPF6_NBW_HZ](#) 8.3f
- #define [ACCEL_DLPF7_3DB_BW_HZ](#) 473.0f
- #define [ACCEL_DLPF7_NBW_HZ](#) 499.0f
- #define [dps250](#) 0
- #define [dps500](#) 1
- #define [dps1000](#) 2
- #define [dps2000](#) 3
- #define [GYRO_FCHOICE_BYPASS](#) 0
- #define [GYRO_FCHOICE_DLPF](#) 1
- #define [GYRO_DLPFCFG_0](#) 0
- #define [GYRO_DLPFCFG_1](#) 1
- #define [GYRO_DLPFCFG_2](#) 2
- #define [GYRO_DLPFCFG_3](#) 3
- #define [GYRO_DLPFCFG_4](#) 4
- #define [GYRO_DLPFCFG_5](#) 5
- #define [GYRO_DLPFCFG_6](#) 6
- #define [GYRO_DLPFCFG_7](#) 7
- #define [GYRO_SMPLRT_MIN_HZ](#) 4.3f
- #define [GYRO_DLPF_BASE_HZ](#) 1100.0f
- #define [GYRO_DLPF_ODR_HZ](#)(rate_request) (rate_request) /* symbolic; your driver computes DIV = round(1100/rate)-1 */
- #define [GYRO_BW_ULTRA_WIDE](#) 0 /* bypass path */
- #define [GYRO_BW_WIDE](#) 1 /* e.g., CFG 0/1 */
- #define [GYRO_BW_MEDIUM](#) 2 /* e.g., CFG 2/3 */
- #define [GYRO_BW_NARROW](#) 3 /* e.g., CFG 4/5/6/7 */
- #define [Hz2](#) 0
- #define [Hz6](#) 1
- #define [Hz8](#) 2
- #define [Hz10](#) 3
- #define [Hz15](#) 4
- #define [Hz20](#) 5
- #define [Hz25](#) 6
- #define [Hz30](#) 7
- #define [LowPower](#) 0
- #define [Regular](#) 1
- #define [EnhancedRegular](#) 2
- #define [HighAccuracy](#) 3

6.40.1 Macro Definition Documentation

6.40.1.1 ACCEL_BYPASS_3DB_BW_HZ

#define [ACCEL_BYPASS_3DB_BW_HZ](#) 1209.0f
Definition at line [369](#) of file [ICM20948_regs.h](#).

6.40.1.2 ACCEL_BYPASS_NBW_HZ

#define [ACCEL_BYPASS_NBW_HZ](#) 1248.0f
Definition at line [370](#) of file [ICM20948_regs.h](#).

6.40.1.3 ACCEL_BYPASS_RATE_HZ

#define ACCEL_BYPASS_RATE_HZ 4500.0f
Definition at line 371 of file [ICM20948_regs.h](#).

6.40.1.4 ACCEL_CONFIG

#define ACCEL_CONFIG 0x14 /* ACCEL_DLPFCFG[2:0] ACCEL_FS_SEL[1:0] ACCEL_FCHOICE */
Definition at line 209 of file [ICM20948_regs.h](#).

6.40.1.5 ACCEL_CONFIG_2

#define ACCEL_CONFIG_2 0x15 /* AX_ST_EN_REG AY_ST_EN_REG AZ_ST_EN_REG DEC3_CFG[1:0] */
Definition at line 220 of file [ICM20948_regs.h](#).

6.40.1.6 ACCEL_DEC3_AVG_16

#define ACCEL_DEC3_AVG_16 2u
Definition at line 356 of file [ICM20948_regs.h](#).

6.40.1.7 ACCEL_DEC3_AVG_32

#define ACCEL_DEC3_AVG_32 3u
Definition at line 357 of file [ICM20948_regs.h](#).

6.40.1.8 ACCEL_DEC3_AVG_4

#define ACCEL_DEC3_AVG_4 0u /* averages 1 or 4 (depends on FCHOICE) */
Definition at line 354 of file [ICM20948_regs.h](#).

6.40.1.9 ACCEL_DEC3_AVG_8

#define ACCEL_DEC3_AVG_8 1u
Definition at line 355 of file [ICM20948_regs.h](#).

6.40.1.10 ACCEL_DLPF0_3DB_BW_HZ

#define ACCEL_DLPF0_3DB_BW_HZ 246.0f
Definition at line 376 of file [ICM20948_regs.h](#).

6.40.1.11 ACCEL_DLPF0_NBW_HZ

#define ACCEL_DLPF0_NBW_HZ 265.0f
Definition at line 377 of file [ICM20948_regs.h](#).

6.40.1.12 ACCEL_DLPF1_3DB_BW_HZ

#define ACCEL_DLPF1_3DB_BW_HZ 246.0f
Definition at line 378 of file [ICM20948_regs.h](#).

6.40.1.13 ACCEL_DLPF1_NBW_HZ

#define ACCEL_DLPF1_NBW_HZ 265.0f
Definition at line 379 of file [ICM20948_regs.h](#).

6.40.1.14 ACCEL_DLPF2_3DB_BW_HZ

#define ACCEL_DLPF2_3DB_BW_HZ 111.4f
Definition at line 380 of file [ICM20948_regs.h](#).

6.40.1.15 ACCEL_DLPF2_NBW_HZ

#define ACCEL_DLPF2_NBW_HZ 136.0f
Definition at line 381 of file [ICM20948_regs.h](#).

6.40.1.16 ACCEL_DLPF3_3DB_BW_HZ

#define ACCEL_DLPF3_3DB_BW_HZ 50.4f
Definition at line 382 of file [ICM20948_regs.h](#).

6.40.1.17 ACCEL_DLPF3_NBW_HZ

#define ACCEL_DLPF3_NBW_HZ 68.8f
Definition at line 383 of file [ICM20948_regs.h](#).

6.40.1.18 ACCEL_DLPF4_3DB_BW_HZ

#define ACCEL_DLPF4_3DB_BW_HZ 23.9f
Definition at line 384 of file [ICM20948_regs.h](#).

6.40.1.19 ACCEL_DLPF4_NBW_HZ

#define ACCEL_DLPF4_NBW_HZ 34.4f
Definition at line 385 of file [ICM20948_regs.h](#).

6.40.1.20 ACCEL_DLPF5_3DB_BW_HZ

#define ACCEL_DLPF5_3DB_BW_HZ 11.5f
Definition at line 386 of file [ICM20948_regs.h](#).

6.40.1.21 ACCEL_DLPF5_NBW_HZ

#define ACCEL_DLPF5_NBW_HZ 17.0f
Definition at line 387 of file [ICM20948_regs.h](#).

6.40.1.22 ACCEL_DLPF6_3DB_BW_HZ

#define ACCEL_DLPF6_3DB_BW_HZ 5.7f
Definition at line 388 of file [ICM20948_regs.h](#).

6.40.1.23 ACCEL_DLPF6_NBW_HZ

#define ACCEL_DLPF6_NBW_HZ 8.3f
Definition at line 389 of file [ICM20948_regs.h](#).

6.40.1.24 ACCEL_DLPF7_3DB_BW_HZ

#define ACCEL_DLPF7_3DB_BW_HZ 473.0f
Definition at line 390 of file [ICM20948_regs.h](#).

6.40.1.25 ACCEL_DLPF7_NBW_HZ

#define ACCEL_DLPF7_NBW_HZ 499.0f
Definition at line 391 of file [ICM20948_regs.h](#).

6.40.1.26 ACCEL_DLPF_BASE_HZ

#define ACCEL_DLPF_BASE_HZ 1125.0f
Definition at line 362 of file [ICM20948_regs.h](#).

6.40.1.27 ACCEL_DLPF_ODR_HZ

```
#define ACCEL_DLPF_ODR_HZ(  
    div ) (ACCEL_DLPF_BASE_HZ / (1.0f + (float)(div)))
```

Definition at line 363 of file [ICM20948_regs.h](#).

6.40.1.28 ACCEL_DLPFCFG_0

```
#define ACCEL_DLPFCFG_0 0
```

ACCEL_DLPFCFG (0..7) — choose cutoff/noise BW set (when DLPF is ON). Tip: start with ACCEL_DLPFCFG_3 for a good noise/latency tradeoff.

Definition at line 345 of file [ICM20948_regs.h](#).

6.40.1.29 ACCEL_DLPFCFG_1

```
#define ACCEL_DLPFCFG_1 1
```

Definition at line 346 of file [ICM20948_regs.h](#).

6.40.1.30 ACCEL_DLPFCFG_2

```
#define ACCEL_DLPFCFG_2 2
```

Definition at line 347 of file [ICM20948_regs.h](#).

6.40.1.31 ACCEL_DLPFCFG_3

```
#define ACCEL_DLPFCFG_3 3
```

Definition at line 348 of file [ICM20948_regs.h](#).

6.40.1.32 ACCEL_DLPFCFG_4

```
#define ACCEL_DLPFCFG_4 4
```

Definition at line 349 of file [ICM20948_regs.h](#).

6.40.1.33 ACCEL_DLPFCFG_5

```
#define ACCEL_DLPFCFG_5 5
```

Definition at line 350 of file [ICM20948_regs.h](#).

6.40.1.34 ACCEL_DLPFCFG_6

```
#define ACCEL_DLPFCFG_6 6
```

Definition at line 351 of file [ICM20948_regs.h](#).

6.40.1.35 ACCEL_DLPFCFG_7

```
#define ACCEL_DLPFCFG_7 7
```

Definition at line 352 of file [ICM20948_regs.h](#).

6.40.1.36 ACCEL_DLPFCFG_MASK

```
#define ACCEL_DLPFCFG_MASK (0x7u << ACCEL_DLPFCFG_SHIFT)
```

Definition at line 218 of file [ICM20948_regs.h](#).

6.40.1.37 ACCEL_DLPFCFG_SHIFT

```
#define ACCEL_DLPFCFG_SHIFT 5
```

Definition at line 217 of file [ICM20948_regs.h](#).

6.40.1.38 ACCEL_FCHOICE

#define ACCEL_FCHOICE BIT(0)

Definition at line 210 of file [ICM20948_regs.h](#).

6.40.1.39 ACCEL_FCHOICE_BYPASS

#define ACCEL_FCHOICE_BYPASS 0

Path select for accel:

- ACCEL_FCHOICE_BYPASS : DLPF bypassed (very wide BW, fastest)
- ACCEL_FCHOICE_DLPF : DLPF enabled (use ACCEL_DLPFCFG + SMPLRT_DIV)

Definition at line 337 of file [ICM20948_regs.h](#).

6.40.1.40 ACCEL_FCHOICE_DLPF

#define ACCEL_FCHOICE_DLPF 1

Definition at line 338 of file [ICM20948_regs.h](#).

6.40.1.41 ACCEL_FS_16G

#define ACCEL_FS_16G (3u << ACCEL_FS_SEL_SHIFT)

Definition at line 216 of file [ICM20948_regs.h](#).

6.40.1.42 ACCEL_FS_2G

#define ACCEL_FS_2G (0u << ACCEL_FS_SEL_SHIFT)

Definition at line 213 of file [ICM20948_regs.h](#).

6.40.1.43 ACCEL_FS_4G

#define ACCEL_FS_4G (1u << ACCEL_FS_SEL_SHIFT)

Definition at line 214 of file [ICM20948_regs.h](#).

6.40.1.44 ACCEL_FS_8G

#define ACCEL_FS_8G (2u << ACCEL_FS_SEL_SHIFT)

Definition at line 215 of file [ICM20948_regs.h](#).

6.40.1.45 ACCEL_FS_SEL_MASK

#define ACCEL_FS_SEL_MASK (0x3u << ACCEL_FS_SEL_SHIFT)

Definition at line 212 of file [ICM20948_regs.h](#).

6.40.1.46 ACCEL_FS_SEL_SHIFT

#define ACCEL_FS_SEL_SHIFT 1

Definition at line 211 of file [ICM20948_regs.h](#).

6.40.1.47 ACCEL_INTEL_CTRL

#define ACCEL_INTEL_CTRL 0x12 /* ACCEL_INTEL_EN ACCEL_INTEL_MODE_INT */

Definition at line 203 of file [ICM20948_regs.h](#).

6.40.1.48 ACCEL_INTEL_EN

#define ACCEL_INTEL_EN BIT(7)

Definition at line 204 of file [ICM20948_regs.h](#).

6.40.1.49 ACCEL_INTEL_MODE_INT

```
#define ACCEL_INTEL_MODE_INT BIT(6)
```

Definition at line 205 of file [ICM20948_regs.h](#).

6.40.1.50 ACCEL_SMPLRT_DIV_1

```
#define ACCEL_SMPLRT_DIV_1 0x10 /* ACCEL_SMPLRT_DIV[11:8] */
```

Definition at line 200 of file [ICM20948_regs.h](#).

6.40.1.51 ACCEL_SMPLRT_DIV_2

```
#define ACCEL_SMPLRT_DIV_2 0x11 /* ACCEL_SMPLRT_DIV[7:0] */
```

Definition at line 201 of file [ICM20948_regs.h](#).

6.40.1.52 ACCEL_SMPLRT_DIV_MAX

```
#define ACCEL_SMPLRT_DIV_MAX 4095u
```

Definition at line 361 of file [ICM20948_regs.h](#).

6.40.1.53 ACCEL_SMPLRT_DIV_MIN

```
#define ACCEL_SMPLRT_DIV_MIN 0u
```

Definition at line 360 of file [ICM20948_regs.h](#).

6.40.1.54 ACCEL_WOM_THR

```
#define ACCEL_WOM_THR 0x13 /* WOM_THRESHOLD[7:0] */
```

Definition at line 207 of file [ICM20948_regs.h](#).

6.40.1.55 ACCEL_XOUT_H

```
#define ACCEL_XOUT_H 0x2D
```

Definition at line 111 of file [ICM20948_regs.h](#).

6.40.1.56 ACCEL_XOUT_L

```
#define ACCEL_XOUT_L 0x2E
```

Definition at line 112 of file [ICM20948_regs.h](#).

6.40.1.57 ACCEL_YOUT_H

```
#define ACCEL_YOUT_H 0x2F
```

Definition at line 113 of file [ICM20948_regs.h](#).

6.40.1.58 ACCEL_YOUT_L

```
#define ACCEL_YOUT_L 0x30
```

Definition at line 114 of file [ICM20948_regs.h](#).

6.40.1.59 ACCEL_ZOUT_H

```
#define ACCEL_ZOUT_H 0x31
```

Definition at line 115 of file [ICM20948_regs.h](#).

6.40.1.60 ACCEL_ZOUT_L

```
#define ACCEL_ZOUT_L 0x32
```

Definition at line 116 of file [ICM20948_regs.h](#).

6.40.1.61 AK09916_I2C_ADDR

```
#define AK09916_I2C_ADDR 0x0C
```

Definition at line 310 of file [ICM20948_regs.h](#).

6.40.1.62 AK_CNTL2

```
#define AK_CNTL2 0x31
```

Definition at line 315 of file [ICM20948_regs.h](#).

6.40.1.63 AK_CNTL3

```
#define AK_CNTL3 0x32
```

Definition at line 316 of file [ICM20948_regs.h](#).

6.40.1.64 AK_HXL

```
#define AK_HXL 0x11
```

Definition at line 313 of file [ICM20948_regs.h](#).

6.40.1.65 AK_ST1

```
#define AK_ST1 0x10
```

Definition at line 312 of file [ICM20948_regs.h](#).

6.40.1.66 AK_ST2

```
#define AK_ST2 0x18
```

Definition at line 314 of file [ICM20948_regs.h](#).

6.40.1.67 AK_WIA2

```
#define AK_WIA2 0x01
```

Definition at line 311 of file [ICM20948_regs.h](#).

6.40.1.68 AK_WIA2_VAL

```
#define AK_WIA2_VAL 0x09 /* AK09916C WIA2 expected */
```

Definition at line 317 of file [ICM20948_regs.h](#).

6.40.1.69 AUTO_SEL

```
#define AUTO_SEL 0x01 /* 1-5: Auto/PLL preferred (use 1 as default) */
```

Definition at line 320 of file [ICM20948_regs.h](#).

6.40.1.70 AX_ST_EN_REG

```
#define AX_ST_EN_REG BIT(7)
```

Definition at line 221 of file [ICM20948_regs.h](#).

6.40.1.71 AY_ST_EN_REG

```
#define AY_ST_EN_REG BIT(6)
```

Definition at line 222 of file [ICM20948_regs.h](#).

6.40.1.72 AZ_ST_EN_REG

```
#define AZ_ST_EN_REG BIT(5)
```

Definition at line 223 of file [ICM20948_regs.h](#).

6.40.1.73 BANK1_REG_BANK_SEL

```
#define BANK1_REG_BANK_SEL 0x7F /* mirror of REG_BANK_SEL */  
Definition at line 163 of file ICM20948_regs.h.
```

6.40.1.74 BANK2_REG_BANK_SEL

```
#define BANK2_REG_BANK_SEL 0x7F /* mirror of REG_BANK_SEL */  
Definition at line 240 of file ICM20948_regs.h.
```

6.40.1.75 BANK3_REG_BANK_SEL

```
#define BANK3_REG_BANK_SEL 0x7F /* mirror of REG_BANK_SEL */  
Definition at line 296 of file ICM20948_regs.h.
```

6.40.1.76 BYPASS_EN

```
#define BYPASS_EN BIT(1) /* I2C bypass to aux devices */  
Definition at line 71 of file ICM20948_regs.h.
```

6.40.1.77 CLK_STOP

```
#define CLK_STOP 0x07 /* 7: Stop clock / timing gen reset */  
Definition at line 321 of file ICM20948_regs.h.
```

6.40.1.78 DEC3_CFG_MASK

```
#define DEC3_CFG_MASK (0x3u << DEC3_CFG_SHIFT)  
Definition at line 225 of file ICM20948_regs.h.
```

6.40.1.79 DEC3_CFG_SHIFT

```
#define DEC3_CFG_SHIFT 0  
Definition at line 224 of file ICM20948_regs.h.
```

6.40.1.80 DELAY_ES_SHADOW

```
#define DELAY_ES_SHADOW BIT(7)  
Definition at line 256 of file ICM20948_regs.h.
```

6.40.1.81 DELAY_TIME_EN

```
#define DELAY_TIME_EN BIT(7)  
Definition at line 228 of file ICM20948_regs.h.
```

6.40.1.82 DELAY_TIMEH

```
#define DELAY_TIMEH 0x28  
Definition at line 107 of file ICM20948_regs.h.
```

6.40.1.83 DELAY_TIMEL

```
#define DELAY_TIMEL 0x29  
Definition at line 108 of file ICM20948_regs.h.
```

6.40.1.84 dps1000

```
#define dps1000 2  
Definition at line 398 of file ICM20948_regs.h.
```

6.40.1.85 dps2000

```
#define dps2000 3
```

Definition at line 399 of file [ICM20948_regs.h](#).

6.40.1.86 dps250

```
#define dps250 0
```

- $\pm 250/\pm 500/\pm 1000/\pm 2000$ dps (maps to FS_SEL 0..3)

Definition at line 396 of file [ICM20948_regs.h](#).

6.40.1.87 dps500

```
#define dps500 1
```

Definition at line 397 of file [ICM20948_regs.h](#).

6.40.1.88 EnhancedRegular

```
#define EnhancedRegular 2
```

Definition at line 456 of file [ICM20948_regs.h](#).

6.40.1.89 EXT_SLV_SENS_DATA_00

```
#define EXT_SLV_SENS_DATA_00 0x3B
```

Definition at line 127 of file [ICM20948_regs.h](#).

6.40.1.90 EXT_SLV_SENS_DATA_23

```
#define EXT_SLV_SENS_DATA_23 0x52 /* contiguous range 0x3B..0x52 */
```

Definition at line 128 of file [ICM20948_regs.h](#).

6.40.1.91 EXT_SYNC_SET_MASK

```
#define EXT_SYNC_SET_MASK 0x0F
```

Definition at line 231 of file [ICM20948_regs.h](#).

6.40.1.92 FIFO_CFG

```
#define FIFO_CFG 0x76
```

Definition at line 138 of file [ICM20948_regs.h](#).

6.40.1.93 FIFO_COUNTH

```
#define FIFO_COUNTH 0x70
```

Definition at line 134 of file [ICM20948_regs.h](#).

6.40.1.94 FIFO_COUNTL

```
#define FIFO_COUNTL 0x71
```

Definition at line 135 of file [ICM20948_regs.h](#).

6.40.1.95 FIFO_EN_1

```
#define FIFO_EN_1 0x66 /* SLV3..SLV0 FIFO_EN */
```

Definition at line 130 of file [ICM20948_regs.h](#).

6.40.1.96 FIFO_EN_2

```
#define FIFO_EN_2 0x67 /* ACCEL GYRO_Z/Y/X TEMP FIFO_EN */
```

Definition at line 131 of file [ICM20948_regs.h](#).

6.40.1.97 FIFO_MODE

```
#define FIFO_MODE 0x69 /* FIFO_MODE[4:0] */
```

Definition at line 133 of file [ICM20948_regs.h](#).

6.40.1.98 FIFO_R_W

```
#define FIFO_R_W 0x72
```

Definition at line 136 of file [ICM20948_regs.h](#).

6.40.1.99 FIFO_RST

```
#define FIFO_RST 0x68 /* FIFO_RESET[4:0] */
```

Definition at line 132 of file [ICM20948_regs.h](#).

6.40.1.100 FSYNC_CONFIG

```
#define FSYNC_CONFIG 0x52 /* DELAY_TIME_EN WOF_DEGLITCH_EN WOF_EDGE_INT EXT_SYNC_SET[3:0] */
```

Definition at line 227 of file [ICM20948_regs.h](#).

6.40.1.101 FSYNC_INT_MODE_EN

```
#define FSYNC_INT_MODE_EN BIT(2)
```

Definition at line 70 of file [ICM20948_regs.h](#).

6.40.1.102 g16

```
#define g16 3
```

Definition at line 329 of file [ICM20948_regs.h](#).

6.40.1.103 g2

```
#define g2 0
```

- $\pm 2g$ / $\pm 4g$ / $\pm 8g$ / $\pm 16g$ (maps to FS_SEL 0..3)

Definition at line 326 of file [ICM20948_regs.h](#).

6.40.1.104 g4

```
#define g4 1
```

Definition at line 327 of file [ICM20948_regs.h](#).

6.40.1.105 g8

```
#define g8 2
```

Definition at line 328 of file [ICM20948_regs.h](#).

6.40.1.106 GYRO_AVGCFG_MASK

```
#define GYRO_AVGCFG_MASK (0x7u << GYRO_AVGCFG_SHIFT)
```

Definition at line 188 of file [ICM20948_regs.h](#).

6.40.1.107 GYRO_AVGCFG_SHIFT

```
#define GYRO_AVGCFG_SHIFT 0
```

Definition at line 187 of file [ICM20948_regs.h](#).

6.40.1.108 GYRO_BW_MEDIUM

```
#define GYRO_BW_MEDIUM 2 /* e.g., CFG 2/3 */
```

Definition at line 435 of file [ICM20948_regs.h](#).

6.40.1.109 GYRO_BW_NARROW

```
#define GYRO_BW_NARROW 3 /* e.g., CFG 4/5/6/7 */
```

Definition at line 436 of file [ICM20948_regs.h](#).

6.40.1.110 GYRO_BW_ULTRA_WIDE

```
#define GYRO_BW_ULTRA_WIDE 0 /* bypass path */
```

Definition at line 433 of file [ICM20948_regs.h](#).

6.40.1.111 GYRO_BW_WIDE

```
#define GYRO_BW_WIDE 1 /* e.g., CFG 0/1 */
```

Definition at line 434 of file [ICM20948_regs.h](#).

6.40.1.112 GYRO_CONFIG_1

```
#define GYRO_CONFIG_1 0x01 /* GYRO_DLPFCFG[2:0] GYRO_FS_SEL[1:0] GYRO_FCHOICE */
```

Definition at line 172 of file [ICM20948_regs.h](#).

6.40.1.113 GYRO_CONFIG_2

```
#define GYRO_CONFIG_2 0x02 /* XGYRO_CTEN YGYRO_CTEN ZGYRO_CTEN GYRO_AVGCFG[2:0] */
```

Definition at line 183 of file [ICM20948_regs.h](#).

6.40.1.114 GYRO_DLPF_BASE_HZ

```
#define GYRO_DLPF_BASE_HZ 1100.0f
```

Definition at line 426 of file [ICM20948_regs.h](#).

6.40.1.115 GYRO_DLPF_ODR_HZ

```
#define GYRO_DLPF_ODR_HZ(  
    rate_request ) (rate_request) /* symbolic; your driver computes DIV = round(1100/rate)-1  
*/
```

Definition at line 427 of file [ICM20948_regs.h](#).

6.40.1.116 GYRO_DLPFCFG_0

```
#define GYRO_DLPFCFG_0 0
```

GYRO_DLPFCFG (0..7) — choose cutoff/noise BW set (when DLPF is ON). Tip: start with GYRO_DLPFCFG_3 or _2 for balanced noise/latency.

Definition at line 415 of file [ICM20948_regs.h](#).

6.40.1.117 GYRO_DLPFCFG_1

```
#define GYRO_DLPFCFG_1 1
```

Definition at line 416 of file [ICM20948_regs.h](#).

6.40.1.118 GYRO_DLPFCFG_2

```
#define GYRO_DLPFCFG_2 2
```

Definition at line 417 of file [ICM20948_regs.h](#).

6.40.1.119 GYRO_DLPFCFG_3

```
#define GYRO_DLPFCFG_3 3
```

Definition at line 418 of file [ICM20948_regs.h](#).

6.40.1.120 GYRO_DLPFCFG_4

```
#define GYRO_DLPFCFG_4 4
```

Definition at line 419 of file [ICM20948_regs.h](#).

6.40.1.121 GYRO_DLPFCFG_5

```
#define GYRO_DLPFCFG_5 5
```

Definition at line 420 of file [ICM20948_regs.h](#).

6.40.1.122 GYRO_DLPFCFG_6

```
#define GYRO_DLPFCFG_6 6
```

Definition at line 421 of file [ICM20948_regs.h](#).

6.40.1.123 GYRO_DLPFCFG_7

```
#define GYRO_DLPFCFG_7 7
```

Definition at line 422 of file [ICM20948_regs.h](#).

6.40.1.124 GYRO_DLPFCFG_MASK

```
#define GYRO_DLPFCFG_MASK (0x7u << GYRO_DLPFCFG_SHIFT)
```

Definition at line 181 of file [ICM20948_regs.h](#).

6.40.1.125 GYRO_DLPFCFG_SHIFT

```
#define GYRO_DLPFCFG_SHIFT 5
```

Definition at line 180 of file [ICM20948_regs.h](#).

6.40.1.126 GYRO_FCHOICE

```
#define GYRO_FCHOICE BIT(0) /* when 0: DLPF on, when 1: off (per datasheet) */
```

Definition at line 173 of file [ICM20948_regs.h](#).

6.40.1.127 GYRO_FCHOICE_BYPASS

```
#define GYRO_FCHOICE_BYPASS 0
```

Path select for gyro:

- GYRO_FCHOICE_BYPASS : DLPF bypassed (widest BW, fastest)
- GYRO_FCHOICE_DLPF : DLPF enabled (use GYRO_DLPFCFG + SMPLRT_DIV)

Definition at line 407 of file [ICM20948_regs.h](#).

6.40.1.128 GYRO_FCHOICE_DLPF

```
#define GYRO_FCHOICE_DLPF 1
```

Definition at line 408 of file [ICM20948_regs.h](#).

6.40.1.129 GYRO_FS_1000DPS

```
#define GYRO_FS_1000DPS (2u << GYRO_FS_SEL_SHIFT)
```

Definition at line 178 of file [ICM20948_regs.h](#).

6.40.1.130 GYRO_FS_2000DPS

```
#define GYRO_FS_2000DPS (3u << GYRO_FS_SEL_SHIFT)
```

Definition at line 179 of file [ICM20948_regs.h](#).

6.40.1.131 GYRO_FS_250DPS

```
#define GYRO_FS_250DPS (0u << GYRO_FS_SEL_SHIFT)
```

Definition at line 176 of file [ICM20948_regs.h](#).

6.40.1.132 GYRO_FS_500DPS

```
#define GYRO_FS_500DPS (1u << GYRO_FS_SEL_SHIFT)
```

Definition at line 177 of file [ICM20948_regs.h](#).

6.40.1.133 GYRO_FS_SEL_MASK

```
#define GYRO_FS_SEL_MASK (0x3u << GYRO_FS_SEL_SHIFT)
```

Definition at line 175 of file [ICM20948_regs.h](#).

6.40.1.134 GYRO_FS_SEL_SHIFT

```
#define GYRO_FS_SEL_SHIFT 1
```

Definition at line 174 of file [ICM20948_regs.h](#).

6.40.1.135 GYRO_SMPLRT_DIV

```
#define GYRO_SMPLRT_DIV 0x00 /* GYRO_SMPLRT_DIV[7:0] */
```

- Gyro/Accel configuration, offsets, FSYNC, temperature filter

Definition at line 170 of file [ICM20948_regs.h](#).

6.40.1.136 GYRO_SMPLRT_MIN_HZ

```
#define GYRO_SMPLRT_MIN_HZ 4.3f
```

Definition at line 425 of file [ICM20948_regs.h](#).

6.40.1.137 GYRO_XOUT_H

```
#define GYRO_XOUT_H 0x33
```

Definition at line 117 of file [ICM20948_regs.h](#).

6.40.1.138 GYRO_XOUT_L

```
#define GYRO_XOUT_L 0x34
```

Definition at line 118 of file [ICM20948_regs.h](#).

6.40.1.139 GYRO_YOUT_H

```
#define GYRO_YOUT_H 0x35
```

Definition at line 119 of file [ICM20948_regs.h](#).

6.40.1.140 GYRO_YOUT_L

```
#define GYRO_YOUT_L 0x36
```

Definition at line 120 of file [ICM20948_regs.h](#).

6.40.1.141 GYRO_ZOUT_H

```
#define GYRO_ZOUT_H 0x37
```

Definition at line 121 of file [ICM20948_regs.h](#).

6.40.1.142 GYRO_ZOUT_L

```
#define GYRO_ZOUT_L 0x38
```

Definition at line 122 of file [ICM20948_regs.h](#).

6.40.1.143 HighAccuracy

```
#define HighAccuracy 3
```

Definition at line 457 of file [ICM20948_regs.h](#).

6.40.1.144 Hz10

```
#define Hz10 3
```

Definition at line 443 of file [ICM20948_regs.h](#).

6.40.1.145 Hz15

```
#define Hz15 4
```

Definition at line 444 of file [ICM20948_regs.h](#).

6.40.1.146 Hz2

```
#define Hz2 0
```

- Friendly ODR presets you can map to dividers

Definition at line 440 of file [ICM20948_regs.h](#).

6.40.1.147 Hz20

```
#define Hz20 5
```

Definition at line 445 of file [ICM20948_regs.h](#).

6.40.1.148 Hz25

```
#define Hz25 6
```

Definition at line 446 of file [ICM20948_regs.h](#).

6.40.1.149 Hz30

```
#define Hz30 7
```

Definition at line 447 of file [ICM20948_regs.h](#).

6.40.1.150 Hz6

```
#define Hz6 1
```

Definition at line 441 of file [ICM20948_regs.h](#).

6.40.1.151 Hz8

```
#define Hz8 2
```

Definition at line 442 of file [ICM20948_regs.h](#).

6.40.1.152 I2C_MST_CLK_MASK

```
#define I2C_MST_CLK_MASK 0x0F
```

Definition at line 253 of file [ICM20948_regs.h](#).

6.40.1.153 I2C_MST_CTRL

```
#define I2C_MST_CTRL 0x01 /* MULT_MST_EN - I2C_MST_P_NSR I2C_MST_CLK[3:0] */
```

Definition at line 250 of file [ICM20948_regs.h](#).

6.40.1.154 I2C_MST_DELAY_CTRL

```
#define I2C_MST_DELAY_CTRL 0x02 /* DELAY_ES_SHADOW + I2C_SLVx_DELAY_EN bits */
```

Definition at line 255 of file [ICM20948_regs.h](#).

6.40.1.155 I2C_MST_ODR_CONFIG

```
#define I2C_MST_ODR_CONFIG 0x00 /* I2C_MST_ODR_CONFIG[3:0] */
```

- I2C master (aux) interface and slave windows

Definition at line 247 of file [ICM20948_regs.h](#).

6.40.1.156 I2C_MST_P_NSR

```
#define I2C_MST_P_NSR BIT(4)
```

Definition at line 252 of file [ICM20948_regs.h](#).

6.40.1.157 I2C_MST_STATUS

```
#define I2C_MST_STATUS 0x17 /* PASS_THROUGH, *_DONE, *_NACK, LOST_ARB */
```

Definition at line 86 of file [ICM20948_regs.h](#).

6.40.1.158 I2C_SLV0_ADDR

```
#define I2C_SLV0_ADDR 0x03 /* RNW + ID[6:0] */
```

Definition at line 263 of file [ICM20948_regs.h](#).

6.40.1.159 I2C_SLV0_CTRL

```
#define I2C_SLV0_CTRL 0x05 /* EN BYTE_SW REG_DIS GRP LENG[3:0] */
```

Definition at line 266 of file [ICM20948_regs.h](#).

6.40.1.160 I2C_SLV0_DELAY_EN

```
#define I2C_SLV0_DELAY_EN BIT(0)
```

Definition at line 261 of file [ICM20948_regs.h](#).

6.40.1.161 I2C_SLV0_DO

```
#define I2C_SLV0_DO 0x06
```

Definition at line 272 of file [ICM20948_regs.h](#).

6.40.1.162 I2C_SLV0_REG

```
#define I2C_SLV0_REG 0x04
```

Definition at line 265 of file [ICM20948_regs.h](#).

6.40.1.163 I2C_SLV1_ADDR

```
#define I2C_SLV1_ADDR 0x07
```

Definition at line 274 of file [ICM20948_regs.h](#).

6.40.1.164 I2C_SLV1_CTRL

```
#define I2C_SLV1_CTRL 0x09
```

Definition at line 276 of file [ICM20948_regs.h](#).

6.40.1.165 I2C_SLV1_DELAY_EN

```
#define I2C_SLV1_DELAY_EN BIT(1)
```

Definition at line 260 of file [ICM20948_regs.h](#).

6.40.1.166 I2C_SLV1_DO

```
#define I2C_SLV1_DO 0x0A
```

Definition at line 277 of file [ICM20948_regs.h](#).

6.40.1.167 I2C_SLV1_REG

```
#define I2C_SLV1_REG 0x08
```

Definition at line 275 of file [ICM20948_regs.h](#).

6.40.1.168 I2C_SLV2_ADDR

```
#define I2C_SLV2_ADDR 0x0B
```

Definition at line 279 of file [ICM20948_regs.h](#).

6.40.1.169 I2C_SLV2_CTRL

```
#define I2C_SLV2_CTRL 0x0D
```

Definition at line 281 of file [ICM20948_regs.h](#).

6.40.1.170 I2C_SLV2_DELAY_EN

```
#define I2C_SLV2_DELAY_EN BIT(2)
```

Definition at line 259 of file [ICM20948_regs.h](#).

6.40.1.171 I2C_SLV2_DO

```
#define I2C_SLV2_DO 0x0E
```

Definition at line 282 of file [ICM20948_regs.h](#).

6.40.1.172 I2C_SLV2_REG

```
#define I2C_SLV2_REG 0x0C
```

Definition at line 280 of file [ICM20948_regs.h](#).

6.40.1.173 I2C_SLV3_ADDR

```
#define I2C_SLV3_ADDR 0x0F
```

Definition at line 284 of file [ICM20948_regs.h](#).

6.40.1.174 I2C_SLV3_CTRL

```
#define I2C_SLV3_CTRL 0x11
```

Definition at line 286 of file [ICM20948_regs.h](#).

6.40.1.175 I2C_SLV3_DELAY_EN

```
#define I2C_SLV3_DELAY_EN BIT(3)
```

Definition at line 258 of file [ICM20948_regs.h](#).

6.40.1.176 I2C_SLV3_DO

```
#define I2C_SLV3_DO 0x12
```

Definition at line 287 of file [ICM20948_regs.h](#).

6.40.1.177 I2C_SLV3_REG

```
#define I2C_SLV3_REG 0x10
```

Definition at line 285 of file [ICM20948_regs.h](#).

6.40.1.178 I2C_SLV4_ADDR

```
#define I2C_SLV4_ADDR 0x13
```

Definition at line 289 of file [ICM20948_regs.h](#).

6.40.1.179 I2C_SLV4_CTRL

#define I2C_SLV4_CTRL 0x15 /* EN BYTE_SW REG_DIS DLY[4:0] */
Definition at line 291 of file [ICM20948_regs.h](#).

6.40.1.180 I2C_SLV4_DELAY_EN

#define I2C_SLV4_DELAY_EN BIT(4)
Definition at line 257 of file [ICM20948_regs.h](#).

6.40.1.181 I2C_SLV4_DI

#define I2C_SLV4_DI 0x17
Definition at line 294 of file [ICM20948_regs.h](#).

6.40.1.182 I2C_SLV4_DLY_MASK

#define I2C_SLV4_DLY_MASK 0x1F
Definition at line 292 of file [ICM20948_regs.h](#).

6.40.1.183 I2C_SLV4_DO

#define I2C_SLV4_DO 0x16
Definition at line 293 of file [ICM20948_regs.h](#).

6.40.1.184 I2C_SLV4_REG

#define I2C_SLV4_REG 0x14
Definition at line 290 of file [ICM20948_regs.h](#).

6.40.1.185 I2C_SLVx_BYTE_SW

#define I2C_SLVx_BYTE_SW BIT(6)
Definition at line 268 of file [ICM20948_regs.h](#).

6.40.1.186 I2C_SLVx_EN

#define I2C_SLVx_EN BIT(7)
Definition at line 267 of file [ICM20948_regs.h](#).

6.40.1.187 I2C_SLVx_GRP

#define I2C_SLVx_GRP BIT(4)
Definition at line 270 of file [ICM20948_regs.h](#).

6.40.1.188 I2C_SLVx LENG_MASK

#define I2C_SLVx LENG_MASK 0x0F
Definition at line 271 of file [ICM20948_regs.h](#).

6.40.1.189 I2C_SLVx_REG_DIS

#define I2C_SLVx_REG_DIS BIT(5)
Definition at line 269 of file [ICM20948_regs.h](#).

6.40.1.190 I2C_SLVx_RNW

#define I2C_SLVx_RNW BIT(7)
Definition at line 264 of file [ICM20948_regs.h](#).

6.40.1.191 INT1_ACTL

```
#define INT1_ACTL BIT(7) /* active low */
```

Definition at line 65 of file [ICM20948_regs.h](#).

6.40.1.192 INT1_LATCH_INT_EN

```
#define INT1_LATCH_INT_EN BIT(5) /* latch until status read */
```

Definition at line 67 of file [ICM20948_regs.h](#).

6.40.1.193 INT1_OPEN

```
#define INT1_OPEN BIT(6) /* open-drain */
```

Definition at line 66 of file [ICM20948_regs.h](#).

6.40.1.194 INT_ACTL_FSYNC

```
#define INT_ACTL_FSYNC BIT(3)
```

Definition at line 69 of file [ICM20948_regs.h](#).

6.40.1.195 INT_ANYRD_2CLEAR

```
#define INT_ANYRD_2CLEAR BIT(4)
```

Definition at line 68 of file [ICM20948_regs.h](#).

6.40.1.196 INT_DMP_INT1_EN

```
#define INT_DMP_INT1_EN BIT(1)
```

Definition at line 77 of file [ICM20948_regs.h](#).

6.40.1.197 INT_ENABLE

```
#define INT_ENABLE 0x10 /* REG_WOF_EN - WOM_INT_EN PLL_RDY_EN DMP_INT1_EN I2C_MST_INT_EN */
```

Definition at line 73 of file [ICM20948_regs.h](#).

6.40.1.198 INT_ENABLE_1

```
#define INT_ENABLE_1 0x11 /* RAW_DATA_0_RDY_EN */
```

Definition at line 80 of file [ICM20948_regs.h](#).

6.40.1.199 INT_ENABLE_2

```
#define INT_ENABLE_2 0x12 /* FIFO_OVERFLOW_EN[4:0] */
```

Definition at line 83 of file [ICM20948_regs.h](#).

6.40.1.200 INT_ENABLE_3

```
#define INT_ENABLE_3 0x13 /* FIFO_WM_EN[4:0] */
```

Definition at line 84 of file [ICM20948_regs.h](#).

6.40.1.201 INT_I2C_MST_INT_EN

```
#define INT_I2C_MST_INT_EN BIT(0)
```

Definition at line 78 of file [ICM20948_regs.h](#).

6.40.1.202 INT_PIN_CFG

```
#define INT_PIN_CFG 0x0F /* INT1_ACTL INT1_OPEN INT1_LATCH_INT_EN INT_ANYRD_2CLEAR ACTL_FSYNC  
FSYNC_INT_MODE_EN BYPASS_EN - */
```

Definition at line 64 of file [ICM20948_regs.h](#).

6.40.1.203 INT_PLL_RDY_EN

```
#define INT_PLL_RDY_EN BIT(2)
```

Definition at line 76 of file [ICM20948_regs.h](#).

6.40.1.204 INT_RAW_DATA_0_RDY_EN

```
#define INT_RAW_DATA_0_RDY_EN BIT(0)
```

Definition at line 81 of file [ICM20948_regs.h](#).

6.40.1.205 INT_REG_WOF_EN

```
#define INT_REG_WOF_EN BIT(7)
```

Definition at line 74 of file [ICM20948_regs.h](#).

6.40.1.206 INT_STATUS

```
#define INT_STATUS 0x19 /* WOM_INT PLL_RDY_INT DMP_INT1 I2C_MST_INT */
```

Definition at line 96 of file [ICM20948_regs.h](#).

6.40.1.207 INT_STATUS_1

```
#define INT_STATUS_1 0x1A /* RAW_DATA_0_RDY_INT */
```

Definition at line 102 of file [ICM20948_regs.h](#).

6.40.1.208 INT_STATUS_2

```
#define INT_STATUS_2 0x1B /* FIFO_OVERFLOW_INT[4:0] */
```

Definition at line 104 of file [ICM20948_regs.h](#).

6.40.1.209 INT_STATUS_3

```
#define INT_STATUS_3 0x1C /* FIFO_WM_INT[4:0] */
```

Definition at line 105 of file [ICM20948_regs.h](#).

6.40.1.210 INT_WOM_INT_EN

```
#define INT_WOM_INT_EN BIT(3)
```

Definition at line 75 of file [ICM20948_regs.h](#).

6.40.1.211 INTERNAL_20MHZ

```
#define INTERNAL_20MHZ 0x00 /* 0: Internal 20 MHz RC */
```

Definition at line 319 of file [ICM20948_regs.h](#).

6.40.1.212 LowPower

```
#define LowPower 0
```

Generic power or fusion quality modes (BNO-style). Map to sensor-specific sequences inside your driver.

Definition at line 454 of file [ICM20948_regs.h](#).

6.40.1.213 LP_ACCEL_CYCLE

```
#define LP_ACCEL_CYCLE BIT(5)
```

Definition at line 47 of file [ICM20948_regs.h](#).

6.40.1.214 LP_CONFIG

```
#define LP_CONFIG 0x05 /* I2C_MST_CYCLE ACCEL_CYCLE GYRO_CYCLE - */
```

Definition at line 45 of file [ICM20948_regs.h](#).

6.40.1.215 LP_GYRO_CYCLE

```
#define LP_GYRO_CYCLE BIT(4)
```

Definition at line 48 of file [ICM20948_regs.h](#).

6.40.1.216 LP_I2C_MST_CYCLE

```
#define LP_I2C_MST_CYCLE BIT(6)
```

Definition at line 46 of file [ICM20948_regs.h](#).

6.40.1.217 MOD_CTRL_USR

```
#define MOD_CTRL_USR 0x54 /* REG_LP_DMP_EN */
```

Definition at line 237 of file [ICM20948_regs.h](#).

6.40.1.218 MST_LOST_ARB

```
#define MST_LOST_ARB BIT(5)
```

Definition at line 89 of file [ICM20948_regs.h](#).

6.40.1.219 MST_ODR_CFG_MASK

```
#define MST_ODR_CFG_MASK 0x0F
```

Definition at line 248 of file [ICM20948_regs.h](#).

6.40.1.220 MST_PASS_THROUGH

```
#define MST_PASS_THROUGH BIT(7)
```

Definition at line 87 of file [ICM20948_regs.h](#).

6.40.1.221 MST_SLV0_NACK

```
#define MST_SLV0_NACK BIT(0)
```

Definition at line 94 of file [ICM20948_regs.h](#).

6.40.1.222 MST_SLV1_NACK

```
#define MST_SLV1_NACK BIT(1)
```

Definition at line 93 of file [ICM20948_regs.h](#).

6.40.1.223 MST_SLV2_NACK

```
#define MST_SLV2_NACK BIT(2)
```

Definition at line 92 of file [ICM20948_regs.h](#).

6.40.1.224 MST_SLV3_NACK

```
#define MST_SLV3_NACK BIT(3)
```

Definition at line 91 of file [ICM20948_regs.h](#).

6.40.1.225 MST_SLV4_DONE

```
#define MST_SLV4_DONE BIT(6)
```

Definition at line 88 of file [ICM20948_regs.h](#).

6.40.1.226 MST_SLV4_NACK

```
#define MST_SLV4_NACK BIT(4)
```

Definition at line 90 of file [ICM20948_regs.h](#).

6.40.1.227 MULT_MST_EN

```
#define MULT_MST_EN BIT(7)
```

Definition at line 251 of file [ICM20948_regs.h](#).

6.40.1.228 ODR_ALIGN_EN

```
#define ODR_ALIGN_EN 0x09 /* ODR_ALIGN_EN */
```

Definition at line 197 of file [ICM20948_regs.h](#).

6.40.1.229 ODR_ALIGN_EN_BIT

```
#define ODR_ALIGN_EN_BIT BIT(0)
```

Definition at line 198 of file [ICM20948_regs.h](#).

6.40.1.230 PWR_CLKSEL_AUTO

```
#define PWR_CLKSEL_AUTO 0x01 /* typical: auto selects best source */
```

Definition at line 57 of file [ICM20948_regs.h](#).

6.40.1.231 PWR_CLKSEL_INT_20MHZ

```
#define PWR_CLKSEL_INT_20MHZ 0x01
```

Definition at line 56 of file [ICM20948_regs.h](#).

6.40.1.232 PWR_CLKSEL_MASK

```
#define PWR_CLKSEL_MASK 0x07
```

Definition at line 55 of file [ICM20948_regs.h](#).

6.40.1.233 PWR_DEVICE_RESET

```
#define PWR_DEVICE_RESET BIT(7)
```

Definition at line 51 of file [ICM20948_regs.h](#).

6.40.1.234 PWR_DISABLE_ACCEL

```
#define PWR_DISABLE_ACCEL BIT(3)
```

Definition at line 60 of file [ICM20948_regs.h](#).

6.40.1.235 PWR_DISABLE_GYRO

```
#define PWR_DISABLE_GYRO BIT(0)
```

Definition at line 61 of file [ICM20948_regs.h](#).

6.40.1.236 PWR_LP_EN

```
#define PWR_LP_EN BIT(5)
```

Definition at line 53 of file [ICM20948_regs.h](#).

6.40.1.237 PWR_MGMT_1

```
#define PWR_MGMT_1 0x06 /* DEVICE_RESET SLEEP LP_EN - TEMP_DIS CLKSEL[2:0] */
```

Definition at line 50 of file [ICM20948_regs.h](#).

6.40.1.238 PWR_MGMT_2

```
#define PWR_MGMT_2 0x07 /* - DISABLE_ACCEL DISABLE_GYRO */
```

Definition at line 59 of file [ICM20948_regs.h](#).

6.40.1.239 PWR_SLEEP

```
#define PWR_SLEEP BIT(6)
```

Definition at line 52 of file [ICM20948_regs.h](#).

6.40.1.240 PWR_TEMP_DIS

```
#define PWR_TEMP_DIS BIT(3)
```

Definition at line 54 of file [ICM20948_regs.h](#).

6.40.1.241 REG_BANK_SEL

```
#define REG_BANK_SEL 0x7F /* USER_BANK[1:0] */
```

Definition at line 140 of file [ICM20948_regs.h](#).

6.40.1.242 REG_BANK_SEL_USER_BANK_MASK

```
#define REG_BANK_SEL_USER_BANK_MASK (0x3u << REG_BANK_SEL_USER_BANK_SHIFT)
```

Definition at line 303 of file [ICM20948_regs.h](#).

6.40.1.243 REG_BANK_SEL_USER_BANK_SHIFT

```
#define REG_BANK_SEL_USER_BANK_SHIFT 4
```

Definition at line 302 of file [ICM20948_regs.h](#).

6.40.1.244 REG_LP_DMP_EN

```
#define REG_LP_DMP_EN BIT(7)
```

Definition at line 238 of file [ICM20948_regs.h](#).

6.40.1.245 Regular

```
#define Regular 1
```

Definition at line 455 of file [ICM20948_regs.h](#).

6.40.1.246 SELF_TEST_X_ACCEL

```
#define SELF_TEST_X_ACCEL 0x0E /* XA_ST_DATA[7:0] */
```

Definition at line 150 of file [ICM20948_regs.h](#).

6.40.1.247 SELF_TEST_X_GYRO

```
#define SELF_TEST_X_GYRO 0x02 /* XG_ST_DATA[7:0] */
```

- Self-test / accel offsets / timebase

Definition at line 147 of file [ICM20948_regs.h](#).

6.40.1.248 SELF_TEST_Y_ACCEL

```
#define SELF_TEST_Y_ACCEL 0x0F /* YA_ST_DATA[7:0] */
```

Definition at line 151 of file [ICM20948_regs.h](#).

6.40.1.249 SELF_TEST_Y_GYRO

```
#define SELF_TEST_Y_GYRO 0x03 /* YG_ST_DATA[7:0] */
```

Definition at line 148 of file [ICM20948_regs.h](#).

6.40.1.250 SELF_TEST_Z_ACCEL

```
#define SELF_TEST_Z_ACCEL 0x10 /* ZA_ST_DATA[7:0] */
```

Definition at line 152 of file [ICM20948_regs.h](#).

6.40.1.251 SELF_TEST_Z_GYRO

```
#define SELF_TEST_Z_GYRO 0x04 /* ZG_ST_DATA[7:0] */
```

Definition at line 149 of file [ICM20948_regs.h](#).

6.40.1.252 STS_DMP_INT1

```
#define STS_DMP_INT1 BIT(1)
```

Definition at line 99 of file [ICM20948_regs.h](#).

6.40.1.253 STS_I2C_MST_INT

```
#define STS_I2C_MST_INT BIT(0)
```

Definition at line 100 of file [ICM20948_regs.h](#).

6.40.1.254 STS_PLL_RDY_INT

```
#define STS_PLL_RDY_INT BIT(2)
```

Definition at line 98 of file [ICM20948_regs.h](#).

6.40.1.255 STS_RAW_DATA_0_RDY_INT

```
#define STS_RAW_DATA_0_RDY_INT BIT(0)
```

Definition at line 103 of file [ICM20948_regs.h](#).

6.40.1.256 STS_WOM_INT

```
#define STS_WOM_INT BIT(3)
```

Definition at line 97 of file [ICM20948_regs.h](#).

6.40.1.257 TEMP_CONFIG

```
#define TEMP_CONFIG 0x53 /* TEMP_DLPFCFG[2:0] */
```

Definition at line 233 of file [ICM20948_regs.h](#).

6.40.1.258 TEMP_DLPFCFG_MASK

```
#define TEMP_DLPFCFG_MASK (0x7u << TEMP_DLPFCFG_SHIFT)
```

Definition at line 235 of file [ICM20948_regs.h](#).

6.40.1.259 TEMP_DLPFCFG_SHIFT

```
#define TEMP_DLPFCFG_SHIFT 5
```

Definition at line 234 of file [ICM20948_regs.h](#).

6.40.1.260 TEMP_OUT_H

```
#define TEMP_OUT_H 0x39
```

Definition at line 123 of file [ICM20948_regs.h](#).

6.40.1.261 TEMP_OUT_L

```
#define TEMP_OUT_L 0x3A
```

Definition at line 124 of file [ICM20948_regs.h](#).

6.40.1.262 TIMEBASE_CORRECTION_PLL

```
#define TIMEBASE_CORRECTION_PLL 0x28 /* TBC_PLL[7:0] */
```

Definition at line 161 of file [ICM20948_regs.h](#).

6.40.1.263 USER_BANK_0

```
#define USER_BANK_0 BANK0
```

Definition at line 304 of file [ICM20948_regs.h](#).

6.40.1.264 USER_BANK_1

```
#define USER_BANK_1 BANK1
```

Definition at line 305 of file [ICM20948_regs.h](#).

6.40.1.265 USER_BANK_2

```
#define USER_BANK_2 BANK2
```

Definition at line 306 of file [ICM20948_regs.h](#).

6.40.1.266 USER_BANK_3

```
#define USER_BANK_3 BANK3
```

Definition at line 307 of file [ICM20948_regs.h](#).

6.40.1.267 USER_CTRL

```
#define USER_CTRL 0x03 /* DMP_EN FIFO_EN I2C_MST_EN I2C_IF_DIS DMP_RST SRAM_RST I2C_MST_RST -  
*/
```

Definition at line 35 of file [ICM20948_regs.h](#).

6.40.1.268 USER_CTRL_DMP_EN

```
#define USER_CTRL_DMP_EN BIT(7)
```

Definition at line 37 of file [ICM20948_regs.h](#).

6.40.1.269 USER_CTRL_DMP_RST

```
#define USER_CTRL_DMP_RST BIT(3)
```

Definition at line 41 of file [ICM20948_regs.h](#).

6.40.1.270 USER_CTRL_FIFO_EN

```
#define USER_CTRL_FIFO_EN BIT(6)
```

Definition at line 38 of file [ICM20948_regs.h](#).

6.40.1.271 USER_CTRL_I2C_IF_DIS

```
#define USER_CTRL_I2C_IF_DIS BIT(4)
```

Definition at line 40 of file [ICM20948_regs.h](#).

6.40.1.272 USER_CTRL_I2C_MST_EN

```
#define USER_CTRL_I2C_MST_EN BIT(5)
```

Definition at line 39 of file [ICM20948_regs.h](#).

6.40.1.273 USER_CTRL_I2C_MST_RST

```
#define USER_CTRL_I2C_MST_RST BIT(1)
```

Definition at line 43 of file [ICM20948_regs.h](#).

6.40.1.274 USER_CTRL_SRAM_RST

```
#define USER_CTRL_SRAM_RST BIT(2)
```

Definition at line 42 of file [ICM20948_regs.h](#).

6.40.1.275 WHO_AM_I

```
#define WHO_AM_I 0x00 /* WHO_AM_I[7:0] */  
7Semi comment style
```

- Full ICM-20948 register map (Banks 0–3) + key bit fields
- Simple, portable #defines for firmware use
- Datasheet naming kept where practical; fields grouped by bank

Notes

- WHO_AM_I expected value: 0xEA
- Select register bank via REG_BANK_SEL in any bank
- Use BANK(n) helper or write raw values 0x00/0x10/0x20/0x30
- Identity / power / interrupts / sensor data

Definition at line 32 of file [ICM20948_regs.h](#).

6.40.1.276 WHO_AM_I_VAL

```
#define WHO_AM_I_VAL 0xEA
```

Definition at line 33 of file [ICM20948_regs.h](#).

6.40.1.277 WOF_DEGLITCH_EN

```
#define WOF_DEGLITCH_EN BIT(6)
```

Definition at line 229 of file [ICM20948_regs.h](#).

6.40.1.278 WOF_EDGE_INT

```
#define WOF_EDGE_INT BIT(5)
```

Definition at line 230 of file [ICM20948_regs.h](#).

6.40.1.279 XA_OFFS_H

```
#define XA_OFFS_H 0x14 /* XA_OFFS[14:7] */
```

Definition at line 154 of file [ICM20948_regs.h](#).

6.40.1.280 XA_OFFS_L

```
#define XA_OFFS_L 0x15 /* XA_OFFS[6:0] */
```

Definition at line 155 of file [ICM20948_regs.h](#).

6.40.1.281 XG_OFFS_USRH

```
#define XG_OFFS_USRH 0x03
```

Definition at line 190 of file [ICM20948_regs.h](#).

6.40.1.282 XG_OFFS_USRL

```
#define XG_OFFS_USRL 0x04
```

Definition at line 191 of file [ICM20948_regs.h](#).

6.40.1.283 XGYRO_CTEN

```
#define XGYRO_CTEN BIT(5)
```

Definition at line 184 of file [ICM20948_regs.h](#).

6.40.1.284 YA_OFFS_H

```
#define YA_OFFS_H 0x17 /* YA_OFFS[14:7] */
Definition at line 156 of file ICM20948_regs.h.
```

6.40.1.285 YA_OFFS_L

```
#define YA_OFFS_L 0x18 /* YA_OFFS[6:0] */
Definition at line 157 of file ICM20948_regs.h.
```

6.40.1.286 YG_OFFS_USRH

```
#define YG_OFFS_USRH 0x05
Definition at line 192 of file ICM20948_regs.h.
```

6.40.1.287 YG_OFFS_USRL

```
#define YG_OFFS_USRL 0x06
Definition at line 193 of file ICM20948_regs.h.
```

6.40.1.288 YGYRO_CTEN

```
#define YGYRO_CTEN BIT(4)
Definition at line 185 of file ICM20948_regs.h.
```

6.40.1.289 ZA_OFFS_H

```
#define ZA_OFFS_H 0x1A /* ZA_OFFS[14:7] */
Definition at line 158 of file ICM20948_regs.h.
```

6.40.1.290 ZA_OFFS_L

```
#define ZA_OFFS_L 0x1B /* ZA_OFFS[6:0] */
Definition at line 159 of file ICM20948_regs.h.
```

6.40.1.291 ZG_OFFS_USRH

```
#define ZG_OFFS_USRH 0x07
Definition at line 194 of file ICM20948_regs.h.
```

6.40.1.292 ZG_OFFS_USRL

```
#define ZG_OFFS_USRL 0x08
Definition at line 195 of file ICM20948_regs.h.
```

6.40.1.293 ZGYRO_CTEN

```
#define ZGYRO_CTEN BIT(3)
Definition at line 186 of file ICM20948_regs.h.
```

6.41 ICM20948_regs.h

[Go to the documentation of this file.](#)

```
00001 #ifndef ICM20948_REGS_H
00002 #define ICM20948_REGS_H
00003
00004 /**
00005  * 7Semi comment style
00006  * - Full ICM-20948 register map (Banks 0-3) + key bit fields
00007  * - Simple, portable #defines for firmware use
00008  * - Datasheet naming kept where practical; fields grouped by bank
00009  *
00010  * Notes
```

```

00011 * - WHO_AM_I expected value: 0xEA
00012 * - Select register bank via REG_BANK_SEL in any bank
00013 * - Use BANK(n) helper or write raw values 0x00/0x10/0x20/0x30
00014 */
00015
00016 /* ----- Common helpers ----- */
00017 #ifndef BIT
00018 #define BIT(n) (1u << (n))
00019 #endif
00020
00021 // #define BANK(n) ((uint8_t)((n) << 4))
00022 // #define BANK0 BANK(0)
00023 // #define BANK1 BANK(1)
00024 // #define BANK2 BANK(2)
00025 // #define BANK3 BANK(3)
00026
00027 /* ===== */
00028 /*                                     BANK 0                                     */
00029 /* ===== */
00030
00031 /** - Identity / power / interrupts / sensor data */
00032 #define WHO_AM_I 0x00 /* WHO_AM_I[7:0] */
00033 #define WHO_AM_I_VAL 0xEA
00034
00035 #define USER_CTRL 0x03 /* DMP_EN FIFO_EN I2C_MST_EN I2C_IF_DIS DMP_RST SRAM_RST I2C_MST_RST - */
00036 /* bits */
00037 #define USER_CTRL_DMP_EN BIT(7)
00038 #define USER_CTRL_FIFO_EN BIT(6)
00039 #define USER_CTRL_I2C_MST_EN BIT(5)
00040 #define USER_CTRL_I2C_IF_DIS BIT(4)
00041 #define USER_CTRL_DMP_RST BIT(3)
00042 #define USER_CTRL_SRAM_RST BIT(2)
00043 #define USER_CTRL_I2C_MST_RST BIT(1)
00044
00045 #define LP_CONFIG 0x05 /* I2C_MST_CYCLE ACCEL_CYCLE GYRO_CYCLE - */
00046 #define LP_I2C_MST_CYCLE BIT(6)
00047 #define LP_ACCEL_CYCLE BIT(5)
00048 #define LP_GYRO_CYCLE BIT(4)
00049
00050 #define PWR_MGMT_1 0x06 /* DEVICE_RESET SLEEP LP_EN - TEMP_DIS CLKSEL[2:0] */
00051 #define PWR_DEVICE_RESET BIT(7)
00052 #define PWR_SLEEP BIT(6)
00053 #define PWR_LP_EN BIT(5)
00054 #define PWR_TEMP_DIS BIT(3)
00055 #define PWR_CLKSEL_MASK 0x07
00056 #define PWR_CLKSEL_INT_20MHZ 0x01
00057 #define PWR_CLKSEL_AUTO 0x01 /* typical: auto selects best source */
00058
00059 #define PWR_MGMT_2 0x07 /* - DISABLE_ACCEL DISABLE_GYRO */
00060 #define PWR_DISABLE_ACCEL BIT(3)
00061 #define PWR_DISABLE_GYRO BIT(0)
00062
00063 //Interrupt Configs register
00064 #define INT_PIN_CFG 0x0F /* INT1_ACTL INT1_OPEN INT1_LATCH_INT_EN INT_ANYRD_2CLEAR ACTL_FSYNC
FSYNC_INT_MODE_EN BYPASS_EN - */
00065 #define INT1_ACTL BIT(7) /* active low */
00066 #define INT1_OPEN BIT(6) /* open-drain */
00067 #define INT1_LATCH_INT_EN BIT(5) /* latch until status read */
00068 #define INT_ANYRD_2CLEAR BIT(4)
00069 #define INT_ACTL_FSYNC BIT(3)
00070 #define FSYNC_INT_MODE_EN BIT(2)
00071 #define BYPASS_EN BIT(1) /* I2C bypass to aux devices */
00072
00073 #define INT_ENABLE 0x10 /* REG_WOF_EN - WOM_INT_EN PLL_RDY_EN DMP_INT1_EN I2C_MST_INT_EN */
00074 #define INT_REG_WOF_EN BIT(7)
00075 #define INT_WOM_INT_EN BIT(3)
00076 #define INT_PLL_RDY_EN BIT(2)
00077 #define INT_DMP_INT1_EN BIT(1)
00078 #define INT_I2C_MST_INT_EN BIT(0)
00079
00080 #define INT_ENABLE_1 0x11 /* RAW_DATA_0_RDY_EN */
00081 #define INT_RAW_DATA_0_RDY_EN BIT(0)
00082
00083 #define INT_ENABLE_2 0x12 /* FIFO_OVERFLOW_EN[4:0] */
00084 #define INT_ENABLE_3 0x13 /* FIFO_WM_EN[4:0] */
00085
00086 #define I2C_MST_STATUS 0x17 /* PASS_THROUGH, *_DONE, *_NACK, LOST_ARB */
00087 #define MST_PASS_THROUGH BIT(7)
00088 #define MST_SLV4_DONE BIT(6)
00089 #define MST_LOST_ARB BIT(5)
00090 #define MST_SLV4_NACK BIT(4)
00091 #define MST_SLV3_NACK BIT(3)
00092 #define MST_SLV2_NACK BIT(2)
00093 #define MST_SLV1_NACK BIT(1)
00094 #define MST_SLV0_NACK BIT(0)
00095
00096 #define INT_STATUS 0x19 /* WOM_INT PLL_RDY_INT DMP_INT1 I2C_MST_INT */

```

```

00097 #define STS_WOM_INT BIT(3)
00098 #define STS_PLL_RDY_INT BIT(2)
00099 #define STS_DMP_INT1 BIT(1)
00100 #define STS_I2C_MST_INT BIT(0)
00101
00102 #define INT_STATUS_1 0x1A /* RAW_DATA_0_RDY_INT */
00103 #define STS_RAW_DATA_0_RDY_INT BIT(0)
00104 #define INT_STATUS_2 0x1B /* FIFO_OVERFLOW_INT[4:0] */
00105 #define INT_STATUS_3 0x1C /* FIFO_WM_INT[4:0] */
00106
00107 #define DELAY_TIMEH 0x28
00108 #define DELAY_TIMEL 0x29
00109
00110 /* - Sensor outputs */
00111 #define ACCEL_XOUT_H 0x2D
00112 #define ACCEL_XOUT_L 0x2E
00113 #define ACCEL_YOUT_H 0x2F
00114 #define ACCEL_YOUT_L 0x30
00115 #define ACCEL_ZOUT_H 0x31
00116 #define ACCEL_ZOUT_L 0x32
00117 #define GYRO_XOUT_H 0x33
00118 #define GYRO_XOUT_L 0x34
00119 #define GYRO_YOUT_H 0x35
00120 #define GYRO_YOUT_L 0x36
00121 #define GYRO_ZOUT_H 0x37
00122 #define GYRO_ZOUT_L 0x38
00123 #define TEMP_OUT_H 0x39
00124 #define TEMP_OUT_L 0x3A
00125
00126 /* - External sensor shadow data (aux I2C) */
00127 #define EXT_SLV_SENS_DATA_00 0x3B
00128 #define EXT_SLV_SENS_DATA_23 0x52 /* contiguous range 0x3B..0x52 */
00129
00130 #define FIFO_EN_1 0x66 /* SLV3..SLV0 FIFO_EN */
00131 #define FIFO_EN_2 0x67 /* ACCEL GYRO_Z/Y/X TEMP FIFO_EN */
00132 #define FIFO_RST 0x68 /* FIFO_RESET[4:0] */
00133 #define FIFO_MODE 0x69 /* FIFO_MODE[4:0] */
00134 #define FIFO_COUNTH 0x70
00135 #define FIFO_COUNTL 0x71
00136 #define FIFO_R_W 0x72
00137 // #define DATA_RDY_STATUS 0x74 /* WOF_STATUS RAW_DATA_RDY[3:0] */
00138 #define FIFO_CFG 0x76
00139
00140 #define REG_BANK_SEL 0x7F /* USER_BANK[1:0] */
00141
00142 /* ===== */
00143 /* BANK 1 */
00144 /* ===== */
00145
00146 /** - Self-test / accel offsets / timebase */
00147 #define SELF_TEST_X_GYRO 0x02 /* XG_ST_DATA[7:0] */
00148 #define SELF_TEST_Y_GYRO 0x03 /* YG_ST_DATA[7:0] */
00149 #define SELF_TEST_Z_GYRO 0x04 /* ZG_ST_DATA[7:0] */
00150 #define SELF_TEST_X_ACCEL 0x0E /* XA_ST_DATA[7:0] */
00151 #define SELF_TEST_Y_ACCEL 0x0F /* YA_ST_DATA[7:0] */
00152 #define SELF_TEST_Z_ACCEL 0x10 /* ZA_ST_DATA[7:0] */
00153
00154 #define XA_OFFS_H 0x14 /* XA_OFFS[14:7] */
00155 #define XA_OFFS_L 0x15 /* XA_OFFS[6:0] */
00156 #define YA_OFFS_H 0x17 /* YA_OFFS[14:7] */
00157 #define YA_OFFS_L 0x18 /* YA_OFFS[6:0] */
00158 #define ZA_OFFS_H 0x1A /* ZA_OFFS[14:7] */
00159 #define ZA_OFFS_L 0x1B /* ZA_OFFS[6:0] */
00160
00161 #define TIMEBASE_CORRECTION_PLL 0x28 /* TBC_PLL[7:0] */
00162
00163 #define BANK1_REG_BANK_SEL 0x7F /* mirror of REG_BANK_SEL */
00164
00165 /* ===== */
00166 /* BANK 2 */
00167 /* ===== */
00168
00169 /** - Gyro/Accel configuration, offsets, FSYNC, temperature filter */
00170 #define GYRO_SMPLRT_DIV 0x00 /* GYRO_SMPLRT_DIV[7:0] */
00171
00172 #define GYRO_CONFIG_1 0x01 /* GYRO_DLPFCFG[2:0] GYRO_FS_SEL[1:0] GYRO_FCHOICE */
00173 #define GYRO_FCHOICE BIT(0) /* when 0: DLPF on, when 1: off (per datasheet) */
00174 #define GYRO_FS_SEL_SHIFT 1
00175 #define GYRO_FS_SEL_MASK (0x3u << GYRO_FS_SEL_SHIFT)
00176 #define GYRO_FS_250DPS (0u << GYRO_FS_SEL_SHIFT)
00177 #define GYRO_FS_500DPS (1u << GYRO_FS_SEL_SHIFT)
00178 #define GYRO_FS_1000DPS (2u << GYRO_FS_SEL_SHIFT)
00179 #define GYRO_FS_2000DPS (3u << GYRO_FS_SEL_SHIFT)
00180 #define GYRO_DLPFCFG_SHIFT 5
00181 #define GYRO_DLPFCFG_MASK (0x7u << GYRO_DLPFCFG_SHIFT)
00182
00183 #define GYRO_CONFIG_2 0x02 /* XGYRO_CTEN YGYRO_CTEN ZGYRO_CTEN GYRO_AVGCFG[2:0] */

```

```

00184 #define XGYRO_CTEN BIT(5)
00185 #define YGYRO_CTEN BIT(4)
00186 #define ZGYRO_CTEN BIT(3)
00187 #define GYRO_AVGCFG_SHIFT 0
00188 #define GYRO_AVGCFG_MASK (0x7u << GYRO_AVGCFG_SHIFT)
00189
00190 #define XG_OFFS_USRH 0x03
00191 #define XG_OFFS_USRL 0x04
00192 #define YG_OFFS_USRH 0x05
00193 #define YG_OFFS_USRL 0x06
00194 #define ZG_OFFS_USRH 0x07
00195 #define ZG_OFFS_USRL 0x08
00196
00197 #define ODR_ALIGN_EN 0x09 /* ODR_ALIGN_EN */
00198 #define ODR_ALIGN_EN_BIT BIT(0)
00199
00200 #define ACCEL_SMPLRT_DIV_1 0x10 /* ACCEL_SMPLRT_DIV[11:8] */
00201 #define ACCEL_SMPLRT_DIV_2 0x11 /* ACCEL_SMPLRT_DIV[7:0] */
00202
00203 #define ACCEL_INTEL_CTRL 0x12 /* ACCEL_INTEL_EN ACCEL_INTEL_MODE_INT */
00204 #define ACCEL_INTEL_EN BIT(7)
00205 #define ACCEL_INTEL_MODE_INT BIT(6)
00206
00207 #define ACCEL_WOM_THR 0x13 /* WOM_THRESHOLD[7:0] */
00208
00209 #define ACCEL_CONFIG 0x14 /* ACCEL_DLPFCFG[2:0] ACCEL_FS_SEL[1:0] ACCEL_FCHOICE */
00210 #define ACCEL_FCHOICE BIT(0)
00211 #define ACCEL_FS_SEL_SHIFT 1
00212 #define ACCEL_FS_SEL_MASK (0x3u << ACCEL_FS_SEL_SHIFT)
00213 #define ACCEL_FS_2G (0u << ACCEL_FS_SEL_SHIFT)
00214 #define ACCEL_FS_4G (1u << ACCEL_FS_SEL_SHIFT)
00215 #define ACCEL_FS_8G (2u << ACCEL_FS_SEL_SHIFT)
00216 #define ACCEL_FS_16G (3u << ACCEL_FS_SEL_SHIFT)
00217 #define ACCEL_DLPFCFG_SHIFT 5
00218 #define ACCEL_DLPFCFG_MASK (0x7u << ACCEL_DLPFCFG_SHIFT)
00219
00220 #define ACCEL_CONFIG_2 0x15 /* AX_ST_EN_REG AY_ST_EN_REG AZ_ST_EN_REG DEC3_CFG[1:0] */
00221 #define AX_ST_EN_REG BIT(7)
00222 #define AY_ST_EN_REG BIT(6)
00223 #define AZ_ST_EN_REG BIT(5)
00224 #define DEC3_CFG_SHIFT 0
00225 #define DEC3_CFG_MASK (0x3u << DEC3_CFG_SHIFT)
00226
00227 #define FSYNC_CONFIG 0x52 /* DELAY_TIME_EN WOF_DEGLITCH_EN WOF_EDGE_INT EXT_SYNC_SET[3:0] */
00228 #define DELAY_TIME_EN BIT(7)
00229 #define WOF_DEGLITCH_EN BIT(6)
00230 #define WOF_EDGE_INT BIT(5)
00231 #define EXT_SYNC_SET_MASK 0x0F
00232
00233 #define TEMP_CONFIG 0x53 /* TEMP_DLPFCFG[2:0] */
00234 #define TEMP_DLPFCFG_SHIFT 5
00235 #define TEMP_DLPFCFG_MASK (0x7u << TEMP_DLPFCFG_SHIFT)
00236
00237 #define MOD_CTRL_USR 0x54 /* REG_LP_DMP_EN */
00238 #define REG_LP_DMP_EN BIT(7)
00239
00240 #define BANK2_REG_BANK_SEL 0x7F /* mirror of REG_BANK_SEL */
00241
00242 /* ===== */
00243 /* BANK 3 */
00244 /* ===== */
00245
00246 /** - I2C master (aux) interface and slave windows */
00247 #define I2C_MST_ODR_CONFIG 0x00 /* I2C_MST_ODR_CONFIG[3:0] */
00248 #define MST_ODR_CFG_MASK 0x0F
00249
00250 #define I2C_MST_CTRL 0x01 /* MULT_MST_EN - I2C_MST_P_NSR I2C_MST_CLK[3:0] */
00251 #define MULT_MST_EN BIT(7)
00252 #define I2C_MST_P_NSR BIT(4)
00253 #define I2C_MST_CLK_MASK 0x0F
00254
00255 #define I2C_MST_DELAY_CTRL 0x02 /* DELAY_ES_SHADOW + I2C_SLVx_DELAY_EN bits */
00256 #define DELAY_ES_SHADOW BIT(7)
00257 #define I2C_SLV4_DELAY_EN BIT(4)
00258 #define I2C_SLV3_DELAY_EN BIT(3)
00259 #define I2C_SLV2_DELAY_EN BIT(2)
00260 #define I2C_SLV1_DELAY_EN BIT(1)
00261 #define I2C_SLV0_DELAY_EN BIT(0)
00262
00263 #define I2C_SLV0_ADDR 0x03 /* RNW + ID[6:0] */
00264 #define I2C_SLVx_RNW BIT(7)
00265 #define I2C_SLV0_REG 0x04
00266 #define I2C_SLV0_CTRL 0x05 /* EN BYTE_SW REG_DIS GRP LENG[3:0] */
00267 #define I2C_SLVx_EN BIT(7)
00268 #define I2C_SLVx_BYTE_SW BIT(6)
00269 #define I2C_SLVx_REG_DIS BIT(5)
00270 #define I2C_SLVx_GRP BIT(4)

```

```

00271 #define I2C_SLVx LENG_MASK 0x0F
00272 #define I2C_SLV0_DO 0x06
00273
00274 #define I2C_SLV1_ADDR 0x07
00275 #define I2C_SLV1_REG 0x08
00276 #define I2C_SLV1_CTRL 0x09
00277 #define I2C_SLV1_DO 0x0A
00278
00279 #define I2C_SLV2_ADDR 0x0B
00280 #define I2C_SLV2_REG 0x0C
00281 #define I2C_SLV2_CTRL 0x0D
00282 #define I2C_SLV2_DO 0x0E
00283
00284 #define I2C_SLV3_ADDR 0x0F
00285 #define I2C_SLV3_REG 0x10
00286 #define I2C_SLV3_CTRL 0x11
00287 #define I2C_SLV3_DO 0x12
00288
00289 #define I2C_SLV4_ADDR 0x13
00290 #define I2C_SLV4_REG 0x14
00291 #define I2C_SLV4_CTRL 0x15 /* EN BYTE_SW REG_DIS DLY[4:0] */
00292 #define I2C_SLV4_DLY_MASK 0x1F
00293 #define I2C_SLV4_DO 0x16
00294 #define I2C_SLV4_DI 0x17
00295
00296 #define BANK3_REG_BANK_SEL 0x7F /* mirror of REG_BANK_SEL */
00297
00298 /* ===== */
00299 /* REG_BANK_SEL (all banks) */
00300 /* ===== */
00301
00302 #define REG_BANK_SEL_USER_BANK_SHIFT 4
00303 #define REG_BANK_SEL_USER_BANK_MASK (0x3u << REG_BANK_SEL_USER_BANK_SHIFT)
00304 #define USER_BANK_0 BANK0
00305 #define USER_BANK_1 BANK1
00306 #define USER_BANK_2 BANK2
00307 #define USER_BANK_3 BANK3
00308
00309 /* AK09916 magnetometer registers (connected internally) */
00310 #define AK09916_I2C_ADDR 0x0C
00311 #define AK_WIA2 0x01
00312 #define AK_ST1 0x10
00313 #define AK_HXL 0x11
00314 #define AK_ST2 0x18
00315 #define AK_CNTL2 0x31
00316 #define AK_CNTL3 0x32
00317 #define AK_WIA2_VAL 0x09 /* AK09916C WIA2 expected */
00318
00319 #define INTERNAL_20MHZ 0x00 /* 0: Internal 20 MHz RC */
00320 #define AUTO_SEL 0x01 /* 1-5: Auto/PLL preferred (use 1 as default) */
00321 #define CLK_STOP 0x07 /* 7: Stop clock / timing gen reset */
00322
00323
00324 /* ----- Accelerometer Range ----- */
00325 /** -  $\pm 2g$  /  $\pm 4g$  /  $\pm 8g$  /  $\pm 16g$  (maps to FS_SEL 0..3) */
00326 #define g2 0
00327 #define g4 1
00328 #define g8 2
00329 #define g16 3
00330
00331 /* ----- Accelerometer Filter Path (FCHOICE) ----- */
00332 /**
00333  * Path select for accel:
00334  * - ACCEL_FCHOICE_BYPASS : DLPF bypassed (very wide BW, fastest)
00335  * - ACCEL_FCHOICE_DLPF : DLPF enabled (use ACCEL_DLPFCFG + SEMPLRT_DIV)
00336  */
00337 #define ACCEL_FCHOICE_BYPASS 0
00338 #define ACCEL_FCHOICE_DLPF 1
00339
00340 /* ----- Accelerometer DLPF Config (CFG) ----- */
00341 /**
00342  * ACCEL_DLPFCFG (0..7) – choose cutoff/noise BW set (when DLPF is ON).
00343  * Tip: start with ACCEL_DLPFCFG_3 for a good noise/latency tradeoff.
00344  */
00345 #define ACCEL_DLPFCFG_0 0
00346 #define ACCEL_DLPFCFG_1 1
00347 #define ACCEL_DLPFCFG_2 2
00348 #define ACCEL_DLPFCFG_3 3
00349 #define ACCEL_DLPFCFG_4 4
00350 #define ACCEL_DLPFCFG_5 5
00351 #define ACCEL_DLPFCFG_6 6
00352 #define ACCEL_DLPFCFG_7 7
00353
00354 #define ACCEL_DEC3_AVG_4 0u /* averages 1 or 4 (depends on FCHOICE) */
00355 #define ACCEL_DEC3_AVG_8 1u
00356 #define ACCEL_DEC3_AVG_16 2u
00357 #define ACCEL_DEC3_AVG_32 3u

```

```

00358
00359 /* ---- Accel DLPF ODR helper (Hz) : base 1125 Hz, DIV 0..4095 ---- */
00360 #define ACCEL_SMPLRT_DIV_MIN      0u
00361 #define ACCEL_SMPLRT_DIV_MAX      4095u
00362 #define ACCEL_DLPF_BASE_HZ        1125.0f
00363 #define ACCEL_DLPF_ODR_HZ(div)    (ACCEL_DLPF_BASE_HZ / (1.0f + (float)(div)))
00364
00365 /* ---- Accel BYPASS quick reference (FCHOICE=0) ----
00366 * 3dB ≈ 1209 Hz, NBW ≈ 1248 Hz, output rate ≈ 4500 Hz
00367 * (No divider/ODR control in bypass path.)
00368 */
00369 #define ACCEL_BYPASS_3DB_BW_HZ     1209.0f
00370 #define ACCEL_BYPASS_NBW_HZ        1248.0f
00371 #define ACCEL_BYPASS_RATE_HZ       4500.0f
00372
00373 /* (Optional) Accel DLPF nominal bandwidth references (FCHOICE=1)
00374 * Keep here for quick docs; exact values depend on datasheet rev.
00375 */
00376 #define ACCEL_DLPF0_3DB_BW_HZ      246.0f
00377 #define ACCEL_DLPF0_NBW_HZ         265.0f
00378 #define ACCEL_DLPF1_3DB_BW_HZ      246.0f
00379 #define ACCEL_DLPF1_NBW_HZ         265.0f
00380 #define ACCEL_DLPF2_3DB_BW_HZ      111.4f
00381 #define ACCEL_DLPF2_NBW_HZ         136.0f
00382 #define ACCEL_DLPF3_3DB_BW_HZ      50.4f
00383 #define ACCEL_DLPF3_NBW_HZ         68.8f
00384 #define ACCEL_DLPF4_3DB_BW_HZ      23.9f
00385 #define ACCEL_DLPF4_NBW_HZ         34.4f
00386 #define ACCEL_DLPF5_3DB_BW_HZ      11.5f
00387 #define ACCEL_DLPF5_NBW_HZ         17.0f
00388 #define ACCEL_DLPF6_3DB_BW_HZ      5.7f
00389 #define ACCEL_DLPF6_NBW_HZ         8.3f
00390 #define ACCEL_DLPF7_3DB_BW_HZ      473.0f
00391 #define ACCEL_DLPF7_NBW_HZ         499.0f
00392
00393
00394 /* ----- Gyroscope Range ----- */
00395 /** - ±250/±500/±1000/±2000 dps (maps to FS_SEL 0..3) */
00396 #define dps250      0
00397 #define dps500      1
00398 #define dps1000     2
00399 #define dps2000     3
00400
00401 /* ----- Gyroscope Filter Path (FCHOICE) ----- */
00402 /**
00403  * Path select for gyro:
00404  * - GYRO_FCHOICE_BYPASS : DLPF bypassed (widest BW, fastest)
00405  * - GYRO_FCHOICE_DLPF   : DLPF enabled (use GYRO_DLPFCFG + SMPLRT_DIV)
00406  */
00407 #define GYRO_FCHOICE_BYPASS      0
00408 #define GYRO_FCHOICE_DLPF       1
00409
00410 /* ----- Gyroscope DLPF Config (CFG) ----- */
00411 /**
00412  * GYRO_DLPFCFG (0..7) – choose cutoff/noise BW set (when DLPF is ON).
00413  * Tip: start with GYRO_DLPFCFG_3 or _2 for balanced noise/latency.
00414  */
00415 #define GYRO_DLPFCFG_0          0
00416 #define GYRO_DLPFCFG_1          1
00417 #define GYRO_DLPFCFG_2          2
00418 #define GYRO_DLPFCFG_3          3
00419 #define GYRO_DLPFCFG_4          4
00420 #define GYRO_DLPFCFG_5          5
00421 #define GYRO_DLPFCFG_6          6
00422 #define GYRO_DLPFCFG_7          7
00423
00424 /* ---- Gyro DLPF ODR helper (Hz) : base 1100 Hz, guard ≥~4.3 Hz ---- */
00425 #define GYRO_SMPLRT_MIN_HZ       4.3f
00426 #define GYRO_DLPF_BASE_HZ        1100.0f
00427 #define GYRO_DLPF_ODR_HZ(rate_request) (rate_request) /* symbolic; your driver computes DIV =
round(1100/rate)-1 */
00428
00429 /* (Optional) Gyro BW “buckets” to tag configs in UI/logs (symbolic)
00430 * Use these if you want a quick human-readable label for NBW tiers.
00431 * Exact Hz depend on datasheet table; map per your implementation.
00432 */
00433 #define GYRO_BW_ULTRA_WIDE      0 /* bypass path */
00434 #define GYRO_BW_WIDE            1 /* e.g., CFG 0/1 */
00435 #define GYRO_BW_MEDIUM         2 /* e.g., CFG 2/3 */
00436 #define GYRO_BW_NARROW         3 /* e.g., CFG 4/5/6/7 */
00437
00438 /* ----- Convenience: presets (symbolic) ----- */
00439 /** - Friendly ODR presets you can map to dividers */
00440 #define Hz2      0
00441 #define Hz6      1
00442 #define Hz8      2
00443 #define Hz10     3

```

```
00444 #define Hz15    4
00445 #define Hz20    5
00446 #define Hz25    6
00447 #define Hz30    7
00448
00449 /* ----- Power / Fusion Modes ----- */
00450 /**
00451  * Generic power or fusion quality modes (BNO-style).
00452  * Map to sensor-specific sequences inside your driver.
00453  */
00454 #define LowPower      0
00455 #define Regular       1
00456 #define EnhancedRegular 2
00457 #define HighAccuracy  3
00458
00459 #endif /* ICM20948_REGS_H */
```


Index

~Interface_7Semi

Interface_7Semi, [67](#)

ACCEL_BYPASS_3DB_BW_HZ

ICM20948_regs.h, [184](#)

ACCEL_BYPASS_NBW_HZ

ICM20948_regs.h, [184](#)

ACCEL_BYPASS_RATE_HZ

ICM20948_regs.h, [184](#)

ACCEL_CONFIG

ICM20948_regs.h, [185](#)

ACCEL_CONFIG_2

ICM20948_regs.h, [185](#)

ACCEL_DEC3_AVG_16

ICM20948_regs.h, [185](#)

ACCEL_DEC3_AVG_32

ICM20948_regs.h, [185](#)

ACCEL_DEC3_AVG_4

ICM20948_regs.h, [185](#)

ACCEL_DEC3_AVG_8

ICM20948_regs.h, [185](#)

ACCEL_DLPF0_3DB_BW_HZ

ICM20948_regs.h, [185](#)

ACCEL_DLPF0_NBW_HZ

ICM20948_regs.h, [185](#)

ACCEL_DLPF1_3DB_BW_HZ

ICM20948_regs.h, [185](#)

ACCEL_DLPF1_NBW_HZ

ICM20948_regs.h, [185](#)

ACCEL_DLPF2_3DB_BW_HZ

ICM20948_regs.h, [185](#)

ACCEL_DLPF2_NBW_HZ

ICM20948_regs.h, [185](#)

ACCEL_DLPF3_3DB_BW_HZ

ICM20948_regs.h, [186](#)

ACCEL_DLPF3_NBW_HZ

ICM20948_regs.h, [186](#)

ACCEL_DLPF4_3DB_BW_HZ

ICM20948_regs.h, [186](#)

ACCEL_DLPF4_NBW_HZ

ICM20948_regs.h, [186](#)

ACCEL_DLPF5_3DB_BW_HZ

ICM20948_regs.h, [186](#)

ACCEL_DLPF5_NBW_HZ

ICM20948_regs.h, [186](#)

ACCEL_DLPF6_3DB_BW_HZ

ICM20948_regs.h, [186](#)

ACCEL_DLPF6_NBW_HZ

ICM20948_regs.h, [186](#)

ACCEL_DLPF7_3DB_BW_HZ

ICM20948_regs.h, [186](#)

ACCEL_DLPF7_NBW_HZ

ICM20948_regs.h, [186](#)

ACCEL_DLPF_BASE_HZ

ICM20948_regs.h, [186](#)

ACCEL_DLPF_ODR_HZ

ICM20948_regs.h, [186](#)

ACCEL_DLPFCFG_0

ICM20948_regs.h, [187](#)

ACCEL_DLPFCFG_1

ICM20948_regs.h, [187](#)

ACCEL_DLPFCFG_2

ICM20948_regs.h, [187](#)

ACCEL_DLPFCFG_3

ICM20948_regs.h, [187](#)

ACCEL_DLPFCFG_4

ICM20948_regs.h, [187](#)

ACCEL_DLPFCFG_5

ICM20948_regs.h, [187](#)

ACCEL_DLPFCFG_6

ICM20948_regs.h, [187](#)

ACCEL_DLPFCFG_7

ICM20948_regs.h, [187](#)

ACCEL_DLPFCFG_MASK

ICM20948_regs.h, [187](#)

ACCEL_DLPFCFG_SHIFT

ICM20948_regs.h, [187](#)

ACCEL_FCHOICE

ICM20948_regs.h, [187](#)

ACCEL_FCHOICE_BYPASS

ICM20948_regs.h, [188](#)

ACCEL_FCHOICE_DLPF

ICM20948_regs.h, [188](#)

ACCEL_FS_16G

ICM20948_regs.h, [188](#)

ACCEL_FS_2G

ICM20948_regs.h, [188](#)

ACCEL_FS_4G

ICM20948_regs.h, [188](#)

ACCEL_FS_8G

ICM20948_regs.h, [188](#)

ACCEL_FS_SEL_MASK

ICM20948_regs.h, [188](#)

ACCEL_FS_SEL_SHIFT

ICM20948_regs.h, [188](#)

ACCEL_INTEL_CTRL

ICM20948_regs.h, [188](#)

ACCEL_INTEL_EN

ICM20948_regs.h, [188](#)

- ACCEL_INTEL_MODE_INT
 - ICM20948_regs.h, [188](#)
- ACCEL_SMPLRT_DIV_1
 - ICM20948_regs.h, [189](#)
- ACCEL_SMPLRT_DIV_2
 - ICM20948_regs.h, [189](#)
- ACCEL_SMPLRT_DIV_MAX
 - ICM20948_regs.h, [189](#)
- ACCEL_SMPLRT_DIV_MIN
 - ICM20948_regs.h, [189](#)
- ACCEL_WOM_THR
 - ICM20948_regs.h, [189](#)
- ACCEL_XOUT_H
 - ICM20948_regs.h, [189](#)
- ACCEL_XOUT_L
 - ICM20948_regs.h, [189](#)
- ACCEL_YOUT_H
 - ICM20948_regs.h, [189](#)
- ACCEL_YOUT_L
 - ICM20948_regs.h, [189](#)
- ACCEL_ZOUT_H
 - ICM20948_regs.h, [189](#)
- ACCEL_ZOUT_L
 - ICM20948_regs.h, [189](#)
- accelAVG
 - test.ino, [123](#)
 - test2.ino, [132](#)
- accelCfg
 - test.ino, [123](#)
 - test2.ino, [133](#)
- accelDLPF
 - test.ino, [123](#)
 - test2.ino, [133](#)
- accelSR
 - test.ino, [123](#)
 - test2.ino, [133](#)
- activeLevel
 - ICM20948_IntPinConfig, [63](#)
- address
 - I2C_Interface, [61](#)
- AK09916_I2C_ADDR
 - ICM20948_regs.h, [189](#)
- AK_CNTL2
 - ICM20948_regs.h, [190](#)
- AK_CNTL3
 - ICM20948_regs.h, [190](#)
- AK_HXL
 - ICM20948_regs.h, [190](#)
- AK_ST1
 - ICM20948_regs.h, [190](#)
- AK_ST2
 - ICM20948_regs.h, [190](#)
- AK_WIA2
 - ICM20948_regs.h, [190](#)
- AK_WIA2_VAL
 - ICM20948_regs.h, [190](#)
- applyBasicDefaults
 - DevLab_ICM20948, [25](#)
- applySettings
 - gyr_test.ino, [75](#)
 - mag_test.ino, [102](#)
 - test.ino, [120](#)
 - test2.ino, [130](#)
- AUTO_PLL_1
 - DevLab_ICM20948.h, [166](#)
- AUTO_PLL_2
 - DevLab_ICM20948.h, [166](#)
- AUTO_PLL_3
 - DevLab_ICM20948.h, [166](#)
- AUTO_PLL_4
 - DevLab_ICM20948.h, [166](#)
- AUTO_PLL_5
 - DevLab_ICM20948.h, [166](#)
- AUTO_SEL
 - ICM20948_regs.h, [190](#)
- auxConfigSlave
 - DevLab_ICM20948, [26](#)
- auxMasterEnable
 - DevLab_ICM20948, [26](#)
- auxRead12bit
 - DevLab_ICM20948, [27](#)
- auxReadByte
 - DevLab_ICM20948, [27](#)
- auxReadResponse
 - DevLab_ICM20948, [28](#)
- auxReadSensorData
 - DevLab_ICM20948, [29](#)
- auxWriteByte
 - DevLab_ICM20948, [30](#)
- auxWriteCommand
 - DevLab_ICM20948, [30](#)
- AVG_1
 - DevLab_ICM20948.h, [166](#)
- AVG_128
 - DevLab_ICM20948.h, [166](#)
- AVG_16
 - DevLab_ICM20948.h, [165](#), [166](#)
- AVG_1_OR_4
 - DevLab_ICM20948.h, [165](#)
- AVG_2
 - DevLab_ICM20948.h, [166](#)
- AVG_32
 - DevLab_ICM20948.h, [165](#), [166](#)
- AVG_4
 - DevLab_ICM20948.h, [166](#)
- AVG_64
 - DevLab_ICM20948.h, [166](#)
- AVG_8
 - DevLab_ICM20948.h, [165](#), [166](#)
- AX_ST_EN_REG
 - ICM20948_regs.h, [190](#)
- AY_ST_EN_REG
 - ICM20948_regs.h, [190](#)
- AZ_ST_EN_REG
 - ICM20948_regs.h, [190](#)
- BANK1_REG_BANK_SEL

- ICM20948_regs.h, [190](#)
- BANK2_REG_BANK_SEL
 - ICM20948_regs.h, [191](#)
- BANK3_REG_BANK_SEL
 - ICM20948_regs.h, [191](#)
- beginI2C
 - DevLab_ICM20948, [31](#)
 - I2C_Interface, [60](#)
 - Interface_7Semi, [67](#)
 - SPI_Interface, [70](#)
- beginSPI
 - DevLab_ICM20948, [31](#)
 - I2C_Interface, [60](#)
 - Interface_7Semi, [67](#)
 - SPI_Interface, [70](#)
- BusIO_7Semi
 - BusIO_7Semi< Bus >, [13](#)
- BusIO_7Semi< Bus >, [13](#)
 - BusIO_7Semi, [13](#)
 - read, [14](#), [15](#)
 - readBit, [15](#), [16](#)
 - readBits, [16](#), [17](#)
 - write, [17](#), [18](#)
 - writeBit, [19](#), [20](#)
 - writeBits, [21](#)
- BYPASS_EN
 - ICM20948_regs.h, [191](#)
- bypassEn
 - ICM20948_IntPinConfig, [63](#)
- checkIntStatus
 - DevLab_ICM20948, [32](#)
- clearMode
 - ICM20948_IntPinConfig, [63](#)
- CLK_STOP
 - ICM20948_regs.h, [191](#)
- configDLPF, [23](#)
 - div, [23](#)
 - dlpf_idx, [23](#)
 - nbw_hz, [23](#)
 - nombbre, [23](#)
 - odr_hz, [23](#)
- configsDLPF
 - gyr_test.ino, [77](#)
 - test.ino, [123](#)
 - test2.ino, [133](#)
- cs
 - SPI_Interface, [71](#)
- CS_PIN
 - spi_accel.ino, [107](#)
 - spi_basic.ino, [111](#)
 - spi_gyro.ino, [114](#)
- DEC3_CFG_MASK
 - ICM20948_regs.h, [191](#)
- DEC3_CFG_SHIFT
 - ICM20948_regs.h, [191](#)
- DELAY_ES_SHADOW
 - ICM20948_regs.h, [191](#)
- delay_ms
 - icm_plat_t, [65](#)
- DELAY_TIME_EN
 - ICM20948_regs.h, [191](#)
- DELAY_TIMEH
 - ICM20948_regs.h, [191](#)
- DELAY_TIMEL
 - ICM20948_regs.h, [191](#)
- delay_us
 - Interface_7Semi, [67](#)
- DevLab ICM20948 Arduino Library, [1](#)
- DevLab_ICM20948, [24](#)
 - applyBasicDefaults, [25](#)
 - auxConfigSlave, [26](#)
 - auxMasterEnable, [26](#)
 - auxRead12bit, [27](#)
 - auxReadByte, [27](#)
 - auxReadResponse, [28](#)
 - auxReadSensorData, [29](#)
 - auxWriteByte, [30](#)
 - auxWriteCommand, [30](#)
 - beginI2C, [31](#)
 - beginSPI, [31](#)
 - checkIntStatus, [32](#)
 - DevLab_ICM20948, [25](#)
 - getAccelAveraging, [33](#)
 - getAccelDLPF, [33](#)
 - getAccelOffset, [34](#)
 - getAccelSampleRate, [34](#)
 - getAccelScale, [35](#)
 - getClock, [35](#)
 - getDLPF, [36](#)
 - getGyroOffset, [36](#)
 - getGyroSampleRate, [37](#)
 - getGyroScale, [37](#)
 - getLowPower, [38](#)
 - getSensors, [38](#)
 - initMag, [39](#)
 - intEnableConfig, [39](#)
 - intInit, [40](#)
 - readAccel, [41](#)
 - readGyro, [42](#)
 - readMag, [42](#)
 - readTemperature, [43](#)
 - readWhoAmI, [44](#)
 - selectBank, [45](#)
 - selfTestAccel, [45](#)
 - selfTestGyro, [46](#)
 - setAccelAveraging, [46](#)
 - setAccelDivRate, [47](#)
 - setAccelDLPF, [48](#)
 - setAccelOffset, [49](#)
 - setAccelSampleRate, [49](#)
 - setAccelScale, [50](#)
 - setClock, [51](#)
 - setDLPF, [51](#)
 - setGyroAveraging, [52](#)
 - setGyroDivRate, [52](#)

- setGyroOffset, [53](#)
- setGyroSampleRate, [54](#)
- setGyroScale, [54](#)
- setLowPower, [55](#)
- setMagOpMode, [55](#)
- setSensors, [56](#)
- sleep, [57](#)
- softReset, [57](#)
- DevLab_ICM20948.h
 - AUTO_PLL_1, [166](#)
 - AUTO_PLL_2, [166](#)
 - AUTO_PLL_3, [166](#)
 - AUTO_PLL_4, [166](#)
 - AUTO_PLL_5, [166](#)
 - AVG_1, [166](#)
 - AVG_128, [166](#)
 - AVG_16, [165](#), [166](#)
 - AVG_1_OR_4, [165](#)
 - AVG_2, [166](#)
 - AVG_32, [165](#), [166](#)
 - AVG_4, [166](#)
 - AVG_64, [166](#)
 - AVG_8, [165](#), [166](#)
 - DLPF0_246HZ, [165](#)
 - DLPF_111HZ, [165](#)
 - DLPF_119HZ, [167](#)
 - DLPF_11HZ, [167](#)
 - DLPF_12HZ, [165](#)
 - DLPF_151HZ, [167](#)
 - DLPF_196HZ, [167](#)
 - DLPF_23HZ, [167](#)
 - DLPF_246HZ, [165](#)
 - DLPF_24HZ, [165](#)
 - DLPF_361HZ, [167](#)
 - DLPF_473HZ, [165](#)
 - DLPF_50HZ, [165](#)
 - DLPF_51HZ, [167](#)
 - DLPF_5HZ, [167](#)
 - DLPF_6HZ, [165](#)
 - DPS_1000, [167](#)
 - DPS_2000, [167](#)
 - DPS_250, [167](#)
 - DPS_500, [167](#)
 - G_16, [166](#)
 - G_2, [166](#)
 - G_4, [166](#)
 - G_8, [166](#)
 - ICM20948_Accel_Average, [165](#)
 - ICM20948_Accel_DLPCFG, [165](#)
 - ICM20948_Accel_FullScale, [166](#)
 - ICM20948_Clock_Source, [166](#)
 - ICM20948_Gyro_Average, [166](#)
 - ICM20948_Gyro_DLPF, [166](#)
 - ICM20948_Gyro_FullScale, [167](#)
 - ICM20948_Op_Mode, [167](#)
 - INTERNAL_20MHZ_0, [166](#)
 - INTERNAL_20MHZ_6, [166](#)
 - MODE_CONTINUOUS_1, [167](#)
 - MODE_CONTINUOUS_2, [167](#)
 - MODE_CONTINUOUS_3, [167](#)
 - MODE_CONTINUOUS_4, [167](#)
 - MODE_POWER_DOWN, [167](#)
 - MODE_SELF_TEST, [167](#)
 - MODE_SINGLE_MEASUREMENT, [167](#)
 - STOP_CLOCK, [166](#)
- div
 - configDLPF, [23](#)
- DLPF0_246HZ
 - DevLab_ICM20948.h, [165](#)
- DLPF_111HZ
 - DevLab_ICM20948.h, [165](#)
- DLPF_119HZ
 - DevLab_ICM20948.h, [167](#)
- DLPF_11HZ
 - DevLab_ICM20948.h, [167](#)
- DLPF_12HZ
 - DevLab_ICM20948.h, [165](#)
- DLPF_151HZ
 - DevLab_ICM20948.h, [167](#)
- DLPF_196HZ
 - DevLab_ICM20948.h, [167](#)
- DLPF_23HZ
 - DevLab_ICM20948.h, [167](#)
- DLPF_246HZ
 - DevLab_ICM20948.h, [165](#)
- DLPF_24HZ
 - DevLab_ICM20948.h, [165](#)
- DLPF_361HZ
 - DevLab_ICM20948.h, [167](#)
- DLPF_473HZ
 - DevLab_ICM20948.h, [165](#)
- DLPF_50HZ
 - DevLab_ICM20948.h, [165](#)
- DLPF_51HZ
 - DevLab_ICM20948.h, [167](#)
- DLPF_5HZ
 - DevLab_ICM20948.h, [167](#)
- DLPF_6HZ
 - DevLab_ICM20948.h, [165](#)
- dlpf_idx
 - configDLPF, [23](#)
- dmpInt1
 - ICM20948_IntStatus, [64](#)
- dmpInt1En
 - ICM20948_IntEnableConfig, [62](#)
- dps1000
 - ICM20948_regs.h, [191](#)
- dps2000
 - ICM20948_regs.h, [191](#)
- dps250
 - ICM20948_regs.h, [192](#)
- dps500
 - ICM20948_regs.h, [192](#)
- DPS_1000
 - DevLab_ICM20948.h, [167](#)
- DPS_2000

- DevLab_ICM20948.h, [167](#)
- DPS_250
 - DevLab_ICM20948.h, [167](#)
- DPS_500
 - DevLab_ICM20948.h, [167](#)
- driveMode
 - ICM20948_IntPinConfig, [63](#)
- EnhancedRegular
 - ICM20948_regs.h, [192](#)
- etiquetasAccelAVG
 - test.ino, [124](#)
 - test2.ino, [133](#)
- etiquetasAccelCfg
 - test.ino, [124](#)
 - test2.ino, [133](#)
- etiquetasAccelDLPCFG
 - test.ino, [124](#)
- etiquetasGyroAVGCfg
 - gyr_test.ino, [78](#)
- etiquetasGyroFSCfg
 - gyr_test.ino, [78](#)
- etiquetasOpModeCfg
 - mag_test.ino, [104](#)
- etiquetasSR
 - test.ino, [124](#)
 - test2.ino, [134](#)
- examples/gyr_test/gyr_test.ino, [73, 79](#)
- examples/i2c_accel/i2c_accel.ino, [82, 84](#)
- examples/i2c_basic/i2c_basic.ino, [85, 87](#)
- examples/i2c_gyro/i2c_gyro.ino, [89, 91](#)
- examples/i2c_magneto/i2c_magneto.ino, [93, 95](#)
- examples/interrupt/interrupt.ino, [96, 99](#)
- examples/mag_test/mag_test.ino, [101, 104](#)
- examples/spi_accel/spi_accel.ino, [106, 108](#)
- examples/spi_basic/spi_basic.ino, [110, 112](#)
- examples/spi_gyro/spi_gyro.ino, [114, 116](#)
- examples/test/test.ino, [117, 125](#)
- examples/test2/test2.ino, [128, 135](#)
- EXT_SLV_SENS_DATA_00
 - ICM20948_regs.h, [192](#)
- EXT_SLV_SENS_DATA_23
 - ICM20948_regs.h, [192](#)
- EXT_SYNC_SET_MASK
 - ICM20948_regs.h, [192](#)
- FIFO_CFG
 - ICM20948_regs.h, [192](#)
- FIFO_COUNTH
 - ICM20948_regs.h, [192](#)
- FIFO_COUNTL
 - ICM20948_regs.h, [192](#)
- FIFO_EN_1
 - ICM20948_regs.h, [192](#)
- FIFO_EN_2
 - ICM20948_regs.h, [192](#)
- FIFO_MODE
 - ICM20948_regs.h, [192](#)
- FIFO_R_W
 - ICM20948_regs.h, [193](#)
- FIFO_RST
 - ICM20948_regs.h, [193](#)
- fifoOvf
 - ICM20948_IntStatus, [64](#)
- fifoOvfEn
 - ICM20948_IntEnableConfig, [62](#)
- fifoWm
 - ICM20948_IntStatus, [64](#)
- fifoWmEn
 - ICM20948_IntEnableConfig, [62](#)
- firstStage
 - gyr_test.ino, [75](#)
 - test.ino, [120](#)
 - test2.ino, [130](#)
- FSYNC_CONFIG
 - ICM20948_regs.h, [193](#)
- FSYNC_INT_MODE_EN
 - ICM20948_regs.h, [193](#)
- fsyncActLevel
 - ICM20948_IntPinConfig, [63](#)
- fsyncIntEn
 - ICM20948_IntPinConfig, [63](#)
- g16
 - ICM20948_regs.h, [193](#)
- g2
 - ICM20948_regs.h, [193](#)
- g4
 - ICM20948_regs.h, [193](#)
- g8
 - ICM20948_regs.h, [193](#)
- G_16
 - DevLab_ICM20948.h, [166](#)
- G_2
 - DevLab_ICM20948.h, [166](#)
- G_4
 - DevLab_ICM20948.h, [166](#)
- G_8
 - DevLab_ICM20948.h, [166](#)
- getAccelAveraging
 - DevLab_ICM20948, [33](#)
- getAccelDLPF
 - DevLab_ICM20948, [33](#)
- getAccelOffset
 - DevLab_ICM20948, [34](#)
- getAccelSampleRate
 - DevLab_ICM20948, [34](#)
- getAccelScale
 - DevLab_ICM20948, [35](#)
- getClock
 - DevLab_ICM20948, [35](#)
- getDLPF
 - DevLab_ICM20948, [36](#)
- getGyroOffset
 - DevLab_ICM20948, [36](#)
- getGyroSampleRate
 - DevLab_ICM20948, [37](#)
- getGyroScale

- DevLab_ICM20948, [37](#)
- getLowPower
 - DevLab_ICM20948, [38](#)
- getSensors
 - DevLab_ICM20948, [38](#)
- gyr_test.ino
 - applySettings, [75](#)
 - configsDLPF, [77](#)
 - etiquetasGyroAVGCfg, [78](#)
 - etiquetasGyroFSCfg, [78](#)
 - firstStage, [75](#)
 - gyroAVGCfg, [78](#)
 - gyroDLPF, [78](#)
 - gyroFSCfg, [78](#)
 - I2C_FREQ, [74](#)
 - ICM_ADDR, [74](#)
 - imu, [78](#)
 - loop, [76](#)
 - N_AVG, [79](#)
 - N_DLPF, [79](#)
 - N_FS, [79](#)
 - printDobleLinea, [76](#)
 - printLinea, [76](#)
 - runSweep, [76](#)
 - SCL_PIN, [75](#)
 - SDA_PIN, [75](#)
 - setup, [77](#)
- GYRO_AVGCFG_MASK
 - ICM20948_regs.h, [193](#)
- GYRO_AVGCFG_SHIFT
 - ICM20948_regs.h, [193](#)
- GYRO_BW_MEDIUM
 - ICM20948_regs.h, [193](#)
- GYRO_BW_NARROW
 - ICM20948_regs.h, [193](#)
- GYRO_BW_ULTRA_WIDE
 - ICM20948_regs.h, [194](#)
- GYRO_BW_WIDE
 - ICM20948_regs.h, [194](#)
- GYRO_CONFIG_1
 - ICM20948_regs.h, [194](#)
- GYRO_CONFIG_2
 - ICM20948_regs.h, [194](#)
- GYRO_DLPF_BASE_HZ
 - ICM20948_regs.h, [194](#)
- GYRO_DLPF_ODR_HZ
 - ICM20948_regs.h, [194](#)
- GYRO_DLPFCFG_0
 - ICM20948_regs.h, [194](#)
- GYRO_DLPFCFG_1
 - ICM20948_regs.h, [194](#)
- GYRO_DLPFCFG_2
 - ICM20948_regs.h, [194](#)
- GYRO_DLPFCFG_3
 - ICM20948_regs.h, [194](#)
- GYRO_DLPFCFG_4
 - ICM20948_regs.h, [194](#)
- GYRO_DLPFCFG_5
 - ICM20948_regs.h, [195](#)
- GYRO_DLPFCFG_6
 - ICM20948_regs.h, [195](#)
- GYRO_DLPFCFG_7
 - ICM20948_regs.h, [195](#)
- GYRO_DLPFCFG_MASK
 - ICM20948_regs.h, [195](#)
- GYRO_DLPFCFG_SHIFT
 - ICM20948_regs.h, [195](#)
- GYRO_FCHOICE
 - ICM20948_regs.h, [195](#)
- GYRO_FCHOICE_BYPASS
 - ICM20948_regs.h, [195](#)
- GYRO_FCHOICE_DLPF
 - ICM20948_regs.h, [195](#)
- GYRO_FS_1000DPS
 - ICM20948_regs.h, [195](#)
- GYRO_FS_2000DPS
 - ICM20948_regs.h, [195](#)
- GYRO_FS_250DPS
 - ICM20948_regs.h, [195](#)
- GYRO_FS_500DPS
 - ICM20948_regs.h, [196](#)
- GYRO_FS_SEL_MASK
 - ICM20948_regs.h, [196](#)
- GYRO_FS_SEL_SHIFT
 - ICM20948_regs.h, [196](#)
- GYRO_SMPLRT_DIV
 - ICM20948_regs.h, [196](#)
- GYRO_SMPLRT_MIN_HZ
 - ICM20948_regs.h, [196](#)
- GYRO_XOUT_H
 - ICM20948_regs.h, [196](#)
- GYRO_XOUT_L
 - ICM20948_regs.h, [196](#)
- GYRO_YOUT_H
 - ICM20948_regs.h, [196](#)
- GYRO_YOUT_L
 - ICM20948_regs.h, [196](#)
- GYRO_ZOUT_H
 - ICM20948_regs.h, [196](#)
- GYRO_ZOUT_L
 - ICM20948_regs.h, [196](#)
- gyroAVGCfg
 - gyr_test.ino, [78](#)
- gyroDLPF
 - gyr_test.ino, [78](#)
- gyroFSCfg
 - gyr_test.ino, [78](#)
- HighAccuracy
 - ICM20948_regs.h, [196](#)
- Hz10
 - ICM20948_regs.h, [197](#)
- Hz15
 - ICM20948_regs.h, [197](#)
- Hz2
 - ICM20948_regs.h, [197](#)
- Hz20
 - ICM20948_regs.h, [197](#)

- ICM20948_regs.h, [197](#)
- Hz25
 - ICM20948_regs.h, [197](#)
- Hz30
 - ICM20948_regs.h, [197](#)
- Hz6
 - ICM20948_regs.h, [197](#)
- Hz8
 - ICM20948_regs.h, [197](#)
- i2c
 - I2C_Interface, [61](#)
- i2c_accel.ino
 - I2C_FREQ, [82](#)
 - ICM_ADDR, [82](#)
 - imu, [84](#)
 - loop, [83](#)
 - SCL_PIN, [82](#)
 - SDA_PIN, [83](#)
 - setup, [83](#)
- i2c_addr
 - icm_plat_t, [65](#)
- i2c_basic.ino
 - I2C_FREQ, [86](#)
 - ICM_ADDR, [86](#)
 - imu, [87](#)
 - loop, [86](#)
 - SCL_PIN, [86](#)
 - SDA_PIN, [86](#)
 - setup, [87](#)
- I2C_FREQ
 - gyr_test.ino, [74](#)
 - i2c_accel.ino, [82](#)
 - i2c_basic.ino, [86](#)
 - i2c_gyro.ino, [90](#)
 - i2c_magneto.ino, [93](#)
 - interrupt.ino, [97](#)
 - mag_test.ino, [102](#)
 - test.ino, [120](#)
 - test2.ino, [129](#)
- i2c_gyro.ino
 - I2C_FREQ, [90](#)
 - ICM_ADDR, [90](#)
 - imu, [91](#)
 - loop, [90](#)
 - SCL_PIN, [90](#)
 - SDA_PIN, [90](#)
 - setup, [90](#)
- I2C_Interface, [58](#)
 - address, [61](#)
 - beginI2C, [60](#)
 - beginSPI, [60](#)
 - i2c, [61](#)
 - read, [60](#)
 - write, [61](#)
- i2c_magneto.ino
 - I2C_FREQ, [93](#)
 - ICM_ADDR, [93](#)
 - imu, [95](#)
 - loop, [94](#)
 - SCL_PIN, [93](#)
 - SDA_PIN, [94](#)
 - setup, [94](#)
- I2C_MST_CLK_MASK
 - ICM20948_regs.h, [197](#)
- I2C_MST_CTRL
 - ICM20948_regs.h, [197](#)
- I2C_MST_DELAY_CTRL
 - ICM20948_regs.h, [197](#)
- I2C_MST_ODR_CONFIG
 - ICM20948_regs.h, [197](#)
- I2C_MST_P_NSR
 - ICM20948_regs.h, [198](#)
- I2C_MST_STATUS
 - ICM20948_regs.h, [198](#)
- I2C_SLV0_ADDR
 - ICM20948_regs.h, [198](#)
- I2C_SLV0_CTRL
 - ICM20948_regs.h, [198](#)
- I2C_SLV0_DELAY_EN
 - ICM20948_regs.h, [198](#)
- I2C_SLV0_DO
 - ICM20948_regs.h, [198](#)
- I2C_SLV0_REG
 - ICM20948_regs.h, [198](#)
- I2C_SLV1_ADDR
 - ICM20948_regs.h, [198](#)
- I2C_SLV1_CTRL
 - ICM20948_regs.h, [198](#)
- I2C_SLV1_DELAY_EN
 - ICM20948_regs.h, [198](#)
- I2C_SLV1_DO
 - ICM20948_regs.h, [198](#)
- I2C_SLV1_REG
 - ICM20948_regs.h, [198](#)
- I2C_SLV2_ADDR
 - ICM20948_regs.h, [199](#)
- I2C_SLV2_CTRL
 - ICM20948_regs.h, [199](#)
- I2C_SLV2_DELAY_EN
 - ICM20948_regs.h, [199](#)
- I2C_SLV2_DO
 - ICM20948_regs.h, [199](#)
- I2C_SLV2_REG
 - ICM20948_regs.h, [199](#)
- I2C_SLV3_ADDR
 - ICM20948_regs.h, [199](#)
- I2C_SLV3_CTRL
 - ICM20948_regs.h, [199](#)
- I2C_SLV3_DELAY_EN
 - ICM20948_regs.h, [199](#)
- I2C_SLV3_DO
 - ICM20948_regs.h, [199](#)
- I2C_SLV3_REG
 - ICM20948_regs.h, [199](#)
- I2C_SLV4_ADDR
 - ICM20948_regs.h, [199](#)

- I2C_SLV4_CTRL
 - ICM20948_regs.h, [199](#)
- I2C_SLV4_DELAY_EN
 - ICM20948_regs.h, [200](#)
- I2C_SLV4_DI
 - ICM20948_regs.h, [200](#)
- I2C_SLV4_DLY_MASK
 - ICM20948_regs.h, [200](#)
- I2C_SLV4_DO
 - ICM20948_regs.h, [200](#)
- I2C_SLV4_REG
 - ICM20948_regs.h, [200](#)
- I2C_SLVx_BYTE_SW
 - ICM20948_regs.h, [200](#)
- I2C_SLVx_EN
 - ICM20948_regs.h, [200](#)
- I2C_SLVx_GRP
 - ICM20948_regs.h, [200](#)
- I2C_SLVx LENG_MASK
 - ICM20948_regs.h, [200](#)
- I2C_SLVx_REG_DIS
 - ICM20948_regs.h, [200](#)
- I2C_SLVx_RNW
 - ICM20948_regs.h, [200](#)
- i2cMstInt
 - ICM20948_IntStatus, [64](#)
- i2cMstIntEn
 - ICM20948_IntEnableConfig, [62](#)
- ICM20948_Accel_Average
 - DevLab_ICM20948.h, [165](#)
- ICM20948_Accel_DLPFCFG
 - DevLab_ICM20948.h, [165](#)
- ICM20948_Accel_FullScale
 - DevLab_ICM20948.h, [166](#)
- ICM20948_Clock_Source
 - DevLab_ICM20948.h, [166](#)
- ICM20948_Gyro_Average
 - DevLab_ICM20948.h, [166](#)
- ICM20948_Gyro_DLPF
 - DevLab_ICM20948.h, [166](#)
- ICM20948_Gyro_FullScale
 - DevLab_ICM20948.h, [167](#)
- ICM20948_IntEnableConfig, [61](#)
 - dmpInt1En, [62](#)
 - fifoOvfEn, [62](#)
 - fifoWmEn, [62](#)
 - i2cMstIntEn, [62](#)
 - pllRdyEn, [62](#)
 - rawDataRdyEn, [62](#)
 - wofEn, [62](#)
 - womIntEn, [62](#)
- ICM20948_IntPinConfig, [62](#)
 - activeLevel, [63](#)
 - bypassEn, [63](#)
 - clearMode, [63](#)
 - driveMode, [63](#)
 - fsyncActLevel, [63](#)
 - fsyncIntEn, [63](#)
 - latchMode, [63](#)
- ICM20948_IntStatus, [64](#)
 - dmpInt1, [64](#)
 - fifoOvf, [64](#)
 - fifoWm, [64](#)
 - i2cMstInt, [64](#)
 - pllRdyInt, [64](#)
 - rawDataRdy, [64](#)
 - womInt, [64](#)
- ICM20948_Op_Mode
 - DevLab_ICM20948.h, [167](#)
- ICM20948_regs.h
 - ACCEL_BYPASS_3DB_BW_HZ, [184](#)
 - ACCEL_BYPASS_NBW_HZ, [184](#)
 - ACCEL_BYPASS_RATE_HZ, [184](#)
 - ACCEL_CONFIG, [185](#)
 - ACCEL_CONFIG_2, [185](#)
 - ACCEL_DEC3_AVG_16, [185](#)
 - ACCEL_DEC3_AVG_32, [185](#)
 - ACCEL_DEC3_AVG_4, [185](#)
 - ACCEL_DEC3_AVG_8, [185](#)
 - ACCEL_DLPF0_3DB_BW_HZ, [185](#)
 - ACCEL_DLPF0_NBW_HZ, [185](#)
 - ACCEL_DLPF1_3DB_BW_HZ, [185](#)
 - ACCEL_DLPF1_NBW_HZ, [185](#)
 - ACCEL_DLPF2_3DB_BW_HZ, [185](#)
 - ACCEL_DLPF2_NBW_HZ, [185](#)
 - ACCEL_DLPF3_3DB_BW_HZ, [186](#)
 - ACCEL_DLPF3_NBW_HZ, [186](#)
 - ACCEL_DLPF4_3DB_BW_HZ, [186](#)
 - ACCEL_DLPF4_NBW_HZ, [186](#)
 - ACCEL_DLPF5_3DB_BW_HZ, [186](#)
 - ACCEL_DLPF5_NBW_HZ, [186](#)
 - ACCEL_DLPF6_3DB_BW_HZ, [186](#)
 - ACCEL_DLPF6_NBW_HZ, [186](#)
 - ACCEL_DLPF7_3DB_BW_HZ, [186](#)
 - ACCEL_DLPF7_NBW_HZ, [186](#)
 - ACCEL_DLPF_BASE_HZ, [186](#)
 - ACCEL_DLPF_ODR_HZ, [186](#)
 - ACCEL_DLPFCFG_0, [187](#)
 - ACCEL_DLPFCFG_1, [187](#)
 - ACCEL_DLPFCFG_2, [187](#)
 - ACCEL_DLPFCFG_3, [187](#)
 - ACCEL_DLPFCFG_4, [187](#)
 - ACCEL_DLPFCFG_5, [187](#)
 - ACCEL_DLPFCFG_6, [187](#)
 - ACCEL_DLPFCFG_7, [187](#)
 - ACCEL_DLPFCFG_MASK, [187](#)
 - ACCEL_DLPFCFG_SHIFT, [187](#)
 - ACCEL_FCHOICE, [187](#)
 - ACCEL_FCHOICE_BYPASS, [188](#)
 - ACCEL_FCHOICE_DLPF, [188](#)
 - ACCEL_FS_16G, [188](#)
 - ACCEL_FS_2G, [188](#)
 - ACCEL_FS_4G, [188](#)
 - ACCEL_FS_8G, [188](#)
 - ACCEL_FS_SEL_MASK, [188](#)
 - ACCEL_FS_SEL_SHIFT, [188](#)

ACCEL_INTEL_CTRL, [188](#)
ACCEL_INTEL_EN, [188](#)
ACCEL_INTEL_MODE_INT, [188](#)
ACCEL_SMPLRT_DIV_1, [189](#)
ACCEL_SMPLRT_DIV_2, [189](#)
ACCEL_SMPLRT_DIV_MAX, [189](#)
ACCEL_SMPLRT_DIV_MIN, [189](#)
ACCEL_WOM_THR, [189](#)
ACCEL_XOUT_H, [189](#)
ACCEL_XOUT_L, [189](#)
ACCEL_YOUT_H, [189](#)
ACCEL_YOUT_L, [189](#)
ACCEL_ZOUT_H, [189](#)
ACCEL_ZOUT_L, [189](#)
AK09916_I2C_ADDR, [189](#)
AK_CNTL2, [190](#)
AK_CNTL3, [190](#)
AK_HXL, [190](#)
AK_ST1, [190](#)
AK_ST2, [190](#)
AK_WIA2, [190](#)
AK_WIA2_VAL, [190](#)
AUTO_SEL, [190](#)
AX_ST_EN_REG, [190](#)
AY_ST_EN_REG, [190](#)
AZ_ST_EN_REG, [190](#)
BANK1_REG_BANK_SEL, [190](#)
BANK2_REG_BANK_SEL, [191](#)
BANK3_REG_BANK_SEL, [191](#)
BYPASS_EN, [191](#)
CLK_STOP, [191](#)
DEC3_CFG_MASK, [191](#)
DEC3_CFG_SHIFT, [191](#)
DELAY_ES_SHADOW, [191](#)
DELAY_TIME_EN, [191](#)
DELAY_TIMEH, [191](#)
DELAY_TIMEL, [191](#)
dps1000, [191](#)
dps2000, [191](#)
dps250, [192](#)
dps500, [192](#)
EnhancedRegular, [192](#)
EXT_SLV_SENS_DATA_00, [192](#)
EXT_SLV_SENS_DATA_23, [192](#)
EXT_SYNC_SET_MASK, [192](#)
FIFO_CFG, [192](#)
FIFO_COUNTH, [192](#)
FIFO_COUNTL, [192](#)
FIFO_EN_1, [192](#)
FIFO_EN_2, [192](#)
FIFO_MODE, [192](#)
FIFO_R_W, [193](#)
FIFO_RST, [193](#)
FSYNC_CONFIG, [193](#)
FSYNC_INT_MODE_EN, [193](#)
g16, [193](#)
g2, [193](#)
g4, [193](#)
g8, [193](#)
GYRO_AVGCFG_MASK, [193](#)
GYRO_AVGCFG_SHIFT, [193](#)
GYRO_BW_MEDIUM, [193](#)
GYRO_BW_NARROW, [193](#)
GYRO_BW_ULTRA_WIDE, [194](#)
GYRO_BW_WIDE, [194](#)
GYRO_CONFIG_1, [194](#)
GYRO_CONFIG_2, [194](#)
GYRO_DLPF_BASE_HZ, [194](#)
GYRO_DLPF_ODR_HZ, [194](#)
GYRO_DLPFCFG_0, [194](#)
GYRO_DLPFCFG_1, [194](#)
GYRO_DLPFCFG_2, [194](#)
GYRO_DLPFCFG_3, [194](#)
GYRO_DLPFCFG_4, [194](#)
GYRO_DLPFCFG_5, [195](#)
GYRO_DLPFCFG_6, [195](#)
GYRO_DLPFCFG_7, [195](#)
GYRO_DLPFCFG_MASK, [195](#)
GYRO_DLPFCFG_SHIFT, [195](#)
GYRO_FCHOICE, [195](#)
GYRO_FCHOICE_BYPASS, [195](#)
GYRO_FCHOICE_DLPF, [195](#)
GYRO_FS_1000DPS, [195](#)
GYRO_FS_2000DPS, [195](#)
GYRO_FS_250DPS, [195](#)
GYRO_FS_500DPS, [196](#)
GYRO_FS_SEL_MASK, [196](#)
GYRO_FS_SEL_SHIFT, [196](#)
GYRO_SMPLRT_DIV, [196](#)
GYRO_SMPLRT_MIN_HZ, [196](#)
GYRO_XOUT_H, [196](#)
GYRO_XOUT_L, [196](#)
GYRO_YOUT_H, [196](#)
GYRO_YOUT_L, [196](#)
GYRO_ZOUT_H, [196](#)
GYRO_ZOUT_L, [196](#)
HighAccuracy, [196](#)
Hz10, [197](#)
Hz15, [197](#)
Hz2, [197](#)
Hz20, [197](#)
Hz25, [197](#)
Hz30, [197](#)
Hz6, [197](#)
Hz8, [197](#)
I2C_MST_CLK_MASK, [197](#)
I2C_MST_CTRL, [197](#)
I2C_MST_DELAY_CTRL, [197](#)
I2C_MST_ODR_CONFIG, [197](#)
I2C_MST_P_NSR, [198](#)
I2C_MST_STATUS, [198](#)
I2C_SLV0_ADDR, [198](#)
I2C_SLV0_CTRL, [198](#)
I2C_SLV0_DELAY_EN, [198](#)
I2C_SLV0_DO, [198](#)
I2C_SLV0_REG, [198](#)

- I2C_SLV1_ADDR, 198
- I2C_SLV1_CTRL, 198
- I2C_SLV1_DELAY_EN, 198
- I2C_SLV1_DO, 198
- I2C_SLV1_REG, 198
- I2C_SLV2_ADDR, 199
- I2C_SLV2_CTRL, 199
- I2C_SLV2_DELAY_EN, 199
- I2C_SLV2_DO, 199
- I2C_SLV2_REG, 199
- I2C_SLV3_ADDR, 199
- I2C_SLV3_CTRL, 199
- I2C_SLV3_DELAY_EN, 199
- I2C_SLV3_DO, 199
- I2C_SLV3_REG, 199
- I2C_SLV4_ADDR, 199
- I2C_SLV4_CTRL, 199
- I2C_SLV4_DELAY_EN, 200
- I2C_SLV4_DI, 200
- I2C_SLV4_DLY_MASK, 200
- I2C_SLV4_DO, 200
- I2C_SLV4_REG, 200
- I2C_SLVx_BYTE_SW, 200
- I2C_SLVx_EN, 200
- I2C_SLVx_GRP, 200
- I2C_SLVx LENG_MASK, 200
- I2C_SLVx_REG_DIS, 200
- I2C_SLVx_RNW, 200
- INT1_ACTL, 200
- INT1_LATCH_INT_EN, 201
- INT1_OPEN, 201
- INT_ACTL_FSYNC, 201
- INT_ANYRD_2CLEAR, 201
- INT_DMP_INT1_EN, 201
- INT_ENABLE, 201
- INT_ENABLE_1, 201
- INT_ENABLE_2, 201
- INT_ENABLE_3, 201
- INT_I2C_MST_INT_EN, 201
- INT_PIN_CFG, 201
- INT_PLL_RDY_EN, 201
- INT_RAW_DATA_0_RDY_EN, 202
- INT_REG_WOF_EN, 202
- INT_STATUS, 202
- INT_STATUS_1, 202
- INT_STATUS_2, 202
- INT_STATUS_3, 202
- INT_WOM_INT_EN, 202
- INTERNAL_20MHZ, 202
- LowPower, 202
- LP_ACCEL_CYCLE, 202
- LP_CONFIG, 202
- LP_GYRO_CYCLE, 202
- LP_I2C_MST_CYCLE, 203
- MOD_CTRL_USR, 203
- MST_LOST_ARB, 203
- MST_ODR_CFG_MASK, 203
- MST_PASS_THROUGH, 203
- MST_SLV0_NACK, 203
- MST_SLV1_NACK, 203
- MST_SLV2_NACK, 203
- MST_SLV3_NACK, 203
- MST_SLV4_DONE, 203
- MST_SLV4_NACK, 203
- MULT_MST_EN, 203
- ODR_ALIGN_EN, 204
- ODR_ALIGN_EN_BIT, 204
- PWR_CLKSEL_AUTO, 204
- PWR_CLKSEL_INT_20MHZ, 204
- PWR_CLKSEL_MASK, 204
- PWR_DEVICE_RESET, 204
- PWR_DISABLE_ACCEL, 204
- PWR_DISABLE_GYRO, 204
- PWR_LP_EN, 204
- PWR_MGMT_1, 204
- PWR_MGMT_2, 204
- PWR_SLEEP, 204
- PWR_TEMP_DIS, 205
- REG_BANK_SEL, 205
- REG_BANK_SEL_USER_BANK_MASK, 205
- REG_BANK_SEL_USER_BANK_SHIFT, 205
- REG_LP_DMP_EN, 205
- Regular, 205
- SELF_TEST_X_ACCEL, 205
- SELF_TEST_X_GYRO, 205
- SELF_TEST_Y_ACCEL, 205
- SELF_TEST_Y_GYRO, 205
- SELF_TEST_Z_ACCEL, 205
- SELF_TEST_Z_GYRO, 205
- STS_DMP_INT1, 206
- STS_I2C_MST_INT, 206
- STS_PLL_RDY_INT, 206
- STS_RAW_DATA_0_RDY_INT, 206
- STS_WOM_INT, 206
- TEMP_CONFIG, 206
- TEMP_DLPFCFG_MASK, 206
- TEMP_DLPFCFG_SHIFT, 206
- TEMP_OUT_H, 206
- TEMP_OUT_L, 206
- TIMEBASE_CORRECTION_PLL, 206
- USER_BANK_0, 206
- USER_BANK_1, 207
- USER_BANK_2, 207
- USER_BANK_3, 207
- USER_CTRL, 207
- USER_CTRL_DMP_EN, 207
- USER_CTRL_DMP_RST, 207
- USER_CTRL_FIFO_EN, 207
- USER_CTRL_I2C_IF_DIS, 207
- USER_CTRL_I2C_MST_EN, 207
- USER_CTRL_I2C_MST_RST, 207
- USER_CTRL_SRAM_RST, 207
- WHO_AM_I, 207
- WHO_AM_I_VAL, 208
- WOF DEGLITCH_EN, 208
- WOF_EDGE_INT, 208

- XA_OFFS_H, [208](#)
- XA_OFFS_L, [208](#)
- XG_OFFS_USRH, [208](#)
- XG_OFFS_USRL, [208](#)
- XGYRO_CTEN, [208](#)
- YA_OFFS_H, [208](#)
- YA_OFFS_L, [209](#)
- YG_OFFS_USRH, [209](#)
- YG_OFFS_USRL, [209](#)
- YGYRO_CTEN, [209](#)
- ZA_OFFS_H, [209](#)
- ZA_OFFS_L, [209](#)
- ZG_OFFS_USRH, [209](#)
- ZG_OFFS_USRL, [209](#)
- ZGYRO_CTEN, [209](#)
- ICM_ADDR
 - gyr_test.ino, [74](#)
 - i2c_accel.ino, [82](#)
 - i2c_basic.ino, [86](#)
 - i2c_gyro.ino, [90](#)
 - i2c_magneto.ino, [93](#)
 - interrupt.ino, [97](#)
 - mag_test.ino, [102](#)
 - test.ino, [120](#)
 - test2.ino, [129](#)
- icm_plat_t, [65](#)
 - delay_ms, [65](#)
 - i2c_addr, [65](#)
 - read, [65](#)
 - spi_cs, [65](#)
 - spi_reg_rw_bits, [66](#)
 - user, [66](#)
 - write, [66](#)
- imu
 - gyr_test.ino, [78](#)
 - i2c_accel.ino, [84](#)
 - i2c_basic.ino, [87](#)
 - i2c_gyro.ino, [91](#)
 - i2c_magneto.ino, [95](#)
 - interrupt.ino, [99](#)
 - mag_test.ino, [104](#)
 - spi_accel.ino, [108](#)
 - spi_basic.ino, [112](#)
 - spi_gyro.ino, [116](#)
 - test.ino, [124](#)
 - test2.ino, [134](#)
- initMag
 - DevLab_ICM20948, [39](#)
- INT1_ACTL
 - ICM20948_regs.h, [200](#)
- INT1_LATCH_INT_EN
 - ICM20948_regs.h, [201](#)
- INT1_OPEN
 - ICM20948_regs.h, [201](#)
- INT_ACTL_FSYNC
 - ICM20948_regs.h, [201](#)
- INT_ANYRD_2CLEAR
 - ICM20948_regs.h, [201](#)
- INT_DMP_INT1_EN
 - ICM20948_regs.h, [201](#)
- INT_ENABLE
 - ICM20948_regs.h, [201](#)
- INT_ENABLE_1
 - ICM20948_regs.h, [201](#)
- INT_ENABLE_2
 - ICM20948_regs.h, [201](#)
- INT_ENABLE_3
 - ICM20948_regs.h, [201](#)
- INT_I2C_MST_INT_EN
 - ICM20948_regs.h, [201](#)
- INT_PIN_CFG
 - ICM20948_regs.h, [201](#)
- INT_PLL_RDY_EN
 - ICM20948_regs.h, [201](#)
- INT_RAW_DATA_0_RDY_EN
 - ICM20948_regs.h, [202](#)
- INT_REG_WOF_EN
 - ICM20948_regs.h, [202](#)
- INT_STATUS
 - ICM20948_regs.h, [202](#)
- INT_STATUS_1
 - ICM20948_regs.h, [202](#)
- INT_STATUS_2
 - ICM20948_regs.h, [202](#)
- INT_STATUS_3
 - ICM20948_regs.h, [202](#)
- INT_WOM_INT_EN
 - ICM20948_regs.h, [202](#)
- intEn
 - interrupt.ino, [99](#)
- intEnableConfig
 - DevLab_ICM20948, [39](#)
- Interface_7Semi, [66](#)
 - ~Interface_7Semi, [67](#)
 - beginI2C, [67](#)
 - beginSPI, [67](#)
 - delay_us, [67](#)
 - read, [68](#)
 - write, [68](#)
- INTERNAL_20MHZ
 - ICM20948_regs.h, [202](#)
- INTERNAL_20MHZ_0
 - DevLab_ICM20948.h, [166](#)
- INTERNAL_20MHZ_6
 - DevLab_ICM20948.h, [166](#)
- interrupt.ino
 - I2C_FREQ, [97](#)
 - ICM_ADDR, [97](#)
 - imu, [99](#)
 - intEn, [99](#)
 - isrCount, [99](#)
 - isrHandler, [98](#)
 - loop, [98](#)
 - PIN_INT, [97](#)
 - pinCfg, [99](#)
 - printDobleLinea, [98](#)

- printLinea, 98
- SCL_PIN, 98
- SDA_PIN, 98
- setup, 98
- intInit
 - DevLab_ICM20948, 40
- isrCount
 - interrupt.ino, 99
- isrHandler
 - interrupt.ino, 98
- latchMode
 - ICM20948_IntPinConfig, 63
- loop
 - gyr_test.ino, 76
 - i2c_accel.ino, 83
 - i2c_basic.ino, 86
 - i2c_gyro.ino, 90
 - i2c_magneto.ino, 94
 - interrupt.ino, 98
 - mag_test.ino, 103
 - spi_accel.ino, 107
 - spi_basic.ino, 111
 - spi_gyro.ino, 114
 - test.ino, 121
 - test2.ino, 131
- LowPower
 - ICM20948_regs.h, 202
- LP_ACCEL_CYCLE
 - ICM20948_regs.h, 202
- LP_CONFIG
 - ICM20948_regs.h, 202
- LP_GYRO_CYCLE
 - ICM20948_regs.h, 202
- LP_I2C_MST_CYCLE
 - ICM20948_regs.h, 203
- mag_test.ino
 - applySettings, 102
 - etiquetasOpModeCfg, 104
 - I2C_FREQ, 102
 - ICM_ADDR, 102
 - imu, 104
 - loop, 103
 - N_OPM, 104
 - opMode, 104
 - printDobleLinea, 103
 - printLinea, 103
 - SCL_PIN, 102
 - SDA_PIN, 102
 - setup, 103
- MOD_CTRL_USR
 - ICM20948_regs.h, 203
- MODE_CONTINUOUS_1
 - DevLab_ICM20948.h, 167
- MODE_CONTINUOUS_2
 - DevLab_ICM20948.h, 167
- MODE_CONTINUOUS_3
 - DevLab_ICM20948.h, 167
- MODE_CONTINUOUS_4
 - DevLab_ICM20948.h, 167
- MODE_POWER_DOWN
 - DevLab_ICM20948.h, 167
- MODE_SELF_TEST
 - DevLab_ICM20948.h, 167
- MODE_SINGLE_MEASUREMENT
 - DevLab_ICM20948.h, 167
- MST_LOST_ARB
 - ICM20948_regs.h, 203
- MST_ODR_CFG_MASK
 - ICM20948_regs.h, 203
- MST_PASS_THROUGH
 - ICM20948_regs.h, 203
- MST_SLV0_NACK
 - ICM20948_regs.h, 203
- MST_SLV1_NACK
 - ICM20948_regs.h, 203
- MST_SLV2_NACK
 - ICM20948_regs.h, 203
- MST_SLV3_NACK
 - ICM20948_regs.h, 203
- MST_SLV4_DONE
 - ICM20948_regs.h, 203
- MST_SLV4_NACK
 - ICM20948_regs.h, 203
- MULT_MST_EN
 - ICM20948_regs.h, 203
- N_AVG
 - gyr_test.ino, 79
 - test2.ino, 134
- N_DLPF
 - gyr_test.ino, 79
 - test.ino, 124
 - test2.ino, 134
- N_FS
 - gyr_test.ino, 79
 - test.ino, 124
 - test2.ino, 134
- N_OPM
 - mag_test.ino, 104
- nbw_hz
 - configDLPF, 23
- nombre
 - configDLPF, 23
- ODR_ALIGN_EN
 - ICM20948_regs.h, 204
- ODR_ALIGN_EN_BIT
 - ICM20948_regs.h, 204
- odr_hz
 - configDLPF, 23
- opMode
 - mag_test.ino, 104
- PIN_INT
 - interrupt.ino, 97
- pinCfg

- interrupt.ino, [99](#)
- pllRdyEn
 - ICM20948_IntEnableConfig, [62](#)
- pllRdyInt
 - ICM20948_IntStatus, [64](#)
- printDobleLinea
 - gyr_test.ino, [76](#)
 - interrupt.ino, [98](#)
 - mag_test.ino, [103](#)
 - test.ino, [121](#)
 - test2.ino, [131](#)
- printLinea
 - gyr_test.ino, [76](#)
 - interrupt.ino, [98](#)
 - mag_test.ino, [103](#)
 - test.ino, [121](#)
 - test2.ino, [131](#)
- PWR_CLKSEL_AUTO
 - ICM20948_regs.h, [204](#)
- PWR_CLKSEL_INT_20MHZ
 - ICM20948_regs.h, [204](#)
- PWR_CLKSEL_MASK
 - ICM20948_regs.h, [204](#)
- PWR_DEVICE_RESET
 - ICM20948_regs.h, [204](#)
- PWR_DISABLE_ACCEL
 - ICM20948_regs.h, [204](#)
- PWR_DISABLE_GYRO
 - ICM20948_regs.h, [204](#)
- PWR_LP_EN
 - ICM20948_regs.h, [204](#)
- PWR_MGMT_1
 - ICM20948_regs.h, [204](#)
- PWR_MGMT_2
 - ICM20948_regs.h, [204](#)
- PWR_SLEEP
 - ICM20948_regs.h, [204](#)
- PWR_TEMP_DIS
 - ICM20948_regs.h, [205](#)
- rawDataRdy
 - ICM20948_IntStatus, [64](#)
- rawDataRdyEn
 - ICM20948_IntEnableConfig, [62](#)
- read
 - BusIO_7Semi< Bus >, [14](#), [15](#)
 - I2C_Interface, [60](#)
 - icm_plat_t, [65](#)
 - Interface_7Semi, [68](#)
 - SPI_Interface, [70](#)
- readAccel
 - DevLab_ICM20948, [41](#)
- readBit
 - BusIO_7Semi< Bus >, [15](#), [16](#)
- readBits
 - BusIO_7Semi< Bus >, [16](#), [17](#)
- readGyro
 - DevLab_ICM20948, [42](#)
- readMag
 - DevLab_ICM20948, [42](#)
- README.md, [138](#)
- readTemperature
 - DevLab_ICM20948, [43](#)
- readWhoAmI
 - DevLab_ICM20948, [44](#)
- REG_BANK_SEL
 - ICM20948_regs.h, [205](#)
- REG_BANK_SEL_USER_BANK_MASK
 - ICM20948_regs.h, [205](#)
- REG_BANK_SEL_USER_BANK_SHIFT
 - ICM20948_regs.h, [205](#)
- REG_LP_DMP_EN
 - ICM20948_regs.h, [205](#)
- Regular
 - ICM20948_regs.h, [205](#)
- runSweep
 - gyr_test.ino, [76](#)
 - test.ino, [122](#)
 - test2.ino, [131](#)
- SCL_PIN
 - gyr_test.ino, [75](#)
 - i2c_accel.ino, [82](#)
 - i2c_basic.ino, [86](#)
 - i2c_gyro.ino, [90](#)
 - i2c_magneto.ino, [93](#)
 - interrupt.ino, [98](#)
 - mag_test.ino, [102](#)
 - test.ino, [120](#)
 - test2.ino, [130](#)
- SDA_PIN
 - gyr_test.ino, [75](#)
 - i2c_accel.ino, [83](#)
 - i2c_basic.ino, [86](#)
 - i2c_gyro.ino, [90](#)
 - i2c_magneto.ino, [94](#)
 - interrupt.ino, [98](#)
 - mag_test.ino, [102](#)
 - test.ino, [120](#)
 - test2.ino, [130](#)
- selectBank
 - DevLab_ICM20948, [45](#)
- SELF_TEST_X_ACCEL
 - ICM20948_regs.h, [205](#)
- SELF_TEST_X_GYRO
 - ICM20948_regs.h, [205](#)
- SELF_TEST_Y_ACCEL
 - ICM20948_regs.h, [205](#)
- SELF_TEST_Y_GYRO
 - ICM20948_regs.h, [205](#)
- SELF_TEST_Z_ACCEL
 - ICM20948_regs.h, [205](#)
- SELF_TEST_Z_GYRO
 - ICM20948_regs.h, [205](#)
- selfTestAccel
 - DevLab_ICM20948, [45](#)
- selfTestGyro
 - DevLab_ICM20948, [46](#)

- setAccelAveraging
 - DevLab_ICM20948, [46](#)
- setAccelDivRate
 - DevLab_ICM20948, [47](#)
- setAccelDLPF
 - DevLab_ICM20948, [48](#)
- setAccelOffset
 - DevLab_ICM20948, [49](#)
- setAccelSampleRate
 - DevLab_ICM20948, [49](#)
- setAccelScale
 - DevLab_ICM20948, [50](#)
- setClock
 - DevLab_ICM20948, [51](#)
- setDLPF
 - DevLab_ICM20948, [51](#)
- setGyroAveraging
 - DevLab_ICM20948, [52](#)
- setGyroDivRate
 - DevLab_ICM20948, [52](#)
- setGyroOffset
 - DevLab_ICM20948, [53](#)
- setGyroSampleRate
 - DevLab_ICM20948, [54](#)
- setGyroScale
 - DevLab_ICM20948, [54](#)
- setLowPower
 - DevLab_ICM20948, [55](#)
- setMagOpMode
 - DevLab_ICM20948, [55](#)
- setSensors
 - DevLab_ICM20948, [56](#)
- setup
 - gyr_test.ino, [77](#)
 - i2c_accel.ino, [83](#)
 - i2c_basic.ino, [87](#)
 - i2c_gyro.ino, [90](#)
 - i2c_magneto.ino, [94](#)
 - interrupt.ino, [98](#)
 - mag_test.ino, [103](#)
 - spi_accel.ino, [107](#)
 - spi_basic.ino, [111](#)
 - spi_gyro.ino, [115](#)
 - test.ino, [122](#)
 - test2.ino, [132](#)
- sleep
 - DevLab_ICM20948, [57](#)
- softReset
 - DevLab_ICM20948, [57](#)
- speed
 - SPI_Interface, [71](#)
- spi
 - SPI_Interface, [71](#)
- spi_accel.ino
 - CS_PIN, [107](#)
 - imu, [108](#)
 - loop, [107](#)
 - setup, [107](#)
 - spi_bus, [108](#)
 - SPI_FAST_SPEED, [107](#)
- spi_basic.ino
 - CS_PIN, [111](#)
 - imu, [112](#)
 - loop, [111](#)
 - setup, [111](#)
 - spi_bus, [112](#)
 - SPI_FAST_SPEED, [111](#)
- spi_bus
 - spi_accel.ino, [108](#)
 - spi_basic.ino, [112](#)
 - spi_gyro.ino, [115](#)
- spi_cs
 - icm_plat_t, [65](#)
- SPI_FAST_SPEED
 - spi_accel.ino, [107](#)
 - spi_basic.ino, [111](#)
 - spi_gyro.ino, [114](#)
- spi_gyro.ino
 - CS_PIN, [114](#)
 - imu, [116](#)
 - loop, [114](#)
 - setup, [115](#)
 - spi_bus, [115](#)
 - SPI_FAST_SPEED, [114](#)
- SPI_Interface, [68](#)
 - beginI2C, [70](#)
 - beginSPI, [70](#)
 - cs, [71](#)
 - read, [70](#)
 - speed, [71](#)
 - spi, [71](#)
 - write, [71](#)
- spi_reg_rw_bits
 - icm_plat_t, [66](#)
- src/7Semi_I2C_Interface.h, [138](#), [139](#)
- src/7Semi_Interface.h, [140](#), [141](#)
- src/7Semi_SPI_Interface.h, [142](#), [143](#)
- src/BusIO_7Semi.h, [145](#)
- src/DevLab_ICM20948.cpp, [147](#)
- src/DevLab_ICM20948.h, [164](#), [167](#)
- src/ICM20948_platform.h, [178](#)
- src/ICM20948_regs.h, [179](#), [209](#)
- STOP_CLOCK
 - DevLab_ICM20948.h, [166](#)
- STS_DMP_INT1
 - ICM20948_regs.h, [206](#)
- STS_I2C_MST_INT
 - ICM20948_regs.h, [206](#)
- STS_PLL_RDY_INT
 - ICM20948_regs.h, [206](#)
- STS_RAW_DATA_0_RDY_INT
 - ICM20948_regs.h, [206](#)
- STS_WOM_INT
 - ICM20948_regs.h, [206](#)
- TEMP_CONFIG
 - ICM20948_regs.h, [206](#)

TEMP_DLPFCFG_MASK
 ICM20948_regs.h, 206
 TEMP_DLPFCFG_SHIFT
 ICM20948_regs.h, 206
 TEMP_OUT_H
 ICM20948_regs.h, 206
 TEMP_OUT_L
 ICM20948_regs.h, 206
 test.ino
 accelAVG, 123
 accelCfg, 123
 accelDLPF, 123
 accelSR, 123
 applySettings, 120
 configsDLPF, 123
 etiquetasAccelAVG, 124
 etiquetasAccelCfg, 124
 etiquetasAccelDLPFCFG, 124
 etiquetasSR, 124
 firstStage, 120
 I2C_FREQ, 120
 ICM_ADDR, 120
 imu, 124
 loop, 121
 N_DLPF, 124
 N_FS, 124
 printDobleLinea, 121
 printLinea, 121
 runSweep, 122
 SCL_PIN, 120
 SDA_PIN, 120
 setup, 122
 total_fail, 124
 total_pass, 124
 total_warn, 125
 test2.ino
 accelAVG, 132
 accelCfg, 133
 accelDLPF, 133
 accelSR, 133
 applySettings, 130
 configsDLPF, 133
 etiquetasAccelAVG, 133
 etiquetasAccelCfg, 133
 etiquetasSR, 134
 firstStage, 130
 I2C_FREQ, 129
 ICM_ADDR, 129
 imu, 134
 loop, 131
 N_AVG, 134
 N_DLPF, 134
 N_FS, 134
 printDobleLinea, 131
 printLinea, 131
 runSweep, 131
 SCL_PIN, 130
 SDA_PIN, 130
 setup, 132
 total_fail, 134
 total_pass, 134
 total_warn, 134
 TIMEBASE_CORRECTION_PLL
 ICM20948_regs.h, 206
 total_fail
 test.ino, 124
 test2.ino, 134
 total_pass
 test.ino, 124
 test2.ino, 134
 total_warn
 test.ino, 125
 test2.ino, 134
 user
 icm_plat_t, 66
 USER_BANK_0
 ICM20948_regs.h, 206
 USER_BANK_1
 ICM20948_regs.h, 207
 USER_BANK_2
 ICM20948_regs.h, 207
 USER_BANK_3
 ICM20948_regs.h, 207
 USER_CTRL
 ICM20948_regs.h, 207
 USER_CTRL_DMP_EN
 ICM20948_regs.h, 207
 USER_CTRL_DMP_RST
 ICM20948_regs.h, 207
 USER_CTRL_FIFO_EN
 ICM20948_regs.h, 207
 USER_CTRL_I2C_IF_DIS
 ICM20948_regs.h, 207
 USER_CTRL_I2C_MST_EN
 ICM20948_regs.h, 207
 USER_CTRL_I2C_MST_RST
 ICM20948_regs.h, 207
 USER_CTRL_SRAM_RST
 ICM20948_regs.h, 207
 WHO_AM_I
 ICM20948_regs.h, 207
 WHO_AM_I_VAL
 ICM20948_regs.h, 208
 WOF DEGLITCH_EN
 ICM20948_regs.h, 208
 WOF_EDGE_INT
 ICM20948_regs.h, 208
 wofEn
 ICM20948_IntEnableConfig, 62
 womInt
 ICM20948_IntStatus, 64
 womIntEn
 ICM20948_IntEnableConfig, 62
 write
 BusIO_7Semi< Bus >, 17, 18

- I2C_Interface, [61](#)
- icm_plat_t, [66](#)
- Interface_7Semi, [68](#)
- SPI_Interface, [71](#)
- writeBit
 - BusIO_7Semi< Bus >, [19](#), [20](#)
- writeBits
 - BusIO_7Semi< Bus >, [21](#)
- XA_OFFS_H
 - ICM20948_regs.h, [208](#)
- XA_OFFS_L
 - ICM20948_regs.h, [208](#)
- XG_OFFS_USRH
 - ICM20948_regs.h, [208](#)
- XG_OFFS_USRL
 - ICM20948_regs.h, [208](#)
- XGYRO_CTEN
 - ICM20948_regs.h, [208](#)
- YA_OFFS_H
 - ICM20948_regs.h, [208](#)
- YA_OFFS_L
 - ICM20948_regs.h, [209](#)
- YG_OFFS_USRH
 - ICM20948_regs.h, [209](#)
- YG_OFFS_USRL
 - ICM20948_regs.h, [209](#)
- YGYRO_CTEN
 - ICM20948_regs.h, [209](#)
- ZA_OFFS_H
 - ICM20948_regs.h, [209](#)
- ZA_OFFS_L
 - ICM20948_regs.h, [209](#)
- ZG_OFFS_USRH
 - ICM20948_regs.h, [209](#)
- ZG_OFFS_USRL
 - ICM20948_regs.h, [209](#)
- ZGYRO_CTEN
 - ICM20948_regs.h, [209](#)