

DevLab\_ICM20948

1.0.0

Generated on Mon Jun 15 2026 18:36:22 for DevLab\_ICM20948 by Doxygen 1.17.0

Mon Jun 15 2026 18:36:22



# Chapter 1

## Directory Hierarchy

### 1.1 Directories

admin_pwm	??
admin_pwm.ino	??
examples	??
admin_pwm	??
admin_pwm.ino	??
I2C_Accel	??
I2C_Accel.ino	??
I2C_Basic	??
I2C_Basic.ino	??
I2C_Magneto	??
I2C_Magneto.ino	??
SPI_Basic	??
SPI_Basic.ino	??
I2C_Accel	??
I2C_Accel.ino	??
I2C_Basic	??
I2C_Basic.ino	??
I2C_Magneto	??
I2C_Magneto.ino	??
SPI_Basic	??
SPI_Basic.ino	??
src	??
7Semi_I2C_Interface.h	??
7Semi_Interface.h	??
7Semi_SPI_Interface.h	??
BusIO_7Semi.h	??
DevLab_ICM20948.cpp	??
DevLab_ICM20948.h	??
ICM20948_platform.h	??
ICM20948_regs.h	??



## Chapter 2

# Hierarchical Index

### 2.1 Class Hierarchy

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

BusIO_7Semi< Bus > . . . . .	??
DevLab_ICM20948 . . . . .	??
ICM20948_IntEnableConfig . . . . .	??
ICM20948_IntPinConfig . . . . .	??
ICM20948_IntStatus . . . . .	??
icm_plat_t . . . . .	??
Interface_7Semi . . . . .	??
I2C_Interface . . . . .	??
SPI_Interface . . . . .	??



## Chapter 3

# Class Index

### 3.1 Class List

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

<a href="#">BusIO_7Semi&lt; Bus &gt;</a>	??
<a href="#">DevLab_ICM20948</a>	??
<a href="#">I2C_Interface</a>	??
<a href="#">ICM20948_IntEnableConfig</a>	??
<a href="#">ICM20948_IntPinConfig</a>	??
<a href="#">ICM20948_IntStatus</a>	??
<a href="#">icm_plat_t</a>	??
<a href="#">Interface_7Semi</a>	??
<a href="#">SPI_Interface</a>	??





# Chapter 4

## File Index

### 4.1 File List

Here is a list of all files with brief descriptions:

examples/admin_pwm/admin_pwm.ino	??
examples/I2C_Accel/I2C_Accel.ino	??
examples/I2C_Basic/I2C_Basic.ino	??
examples/I2C_Magneto/I2C_Magneto.ino	??
examples/SPI_Basic/SPI_Basic.ino	??
src/7Semi_I2C_Interface.h	??
src/7Semi_Interface.h	??
src/7Semi_SPI_Interface.h	??
src/BusIO_7Semi.h	??
src/DevLab_ICM20948.cpp	??
src/DevLab_ICM20948.h	??
src/ICM20948_platform.h	??
src/ICM20948_regs.h	??

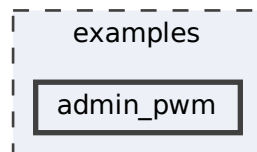


## Chapter 5

# Directory Documentation

### 5.1 examples/admin\_pwm Directory Reference

Directory dependency graph for admin\_pwm:



#### Files

- file [admin\\_pwm.ino](#)

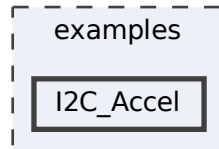
### 5.2 examples Directory Reference

#### Directories

- directory [admin\\_pwm](#)
- directory [I2C\\_Accel](#)
- directory [I2C\\_Basic](#)
- directory [I2C\\_Magneto](#)
- directory [SPI\\_Basic](#)

### 5.3 examples/I2C\_Accel Directory Reference

Directory dependency graph for I2C\_Accel:

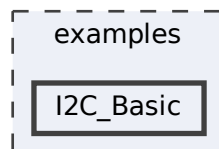


#### Files

- file [I2C\\_Accel.ino](#)

### 5.4 examples/I2C\_Basic Directory Reference

Directory dependency graph for I2C\_Basic:

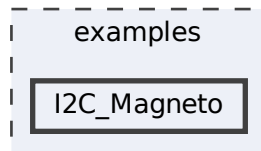


#### Files

- file [I2C\\_Basic.ino](#)

## 5.5 examples/I2C\_Magneto Directory Reference

Directory dependency graph for I2C\_Magneto:

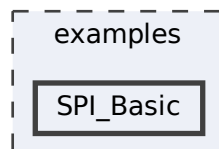


### Files

- file [I2C\\_Magneto.ino](#)

## 5.6 examples/SPI\_Basic Directory Reference

Directory dependency graph for SPI\_Basic:



### Files

- file [SPI\\_Basic.ino](#)

## 5.7 src Directory Reference

### Files

- file [7Semi\\_I2C\\_Interface.h](#)
- file [7Semi\\_Interface.h](#)
- file [7Semi\\_SPI\\_Interface.h](#)
- file [BusIO\\_7Semi.h](#)
- file [DevLab\\_ICM20948.cpp](#)
- file [DevLab\\_ICM20948.h](#)
- file [ICM20948\\_platform.h](#)
- file [ICM20948\\_regs.h](#)

# Chapter 6

## Class Documentation

### 6.1 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)

#### 6.1.1 Detailed Description

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

Definition at line 5 of file [BusIO\\_7Semi.h](#).

## 6.1.2 Constructor & Destructor Documentation

### 6.1.2.1 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](#).

## 6.1.3 Member Function Documentation

### 6.1.3.1 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](#).

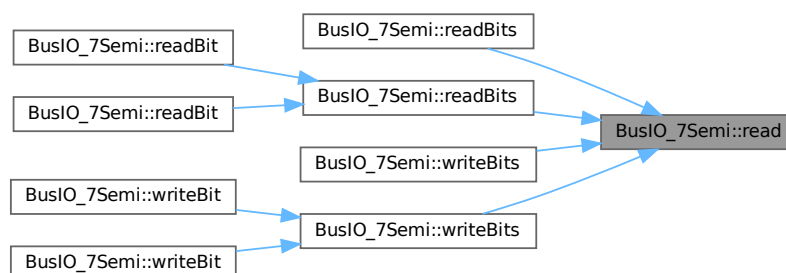
### 6.1.3.2 read() [2/3]

```
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](#).

Here is the caller graph for this function:





### 6.1.3.3 read() [3/3]

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

read burst

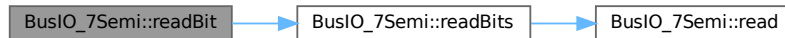
Definition at line 33 of file [BusIO\\_7Semi.h](#).

### 6.1.3.4 readBit() [1/2]

```
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:

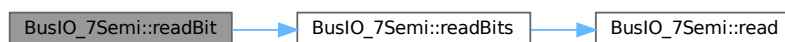


### 6.1.3.5 readBit() [2/2]

```
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:



#### 6.1.3.6 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:



#### 6.1.3.7 readBits() [2/2]

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

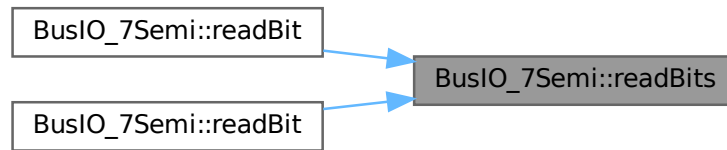
readBits (8-bit register)

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:



#### 6.1.3.8 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](#).

#### 6.1.3.9 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](#).

#### 6.1.3.10 write() [3/3]

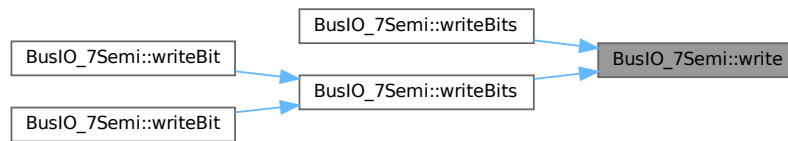
```

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:



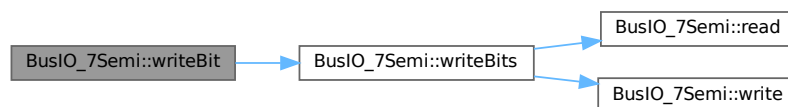
### 6.1.3.11 `writeBit()` [1/2]

```

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:



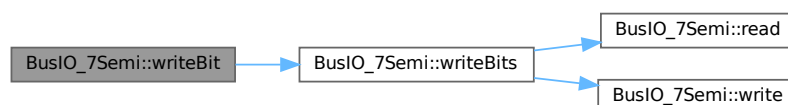
### 6.1.3.12 `writeBit()` [2/2]

```

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:



### 6.1.3.13 writeBits() [1/2]

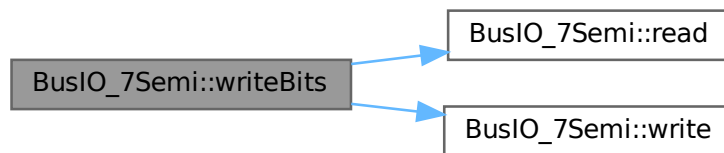
```
template<typename Bus>
bool BusIO_7Semi< Bus >::writeBits (
    uint8_t reg,
    uint8_t pos,
    uint8_t len,
    uint16_t value) [inline]
```

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:



### 6.1.3.14 writeBits() [2/2]

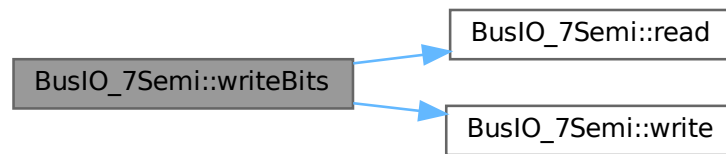
```
template<typename Bus>
bool BusIO_7Semi< Bus >::writeBits (
    uint8_t reg,
    uint8_t pos,
    uint8_t len,
    uint8_t value) [inline]
```

writeBits (8-bit register)

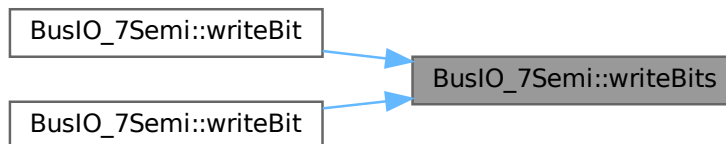
- Modifies specific bits in 8-bit register

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](#)

## 6.2 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 [intlInit](#) (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)

### 6.2.1 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](#).

## 6.2.2 Constructor & Destructor Documentation

### 6.2.2.1 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](#).

## 6.2.3 Member Function Documentation

### 6.2.3.1 applyBasicDefaults()

```
bool DevLab_ICM20948::applyBasicDefaults ()
```

applyBasicDefaults

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

Returns:

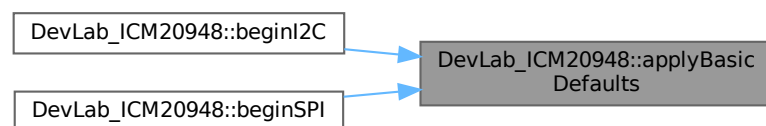
- true → Configuration successful
- false → Any register write failed

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:





### 6.2.3.2 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:



### 6.2.3.3 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 345.6 kHz, 0x0D 400 kHz)

Returns:

- true → Master enabled successfully
- false → Operation failed

Definition at line 1260 of file [DevLab\\_ICM20948.cpp](#).

Here is the call graph for this function:



### 6.2.3.4 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 → Read successful
- false → Read failed

Definition at line 1424 of file [DevLab\\_ICM20948.cpp](#).

Here is the call graph for this function:



### 6.2.3.5 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:



### 6.2.3.6 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:



### 6.2.3.7 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:

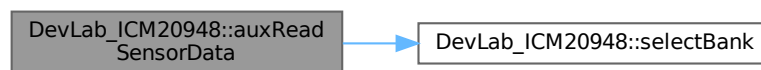
- buf: output buffer to store the received bytes
- len: number of bytes to read (must match numBytes configured in auxConfigSlave)

Returns:

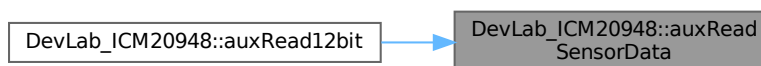
- true → Read successful
- false → Bus error

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:



### 6.2.3.8 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:



### 6.2.3.9 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:

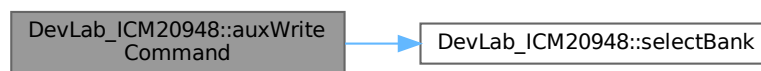
- slaveAddr: 7-bit I2C address of the auxiliary slave
- cmd: command byte to send

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:



### 6.2.3.10 beginI2C()

```

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

`beginI2C`

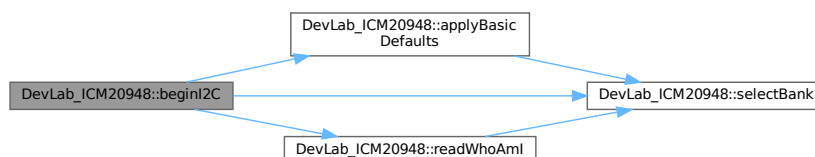
- Initialize ICM20948 using I2C interface
- Sets up communication layer (Interface + BusIO)
- Verifies device identity (WHO\_AM\_I)
- Configures basic power and clock settings

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:

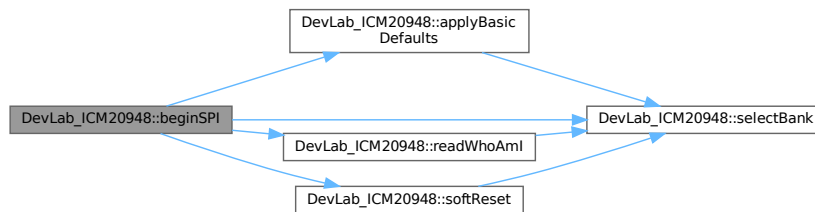


### 6.2.3.11 beginSPI()

```
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:



### 6.2.3.12 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:





### 6.2.3.13 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:



### 6.2.3.14 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:



#### 6.2.3.15 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:



### 6.2.3.16 getAccelSampleRate()

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

getAccelSampleRate

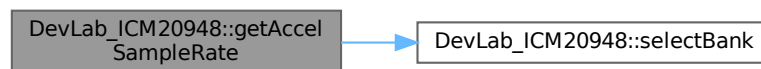
- Get accelerometer output data rate (ODR)
- Computes value from divider registers

Returns:

- true → Read successful
- false → Operation failed

Definition at line 796 of file [DevLab\\_ICM20948.cpp](#).

Here is the call graph for this function:



### 6.2.3.17 getAccelScale()

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

getAccelScale

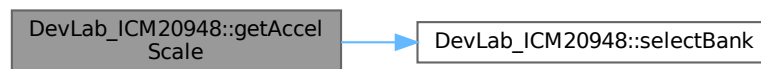
- Get current accelerometer full-scale range (FS\_SEL)

Returns:

- true → Read successful
- false → Operation failed

Definition at line 878 of file [DevLab\\_ICM20948.cpp](#).

Here is the call graph for this function:



### 6.2.3.18 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:



### 6.2.3.19 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:



### 6.2.3.20 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:



#### 6.2.3.21 `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:



### 6.2.3.22 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:



### 6.2.3.23 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:



#### 6.2.3.24 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:



#### 6.2.3.25 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

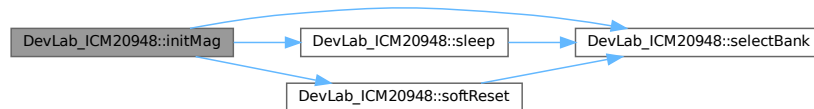


Returns:

- true → Initialization successful
- false → Operation failed

Definition at line 334 of file [DevLab\\_ICM20948.cpp](#).

Here is the call graph for this function:



### 6.2.3.26 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:



### 6.2.3.27 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:



### 6.2.3.28 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:



### 6.2.3.29 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:



### 6.2.3.30 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  $\mu\text{T}$

Returns:

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

Definition at line 310 of file [DevLab\\_ICM20948.cpp](#).

Here is the call graph for this function:



### 6.2.3.31 readTemperature()

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

readTemperature

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

Returns:

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

Definition at line 278 of file [DevLab\\_ICM20948.cpp](#).

Here is the call graph for this function:



### 6.2.3.32 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:

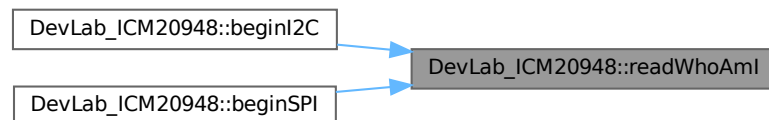
- true → Read successful
- false → Operation failed

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:



### 6.2.3.33 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](#).

#### 6.2.3.34 selfTestAccel()

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

selfTestAccel

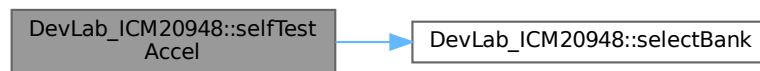
- Enable or disable accelerometer self-test per axis

Returns:

- true → Operation successful
- false → Operation failed

Definition at line 826 of file [DevLab\\_ICM20948.cpp](#).

Here is the call graph for this function:



#### 6.2.3.35 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:



### 6.2.3.36 setAccelAveraging()

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

setAccelAveraging

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

Flow:

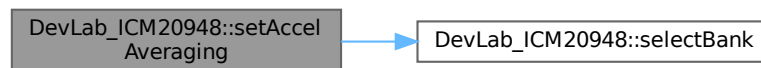
- Validate bus pointer
- Select USER BANK 2
- Write DEC3 bits [1:0]

Returns:

- true → Operation successful
- false → Operation failed

Definition at line 952 of file [DevLab\\_ICM20948.cpp](#).

Here is the call graph for this function:



### 6.2.3.37 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:



### 6.2.3.38 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:



### 6.2.3.39 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:

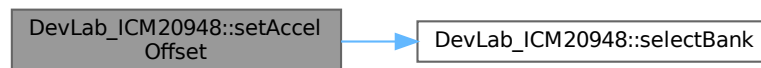
- Validate bus pointer
- Select USER BANK 1
- Pack offsets into byte array
- Write to offset registers

Returns:

- true → Operation successful
- false → Operation failed

Definition at line 1124 of file [DevLab\\_ICM20948.cpp](#).

Here is the call graph for this function:



#### 6.2.3.40 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:



#### 6.2.3.41 setAccelScale()

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

setAccelScale

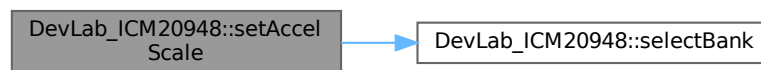
- Set accelerometer full-scale range (FS\_SEL)
- Controls measurement range ( $\pm 2g$ ,  $\pm 4g$ ,  $\pm 8g$ ,  $\pm 16g$ )

Returns:

- true → Operation successful
- false → Operation failed

Definition at line 857 of file [DevLab\\_ICM20948.cpp](#).

Here is the call graph for this function:



#### 6.2.3.42 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:



### 6.2.3.43 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:



### 6.2.3.44 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:

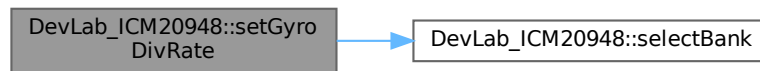


#### 6.2.3.45 setGyroDivRate()

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

Definition at line 780 of file [DevLab\\_ICM20948.cpp](#).

Here is the call graph for this function:



#### 6.2.3.46 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:

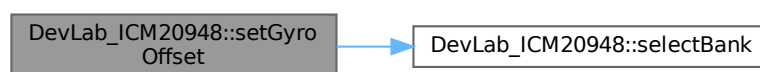
- Validate bus pointer
- Select USER BANK 2
- Pack offsets into byte array
- Write to offset registers

Returns:

- true → Operation successful
- false → Operation failed

Definition at line 1072 of file [DevLab\\_ICM20948.cpp](#).

Here is the call graph for this function:



### 6.2.3.47 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:



### 6.2.3.48 setGyroScale()

```
bool DevLab_ICM20948::setGyroScale (
    ICM20948_Gyro_FullScale fullScale)
```

setGyroScale

- Set gyroscope full-scale range (FS\_SEL)
- Controls sensitivity (dps range)

Flow:

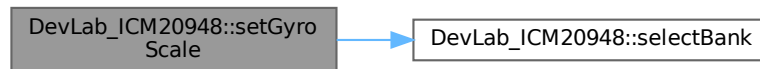
- Validate bus pointer
- Select USER BANK 2
- Write FS\_SEL bits [2:1]

Returns:

- true → Operation successful
- false → Operation failed

Definition at line 644 of file [DevLab\\_ICM20948.cpp](#).

Here is the call graph for this function:



#### 6.2.3.49 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:



### 6.2.3.50 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](#).

### 6.2.3.51 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:



### 6.2.3.52 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:





### 6.2.3.53 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:

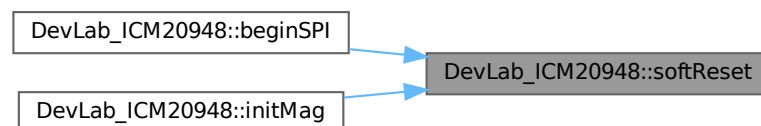
- true → Reset successful
- false → Operation failed

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:



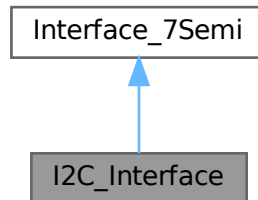
The documentation for this class was generated from the following files:

- [src/DevLab\\_ICM20948.h](#)
- [src/DevLab\\_ICM20948.cpp](#)

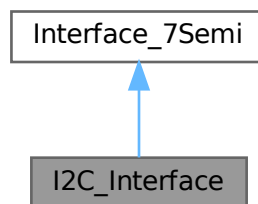
## 6.3 I2C\_Interface Class Reference

```
#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)

### 6.3.1 Detailed Description

Definition at line 5 of file [7Semi\\_I2C\\_Interface.h](#).

### 6.3.2 Member Function Documentation

#### 6.3.2.1 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](#).

#### 6.3.2.2 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](#).

### 6.3.2.3 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](#).

### 6.3.2.4 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](#).

### 6.3.3 Member Data Documentation

#### 6.3.3.1 address

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

Definition at line 9 of file [7Semi\\_I2C\\_Interface.h](#).

#### 6.3.3.2 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](#)

## 6.4 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]

### 6.4.1 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](#).

## 6.4.2 Member Data Documentation

### 6.4.2.1 dmpInt1En

```
uint8_t ICM20948_IntEnableConfig::dmpInt1En
```

Definition at line 128 of file [DevLab\\_ICM20948.h](#).

### 6.4.2.2 fifoOvfEn

```
uint8_t ICM20948_IntEnableConfig::fifoOvfEn[5]
```

Definition at line 135 of file [DevLab\\_ICM20948.h](#).

### 6.4.2.3 fifoWmEn

```
uint8_t ICM20948_IntEnableConfig::fifoWmEn[5]
```

Definition at line 138 of file [DevLab\\_ICM20948.h](#).

### 6.4.2.4 i2cMstIntEn

```
uint8_t ICM20948_IntEnableConfig::i2cMstIntEn
```

Definition at line 129 of file [DevLab\\_ICM20948.h](#).

### 6.4.2.5 pllRdyEn

```
uint8_t ICM20948_IntEnableConfig::pllRdyEn
```

Definition at line 127 of file [DevLab\\_ICM20948.h](#).

### 6.4.2.6 rawDataRdyEn

```
uint8_t ICM20948_IntEnableConfig::rawDataRdyEn
```

Definition at line 132 of file [DevLab\\_ICM20948.h](#).

### 6.4.2.7 wofEn

```
uint8_t ICM20948_IntEnableConfig::wofEn
```

Definition at line 125 of file [DevLab\\_ICM20948.h](#).

#### 6.4.2.8 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](#)

## 6.5 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

### 6.5.1 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](#).

### 6.5.2 Member Data Documentation

#### 6.5.2.1 activeLevel

```
uint8_t ICM20948_IntPinConfig::activeLevel
```

Definition at line 105 of file [DevLab\\_ICM20948.h](#).

### 6.5.2.2 bypassEn

```
uint8_t ICM20948_IntPinConfig::bypassEn
```

Definition at line 111 of file [DevLab\\_ICM20948.h](#).

### 6.5.2.3 clearMode

```
uint8_t ICM20948_IntPinConfig::clearMode
```

Definition at line 108 of file [DevLab\\_ICM20948.h](#).

### 6.5.2.4 driveMode

```
uint8_t ICM20948_IntPinConfig::driveMode
```

Definition at line 106 of file [DevLab\\_ICM20948.h](#).

### 6.5.2.5 fsyncActLevel

```
uint8_t ICM20948_IntPinConfig::fsyncActLevel
```

Definition at line 109 of file [DevLab\\_ICM20948.h](#).

### 6.5.2.6 fsyncIntEn

```
uint8_t ICM20948_IntPinConfig::fsyncIntEn
```

Definition at line 110 of file [DevLab\\_ICM20948.h](#).

### 6.5.2.7 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](#)

## 6.6 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]

## 6.6.1 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](#).

## 6.6.2 Member Data Documentation

### 6.6.2.1 [dmpInt1](#)

```
uint8_t ICM20948_IntStatus::dmpInt1
```

Definition at line 154 of file [DevLab\\_ICM20948.h](#).

### 6.6.2.2 [fifoOvf](#)

```
uint8_t ICM20948_IntStatus::fifoOvf[5]
```

Definition at line 161 of file [DevLab\\_ICM20948.h](#).

### 6.6.2.3 [fifoWm](#)

```
uint8_t ICM20948_IntStatus::fifoWm[5]
```

Definition at line 164 of file [DevLab\\_ICM20948.h](#).

### 6.6.2.4 [i2cMstInt](#)

```
uint8_t ICM20948_IntStatus::i2cMstInt
```

Definition at line 155 of file [DevLab\\_ICM20948.h](#).

### 6.6.2.5 pllRdyInt

```
uint8_t ICM20948_IntStatus::pllRdyInt
```

Definition at line 153 of file [DevLab\\_ICM20948.h](#).

### 6.6.2.6 rawDataRdy

```
uint8_t ICM20948_IntStatus::rawDataRdy
```

Definition at line 158 of file [DevLab\\_ICM20948.h](#).

### 6.6.2.7 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](#)

## 6.7 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](#)

### 6.7.1 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](#).

## 6.7.2 Member Data Documentation

### 6.7.2.1 delay\_ms

```
void(* icm_plat_t::delay_ms) (uint32_t ms)
```

- Millisecond delay

Definition at line 32 of file [ICM20948\\_platform.h](#).

### 6.7.2.2 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](#).

### 6.7.2.3 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](#).

### 6.7.2.4 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](#).

### 6.7.2.5 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](#).

### 6.7.2.6 user

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

- User context pointer passed to read/write

Definition at line 36 of file [ICM20948\\_platform.h](#).

### 6.7.2.7 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](#).

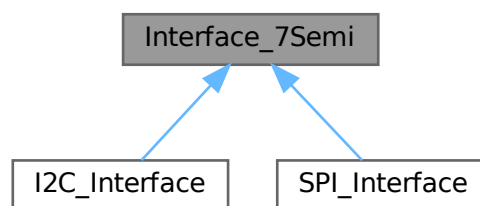
The documentation for this struct was generated from the following file:

- [src/ICM20948\\_platform.h](#)

## 6.8 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)

## 6.8.1 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](#).

## 6.8.2 Constructor & Destructor Documentation

### 6.8.2.1 ~Interface\_7Semi()

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

Definition at line 18 of file [7Semi\\_Interface.h](#).

## 6.8.3 Member Function Documentation

### 6.8.3.1 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](#).

### 6.8.3.2 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 [I2C\\_Interface](#), and [SPI\\_Interface](#).

### 6.8.3.3 delay\_us()

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

Delay wrapper

Definition at line 82 of file [7Semi\\_Interface.h](#).

### 6.8.3.4 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](#).

### 6.8.3.5 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](#).

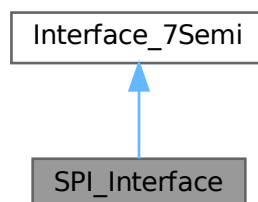
The documentation for this class was generated from the following file:

- [src/7Semi\\_Interface.h](#)

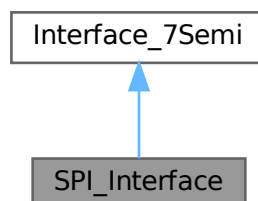
## 6.9 SPI\_Interface Class Reference

```
#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)

## 6.9.1 Detailed Description

7Semi SPI Interface Implementation

Definition at line 7 of file [7Semi\\_SPI\\_Interface.h](#).

## 6.9.2 Member Function Documentation

### 6.9.2.1 [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](#).



### 6.9.2.2 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](#).

### 6.9.2.3 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](#).

### 6.9.2.4 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](#).

## 6.9.3 Member Data Documentation

### 6.9.3.1 cs

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

Definition at line 12 of file [7Semi\\_SPI\\_Interface.h](#).

### 6.9.3.2 speed

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

Definition at line 13 of file [7Semi\\_SPI\\_Interface.h](#).

### 6.9.3.3 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 7

## File Documentation

### 7.1 examples/admin\_pwm/admin\_pwm.ino File Reference

### 7.2 admin\_pwm.ino

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

```
00001 //
00002 // ADMIN SIMPLE - Solo SCAN y TOGGLE RELAY
00003 // Version minimalista para control basico I2C
00004 //
00005
00006 #include <Wire.h>
00007 #include <DevLab_ICM20948.h>
00008 // Configuracion I2C - Wire1 con GPIO2 (SDA) y GPIO3 (SCL)
00009 #define WIRE Wire
00010 #define I2C_SDA 6
00011 #define I2C_SCL 7
00012 #define I2C_FREQ 100000 // 100 kHz - SEGURO para inicialización (luego se puede subir a 400kHz)
00013 #define ICM_ADDR 0x69
00014 //
00015 //          COMANDOS CRÍTICOS DE SEGURIDAD (0xA0-0xAF)
00016 //
00017 #define CMD_RELAY_OFF      0xA0 // Apagar relé PA2 y PB5
00018 #define CMD_RELAY_ON      0xA1 // Encender relé PA2 y PB5
00019 #define CMD_RELAY_TOGGLE  0xA6 // Pulso/Disparo relé PA2 y PB5
00020
00021 //
00022 //          COMANDOS NEOPIXEL
00023 //
00024 #define CMD_RED            0x02 // NeoPixels rojos
00025 #define CMD_GREEN         0x03 // NeoPixels verdes
00026 #define CMD_BLUE          0x04 // NeoPixels azules
00027 #define CMD_OFF           0x05 // NeoPixels apagados + PWM OFF
00028 #define CMD_WHITE         0x08 // NeoPixels blancos
00029
00030 //
00031 //          COMANDOS PWM / BUZZER
00032 //
00033 #define CMD_PWM_OFF        0x20 // PWM OFF - Silencio
00034 #define CMD_PWM_25         0x21 // PWM 25% - Tono grave 200Hz
00035 #define CMD_PWM_50         0x22 // PWM 50% - Tono medio 500Hz
00036 #define CMD_PWM_75         0x23 // PWM 75% - Tono agudo 1000Hz
00037 #define CMD_PWM_100        0x24 // PWM 100% - Tono muy agudo 2000Hz
00038
00039 // Comandos deshabilitados - Causan problemas de estabilidad I2C
00040 /*
00041 // Notas musicales
00042 #define CMD_TONE_DO        0x25 // Tono DO (261Hz)
00043 #define CMD_TONE_RE        0x26 // Tono RE (294Hz)
00044 #define CMD_TONE_MI        0x27 // Tono MI (330Hz)
00045 #define CMD_TONE_FA        0x28 // Tono FA (349Hz)
00046 #define CMD_TONE_SOL       0x29 // Tono SOL (392Hz)
00047 #define CMD_TONE_LA        0x2A // Tono LA (440Hz)
00048 #define CMD_TONE_SI        0x2B // Tono SI (494Hz)
00049
```

```

00050 // Alertas del sistema
00051 #define CMD_SUCCESS          0x2C    // Success - Tono positivo 800Hz
00052 #define CMD_OK               0x2D    // OK - Confirmación 1000Hz
00053 #define CMD_WARNING          0x2E    // Warning - Advertencia 1200Hz
00054 #define CMD_ALERT            0x2F    // Alert - Alerta crítica 1500Hz
00055 */
00056
00057 //
00058 //          COMANDOS LECTURA DIGITAL
00059 //
00060 #define CMD_PA4_DIGITAL      0x07    // Leer PA0 como entrada digital
00061 #define RESP_PA4_DIGITAL     0x09    // Respuesta PA4 digital
00062
00063 //
00064 //          COMANDOS ADC 12-bit (PA1 interno del slave)
00065 //
00066 #define CMD_ADC_PA1_HSB      0xD8    // Leer ADC - bits 11-8 (nibble alto)
00067 #define CMD_ADC_PA1_LSB      0xD9    // Leer ADC - bits 7-0
00068
00069 //
00070 //          COMANDOS I2C ADDRESS MANAGEMENT
00071 //
00072 #define CMD_SET_I2C_ADDR      0x3D    // Establecer nueva dirección I2C (se guarda en Flash)
00073 #define CMD_RESET_FACTORY     0x3E    // Reset factory (usar UID por defecto)
00074 #define CMD_GET_I2C_STATUS    0x3F    // Obtener estado I2C (Flash vs UID)
00075 #define CMD_SAVE_COLOR        0x46    // Guardar color por slot (1..3 en admin, 0..2 en esclavo)
00076
00077 #define RESP_I2C_ADDR_SET      0x0D    // Nueva dirección I2C establecida
00078 #define RESP_FACTORY_RESET     0x0E    // Reset factory ejecutado (usar UID)
00079 #define RESP_I2C_FROM_FLASH    0x0F    // I2C usando dirección desde Flash
00080 #define RESP_I2C_FROM_UID      0x0A    // I2C usando dirección desde UID
00081 #define RESP_COLOR_STAGE_READY 0x15    // Etapa de save_color lista
00082 #define RESP_COLOR_SAVED       0x16    // Color guardado en Flash
00083
00084 // Variables globales
00085 uint8_t found_devices[128];
00086 uint8_t device_count = 0;
00087 bool scan_loop_active = false; // Bandera para scan continuo
00088 uint32_t last_scan_time = 0;   // Timestamp del último scan
00089 uint32_t scan_interval = 1000; // Intervalo entre scans (1 segundo)
00090 bool scan_in_progress = false; // Bandera para evitar scans simultáneos
00091 #define DEVICE_DELAY 30         // Delay óptimo entre dispositivos (ms)
00092 struct DeviceError {
00093     uint8_t address;
00094     uint32_t error_count;
00095     uint32_t success_count;
00096     uint32_t last_error_time;
00097 };
00098 DeviceError device_errors[128];
00099 int device_error_count = 0;
00100
00101 // Test mode
00102 bool test_mode_active = false;
00103 uint32_t test_interval = 100; // Intervalo de test (100ms por defecto)
00104 uint32_t test_start_time = 0;
00105 uint32_t test_duration = 0;
00106
00107 // PA0 Digital reading stats
00108 struct DigitalReadStats {
00109     uint8_t address;
00110     uint32_t read_count;
00111     uint32_t fail_count;
00112     uint8_t last_state; // 0=LOW, 1=HIGH
00113     uint32_t last_read_time;
00114 };
00115 DigitalReadStats digital_stats[128];
00116 int digital_stats_count = 0;
00117
00118 // Test mode para lecturas digitales
00119 bool read_test_active = false;
00120 uint32_t read_test_interval = 100;
00121 uint32_t read_test_start_time = 0;
00122 uint32_t read_test_duration = 0;
00123
00124 // Flag para reimprimir menú cuando se conecta Serial
00125 bool menu_shown = false;
00126 uint32_t last_serial_check = 0;
00127 bool i2c_safe_mode = false; // Modo seguro si I2C está bloqueado
00128
00129 DevLab_ICM20948 imu;
00130
00131 void setup() {
00132     //
00133     // PASO 1: INICIALIZAR USB/CDC PRIMERO (CRÍTICO EN RP2040/RP2350)
00134     //
00135     Serial.begin(115200);
00136

```

```

00137  #if defined(ARDUINO_ARCH_RP2040)
00138      // RP2040/RP2350: Esperar a que USB enumere ANTES de tocar I2C
00139      // Esto evita que I2C bloquee IRQs y mate el CDC
00140      uint32_t t0 = millis();
00141      while (!Serial && millis() - t0 < 2000) {
00142          delay(10);
00143      }
00144      delay(200); // Margen extra de seguridad
00145
00146      Serial.println("\n[RP2040/RP2350] USB CDC listo");
00147  #else
00148      delay(500);
00149  #endif
00150
00151  Serial.println("\n\n");
00152  Serial.println("");
00153  Serial.println("    ADMIN PWM - I2C Control          ");
00154  Serial.println("    Scan + Relay + PWM/Buzzer          ");
00155  Serial.println("");
00156
00157  //
00158  // PASO 2: AHORA SÍ INICIALIZAR I2C (USB ya está vivo)
00159  //
00160  Serial.println("\n[INFO] Inicializando I2C...");
00161
00162  // Verificar que SDA/SCL no estén bloqueados ANTES de WIRE.begin()
00163  Serial.println("[CHECK] Verificando bus I2C...");
00164  pinMode(I2C_SDA, INPUT_PULLUP);
00165  pinMode(I2C_SCL, INPUT_PULLUP);
00166  delay(10);
00167
00168  bool sda_ok = digitalRead(I2C_SDA);
00169  bool scl_ok = digitalRead(I2C_SCL);
00170
00171  Serial.printf("    SDA (GPIO%d): %s\n", I2C_SDA, sda_ok ? "HIGH " : "LOW  BLOQUEADO");
00172  Serial.printf("    SCL (GPIO%d): %s\n", I2C_SCL, scl_ok ? "HIGH " : "LOW  BLOQUEADO");
00173
00174  if (!sda_ok || !scl_ok) {
00175      Serial.println("\n[ERROR] Bus I2C bloqueado!");
00176      Serial.println("CAUSA: Dispositivo esclavo colgado o sin pull-ups");
00177      Serial.println("SOLUCION:");
00178      Serial.println("  1. Desconecta todos los dispositivos I2C");
00179      Serial.println("  2. Verifica resistencias pull-up (4.7k ohm)");
00180      Serial.println("  3. Resetea el RP2350");
00181      Serial.println("\n[MODO SEGURO] I2C NO inicializado");
00182      i2c_safe_mode = true;
00183      // NO return - continuar para procesar comandos seriales
00184  } else {
00185      Serial.println("[OK] Bus I2C libre");
00186      i2c_safe_mode = false;
00187  }
00188
00189  // Solo inicializar I2C si el bus está libre
00190  if (!i2c_safe_mode) {
00191      // Inicializar I2C con Wire1
00192      // RP2040 / RP2350: configurar pines ANTES de begin()
00193      // WIRE.setSDA(I2C_SDA);
00194      // WIRE.setSCL(I2C_SCL);
00195      WIRE.begin(I2C_SDA, I2C_SCL);
00196      WIRE.setClock(I2C_FREQ);
00197
00198      if(!imu.beginI2C(ICM_ADDR, WIRE, I2C_FREQ)){
00199          Serial.println(F("ERROR: beginI2C() failed"));
00200          while (1) delay(200);
00201      }
00202      if(!imu.auxMasterEnable(0x07)){
00203          Serial.println(F("ERROR: enableAuxMaster() failed"));
00204          while (1) delay(200);
00205      }
00206      // Deshabilitado temporalmente: bloqueaba el arranque (while(1)) si el
00207      // esclavo en 0x2B no respondia al registro 0x31 (valores eran para el
00208      // magnetometro AK09916, no para el esclavo de relay/PWM).
00209      /*
00210      if(!imu.auxWriteByte(0x2B, 0x31, 0x08)){
00211          Serial.println(F("ERROR: auxwritebyte() failed"));
00212          while (1) delay(200);
00213      }
00214      */
00215      Serial.println("[OK] I2C inicializado sin bloquear USB");
00216      Serial.printf("    Wire1 - SDA:GPIO%d SCL:GPIO%d @ %d kHz\n", I2C_SDA, I2C_SCL, I2C_FREQ/1000);
00217
00218  } // Fin de if (!i2c_safe_mode)
00219  bool mag_ok = imu.initMag();
00220  Serial.printf("[TEST] initMag() = %s\n", mag_ok ? "OK" : "FAIL");
00221
00222  Serial.println("\nComandos disponibles:");
00223  Serial.println("  s          - Scan dispositivos I2C");

```

```

00224 Serial.println(" loop           - Scan continuo (cada 1 seg)");
00225 Serial.println(" stop           - Detener scan continuo");
00226 Serial.println(" t XX          - Pulso relay (XX = addr hex)");
00227 Serial.println(" on XX          - Encender relay");
00228 Serial.println(" off XX         - Apagar relay");
00229 Serial.println(" help          - Mostrar ayuda");
00230 Serial.println("\nEjemplos:");
00231 Serial.println(" s              (scan de dispositivos)");
00232 Serial.println(" loop          (scan continuo)");
00233 Serial.println(" stop          (detener scan)");
00234 Serial.println(" t 32          (toggle relay en 0x20)");
00235 Serial.println(" off 32        (apagar relay en 0x20)");
00236 Serial.println("\n");
00237 }
00238
00239 void loop() {
00240     // Reimprimir menú si se reconecta el Serial Monitor
00241     #if defined(ARDUINO_ARCH_RP2040)
00242         if (Serial && !menu_shown && millis() - last_serial_check > 1000) {
00243             showMenu();
00244             menu_shown = true;
00245             last_serial_check = millis();
00246         }
00247         if (!Serial) {
00248             menu_shown = false;
00249             last_serial_check = millis();
00250         }
00251     #endif
00252
00253     // Procesar comandos desde Serial
00254     if (Serial.available()) {
00255         String cmd = Serial.readStringUntil('\n');
00256         cmd.trim();
00257         cmd.toLowerCase();
00258
00259         if (cmd.length() > 0) {
00260             processCommand(cmd);
00261         }
00262     }
00263
00264     // Ejecutar scan continuo si está activo
00265     if (scan_loop_active && !scan_in_progress) {
00266         if (millis() - last_scan_time >= scan_interval) {
00267             scanDevices();
00268             last_scan_time = millis();
00269         }
00270     }
00271
00272     // Ejecutar test asíncrono si está activo
00273     if (test_mode_active) {
00274         if (millis() - last_scan_time >= test_interval) {
00275             testDevices();
00276             last_scan_time = millis();
00277         }
00278         // Verificar si el test debe terminar
00279         if (test_duration > 0 && (millis() - test_start_time >= test_duration)) {
00280             stopTest();
00281             showTestResults();
00282         }
00283     }
00284
00285     // Ejecutar test de lectura digital si está activo
00286     if (read_test_active) {
00287         if (millis() - last_scan_time >= read_test_interval) {
00288             testReadDigital();
00289             last_scan_time = millis();
00290         }
00291         // Verificar si el test debe terminar
00292         if (read_test_duration > 0 && (millis() - read_test_start_time >= read_test_duration)) {
00293             stopReadTest();
00294         }
00295     }
00296 }
00297
00298 void processCommand(String cmd) {
00299     if (cmd == "s" || cmd == "scan") {
00300         scanDevices();
00301     } else if (cmd == "loop") {
00302         startScanLoop();
00303     } else if (cmd == "stop") {
00304         stopScanLoop();
00305     } else if (cmd == "menu" || cmd == "m") {
00306         showMenu();
00307     } else if (cmd.startsWith("test ")) {
00308         startTest(cmd);
00309     } else if (cmd == "stoptest") {
00310         stopTest();

```

```

00311     showTestResults();
00312 } else if (cmd == "errors") {
00313     showTestResults();
00314 } else if (cmd.startsWith("adc ")) {
00315     readADC(cmd);
00316 } else if (cmd.startsWith("read ")) {
00317     readDigital(cmd);
00318 } else if (cmd == "dstats") {
00319     showDigitalStats();
00320 } else if (cmd.startsWith("readtest ")) {
00321     startReadTest(cmd);
00322 } else if (cmd == "stopread") {
00323     stopReadTest();
00324 } else if (cmd.startsWith("t ")) {
00325     toggleRelay(cmd);
00326 } else if (cmd.startsWith("on ")) {
00327     relayOn(cmd);
00328 } else if (cmd.startsWith("off ")) {
00329     relayOff(cmd);
00330 } else if (cmd.startsWith("pwm ")) {
00331     pwmCommand(cmd);
00332 }
00333 // Comandos deshabilitados
00334 } else if (cmd.startsWith("note ")) {
00335     playNote(cmd);
00336 } else if (cmd.startsWith("alert ")) {
00337     playAlert(cmd);
00338 }
00339 } else if (cmd == "silence" || cmd == "pwmoff") {
00340     pwmOff();
00341 } else if (cmd.startsWith("neo ")) {
00342     neoCommand(cmd);
00343 } else if (cmd.startsWith("red ")) {
00344     neoRed(cmd);
00345 } else if (cmd.startsWith("green ")) {
00346     neoGreen(cmd);
00347 } else if (cmd.startsWith("blue ")) {
00348     neoBlue(cmd);
00349 } else if (cmd.startsWith("white ")) {
00350     neoWhite(cmd);
00351 } else if (cmd == "neoff") {
00352     neoOff();
00353 } else if (cmd.startsWith("ch ")) {
00354     changeI2CAddress(cmd);
00355 } else if (cmd.startsWith("save_color ")) {
00356     saveColorPreset(cmd);
00357 } else if (cmd == "help" || cmd == "h") {
00358     showHelp();
00359 } else {
00360     Serial.println("ERROR: Comando desconocido. Usa 'help' o 'menu' para ver comandos.");
00361 }
00362 }
00363
00364 void showMenu() {
00365     Serial.println("\n\n");
00366     Serial.println("");
00367     Serial.println("    ADMIN PWM - I2C Control          ");
00368     Serial.println("    Scan + Relay + PWM/Buzzer      ");
00369     Serial.println("");
00370     Serial.println("[OK] I2C inicializado sin bloquear USB");
00371     Serial.printf("    SDA:GPIO%d SCL:GPIO%d @ %d kHz\n", I2C_SDA, I2C_SCL, I2C_FREQ/1000);
00372     Serial.println("\nComandos disponibles:");
00373     Serial.println("  s / menu          - Scan dispositivos / mostrar menu");
00374     Serial.println("  loop              - Scan continuo (cada 1 seg)");
00375     Serial.println("  stop              - Detener scan continuo");
00376     Serial.println("  t XX              - Pulso relay (XX = addr hex)");
00377     Serial.println("  on XX             - Encender relay");
00378     Serial.println("  off XX            - Apagar relay");
00379     Serial.println("  pwm XX NIVEL      - PWM (25, 50, 75, 100, off)");
00380     Serial.println("  silence           - Apagar PWM en todos");
00381     Serial.println("  neo XX COLOR      - NeoPixel (red/green/blue/white/off o 1/2/3/4/0)");
00382     Serial.println("  ch XX YY          - Cambiar dir I2C (XX=old, YY=new)");
00383     Serial.println("  save_color A P R G B - Guardar color por slot (1..3)");
00384     Serial.println("  help              - Mostrar ayuda completa");
00385     Serial.println("\nEjemplos:");
00386     Serial.println("  s                  (scan de dispositivos)");
00387     Serial.println("  menu               (mostrar este menu)");
00388     Serial.println("  pwm 20 50          (PWM 50% en 0x20)");
00389     Serial.println("  silence            (apagar PWM en todos)");
00390     Serial.println("  neo 20 1           (NeoPixel rojo en 0x20)");
00391     Serial.println("  save_color 20 1 0x00 0xff 0x60");
00392     Serial.println("");
00393 }
00394
00395 void scanDevices() {
00396     // Verificar si estamos en modo seguro
00397     if (i2c_safe_mode) {

```

```

00398     Serial.println("[ERROR] I2C en MODO SEGURO - comando no disponible");
00399     Serial.println("Desconecta dispositivos I2C y resetea el dispositivo");
00400     return;
00401 }
00402
00403 // Evitar scans simultáneos
00404 if (scan_in_progress) {
00405     Serial.println(" Scan ya en progreso, esperando...");
00406     return;
00407 }
00408
00409 scan_in_progress = true;
00410 Serial.println("\n SCAN I2C + TOGGLE ");
00411
00412 device_count = 0;
00413 memset(found_devices, 0, sizeof(found_devices));
00414
00415 WIRE.setTimeout(50); // Timeout corto para velocidad
00416
00417 for (uint8_t addr = 0x08; addr <= 0x77; addr++) {
00418     /*
00419     WIRE.beginTransaction(addr);
00420     uint8_t error = WIRE.endTransmission();
00421
00422     if (error == 0) {
00423         found_devices[device_count++] = addr;
00424         Serial.printf(" [%02d] 0x%02X (%3d) → ", device_count, addr, addr);
00425
00426         // Enviar toggle automático SIN esperar respuesta
00427         if (sendCommandFast(addr, CMD_RELAY_TOGGLE)) {
00428             Serial.println("TOGGLE ");
00429         } else {
00430             Serial.println("TOGGLE ");
00431         }
00432
00433         delay(DEVICE_DELAY); // Delay optimizado: 30ms
00434     }
00435     */
00436     uint8_t dummy;
00437     // auxReadByte retorna true si el dispositivo hace ACK
00438     if (imu.auxReadByte(addr, 0x00, dummy)) {
00439         found_devices[device_count++] = addr;
00440         Serial.printf(" [%02d] 0x%02X (%3d) → ", device_count, addr, addr);
00441
00442         if (imu.auxWriteCommand(addr, CMD_RELAY_TOGGLE)) {
00443             Serial.println("TOGGLE ");
00444         } else {
00445             Serial.println("TOGGLE ");
00446         }
00447         delay(DEVICE_DELAY);
00448     }
00449 }
00450
00451 Serial.println("");
00452 Serial.printf("Total: %d dispositivos\n\n", device_count);
00453
00454 if (device_count == 0) {
00455     Serial.println(" No se encontraron dispositivos I2C");
00456     Serial.println(" Verifica conexiones y pull-ups\n");
00457 }
00458
00459 scan_in_progress = false;
00460 }
00461
00462
00463
00464 void toggleRelay(String cmd) {
00465     // Verificar si estamos en modo seguro
00466     if (i2c_safe_mode) {
00467         Serial.println("[ERROR] I2C en MODO SEGURO - comando no disponible");
00468         return;
00469     }
00470
00471     String addr_str = cmd.substring(2);
00472     addr_str.trim();
00473
00474     uint8_t address = parseHex(addr_str);
00475     if (address == 0) {
00476         Serial.println("ERROR: Direccion invalida");
00477         return;
00478     }
00479
00480     if (sendCommand(address, CMD_RELAY_TOGGLE)) {
00481         Serial.printf("[OK] Pulso relay en 0x%02X (10ms)\n", address);
00482     } else {
00483         Serial.printf("[FAIL] No se pudo enviar pulso a 0x%02X\n", address);
00484     }

```



```

00485 }
00486
00487 void relayOn(String cmd) {
00488     String addr_str = cmd.substring(3);
00489     addr_str.trim();
00490
00491     uint8_t address = parseHex(addr_str);
00492     if (address == 0) {
00493         Serial.println("ERROR: Direccion invalida");
00494         return;
00495     }
00496
00497     if (sendCommand(address, CMD_RELAY_ON)) {
00498         Serial.printf("[OK] Relay ON en 0x%02X\n", address);
00499     } else {
00500         Serial.printf("[FAIL] No se pudo encender relay en 0x%02X\n", address);
00501     }
00502 }
00503
00504 void relayOff(String cmd) {
00505     String addr_str = cmd.substring(4);
00506     addr_str.trim();
00507
00508     uint8_t address = parseHex(addr_str);
00509     if (address == 0) {
00510         Serial.println("ERROR: Direccion invalida");
00511         return;
00512     }
00513
00514     if (sendCommand(address, CMD_RELAY_OFF)) {
00515         Serial.printf("[OK] Relay OFF en 0x%02X\n", address);
00516     } else {
00517         Serial.printf("[FAIL] No se pudo apagar relay en 0x%02X\n", address);
00518     }
00519 }
00520
00521 //
00522 //          LECTURA DIGITAL PA0
00523 //
00524
00525 void readADC(String cmd) {
00526     String addr_str = cmd.substring(4);
00527     addr_str.trim();
00528
00529     uint8_t address = parseHex(addr_str);
00530     if (address == 0) {
00531         Serial.println("ERROR: Direccion invalida");
00532         return;
00533     }
00534
00535     uint16_t raw = 0;
00536     if (readADC12(address, raw)) {
00537         float voltage = raw * (3.3f / 4095.0f);
00538         Serial.printf("[OK] 0x%02X - ADC: %u (%.3f V)\n", address, raw, voltage);
00539     } else {
00540         Serial.printf("[FAIL] No se pudo leer ADC en 0x%02X\n", address);
00541     }
00542 }
00543
00544 void readDigital(String cmd) {
00545     String addr_str = cmd.substring(5);
00546     addr_str.trim();
00547
00548     uint8_t address = parseHex(addr_str);
00549     if (address == 0) {
00550         Serial.println("ERROR: Direccion invalida");
00551         return;
00552     }
00553
00554     uint8_t state = 0;
00555     bool success = readPA0Digital(address, &state);
00556
00557     if (success) {
00558         Serial.printf("[OK] 0x%02X - PA0 Digital: %s\n", address, state ? "HIGH" : "LOW");
00559     } else {
00560         Serial.printf("[FAIL] No se pudo leer PA0 digital en 0x%02X\n", address);
00561     }
00562 }
00563
00564 bool readPA0Digital(uint8_t address, uint8_t* state) {
00565     // Enviar comando de lectura PA0 digital
00566     WIRE.beginTransmission(address);
00567     WIRE.write(CMD_PA4_DIGITAL);
00568     uint8_t error = WIRE.endTransmission();
00569
00570     if (error != 0) {
00571         updateDigitalStats(address, false, 0);

```

```

00572     return false;
00573 }
00574
00575 // Esperar procesamiento
00576 delay(5);
00577
00578 // Leer respuesta
00579 WIRE.setTimeout(50);
00580 uint8_t bytesRead = WIRE.requestFrom(address, (uint8_t)1);
00581
00582 if (bytesRead == 1) {
00583     uint8_t response = WIRE.read();
00584
00585     // El estado digital está en el nibble alto (bit 7-4)
00586     // 0xF0 = HIGH, 0x00 = LOW
00587     *state = (response & 0xF0) ? 1 : 0;
00588
00589     updateDigitalStats(address, true, *state);
00590     return true;
00591 }
00592
00593 updateDigitalStats(address, false, 0);
00594 return false;
00595 }
00596
00597 bool readADC12(uint8_t address, uint16_t &raw) {
00598     uint8_t hi = 0, lo = 0;
00599
00600     // El slave necesita transacciones SEPARADAS: write (comando) → STOP,
00601     // slave procesa + convierte ADC (~20ms), luego read independiente.
00602     // auxReadByte hace write+RSTART+read combinado → el slave no alcanza
00603     // a llamar HAL_I2C_Slave_Transmit → NACK.
00604     // auxReadResponse hace solo [START, addr+R, 1byte, STOP].
00605
00606     if (!imu.auxWriteCommand(address, CMD_ADC_PA1_HSB)) return false;
00607     delay(25); // slave: main loop + conversión ADC (HAL_ADC_PollForConversion, max 20ms)
00608     if (!imu.auxReadResponse(address, hi)) return false;
00609
00610     // LSB: el slave usa valor cacheado (adc_last_read < 50ms → mismo sample que HSB)
00611     if (!imu.auxWriteCommand(address, CMD_ADC_PA1_LSB)) return false;
00612     delay(5); // sin conversión nueva, solo tiempo de main loop
00613     if (!imu.auxReadResponse(address, lo)) return false;
00614
00615     raw = ((uint16_t)(hi & 0x0F) << 8) | lo; // 0-4095
00616     return true;
00617 }
00618
00619 void updateDigitalStats(uint8_t address, bool success, uint8_t state) {
00620     // Buscar dispositivo en el array
00621     int index = -1;
00622     for (int i = 0; i < digital_stats_count; i++) {
00623         if (digital_stats[i].address == address) {
00624             index = i;
00625             break;
00626         }
00627     }
00628
00629     // Si no existe, agregarlo
00630     if (index == -1 && digital_stats_count < 128) {
00631         index = digital_stats_count++;
00632         digital_stats[index].address = address;
00633         digital_stats[index].read_count = 0;
00634         digital_stats[index].fail_count = 0;
00635         digital_stats[index].last_state = 0;
00636         digital_stats[index].last_read_time = 0;
00637     }
00638
00639     if (index >= 0) {
00640         digital_stats[index].read_count++;
00641         if (!success) {
00642             digital_stats[index].fail_count++;
00643         } else {
00644             digital_stats[index].last_state = state;
00645         }
00646         digital_stats[index].last_read_time = millis();
00647     }
00648 }
00649
00650 void showDigitalStats() {
00651     if (digital_stats_count == 0) {
00652         Serial.println("No hay estadísticas de lectura digital");
00653         return;
00654     }
00655
00656     Serial.println("\n");
00657     Serial.println("    ESTADÍSTICAS DE LECTURA DIGITAL PA0    ");
00658     Serial.println("");

```

```

00659 Serial.println(" Addr  Lecturas  Fallos  Tasa OK  Estado  Tiempo ");
00660 Serial.println("");
00661
00662 for (int i = 0; i < digital_stats_count; i++) {
00663     DigitalReadStats* stats = &digital_stats[i];
00664
00665     float success_rate = 0.0;
00666     if (stats->read_count > 0) {
00667         success_rate = ((float)(stats->read_count - stats->fail_count) / stats->read_count) * 100.0;
00668     }
00669
00670     uint32_t time_since = (millis() - stats->last_read_time) / 1000; // segundos
00671
00672     Serial.printf(" 0x%02X %8lu %6lu %6.1f%% %4s %4lus  \n",
00673         stats->address,
00674         stats->read_count,
00675         stats->fail_count,
00676         success_rate,
00677         stats->last_state ? "HIGH" : "LOW",
00678         time_since);
00679 }
00680
00681 Serial.println("");
00682 }
00683
00684 //
00685 //          TEST DE LECTURA DIGITAL CONTINUA
00686 //
00687
00688 void startReadTest(String cmd) {
00689     // Formato: "readtest <intervalo_ms> <duracion_ms>"
00690     // Ejemplo: "readtest 100 10000" = leer cada 100ms durante 10 segundos
00691     cmd.trim();
00692     int spacel = cmd.indexOf(' ', 9);
00693
00694     if (spacel == -1) {
00695         Serial.println("ERROR: Formato incorrecto");
00696         Serial.println("Uso: readtest <intervalo_ms> <duracion_ms>");
00697         Serial.println("Ejemplo: readtest 100 10000 (leer cada 100ms por 10 seg)");
00698         return;
00699     }
00700
00701     String interval_str = cmd.substring(9, spacel);
00702     String duration_str = cmd.substring(spacel + 1);
00703
00704     read_test_interval = interval_str.toInt();
00705     read_test_duration = duration_str.toInt();
00706
00707     // Validar rango
00708     if (read_test_interval < 30) read_test_interval = 30;
00709     if (read_test_interval > 1000) read_test_interval = 1000;
00710     if (read_test_duration < 1000) read_test_duration = 1000;
00711
00712     // Reset contadores de lectura digital
00713     digital_stats_count = 0;
00714     memset(digital_stats, 0, sizeof(digital_stats));
00715
00716     read_test_active = true;
00717     read_test_start_time = millis();
00718     last_scan_time = millis();
00719
00720     Serial.println("\n[READ TEST MODE ACTIVADO]");
00721     Serial.printf("Intervalo: %lu ms\n", read_test_interval);
00722     Serial.printf("Duración: %lu ms (%.1f seg)\n", read_test_duration, read_test_duration / 1000.0);
00723     Serial.println("Dispositivos a leer: ");
00724     for (int i = 0; i < device_count; i++) {
00725         Serial.printf(" 0x%02X\n", found_devices[i]);
00726     }
00727     Serial.println("Usa 'stopread' para detener antes\n");
00728 }
00729
00730 void stopReadTest() {
00731     read_test_active = false;
00732     uint32_t elapsed = millis() - read_test_start_time;
00733     Serial.println("\n[READ TEST MODE DETENIDO]");
00734     Serial.printf("Tiempo transcurrido: %.2f segundos\n", elapsed / 1000.0);
00735     showDigitalStats();
00736 }
00737
00738 void testReadDigital() {
00739     WIRE.setTimeout(50);
00740
00741     for (int i = 0; i < device_count; i++) {
00742         uint8_t addr = found_devices[i];
00743         uint8_t state = 0;
00744         readPA0Digital(addr, &state);
00745         delay(read_test_interval / device_count); // Distribuir tiempo entre dispositivos

```

```

00746     }
00747 }
00748
00749 bool sendCommand(uint8_t address, uint8_t command) {
00750     /*WIRE.beginTransaction(address);
00751     WIRE.write(command);
00752     uint8_t error = WIRE.endTransmission();
00753
00754     if (error != 0) return false;
00755
00756     // Esperar respuesta con timeout más generoso
00757     delay(20); // Dar tiempo al esclavo para procesar
00758     WIRE.setTimeout(100);
00759     uint8_t bytesRead = WIRE.requestFrom(address, (uint8_t)1);
00760     if (bytesRead == 1) {
00761         WIRE.read(); // Leer y descartar respuesta
00762         return true;
00763     }
00764
00765     return false;*/
00766     return imu.auxWriteCommand(address, command);
00767 }
00768
00769 // Función para leer respuesta del dispositivo
00770 uint8_t readResponse(uint8_t address) {
00771     delay(10); // Pequeño delay para que el dispositivo prepare respuesta
00772     WIRE.setTimeout(100);
00773     uint8_t bytesRead = WIRE.requestFrom(address, (uint8_t)1);
00774     if (bytesRead == 1) {
00775         return WIRE.read();
00776     }
00777     return 0xFF; // Error: no se recibió respuesta
00778 }
00779
00780 // Versión rápida: enviar comando SIN esperar respuesta
00781 bool sendCommandFast(uint8_t address, uint8_t command) {
00782     WIRE.beginTransaction(address);
00783     WIRE.write(command);
00784     uint8_t error = WIRE.endTransmission();
00785
00786     // Actualizar contador de errores
00787     updateDeviceError(address, error == 0);
00788
00789     return (error == 0); // Solo verificar que se envió
00790 }
00791
00792 void updateDeviceError(uint8_t address, bool success) {
00793     // Buscar dispositivo en el array
00794     int index = -1;
00795     for (int i = 0; i < device_error_count; i++) {
00796         if (device_errors[i].address == address) {
00797             index = i;
00798             break;
00799         }
00800     }
00801
00802     // Si no existe, agregarlo
00803     if (index == -1 && device_error_count < 128) {
00804         index = device_error_count++;
00805         device_errors[index].address = address;
00806         device_errors[index].error_count = 0;
00807         device_errors[index].success_count = 0;
00808         device_errors[index].last_error_time = 0;
00809     }
00810
00811     // Actualizar contadores
00812     if (index != -1) {
00813         if (success) {
00814             device_errors[index].success_count++;
00815         } else {
00816             device_errors[index].error_count++;
00817             device_errors[index].last_error_time = millis();
00818         }
00819     }
00820 }
00821
00822 void startTest(String cmd) {
00823     // Formato: "test <intervalo_ms> <duracion_ms>"
00824     // Ejemplo: "test 50 5000" = test cada 50ms durante 5 segundos
00825     cmd.trim();
00826     int space1 = cmd.indexOf(' ', 5);
00827
00828     if (space1 == -1) {
00829         Serial.println("ERROR: Formato incorrecto");
00830         Serial.println("Uso: test <intervalo_ms> <duracion_ms>");
00831         Serial.println("Ejemplo: test 50 5000 (test cada 50ms por 5 seg)");
00832         return;

```

```

00833     }
00834
00835     String interval_str = cmd.substring(5, space1);
00836     String duration_str = cmd.substring(space1 + 1);
00837
00838     test_interval = interval_str.toInt();
00839     test_duration = duration_str.toInt();
00840
00841     // Validar rango
00842     if (test_interval < 30) test_interval = 30;
00843     if (test_interval > 1000) test_interval = 1000;
00844     if (test_duration < 1000) test_duration = 1000;
00845
00846     // Reset contadores
00847     device_error_count = 0;
00848     memset(device_errors, 0, sizeof(device_errors));
00849
00850     test_mode_active = true;
00851     test_start_time = millis();
00852     last_scan_time = millis();
00853
00854     Serial.println("\n[TEST MODE ACTIVADO]");
00855     Serial.printf("Intervalo: %lu ms\n", test_interval);
00856     Serial.printf("Duración: %lu ms (%.1f seg)\n", test_duration, test_duration / 1000.0);
00857     Serial.println("Usa 'stoptest' para detener antes\n");
00858 }
00859
00860 void stopTest() {
00861     test_mode_active = false;
00862     uint32_t elapsed = millis() - test_start_time;
00863     Serial.println("\n[TEST MODE DETENIDO]");
00864     Serial.printf("Tiempo transcurrido: %.2f segundos\n\n", elapsed / 1000.0);
00865 }
00866
00867 void testDevices() {
00868     // Verificar si estamos en modo seguro
00869     if (i2c_safe_mode) {
00870         Serial.println("[ERROR] I2C en MODO SEGURO - comando no disponible");
00871         return;
00872     }
00873
00874     WIRE.setTimeout(50);
00875
00876     for (int i = 0; i < device_count; i++) {
00877         uint8_t addr = found_devices[i];
00878         sendCommandFast(addr, CMD_RELAY_TOGGLE);
00879         delay(test_interval / device_count); // Distribuir tiempo entre dispositivos
00880     }
00881 }
00882
00883 void showTestResults() {
00884     Serial.println("\n");
00885     Serial.println("                RESULTADOS DE TEST I2C                ");
00886     Serial.println("\n");
00887
00888     if (device_error_count == 0) {
00889         Serial.println("No hay datos de test disponibles\n");
00890         return;
00891     }
00892
00893     uint32_t total_success = 0;
00894     uint32_t total_errors = 0;
00895
00896     Serial.println("Addr | Success | Errors | Tasa Éxito | Último Error");
00897     Serial.println("-----|-----|-----|-----|-----");
00898
00899     for (int i = 0; i < device_error_count; i++) {
00900         uint8_t addr = device_errors[i].address;
00901         uint32_t success = device_errors[i].success_count;
00902         uint32_t errors = device_errors[i].error_count;
00903         uint32_t total = success + errors;
00904         float rate = total > 0 ? (success * 100.0 / total) : 0;
00905
00906         total_success += success;
00907         total_errors += errors;
00908
00909         Serial.printf("0x%02X | %7lu | %6lu | %6.2f%% | ",
00910             addr, success, errors, rate);
00911
00912         if (errors > 0) {
00913             uint32_t since_error = (millis() - device_errors[i].last_error_time) / 1000;
00914             Serial.printf("%6lu seg\n", since_error);
00915         } else {
00916             Serial.println("Nunca");
00917         }
00918     }
00919

```

```

00920 Serial.println("-----|-----|-----|-----|-----");
00921 uint32_t grand_total = total_success + total_errors;
00922 float overall_rate = grand_total > 0 ? (total_success * 100.0 / grand_total) : 0;
00923 Serial.printf("TOTAL | %7lu | %6lu | %6.2f%% |\n\n",
00924               total_success, total_errors, overall_rate);
00925 }
00926
00927 uint8_t parseHex(String hex_str) {
00928     hex_str.trim();
00929
00930     // Convertir de hexadecimal
00931     char* endptr;
00932     long addr = strtol(hex_str.c_str(), &endptr, 16);
00933
00934     if (*endptr != '\0' || addr < 0x08 || addr > 0x77) {
00935         return 0;
00936     }
00937
00938     return (uint8_t)addr;
00939 }
00940
00941 bool parseByteAuto(String value_str, uint8_t* out) {
00942     value_str.trim();
00943     char* endptr;
00944     long value = strtol(value_str.c_str(), &endptr, 0);
00945
00946     if (*endptr != '\0' || value < 0 || value > 0xFF) {
00947         return false;
00948     }
00949
00950     *out = (uint8_t)value;
00951     return true;
00952 }
00953
00954 void startScanLoop() {
00955     scan_loop_active = true;
00956     last_scan_time = millis();
00957     Serial.println("\n[OK] Scan continuo ACTIVADO");
00958     Serial.printf("Intervalo: %lu ms\n", scan_interval);
00959     Serial.println("Usa 'stop' para detener\n");
00960 }
00961
00962 void stopScanLoop() {
00963     scan_loop_active = false;
00964     Serial.println("\n[OK] Scan continuo DETENIDO\n");
00965 }
00966
00967 //
00968 //     FUNCIONES PWM / BUZZER
00969 //
00970
00971 void pwmOff() {
00972     Serial.println("\n APAGAR PWM (TODOS) ");
00973     int count = 0;
00974     for (int i = 0; i < device_count; i++) {
00975         uint8_t addr = found_devices[i];
00976         if (sendCommand(addr, CMD_PWM_OFF)) {
00977             Serial.printf(" [OK] 0x%02X - PWM apagado\n", addr);
00978             count++;
00979         } else {
00980             Serial.printf(" [FAIL] 0x%02X\n", addr);
00981         }
00982         delay(10);
00983     }
00984     Serial.printf("\nTotal: %d/%d dispositivos\n", count, device_count);
00985 }
00986
00987 void pwmCommand(String cmd) {
00988     // Formato: "pwm XX NIVEL"
00989     // XX = dirección hex, NIVEL = 25, 50, 75, 100, off
00990     cmd.trim();
00991     int space = cmd.indexOf(' ', 4);
00992
00993     if (space == -1) {
00994         Serial.println("ERROR: Formato incorrecto");
00995         Serial.println("Uso: pwm XX NIVEL");
00996         Serial.println("Ejemplo: pwm 20 50 (PWM 50% en 0x20)");
00997         Serial.println("Niveles: 25, 50, 75, 100, off");
00998         return;
00999     }
01000
01001     String addr_str = cmd.substring(4, space);
01002     String level_str = cmd.substring(space + 1);
01003     addr_str.trim();
01004     level_str.trim();
01005
01006     uint8_t address = parseHex(addr_str);

```

```

01007   if (address == 0) {
01008       Serial.println("ERROR: Direccion invalida");
01009       return;
01010   }
01011
01012   uint8_t pwm_cmd = CMD_PWM_OFF;
01013   String level_name = "OFF";
01014
01015   if (level_str == "off" || level_str == "0") {
01016       pwm_cmd = CMD_PWM_OFF;
01017       level_name = "OFF";
01018   } else if (level_str == "25") {
01019       pwm_cmd = CMD_PWM_25;
01020       level_name = "25% (200Hz)";
01021   } else if (level_str == "50") {
01022       pwm_cmd = CMD_PWM_50;
01023       level_name = "50% (500Hz)";
01024   } else if (level_str == "75") {
01025       pwm_cmd = CMD_PWM_75;
01026       level_name = "75% (1000Hz)";
01027   } else if (level_str == "100") {
01028       pwm_cmd = CMD_PWM_100;
01029       level_name = "100% (2000Hz)";
01030   } else {
01031       Serial.println("ERROR: Nivel invalido. Usa: 25, 50, 75, 100, off");
01032       return;
01033   }
01034
01035   if (sendCommand(address, pwm_cmd)) {
01036       Serial.printf("[OK] PWM %s en 0x%02X\n", level_name.c_str(), address);
01037   } else {
01038       Serial.printf("[FAIL] No se pudo configurar PWM en 0x%02X\n", address);
01039   }
01040 }
01041
01042 //
01043 //      FUNCIONES NEOPIXEL
01044 //
01045
01046 void neoCommand(String cmd) {
01047     // Formato: "neo XX COLOR"
01048     // XX = dirección hex, COLOR = red/green/blue/white/off o 1/2/3/4/0
01049     cmd.trim();
01050     int space = cmd.indexOf(' ', 4);
01051
01052     if (space == -1) {
01053         Serial.println("ERROR: Formato incorrecto");
01054         Serial.println("Uso: neo XX COLOR");
01055         Serial.println("Ejemplo: neo 20 1 (NeoPixel rojo en 0x20)");
01056         Serial.println("Colores: red, green, blue, white, off");
01057         Serial.println("Numeros: 1=red, 2=green, 3=blue, 4=white, 0=off");
01058         return;
01059     }
01060
01061     String addr_str = cmd.substring(4, space);
01062     String color_str = cmd.substring(space + 1);
01063     addr_str.trim();
01064     color_str.trim();
01065     color_str.toLowerCase();
01066
01067     uint8_t address = parseHex(addr_str);
01068     if (address == 0) {
01069         Serial.println("ERROR: Direccion invalida");
01070         return;
01071     }
01072
01073     uint8_t neo_cmd = 0;
01074     String color_name = "";
01075
01076     if (color_str == "red") {
01077         neo_cmd = CMD_RED;
01078         color_name = "ROJO";
01079     } else if (color_str == "1") {
01080         neo_cmd = CMD_RED;
01081         color_name = "ROJO";
01082     } else if (color_str == "green") {
01083         neo_cmd = CMD_GREEN;
01084         color_name = "VERDE";
01085     } else if (color_str == "2") {
01086         neo_cmd = CMD_GREEN;
01087         color_name = "VERDE";
01088     } else if (color_str == "blue") {
01089         neo_cmd = CMD_BLUE;
01090         color_name = "AZUL";
01091     } else if (color_str == "3") {
01092         neo_cmd = CMD_BLUE;
01093         color_name = "AZUL";

```

```

01094 } else if (color_str == "white") {
01095     neo_cmd = CMD_WHITE;
01096     color_name = "BLANCO";
01097 } else if (color_str == "4") {
01098     neo_cmd = CMD_WHITE;
01099     color_name = "BLANCO";
01100 } else if (color_str == "off") {
01101     neo_cmd = CMD_OFF;
01102     color_name = "APAGADO";
01103 } else if (color_str == "0") {
01104     neo_cmd = CMD_OFF;
01105     color_name = "APAGADO";
01106 } else {
01107     Serial.println("ERROR: Color invalido. Usa: red, green, blue, white, off");
01108     Serial.println("      0 numeros: 1=red, 2=green, 3=blue, 4=white, 0=off");
01109     return;
01110 }
01111
01112 if (sendCommand(address, neo_cmd)) {
01113     Serial.printf("[OK] NeoPixel %s en 0x%02X\n", color_name.c_str(), address);
01114 } else {
01115     Serial.printf("[FAIL] No se pudo configurar NeoPixel en 0x%02X\n", address);
01116 }
01117 }
01118
01119 void neoRed(String cmd) {
01120     String addr_str = cmd.substring(4);
01121     addr_str.trim();
01122
01123     uint8_t address = parseHex(addr_str);
01124     if (address == 0) {
01125         Serial.println("ERROR: Direccion invalida");
01126         return;
01127     }
01128
01129     if (sendCommand(address, CMD_RED)) {
01130         Serial.printf("[OK] NeoPixel ROJO en 0x%02X\n", address);
01131     } else {
01132         Serial.printf("[FAIL] No se pudo configurar NeoPixel en 0x%02X\n", address);
01133     }
01134 }
01135
01136 void neoGreen(String cmd) {
01137     String addr_str = cmd.substring(6);
01138     addr_str.trim();
01139
01140     uint8_t address = parseHex(addr_str);
01141     if (address == 0) {
01142         Serial.println("ERROR: Direccion invalida");
01143         return;
01144     }
01145
01146     if (sendCommand(address, CMD_GREEN)) {
01147         Serial.printf("[OK] NeoPixel VERDE en 0x%02X\n", address);
01148     } else {
01149         Serial.printf("[FAIL] No se pudo configurar NeoPixel en 0x%02X\n", address);
01150     }
01151 }
01152
01153 void neoBlue(String cmd) {
01154     String addr_str = cmd.substring(5);
01155     addr_str.trim();
01156
01157     uint8_t address = parseHex(addr_str);
01158     if (address == 0) {
01159         Serial.println("ERROR: Direccion invalida");
01160         return;
01161     }
01162
01163     if (sendCommand(address, CMD_BLUE)) {
01164         Serial.printf("[OK] NeoPixel AZUL en 0x%02X\n", address);
01165     } else {
01166         Serial.printf("[FAIL] No se pudo configurar NeoPixel en 0x%02X\n", address);
01167     }
01168 }
01169
01170 void neoWhite(String cmd) {
01171     String addr_str = cmd.substring(6);
01172     addr_str.trim();
01173
01174     uint8_t address = parseHex(addr_str);
01175     if (address == 0) {
01176         Serial.println("ERROR: Direccion invalida");
01177         return;
01178     }
01179
01180     if (sendCommand(address, CMD_WHITE)) {

```



```

01181     Serial.printf("[OK] NeoPixel BLANCO en 0x%02X\n", address);
01182 } else {
01183     Serial.printf("[FAIL] No se pudo configurar NeoPixel en 0x%02X\n", address);
01184 }
01185 }
01186
01187 void neoOff() {
01188     Serial.println("\n APAGAR NEOPIXEL (TODOS) ");
01189     int count = 0;
01190     for (int i = 0; i < device_count; i++) {
01191         uint8_t addr = found_devices[i];
01192         if (sendCommand(addr, CMD_OFF)) {
01193             Serial.printf(" [OK] 0x%02X - NeoPixel apagado\n", addr);
01194             count++;
01195         } else {
01196             Serial.printf(" [FAIL] 0x%02X\n", addr);
01197         }
01198         delay(10);
01199     }
01200     Serial.printf("\nTotal: %d/%d dispositivos\n", count, device_count);
01201 }
01202
01203 //
01204 //             FUNCIÓN: CAMBIAR DIRECCIÓN I2C
01205 //
01206 void changeI2CAddress(String cmd) {
01207     // Formato: "ch XX YY" donde XX=dirección actual, YY=nueva dirección
01208     cmd.trim();
01209     int firstSpace = cmd.indexOf(' ');
01210     int secondSpace = cmd.indexOf(' ', firstSpace + 1);
01211
01212     if (firstSpace == -1 || secondSpace == -1) {
01213         Serial.println("[ERROR] Formato: ch XX YY");
01214         Serial.println("      XX = dirección actual (hex)");
01215         Serial.println("      YY = nueva dirección (hex)");
01216         return;
01217     }
01218
01219     String oldAddrStr = cmd.substring(firstSpace + 1, secondSpace);
01220     String newAddrStr = cmd.substring(secondSpace + 1);
01221
01222     oldAddrStr.trim();
01223     newAddrStr.trim();
01224
01225     // Convertir direcciones de hexadecimal
01226     uint8_t oldAddr = (uint8_t)strtol(oldAddrStr.c_str(), NULL, 16);
01227     uint8_t newAddr = (uint8_t)strtol(newAddrStr.c_str(), NULL, 16);
01228
01229     // Validar rangos
01230     if (oldAddr < 0x08 || oldAddr > 0x77) {
01231         Serial.printf("[ERROR] Dirección actual 0x%02X fuera de rango (0x08-0x77)\n", oldAddr);
01232         return;
01233     }
01234
01235     if (newAddr < 0x08 || newAddr > 0x77) {
01236         Serial.printf("[ERROR] Nueva dirección 0x%02X fuera de rango (0x08-0x77)\n", newAddr);
01237         return;
01238     }
01239
01240     Serial.println("\n CAMBIAR DIRECCIÓN I2C ");
01241     Serial.printf("Dispositivo: 0x%02X → 0x%02X\n", oldAddr, newAddr);
01242     Serial.println(" ADVERTENCIA: Esta operación es PERMANENTE (guarda en Flash)");
01243     Serial.println("");
01244
01245     // PROTOCOLO CORRECTO según firmware i2c_slave/main.c:
01246     // 1. Enviar 0x3D (CMD_SET_I2C_ADDR) - activa modo "esperar nueva dirección"
01247     // 2. El firmware responde 0x0D y espera el siguiente byte como nueva dirección
01248     // 3. Enviar la nueva dirección como siguiente comando
01249     // 4. Firmware guarda en Flash y responde 0x0D
01250     // 5. Requiere reset para aplicar
01251
01252     Serial.print("Paso 1: Activando modo cambio dirección... ");
01253
01254     WIRE.beginTransmission(oldAddr);
01255     WIRE.write(CMD_SET_I2C_ADDR); // 0x3D
01256     uint8_t error = WIRE.endTransmission();
01257
01258     if (error != 0) {
01259         Serial.printf("[FAIL] Error I2C: %d\n", error);
01260         Serial.println(" El dispositivo no respondió. Verifica:");
01261         Serial.println("   - Dirección correcta (usa 'scan' para confirmar)");
01262         Serial.println("   - Dispositivo conectado y funcionando");
01263         return;
01264     }
01265
01266     delay(50); // Delay importante: firmware necesita procesar comando
01267

```

```

01268 // Leer respuesta
01269 uint8_t bytesRead = WIRE.requestFrom(oldAddr, (uint8_t)1);
01270 uint8_t response = 0xFF;
01271 if (bytesRead == 1) {
01272     response = WIRE.read();
01273 }
01274
01275 if (response == 0x0D) { // RESP_I2C_ADDR_SET
01276     Serial.println("[OK] Modo activado");
01277 } else {
01278     Serial.printf("[WARN] Respuesta: 0x%02X (esperaba 0x0D)\n", response);
01279     // Continuamos, puede funcionar igual
01280 }
01281
01282 delay(50); // Delay adicional antes del siguiente comando
01283
01284 // Paso 2: Enviar nueva dirección
01285 Serial.printf("Paso 2: Enviando nueva dirección 0x%02X... ", newAddr);
01286
01287 WIRE.beginTransmission(oldAddr);
01288 WIRE.write(newAddr); // Nueva dirección como dato
01289 error = WIRE.endTransmission();
01290
01291 if (error != 0) {
01292     Serial.printf("[FAIL] Error I2C: %d\n", error);
01293     Serial.println(" No se pudo completar el cambio");
01294     return;
01295 }
01296
01297 delay(100); // Delay largo: escritura Flash puede tomar tiempo
01298
01299 // Leer confirmación
01300 bytesRead = WIRE.requestFrom(oldAddr, (uint8_t)1);
01301 response = 0xFF;
01302 if (bytesRead == 1) {
01303     response = WIRE.read();
01304 }
01305
01306 if (response == 0x0D) { // RESP_I2C_ADDR_SET
01307     Serial.println("[OK] Guardado en Flash");
01308 } else {
01309     Serial.printf("[WARN] Respuesta: 0x%02X\n", response);
01310 }
01311
01312 Serial.println("");
01313 Serial.println("");
01314 Serial.println(" CAMBIO DE DIRECCIÓN COMPLETADO ");
01315 Serial.println("");
01316 Serial.println("");
01317 Serial.println(" IMPORTANTE - PRÓXIMOS PASOS:");
01318 Serial.println(" 1. La nueva dirección está GUARDADA EN FLASH");
01319 Serial.println(" 2. DEBES RESETEAR el dispositivo para aplicar");
01320 Serial.println(" 3. Opciones de reset:");
01321 Serial.println("     • Desconecta y reconecta el dispositivo");
01322 Serial.println("     • Usa el botón de reset físico");
01323 Serial.println("     • Cicla la alimentación");
01324 Serial.printf(" 4. Después del reset: dispositivo en 0x%02X\n", newAddr);
01325 Serial.println("");
01326 Serial.println("Verificación:");
01327 Serial.println(" 1. Resetea el dispositivo");
01328 Serial.println(" 2. Ejecuta: scan");
01329 Serial.printf(" 3. Busca la dirección 0x%02X en la lista\n", newAddr);
01330 Serial.println("");
01331 }
01332
01333 void saveColorPreset(String cmd) {
01334     // Formato: save_color <addr> <pos> <r> <g> <b>
01335     cmd.trim();
01336
01337     String tokens[6];
01338     int token_count = 0;
01339     int i = 0;
01340     while (i < cmd.length() && token_count < 6) {
01341         while (i < cmd.length() && cmd.charAt(i) == ' ') {
01342             i++;
01343         }
01344         if (i >= cmd.length()) {
01345             break;
01346         }
01347
01348         int start = i;
01349         while (i < cmd.length() && cmd.charAt(i) != ' ') {
01350             i++;
01351         }
01352         tokens[token_count++] = cmd.substring(start, i);
01353     }
01354

```

```

01355 if (token_count < 6 || tokens[0] != "save_color") {
01356     Serial.println("[ERROR] Formato: save_color <addr> <pos> <r> <g> <b>");
01357     Serial.println("      pos: 1=slot rojo, 2=slot verde, 3=slot azul");
01358     Serial.println("      r/g/b: 0..255 (decimal) o 0x00..0xFF");
01359     return;
01360 }
01361
01362 String addr_str = tokens[1];
01363 String pos_str = tokens[2];
01364 String r_str = tokens[3];
01365 String g_str = tokens[4];
01366 String b_str = tokens[5];
01367
01368 addr_str.trim();
01369 pos_str.trim();
01370 r_str.trim();
01371 g_str.trim();
01372 b_str.trim();
01373
01374 uint8_t addr = parseHex(addr_str);
01375 if (addr == 0) {
01376     Serial.println("[ERROR] Direccion invalida (0x08-0x77)");
01377     return;
01378 }
01379
01380 uint8_t pos_user = 0;
01381 uint8_t pos = 0;
01382 uint8_t r = 0;
01383 uint8_t g = 0;
01384 uint8_t b = 0;
01385
01386 if (!parseByteAuto(pos_str, &pos_user) || pos_user < 1 || pos_user > 3) {
01387     Serial.println("[ERROR] Posicion invalida. Usa 1, 2 o 3");
01388     return;
01389 }
01390 pos = (uint8_t)(pos_user - 1);
01391 if (!parseByteAuto(r_str, &r) || !parseByteAuto(g_str, &g) || !parseByteAuto(b_str, &b)) {
01392     Serial.println("[ERROR] RGB invalido. Usa 0..255 o 0x00..0xFF");
01393     return;
01394 }
01395
01396 Serial.println("\n SAVE COLOR PRESET ");
01397 Serial.printf("Dispositivo 0x%02X, slot %u, RGB=(%u,%u,%u)\n", addr, pos_user, r, g, b);
01398
01399 if (!sendCommand(addr, CMD_SAVE_COLOR)) {
01400     Serial.println("[FAIL] No se pudo iniciar save_color");
01401     return;
01402 }
01403
01404 delay(5);
01405 if (!sendCommand(addr, pos)) {
01406     Serial.println("[FAIL] No se pudo enviar posicion");
01407     return;
01408 }
01409
01410 delay(5);
01411 if (!sendCommand(addr, r)) {
01412     Serial.println("[FAIL] No se pudo enviar canal R");
01413     return;
01414 }
01415
01416 delay(5);
01417 if (!sendCommand(addr, g)) {
01418     Serial.println("[FAIL] No se pudo enviar canal G");
01419     return;
01420 }
01421
01422 delay(5);
01423 if (!sendCommand(addr, b)) {
01424     Serial.println("[FAIL] No se pudo enviar canal B");
01425     return;
01426 }
01427
01428 Serial.println("[OK] Color guardado en Flash");
01429 Serial.println("Usa comandos red/green/blue para probar los 3 slots.");
01430 }
01431
01432 void showHelp() {
01433     Serial.println("\n");
01434     Serial.println("      COMANDOS DISPONIBLES      ");
01435     Serial.println("");
01436     Serial.println("");
01437     Serial.println("SCAN:");
01438     Serial.println("  s      - Escanear dispositivos I2C");
01439     Serial.println("  loop   - Scan continuo (1 seg)");
01440     Serial.println("  stop   - Detener scan continuo");
01441     Serial.println("");

```

```

01442 Serial.println("TEST I2C:");
01443 Serial.println(" test INT DUR - Test asíncrono");
01444 Serial.println("          INT = intervalo (30-1000ms)");
01445 Serial.println("          DUR = duración (ms)");
01446 Serial.println(" stoptest - Detener test y mostrar resultados");
01447 Serial.println(" errors - Mostrar estadísticas de errores");
01448 Serial.println("");
01449 Serial.println("LECTURA DIGITAL PA0:");
01450 Serial.println(" read XX - Leer estado digital PA0");
01451 Serial.println(" dstats - Estadísticas de lecturas PA0");
01452 Serial.println(" readtest I D - Test continuo de lectura PA0");
01453 Serial.println("          I = intervalo (30-1000ms)");
01454 Serial.println("          D = duración (ms)");
01455 Serial.println(" stopread - Detener test de lectura");
01456 Serial.println("");
01457 Serial.println("RELAY:");
01458 Serial.println(" t XX - Pulso relay PB5 (10ms)");
01459 Serial.println(" on XX - Encender relay PB5");
01460 Serial.println(" off XX - Apagar relay PB5");
01461 Serial.println("");
01462 Serial.println("PWM / BUZZER:");
01463 Serial.println(" pwm XX NIVEL - PWM en PA2 (25, 50, 75, 100, off)");
01464 Serial.println(" silence - Apagar PWM en todos los dispositivos");
01465 Serial.println("");
01466 Serial.println("NEOPIXEL:");
01467 Serial.println(" neo XX COLOR - Control NeoPixel (red/green/blue/white/off o 1/2/3/4/0)");
01468 Serial.println(" red XX - NeoPixel rojo");
01469 Serial.println(" green XX - NeoPixel verde");
01470 Serial.println(" blue XX - NeoPixel azul");
01471 Serial.println(" white XX - NeoPixel blanco");
01472 Serial.println(" neooff - Apagar NeoPixel en todos");
01473 Serial.println("");
01474 Serial.println("I2C ADDRESS:");
01475 Serial.println(" ch XX YY - Cambiar dirección I2C");
01476 Serial.println("          XX = dirección actual (hex)");
01477 Serial.println("          YY = nueva dirección (hex)");
01478 Serial.println("          (se guarda en Flash, requiere reset)");
01479 Serial.println(" save_color A P R G B - Guardar color por slot en Flash");
01480 Serial.println("          P: 1=red slot, 2=green slot, 3=blue slot");
01481 Serial.println("          R/G/B: decimal o 0x00..0xFF");
01482 Serial.println("");
01483 Serial.println("FORMATO:");
01484 Serial.println(" XX = Dirección I2C en hexadecimal");
01485 Serial.println("");
01486 Serial.println("EJEMPLOS:");
01487 Serial.println(" s → Scan completo");
01488 Serial.println(" loop → Scan automatico cada 1s");
01489 Serial.println(" stop → Detener scan automatico");
01490 Serial.println(" test 50 10000 → Test cada 50ms por 10 seg");
01491 Serial.println(" stoptest → Detener test y ver errores");
01492 Serial.println(" errors → Ver estadísticas");
01493 Serial.println(" read 20 → Leer PA0 digital en 0x20");
01494 Serial.println(" readtest 50 10000 → Test lectura cada 50ms por 10s");
01495 Serial.println(" stopread → Detener test de lectura");
01496 Serial.println(" dstats → Ver estadísticas PA0");
01497 Serial.println(" pwm 20 50 → PWM 50% (500Hz) en 0x20");
01498 Serial.println(" silence → Apagar PWM en todos");
01499 Serial.println(" neo 20 1 → NeoPixel rojo en 0x20");
01500 Serial.println(" red 20 → NeoPixel rojo en 0x20");
01501 Serial.println(" neooff → Apagar NeoPixel en todos");
01502 Serial.println(" t 20 → Pulso en 0x20");
01503 Serial.println(" on 32 → Encender en 0x32");
01504 Serial.println(" off 20 → Apagar relay en 0x20");
01505 Serial.println(" ch 19 25 → Cambiar 0x19 a 0x25");
01506 Serial.println(" save_color 20 1 0x00 0xff 0x60 → Slot 1 personalizado");
01507 Serial.println("");
01508 }

```

## 7.3 examples/I2C\_Accel/I2C\_Accel.ino File Reference

## 7.4 I2C\_Accel.ino

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

```

00001 /*****
00002  * @file I2C_Accel.ino
00003  * @brief Minimal I2C bring-up for 7Semi ICM-20948 on UNO/ESP32 +
00004  * continuous accelerometer readout
00005  *

```

```

00006 * Features
00007 * - I2C init on default/custom pins
00008 * - IMU beginI2C() on I2C (addr 0x68/0x69)
00009 * - Manual sensor enable (setSensors)
00010 * - Accel config: scale, DLPF, averaging, sample rate
00011 * - Read accel (g) at ~2 Hz print
00012 *
00013 * Notes
00014 * - WHO_AM_I must be 0xEA
00015 * - Sensors must be explicitly enabled
00016 * - Bank switching handled internally by library
00017 *
00018 * Wiring (Arduino UNO I2C)
00019 * - SDA -> A4
00020 * - SCL -> A5
00021 * - VCC -> 3V3 (or 5V if your board supports it)
00022 * - GND -> GND
00023 *
00024 * Wiring (ESP32 default I2C)
00025 * - SDA -> GPIO21
00026 * - SCL -> GPIO22
00027 * - VCC -> 3V3
00028 * - GND -> GND
00029 *****/
00030
00031 #include <Wire.h>
00032 #include <DevLab_ICM20948.h>
00033
00034 /* ===== User Config ===== */
00035 #define SDA_PIN 6
00036 #define SCL_PIN 7
00037 #define I2C_FREQ 400000UL
00038 #define ICM_ADDR 0x69
00039
00040 DevLab_ICM20948 imu;
00041
00042 void setup() {
00043     Serial.begin(115200);
00044     delay(200);
00045     Serial.println(F("ICM-20948 (I2C) -- Accel Example"));
00046     Wire.begin(SDA_PIN, SCL_PIN);
00047     /** Initialize IMU using I2C */
00048     if (!imu.beginI2C(ICM_ADDR, Wire, 400000)) {
00049         Serial.println(F("ERROR: ICM-20948 beginI2C() failed."));
00050         while (1) delay(200);
00051     }
00052
00053     Serial.println(F("ICM-20948 ready."));
00054
00055     /** Verify device identity */
00056     uint8_t who;
00057     if (imu.readWhoAmI(who)) {
00058         Serial.print("WHO_AM_I: 0x");
00059         Serial.println(who, HEX);
00060     }
00061
00062     /** Enable only accelerometer
00063     * - accel = true
00064     * - gyro = false
00065     * - temp = false
00066     */
00067     if (!imu.setSensors(true, false, false)) {
00068         Serial.println(F("setSensors failed."));
00069     }
00070
00071     /** ----- ACCEL CONFIG -----
00072     * - Scale:  $\pm 2/\pm 4/\pm 8/\pm 16$  g
00073     * - DLPF: noise filtering
00074     * - Averaging: noise reduction
00075     * - Sample rate: output frequency
00076     */
00077
00078     /** Set accelerometer full-scale range */
00079     if (!imu.setAccelScale(ICM20948_Accel_FullScale::G_4)) {
00080         Serial.println(F("setAccelScale failed."));
00081     }
00082
00083     /** Enable DLPF (recommended)
00084     * - bypass = false → DLPF enabled
00085     */
00086     if (!imu.setAccelDLPF(ACCEL_DLPFCFG_3, false)) {
00087         Serial.println(F("setAccelDLPF failed."));
00088     }
00089
00090     /** Set averaging (noise reduction) */
00091     if (!imu.setAccelAveraging(ICM20948_Accel_Average::AVG_8)) {
00092         Serial.println(F("setAccelAveraging failed."));

```

```

00093     }
00094
00095     /** Set output data rate (ODR)
00096     * - Base = 1125 Hz
00097     * - ODR = 1125 / (1 + divider)
00098     * - Example: 225 Hz
00099     */
00100     if (!imu.setAccelSampleRate(225)) {
00101         Serial.println(F("setAccelSampleRate failed."));
00102     }
00103
00104     delay(10);
00105 }
00106
00107 void loop() {
00108     float ax, ay, az;
00109
00110     /** Read accelerometer data
00111     * - Output in g units
00112     * - Returns true on success
00113     */
00114     if (imu.readAccel(ax, ay, az)) {
00115         Serial.print("ACCEL [g]: ");
00116         Serial.print(ax, 3);
00117         Serial.print(", ");
00118         Serial.print(ay, 3);
00119         Serial.print(", ");
00120         Serial.println(az, 3);
00121     } else {
00122         Serial.println(F("ACCEL read failed"));
00123     }
00124
00125     Serial.println(F("-----"));
00126     delay(500);
00127 }

```

## 7.5 examples/I2C\_Basic/I2C\_Basic.ino File Reference

## 7.6 I2C\_Basic.ino

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

```

00001 /*****
00002  * @file      I2C_Basic.ino
00003  * @brief     Minimal I2C bring-up for 7Semi ICM-20948 +
00004  *            basic Accel/Gyro/Temp/Mag readout (updated API).
00005  *
00006  * Features
00007  * - I2C init (UNO / ESP32)
00008  * - beginI2C() initialization
00009  * - Manual sensor enable (setSensors)
00010  * - Read Accel / Gyro / Temp / Mag
00011  *
00012  * Notes
00013  * - WHO_AM_I must be 0xEA
00014  * - Sensors must be explicitly enabled
00015  * - Magnetometer requires initMag()
00016  *
00017  * Wiring (Arduino UNO I2C)
00018  * - SDA -> A4
00019  * - SCL -> A5
00020  * - VCC -> 3V3 (or 5V if supported)
00021  * - GND -> GND
00022  *
00023  * Wiring (ESP32 default I2C)
00024  * - SDA -> GPIO21
00025  * - SCL -> GPIO22
00026  *****/
00027 #include <Wire.h>
00028 #include <DevLab_ICM20948.h>
00029
00030 /* ===== User Config ===== */
00031 #define SDA_PIN      6
00032 #define SCL_PIN      7
00033 #define I2C_FREQ     400000UL
00034 #define ICM_ADDR     0x69
00035
00036 DevLab_ICM20948 imu;
00037

```

```

00038 void setup() {
00039     Serial.begin(115200);
00040     delay(200);
00041     Serial.println(F("ICM-20948 -- I2C Basic"));
00042     Wire.begin(SDA_PIN, SCL_PIN);
00043     /** Initialize IMU */
00044     if (!imu.beginI2C(ICM_ADDR, Wire, 400000)) {
00045         Serial.println(F("ERROR: beginI2C() failed"));
00046         while (1) delay(200);
00047     }
00048
00049     /** WHO_AM_I check */
00050     uint8_t who;
00051     if (!imu.readWhoAmI(who) || who != 0xEA) {
00052         Serial.println(F("ERROR: WHO_AM_I mismatch"));
00053         while (1) delay(200);
00054     }
00055
00056     Serial.print(F("WHO_AM_I = 0x"));
00057     Serial.println(who, HEX);
00058
00059     /** Enable all sensors */
00060     if (!imu.setSensors(true, true, true)) {
00061         Serial.println(F("ERROR: setSensors failed"));
00062     }
00063
00064     /** Initialize magnetometer */
00065     if (!imu.initMag()) {
00066         Serial.println(F("Mag init failed"));
00067     } else {
00068         Serial.println(F("Mag initialized"));
00069     }
00070
00071     Serial.println(F("Ready.\n"));
00072 }
00073
00074 void loop() {
00075     float ax, ay, az;
00076     float gx, gy, gz;
00077     float mx, my, mz;
00078     float tC;
00079
00080     /** Read Accelerometer */
00081     if (imu.readAccel(ax, ay, az)) {
00082         Serial.print(F("ACC [g]: "));
00083         Serial.print(ax, 3); Serial.print(", ");
00084         Serial.print(ay, 3); Serial.print(", ");
00085         Serial.println(az, 3);
00086     } else {
00087         Serial.println(F("ACC read failed"));
00088     }
00089
00090     /** Read Gyroscope */
00091     if (imu.readGyro(gx, gy, gz)) {
00092         Serial.print(F("GYR [dps]: "));
00093         Serial.print(gx, 2); Serial.print(", ");
00094         Serial.print(gy, 2); Serial.print(", ");
00095         Serial.println(gz, 2);
00096     } else {
00097         Serial.println(F("GYR read failed"));
00098     }
00099
00100     /** Read Magnetometer */
00101     if (imu.readMag(mx, my, mz)) {
00102         Serial.print(F("MAG [uT]: "));
00103         Serial.print(mx, 2); Serial.print(", ");
00104         Serial.print(my, 2); Serial.print(", ");
00105         Serial.println(mz, 2);
00106     } else {
00107         Serial.println(F("MAG read failed"));
00108     }
00109
00110     /** Read Temperature */
00111     if (imu.readTemperature(tC)) {
00112         Serial.print(F("TMP [C]: "));
00113         Serial.println(tC, 2);
00114     } else {
00115         Serial.println(F("TMP read failed"));
00116     }
00117
00118     Serial.println(F("-----"));
00119     delay(500);
00120 }
00121
00122 /* Note on magnetometer:
00123 * - Uses internal I2C master (AK09916 via ICM20948)
00124 * - initMag() must be called before readMag()

```

```

00125 * - If values are zero:
00126 *   → check initMag()
00127 *   → verify power + pull-ups
00128 */

```

## 7.7 examples/I2C\_Magneto/I2C\_Magneto.ino File Reference

## 7.8 I2C\_Magneto.ino

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

```

00001 /*****
00002 * @file      I2C_Magneto.ino
00003 * @brief     Minimal I2C bring-up for 7Semi ICM-20948 +
00004 *           AK09916 magnetometer readout (updated API).
00005 *
00006 * Features
00007 * - I2C init on default/custom pins
00008 * - IMU beginI2C() initialization
00009 * - Manual sensor enable
00010 * - Magnetometer initialization (initMag)
00011 * - Read magnetometer (uT) at ~2 Hz
00012 *
00013 * Wiring (Arduino UNO, I2C)
00014 * - SDA  -> A4
00015 * - SCL  -> A5
00016 * - VCC  -> 3V3 (or 5V if board supports it)
00017 * - GND  -> GND
00018 *
00019 * Wiring (ESP32, I2C default)
00020 * - SDA  -> GPIO21
00021 * - SCL  -> GPIO22
00022 * - VCC  -> 3V3
00023 * - GND  -> GND
00024 *
00025 * Notes
00026 * - WHO_AM_I must be 0xEA
00027 * - initMag() must be called before readMag()
00028 * - Magnetometer runs via internal I2C master
00029 *
00030 * Author   : 7Semi
00031 * License  : MIT
00032 *****/
00033 #include <Wire.h>
00034 #include <DevLab_ICM20948.h>
00035
00036 /* ===== User Config ===== */
00037 #define SDA_PIN    6
00038 #define SCL_PIN    7
00039 #define I2C_FREQ   400000UL
00040 #define ICM_ADDR   0x69
00041
00042 DevLab_ICM20948 imu;
00043
00044 void setup() {
00045   Serial.begin(115200);
00046   delay(200);
00047   Serial.println(F("ICM-20948 (I2C) -- Magnetometer Example"));
00048   Wire.begin(SDA_PIN, SCL_PIN);
00049   /** Initialize IMU */
00050   if (!imu.beginI2C(ICM_ADDR, Wire, 400000)) {
00051     Serial.println(F("ERROR: beginI2C() failed."));
00052     while (1) delay(200);
00053   }
00054
00055   Serial.println(F("ICM-20948 ready (I2C)."));
00056
00057   /** Verify device */
00058   uint8_t who;
00059   if (!imu.readWhoAmI(who) || who != 0xEA) {
00060     Serial.println(F("ERROR: WHO_AM_I mismatch"));
00061     while (1) delay(200);
00062   }
00063
00064   Serial.print(F("WHO_AM_I = 0x"));
00065   Serial.println(who, HEX);
00066
00067   /** Enable required sensors (mag uses internal I2C master)
00068    * - accel/gyro not strictly required but safe to keep ON

```



```

00069  */
00070  if (!imu.setSensors(true, true, false)) {
00071      Serial.println(F("setSensors failed."));
00072  }
00073
00074  /** Initialize magnetometer */
00075  if (!imu.initMag()) {
00076      Serial.println(F("ERROR: initMag() failed"));
00077      while (1) delay(200);
00078  }
00079
00080  Serial.println(F("Magnetometer ready"));
00081 }
00082
00083 void loop() {
00084     float mx, my, mz;
00085
00086     /** Read magnetometer data
00087      * - Output in microtesla (uT)
00088      * - Returns true on success
00089      */
00090     if (imu.readMag(mx, my, mz)) {
00091         Serial.print(F("MAG [uT]: "));
00092         Serial.print(mx, 2); Serial.print(F(", "));
00093         Serial.print(my, 2); Serial.print(F(", "));
00094         Serial.println(mz, 2);
00095     } else {
00096         Serial.println(F("Mag read failed"));
00097     }
00098
00099     Serial.println(F("-----"));
00100     delay(500);
00101 }

```

## 7.9 examples/SPI\_Basic/SPI\_Basic.ino File Reference

### 7.10 SPI\_Basic.ino

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

```

00001 /*****
00002  * @file      SPI_Basic.ino
00003  * @brief     Minimal SPI bring-up for 7Semi ICM-20948 +
00004  *           basic Accel/Gyro/Temp readout (updated API).
00005  *
00006  * Features
00007  * - SPI init (UNO / ESP32)
00008  * - beginSPI() initialization
00009  * - Manual sensor enable
00010  * - Read Accel / Gyro / Temp
00011  *
00012  * Notes
00013  * - WHO_AM_I must be 0xEA
00014  * - Sensors must be explicitly enabled
00015  * - SPI speed ~1MHz recommended during init
00016  *
00017  * Wiring (Arduino UNO SPI)
00018  * - SCK   -> D13
00019  * - MOSI  -> D11
00020  * - MISO  -> D12
00021  * - CS    -> D10 (changeable; update CS_PIN)
00022  * - VCC   -> 3V3
00023  * - GND   -> GND
00024  *
00025  * Wiring (ESP32 VSPI default)
00026  * - SCK   -> D13
00027  * - MOSI  -> D11
00028  * - MISO  -> D12
00029  * - CS    -> D10
00030  *****/
00031 #include <DevLab_ICM20948.h>
00032
00033 #define SPI_FAST_SPEED 1000000
00034
00035 /** IMU instance */
00036 DevLab_ICM20948 imu;
00037
00038 SPIClass spi_bus(SPI); // ← usa el bus SPI por defecto del Arduino
00039

```

```

00040 void setup() {
00041     Serial.begin(115200);
00042     delay(200);
00043
00044
00045     if (!imu.beginSPI(CS_PIN, spi_bus, 1000000)) {
00046         Serial.println("ERROR: beginSPI() failed");
00047         while (1) delay(200);
00048     }
00049
00050     imu.setSensors(true, true, true);
00051     Serial.println("Ready.");
00052 }
00053
00054 void loop() {
00055     float ax, ay, az;
00056     float gx, gy, gz;
00057     float tC;
00058
00059     /** Read Accelerometer */
00060     if (imu.readAccel(ax, ay, az)) {
00061         Serial.print(F("ACC [g]: "));
00062         Serial.print(ax, 3); Serial.print(", ");
00063         Serial.print(ay, 3); Serial.print(", ");
00064         Serial.println(az, 3);
00065     } else {
00066         Serial.println(F("ACC read failed"));
00067     }
00068
00069     /** Read Gyroscope */
00070     if (imu.readGyro(gx, gy, gz)) {
00071         Serial.print(F("GYR [dps]: "));
00072         Serial.print(gx, 2); Serial.print(", ");
00073         Serial.print(gy, 2); Serial.print(", ");
00074         Serial.println(gz, 2);
00075     } else {
00076         Serial.println(F("GYR read failed"));
00077     }
00078
00079     /** Read Temperature */
00080     if (imu.readTemperature(tC)) {
00081         Serial.print(F("TMP [C]: "));
00082         Serial.println(tC, 2);
00083     } else {
00084         Serial.println(F("TMP read failed"));
00085     }
00086
00087     Serial.println(F("-----"));
00088     delay(500);
00089 }

```

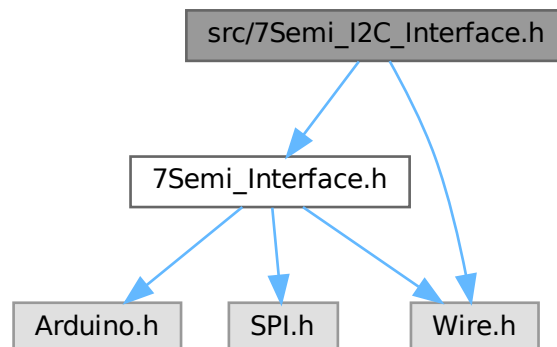
## 7.11 src/7Semi\_I2C\_Interface.h File Reference

```

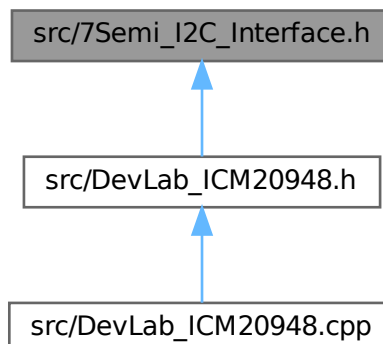
#include "7Semi_Interface.h"
#include <Wire.h>

```

Include dependency graph for 7Semi\_I2C\_Interface.h:



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



## Classes

- class [I2C\\_Interface](#)

## 7.12 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->beginTransmission(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
00066         return 0;
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->beginTransmission(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 };

```

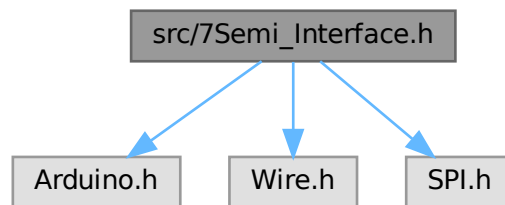
## 7.13 src/7Semi\_Interface.h File Reference

```

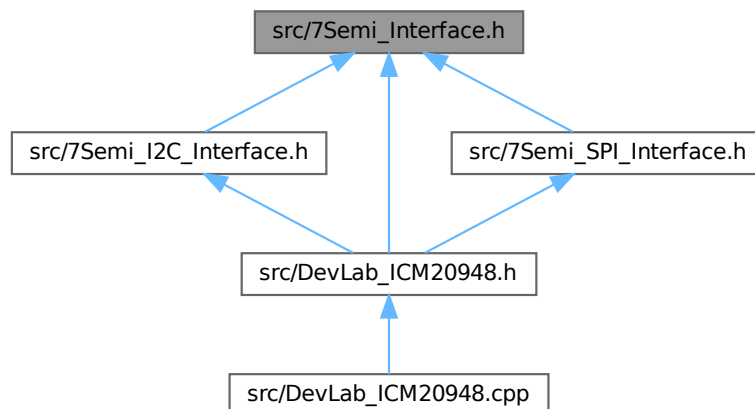
#include <Arduino.h>
#include <Wire.h>
#include <SPI.h>

```

Include dependency graph for 7Semi\_Interface.h:



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



### Classes

- class [Interface\\_7Semi](#)

## 7.14 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
00018     virtual ~Interface_7Semi() {}
00019
00020     /**
00021      * beginI2C()
00022      *
00023      * - Initializes I2C peripheral
00024      * - Configures SDA / SCL pins if supported
00025      * - Sets communication clock
00026      *
00027      * Returns:
00028      * - true → Initialization successful
00029      * - false → Initialization failed
00030      */
00031     virtual bool beginI2C(
00032         uint8_t address,
00033         TwoWire &wire,
00034         uint32_t speed,
00035         uint8_t sda = 255,
00036         uint8_t scl = 255) = 0;
00037
00038     virtual bool beginSPI(
00039         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      * read()
00048      *
00049      * - Reads data from device register
00050      *
00051      * Parameters:
00052      * - reg : Register address
00053      * - data : Output buffer
00054      * - len : Number of bytes
00055      *
00056      * Returns:
00057      * - 0 → Success
00058      * - <0 → Error
00059      */
00060     virtual int8_t read(
00061         uint8_t reg,
00062         uint8_t *data,
00063         uint32_t len) = 0;
00064
00065     /**
00066      * write()
00067      *
00068      * - Writes data to device register
00069      *
00070      * Returns:
00071      * - 0 → Success
00072      * - <0 → Error
00073      */
00074     virtual int8_t write(
00075         uint8_t reg,
00076         const uint8_t *data,
00077         uint32_t len) = 0;
00078
00079 protected:
00080     /** Delay wrapper */
00081     static void delay_us(uint32_t us)

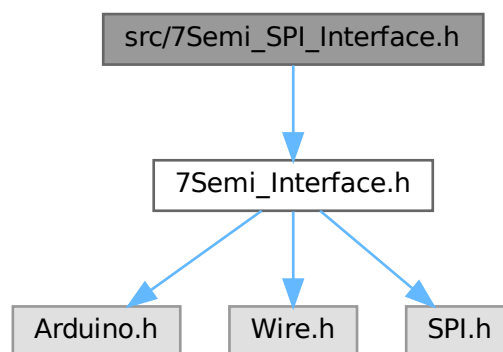
```

```
00083     {  
00084         delayMicroseconds(us);  
00085     }  
00086 };  
00087 #endif
```

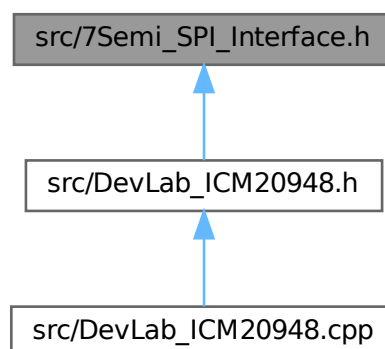
## 7.15 src/7Semi\_SPI\_Interface.h File Reference

```
#include "7Semi_Interface.h"
```

Include dependency graph for 7Semi\_SPI\_Interface.h:



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



### Classes

- class [SPI\\_Interface](#)

## 7.16 7Semi\_SPI\_Interface.h

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

```

00001 #pragma once
00002 #include "7Semi_Interface.h"
00003
00004 /**
00005  * 7Semi SPI Interface Implementation
00006  */
00007 class SPI_Interface : public Interface_7Semi
00008 {
00009
00010 public:
00011     SPIClass *spi = nullptr;
00012     uint8_t cs = 255;
00013     uint32_t speed = 1000000;
00014
00015     /**
00016      * beginSPI()
00017      *
00018      * - Initialize SPI interface
00019      * - Configure CS and SPI bus
00020      */
00021     bool beginSPI(uint8_t csPin,
00022                  SPIClass &spiPort,
00023                  uint32_t spiSpeed,
00024                  uint8_t sck = 255,
00025                  uint8_t miso = 255,
00026                  uint8_t mosi = 255) override
00027     {
00028         if (csPin == 255)
00029             return false;
00030
00031         spi = &spiPort;
00032         cs = csPin;
00033         speed = spiSpeed;
00034
00035         pinMode(cs, OUTPUT);
00036         digitalWrite(cs, HIGH);
00037
00038         #if defined(ESP32)
00039             if (sck != 255 && miso != 255 && mosi != 255)
00040                 spi->begin(sck, miso, mosi, cs);
00041             else
00042                 spi->begin();
00043         #else
00044             spi->begin();
00045         #endif
00046
00047         delay(1);
00048         return true;
00049     }
00050
00051     bool beginI2C(uint8_t, TwoWire &, uint32_t,
00052                  uint8_t, uint8_t) override
00053     {
00054         return false;
00055     }
00056
00057     /**
00058      * read()
00059      *
00060      * - Read multiple bytes from register
00061      */
00062     int8_t read(uint8_t reg, uint8_t *data, uint32_t len) override
00063     {
00064         if (!spi || !data || len == 0)
00065             return -1;
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          */

```

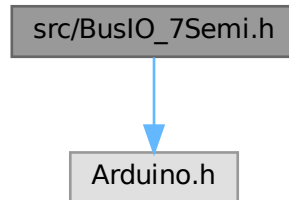


```
00083         for (uint32_t i = 0; i < len; i++)
00084             data[i] = spi->transfer(0x00);
00085
00086         digitalWrite(cs, HIGH);
00087         delayMicroseconds(10);
00088         spi->endTransaction();
00089
00090         // Serial.print("R-Reg: ");
00091         // Serial.print(reg, 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 };
```

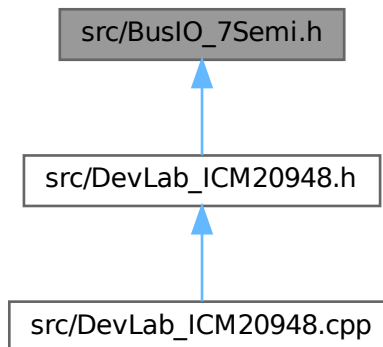
## 7.17 src/BusIO\_7Semi.h File Reference

```
#include <Arduino.h>
```

Include dependency graph for BusIO\_7Semi.h:



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



### Classes

- class [BusIO\\_7Semi< Bus >](#)

## 7.18 BusIO\_7Semi.h

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

```
00001 #pragma once
00002 #include <Arduino.h>
00003
00004 template <typename Bus>
00005 class BusIO_7Semi
00006 {
```

```

00007
00008 public:
00009     /**
00010      * Constructor
00011      *
00012      * - Stores interface reference
00013      */
00014     BusIO_7Semi(Bus &busRef) : bus(busRef) {}
00015
00016     /** read 8-bit */
00017     inline bool read(uint8_t reg, uint8_t &value)
00018     {
00019         return (bus.read(reg, &value, 1) == 0);
00020     }
00021
00022     /** read 16-bit */
00023     inline bool read(uint8_t reg, uint16_t &value)
00024     {
00025         uint8_t data[2];
00026         if (bus.read(reg, data, 2) != 0)
00027             return false;
00028         value = (data[0] << 8) | data[1];
00029         return true;
00030     }
00031
00032     /** read burst */
00033     inline bool read(uint8_t reg, uint8_t *data, uint32_t len)
00034     {
00035         return (bus.read(reg, data, len) == 0);
00036     }
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     /**

```

```

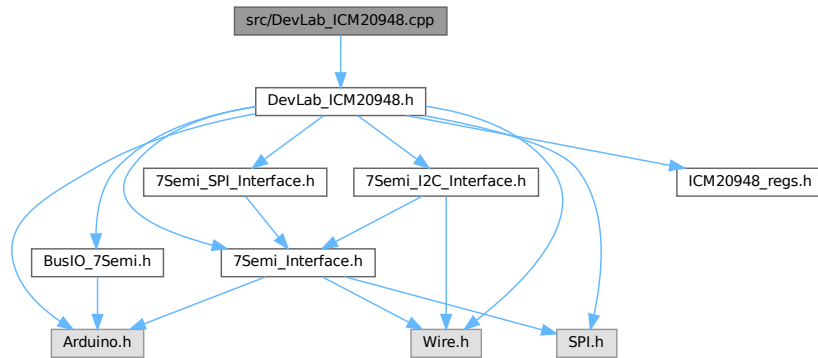
00094     * writeBits (8-bit register)
00095     *
00096     * - Modifies specific bits in 8-bit register
00097     */
00098 bool writeBits(uint8_t reg, uint8_t pos, uint8_t len, uint8_t value)
00099 {
00100     if (len == 0 || len > 8 || (pos + len) > 8)
00101         return false;
00102
00103     uint8_t 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 };

```

## 7.19 src/DevLab\_ICM20948.cpp File Reference

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

Include dependency graph for DevLab\_ICM20948.cpp:



## 7.20 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     if (!bus->write(ODR_ALIGN_EN, (uint8_t)0x01))
00100         return false;
00101
00102     // Enable I2C interface (I2C_IF_DIS = 0)
00103     if (!bus->writeBit(USER_CTRL, 4, (uint8_t)0x01))
00104         return false;
00105
00106     // Wake device from sleep mode (SLEEP = 0)
00107     if (!bus->write(PWR_MGMT_1, (uint8_t)0x01))
00108         return false;
00109
00110     if(!applyBasicDefaults())
00111         return false;
00112
00113     // Initialization successful
00114     return true;
00115 }
00116
00117 bool DevLab_ICM20948::readWhoAmI(uint8_t &whoAmI)
00118 {
00119     // Select USER BANK 0
00120     if (!selectBank(0))
00121         return false;
00122
00123     // Read WHO_AM_I register
00124     if (!bus->read(WHO_AM_I, whoAmI))
00125         return false;
00126
00127     // Success
00128     return true;
00129 }
00130
00131 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

```

```

00220     return false;
00221
00222 // Set accel sample rate divider low byte
00223 if (!bus->write(ACCEL_SMPLRT_DIV_2, (uint8_t)0x00))
00224     return false;
00225
00226 // Select USER BANK 0
00227 if (!selectBank(0))
00228     return false;
00229
00230 // Configure interrupt pin (open-drain, active-low, latch)
00231 if (!bus->write(INT_PIN_CFG, (uint8_t)0x30))
00232     return false;
00233
00234 // Configuration successful
00235 return true;
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  $\mu$ T (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     // Wait for completion (with timeout)
00442     uint32_t start = millis();
00443     uint8_t status;
00444
00445     while (millis() - start < 10)
00446     {
00447         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) & 0x0F);
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) & 0x0F);
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     // Success
00854     return true;
00855 }
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 // Initialize INT1
01192 // Configure interrupt pin (open-drain, active-low, latch)
01193 if (!bus->write(INT_PIN_CFG, reg))
01194     return false;
01195
01196 // Configuration successful
01197 return true;
01198 }
01199
01200 bool DevLab_ICM20948::intEnableConfig(const ICM20948_IntEnableConfig &cfg)
01201 {
01202     if (!bus) return false;
01203     if (!selectBank(0)) return false;
01204
01205     // INT_ENABLE
01206     uint8_t reg0 = (cfg.wofEn << 7) |
01207                   (cfg.womIntEn << 3) |
01208                   (cfg.pllRdyEn << 2) |
01209                   (cfg.dmpInt1En << 1) |
01210                   (cfg.i2cMstIntEn << 0);
01211     if (!bus->write(INT_ENABLE, reg0)) return false;
01212
01213     // INT_ENABLE_1
01214     if (!bus->write(INT_ENABLE_1, (uint8_t)(cfg.rawDataRdyEn & 0x01))) return false;
01215
01216     // INT_ENABLE_2 -- construye máscara desde el array
01217     uint8_t ovfMask = 0;
01218     for (uint8_t i = 0; i < 5; i++)
01219         ovfMask |= (cfg.fifoOvfEn[i] ? BIT(i) : 0);
01220     if (!bus->write(INT_ENABLE_2, ovfMask)) return false;
01221
01222     // INT_ENABLE_3 -- igual
01223     uint8_t wmMask = 0;
01224     for (uint8_t i = 0; i < 5; i++)
01225         wmMask |= (cfg.fifoWmEn[i] ? BIT(i) : 0);
01226     if (!bus->write(INT_ENABLE_3, wmMask)) return false;
01227
01228     return true;
01229 }
01230
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

```

```

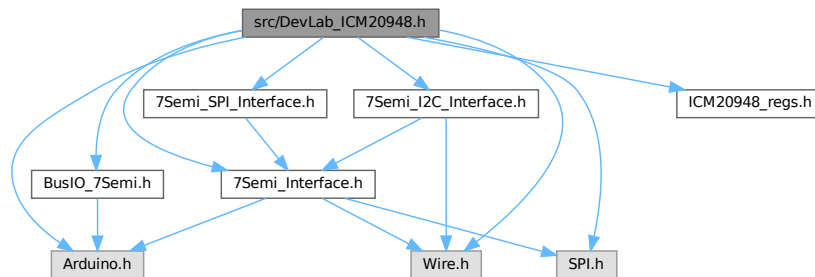
01351     if (!bus->write(I2C_SLV4_ADDR, (uint8_t)(slaveAddr & 0x7F))) return false;
01352
01353     // Byte de comando a enviar (único byte de la transacción)
01354     if (!bus->write(I2C_SLV4_DO, cmd)) return false;
01355
01356     // EN + REG_DIS: el SLV4 envía solo I2C_SLV4_DO, sin byte de registro
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 }

```

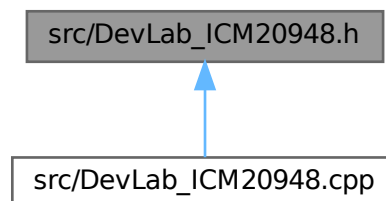
## 7.21 src/DevLab\_ICM20948.h File Reference

```
#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 }

## 7.21.1 Enumeration Type Documentation

### 7.21.1.1 ICM20948\_Accel\_Average

```
enum class ICM20948\_Accel\_Average : uint8_t [strong]
```

#### Enumerator

<a href="#">AVG_1_OR_4</a>	
<a href="#">AVG_8</a>	
<a href="#">AVG_16</a>	
<a href="#">AVG_32</a>	

Definition at line 76 of file [DevLab\\_ICM20948.h](#).

### 7.21.1.2 ICM20948\_Accel\_DLPFCFG

```
enum class ICM20948\_Accel\_DLPFCFG : uint8_t [strong]
```

#### Enumerator

<a href="#">DLPF0_246HZ</a>	
<a href="#">DLPF_246HZ</a>	

DLPF_111HZ	
DLPF_50HZ	
DLPF_24HZ	
DLPF_12HZ	
DLPF_6HZ	
DLPF_473HZ	

Definition at line 84 of file [DevLab\\_ICM20948.h](#).

#### 7.21.1.3 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](#).

#### 7.21.1.4 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](#).

#### 7.21.1.5 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](#).

#### 7.21.1.6 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](#).

### 7.21.1.7 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](#).

### 7.21.1.8 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](#).

## 7.22 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
00176      DevLab_ICM20948();
00177
00178      /**
00179       * beginI2C
00180       *
00181       * - Initialize ICM20948 using I2C interface
00182       * - Sets up communication layer (Interface + BusIO)
00183       * - Verifies device identity (WHO_AM_I)
00184       * - Configures basic power and clock settings
00185       *
00186       * Returns:
00187       * - true → Device initialized successfully
00188       * - false → Communication or configuration failed
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     * auxReadI2bit
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 */

```

## 7.23 src/ICM20948\_platform.h File Reference

### Classes

- struct [icm\\_plat\\_t](#)

## 7.24 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)

```

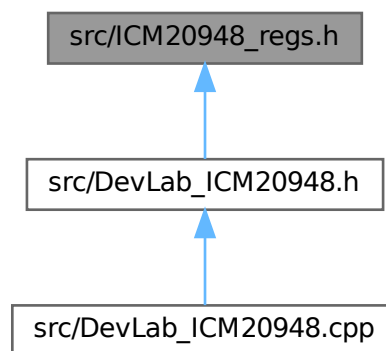
```

00039     * - For I2C: leave as NULL
00040     */
00041 void (*spi_cs)(int level, void *user);
00042 /**
00043     * - Device address (I2C) or ignored for SPI if you wrap inside user
00044     */
00045 uint8_t i2c_addr;
00046 } icm_plat_t;
00047
00048 #endif

```

## 7.25 src/ICM20948\_regs.h File Reference

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



### Macros

- #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_OFFS_H 0x14 /* XA_OFFS[14:7] */
• #define XA_OFFS_L 0x15 /* XA_OFFS[6:0] */
• #define YA_OFFS_H 0x17 /* YA_OFFS[14:7] */
• #define YA_OFFS_L 0x18 /* YA_OFFS[6:0] */
• #define ZA_OFFS_H 0x1A /* ZA_OFFS[14:7] */
• #define ZA_OFFS_L 0x1B /* ZA_OFFS[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_OFFS_USRH 0x03
• #define XG_OFFS_USRL 0x04
• #define YG_OFFS_USRH 0x05
• #define YG_OFFS_USRL 0x06
• #define ZG_OFFS_USRH 0x07
• #define ZG_OFFS_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)
- #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)
- #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`

## 7.25.1 Macro Definition Documentation

### 7.25.1.1 ACCEL\_BYPASS\_3DB\_BW\_HZ

```
#define ACCEL_BYPASS_3DB_BW_HZ 1209.0f
```

Definition at line 369 of file [ICM20948\\_regs.h](#).

### 7.25.1.2 ACCEL\_BYPASS\_NBW\_HZ

```
#define ACCEL_BYPASS_NBW_HZ 1248.0f
```

Definition at line 370 of file [ICM20948\\_regs.h](#).

### 7.25.1.3 ACCEL\_BYPASS\_RATE\_HZ

```
#define ACCEL_BYPASS_RATE_HZ 4500.0f
```

Definition at line 371 of file [ICM20948\\_regs.h](#).

### 7.25.1.4 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](#).

### 7.25.1.5 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](#).

### 7.25.1.6 ACCEL\_DEC3\_AVG\_16

```
#define ACCEL_DEC3_AVG_16 2u
```

Definition at line 356 of file [ICM20948\\_regs.h](#).

#### 7.25.1.7 ACCEL\_DEC3\_AVG\_32

```
#define ACCEL_DEC3_AVG_32 3u
```

Definition at line 357 of file [ICM20948\\_regs.h](#).

#### 7.25.1.8 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](#).

#### 7.25.1.9 ACCEL\_DEC3\_AVG\_8

```
#define ACCEL_DEC3_AVG_8 1u
```

Definition at line 355 of file [ICM20948\\_regs.h](#).

#### 7.25.1.10 ACCEL\_DLPF0\_3DB\_BW\_HZ

```
#define ACCEL_DLPF0_3DB_BW_HZ 246.0f
```

Definition at line 376 of file [ICM20948\\_regs.h](#).

#### 7.25.1.11 ACCEL\_DLPF0\_NBW\_HZ

```
#define ACCEL_DLPF0_NBW_HZ 265.0f
```

Definition at line 377 of file [ICM20948\\_regs.h](#).

#### 7.25.1.12 ACCEL\_DLPF1\_3DB\_BW\_HZ

```
#define ACCEL_DLPF1_3DB_BW_HZ 246.0f
```

Definition at line 378 of file [ICM20948\\_regs.h](#).

#### 7.25.1.13 ACCEL\_DLPF1\_NBW\_HZ

```
#define ACCEL_DLPF1_NBW_HZ 265.0f
```

Definition at line 379 of file [ICM20948\\_regs.h](#).

#### 7.25.1.14 ACCEL\_DLPF2\_3DB\_BW\_HZ

```
#define ACCEL_DLPF2_3DB_BW_HZ 111.4f
```

Definition at line 380 of file [ICM20948\\_regs.h](#).

#### 7.25.1.15 ACCEL\_DLPF2\_NBW\_HZ

```
#define ACCEL_DLPF2_NBW_HZ 136.0f
```

Definition at line 381 of file [ICM20948\\_regs.h](#).

#### 7.25.1.16 ACCEL\_DLPF3\_3DB\_BW\_HZ

```
#define ACCEL_DLPF3_3DB_BW_HZ 50.4f
```

Definition at line 382 of file [ICM20948\\_regs.h](#).

#### 7.25.1.17 ACCEL\_DLPF3\_NBW\_HZ

```
#define ACCEL_DLPF3_NBW_HZ 68.8f
```

Definition at line 383 of file [ICM20948\\_regs.h](#).

#### 7.25.1.18 ACCEL\_DLPF4\_3DB\_BW\_HZ

```
#define ACCEL_DLPF4_3DB_BW_HZ 23.9f
```

Definition at line 384 of file [ICM20948\\_regs.h](#).

#### 7.25.1.19 ACCEL\_DLPF4\_NBW\_HZ

```
#define ACCEL_DLPF4_NBW_HZ 34.4f
```

Definition at line 385 of file [ICM20948\\_regs.h](#).

#### 7.25.1.20 ACCEL\_DLPF5\_3DB\_BW\_HZ

```
#define ACCEL_DLPF5_3DB_BW_HZ 11.5f
```

Definition at line 386 of file [ICM20948\\_regs.h](#).

#### 7.25.1.21 ACCEL\_DLPF5\_NBW\_HZ

```
#define ACCEL_DLPF5_NBW_HZ 17.0f
```

Definition at line 387 of file [ICM20948\\_regs.h](#).

#### 7.25.1.22 ACCEL\_DLPF6\_3DB\_BW\_HZ

```
#define ACCEL_DLPF6_3DB_BW_HZ 5.7f
```

Definition at line 388 of file [ICM20948\\_regs.h](#).

### 7.25.1.23 ACCEL\_DLPF6\_NBW\_HZ

```
#define ACCEL_DLPF6_NBW_HZ 8.3f
```

Definition at line 389 of file [ICM20948\\_regs.h](#).

### 7.25.1.24 ACCEL\_DLPF7\_3DB\_BW\_HZ

```
#define ACCEL_DLPF7_3DB_BW_HZ 473.0f
```

Definition at line 390 of file [ICM20948\\_regs.h](#).

### 7.25.1.25 ACCEL\_DLPF7\_NBW\_HZ

```
#define ACCEL_DLPF7_NBW_HZ 499.0f
```

Definition at line 391 of file [ICM20948\\_regs.h](#).

### 7.25.1.26 ACCEL\_DLPF\_BASE\_HZ

```
#define ACCEL_DLPF_BASE_HZ 1125.0f
```

Definition at line 362 of file [ICM20948\\_regs.h](#).

### 7.25.1.27 ACCEL\_DLPF\_ODR\_HZ

```
#define ACCEL_DLPF_ODR_HZ(  
    div)
```

#### Value:

```
(ACCEL_DLPF_BASE_HZ / (1.0f + (float)(div)))
```

Definition at line 363 of file [ICM20948\\_regs.h](#).

### 7.25.1.28 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](#).

### 7.25.1.29 ACCEL\_DLPFCFG\_1

```
#define ACCEL_DLPFCFG_1 1
```

Definition at line 346 of file [ICM20948\\_regs.h](#).

#### 7.25.1.30 ACCEL\_DLPFCFG\_2

```
#define ACCEL_DLPFCFG_2 2
```

Definition at line 347 of file [ICM20948\\_regs.h](#).

#### 7.25.1.31 ACCEL\_DLPFCFG\_3

```
#define ACCEL_DLPFCFG_3 3
```

Definition at line 348 of file [ICM20948\\_regs.h](#).

#### 7.25.1.32 ACCEL\_DLPFCFG\_4

```
#define ACCEL_DLPFCFG_4 4
```

Definition at line 349 of file [ICM20948\\_regs.h](#).

#### 7.25.1.33 ACCEL\_DLPFCFG\_5

```
#define ACCEL_DLPFCFG_5 5
```

Definition at line 350 of file [ICM20948\\_regs.h](#).

#### 7.25.1.34 ACCEL\_DLPFCFG\_6

```
#define ACCEL_DLPFCFG_6 6
```

Definition at line 351 of file [ICM20948\\_regs.h](#).

#### 7.25.1.35 ACCEL\_DLPFCFG\_7

```
#define ACCEL_DLPFCFG_7 7
```

Definition at line 352 of file [ICM20948\\_regs.h](#).

#### 7.25.1.36 ACCEL\_DLPFCFG\_MASK

```
#define ACCEL_DLPFCFG_MASK (0x7u << ACCEL_DLPFCFG_SHIFT)
```

Definition at line 218 of file [ICM20948\\_regs.h](#).

#### 7.25.1.37 ACCEL\_DLPFCFG\_SHIFT

```
#define ACCEL_DLPFCFG_SHIFT 5
```

Definition at line 217 of file [ICM20948\\_regs.h](#).



### 7.25.1.38 ACCEL\_FCHOICE

```
#define ACCEL_FCHOICE BIT(0)
```

Definition at line 210 of file [ICM20948\\_regs.h](#).

### 7.25.1.39 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](#).

### 7.25.1.40 ACCEL\_FCHOICE\_DLPF

```
#define ACCEL_FCHOICE_DLPF 1
```

Definition at line 338 of file [ICM20948\\_regs.h](#).

### 7.25.1.41 ACCEL\_FS\_16G

```
#define ACCEL_FS_16G (3u << ACCEL_FS_SEL_SHIFT)
```

Definition at line 216 of file [ICM20948\\_regs.h](#).

### 7.25.1.42 ACCEL\_FS\_2G

```
#define ACCEL_FS_2G (0u << ACCEL_FS_SEL_SHIFT)
```

Definition at line 213 of file [ICM20948\\_regs.h](#).

### 7.25.1.43 ACCEL\_FS\_4G

```
#define ACCEL_FS_4G (1u << ACCEL_FS_SEL_SHIFT)
```

Definition at line 214 of file [ICM20948\\_regs.h](#).

### 7.25.1.44 ACCEL\_FS\_8G

```
#define ACCEL_FS_8G (2u << ACCEL_FS_SEL_SHIFT)
```

Definition at line 215 of file [ICM20948\\_regs.h](#).

#### 7.25.1.45 ACCEL\_FS\_SEL\_MASK

```
#define ACCEL_FS_SEL_MASK (0x3u << ACCEL_FS_SEL_SHIFT)
```

Definition at line 212 of file [ICM20948\\_regs.h](#).

#### 7.25.1.46 ACCEL\_FS\_SEL\_SHIFT

```
#define ACCEL_FS_SEL_SHIFT 1
```

Definition at line 211 of file [ICM20948\\_regs.h](#).

#### 7.25.1.47 ACCEL\_INTEL\_CTRL

```
#define ACCEL_INTEL_CTRL 0x12 /* ACCEL_INTEL_EN ACCEL_INTEL_MODE_INT */
```

Definition at line 203 of file [ICM20948\\_regs.h](#).

#### 7.25.1.48 ACCEL\_INTEL\_EN

```
#define ACCEL_INTEL_EN BIT(7)
```

Definition at line 204 of file [ICM20948\\_regs.h](#).

#### 7.25.1.49 ACCEL\_INTEL\_MODE\_INT

```
#define ACCEL_INTEL_MODE_INT BIT(6)
```

Definition at line 205 of file [ICM20948\\_regs.h](#).

#### 7.25.1.50 ACCEL\_SMPLRT\_DIV\_1

```
#define ACCEL_SMPLRT_DIV_1 0x10 /* ACCEL_SMPLRT_DIV[11:8] */
```

Definition at line 200 of file [ICM20948\\_regs.h](#).

#### 7.25.1.51 ACCEL\_SMPLRT\_DIV\_2

```
#define ACCEL_SMPLRT_DIV_2 0x11 /* ACCEL_SMPLRT_DIV[7:0] */
```

Definition at line 201 of file [ICM20948\\_regs.h](#).

#### 7.25.1.52 ACCEL\_SMPLRT\_DIV\_MAX

```
#define ACCEL_SMPLRT_DIV_MAX 4095u
```

Definition at line 361 of file [ICM20948\\_regs.h](#).

#### 7.25.1.53 ACCEL\_SMPLRT\_DIV\_MIN

```
#define ACCEL_SMPLRT_DIV_MIN 0u
```

Definition at line 360 of file [ICM20948\\_regs.h](#).

#### 7.25.1.54 ACCEL\_WOM\_THR

```
#define ACCEL_WOM_THR 0x13 /* WOM_THRESHOLD[7:0] */
```

Definition at line 207 of file [ICM20948\\_regs.h](#).

#### 7.25.1.55 ACCEL\_XOUT\_H

```
#define ACCEL_XOUT_H 0x2D
```

Definition at line 111 of file [ICM20948\\_regs.h](#).

#### 7.25.1.56 ACCEL\_XOUT\_L

```
#define ACCEL_XOUT_L 0x2E
```

Definition at line 112 of file [ICM20948\\_regs.h](#).

#### 7.25.1.57 ACCEL\_YOUT\_H

```
#define ACCEL_YOUT_H 0x2F
```

Definition at line 113 of file [ICM20948\\_regs.h](#).

#### 7.25.1.58 ACCEL\_YOUT\_L

```
#define ACCEL_YOUT_L 0x30
```

Definition at line 114 of file [ICM20948\\_regs.h](#).

#### 7.25.1.59 ACCEL\_ZOUT\_H

```
#define ACCEL_ZOUT_H 0x31
```

Definition at line 115 of file [ICM20948\\_regs.h](#).

#### 7.25.1.60 ACCEL\_ZOUT\_L

```
#define ACCEL_ZOUT_L 0x32
```

Definition at line 116 of file [ICM20948\\_regs.h](#).

#### 7.25.1.61 AK09916\_I2C\_ADDR

```
#define AK09916_I2C_ADDR 0x0C
```

Definition at line 310 of file [ICM20948\\_regs.h](#).

#### 7.25.1.62 AK\_CNTL2

```
#define AK_CNTL2 0x31
```

Definition at line 315 of file [ICM20948\\_regs.h](#).

#### 7.25.1.63 AK\_CNTL3

```
#define AK_CNTL3 0x32
```

Definition at line 316 of file [ICM20948\\_regs.h](#).

#### 7.25.1.64 AK\_HXL

```
#define AK_HXL 0x11
```

Definition at line 313 of file [ICM20948\\_regs.h](#).

#### 7.25.1.65 AK\_ST1

```
#define AK_ST1 0x10
```

Definition at line 312 of file [ICM20948\\_regs.h](#).

#### 7.25.1.66 AK\_ST2

```
#define AK_ST2 0x18
```

Definition at line 314 of file [ICM20948\\_regs.h](#).

#### 7.25.1.67 AK\_WIA2

```
#define AK_WIA2 0x01
```

Definition at line 311 of file [ICM20948\\_regs.h](#).

#### 7.25.1.68 AK\_WIA2\_VAL

```
#define AK_WIA2_VAL 0x09 /* AK09916C WIA2 expected */
```

Definition at line 317 of file [ICM20948\\_regs.h](#).

#### 7.25.1.69 AUTO\_SEL

```
#define AUTO_SEL 0x01 /* 1-5: Auto/PLL preferred (use 1 as default) */
```

Definition at line 320 of file [ICM20948\\_regs.h](#).

#### 7.25.1.70 AX\_ST\_EN\_REG

```
#define AX_ST_EN_REG BIT(7)
```

Definition at line 221 of file [ICM20948\\_regs.h](#).

#### 7.25.1.71 AY\_ST\_EN\_REG

```
#define AY_ST_EN_REG BIT(6)
```

Definition at line 222 of file [ICM20948\\_regs.h](#).

#### 7.25.1.72 AZ\_ST\_EN\_REG

```
#define AZ_ST_EN_REG BIT(5)
```

Definition at line 223 of file [ICM20948\\_regs.h](#).

#### 7.25.1.73 BANK1\_REG\_BANK\_SEL

```
#define BANK1_REG_BANK_SEL 0x7F /* mirror of REG_BANK_SEL */
```

Definition at line 163 of file [ICM20948\\_regs.h](#).

#### 7.25.1.74 BANK2\_REG\_BANK\_SEL

```
#define BANK2_REG_BANK_SEL 0x7F /* mirror of REG_BANK_SEL */
```

Definition at line 240 of file [ICM20948\\_regs.h](#).

#### 7.25.1.75 BANK3\_REG\_BANK\_SEL

```
#define BANK3_REG_BANK_SEL 0x7F /* mirror of REG_BANK_SEL */
```

Definition at line 296 of file [ICM20948\\_regs.h](#).

#### 7.25.1.76 BYPASS\_EN

```
#define BYPASS_EN BIT(1) /* I2C bypass to aux devices */
```

Definition at line 71 of file [ICM20948\\_regs.h](#).

### 7.25.1.77 CLK\_STOP

```
#define CLK_STOP 0x07 /* 7: Stop clock / timing gen reset */
```

Definition at line 321 of file [ICM20948\\_regs.h](#).

### 7.25.1.78 DEC3\_CFG\_MASK

```
#define DEC3_CFG_MASK (0x3u << DEC3_CFG_SHIFT)
```

Definition at line 225 of file [ICM20948\\_regs.h](#).

### 7.25.1.79 DEC3\_CFG\_SHIFT

```
#define DEC3_CFG_SHIFT 0
```

Definition at line 224 of file [ICM20948\\_regs.h](#).

### 7.25.1.80 DELAY\_ES\_SHADOW

```
#define DELAY_ES_SHADOW BIT(7)
```

Definition at line 256 of file [ICM20948\\_regs.h](#).

### 7.25.1.81 DELAY\_TIME\_EN

```
#define DELAY_TIME_EN BIT(7)
```

Definition at line 228 of file [ICM20948\\_regs.h](#).

### 7.25.1.82 DELAY\_TIMEH

```
#define DELAY_TIMEH 0x28
```

Definition at line 107 of file [ICM20948\\_regs.h](#).

### 7.25.1.83 DELAY\_TIMEL

```
#define DELAY_TIMEL 0x29
```

Definition at line 108 of file [ICM20948\\_regs.h](#).

### 7.25.1.84 dps1000

```
#define dps1000 2
```

Definition at line 398 of file [ICM20948\\_regs.h](#).

#### 7.25.1.85 dps2000

```
#define dps2000 3
```

Definition at line 399 of file [ICM20948\\_regs.h](#).

#### 7.25.1.86 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](#).

#### 7.25.1.87 dps500

```
#define dps500 1
```

Definition at line 397 of file [ICM20948\\_regs.h](#).

#### 7.25.1.88 EnhancedRegular

```
#define EnhancedRegular 2
```

Definition at line 456 of file [ICM20948\\_regs.h](#).

#### 7.25.1.89 EXT\_SLV\_SENS\_DATA\_00

```
#define EXT_SLV_SENS_DATA_00 0x3B
```

Definition at line 127 of file [ICM20948\\_regs.h](#).

#### 7.25.1.90 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](#).

#### 7.25.1.91 EXT\_SYNC\_SET\_MASK

```
#define EXT_SYNC_SET_MASK 0x0F
```

Definition at line 231 of file [ICM20948\\_regs.h](#).

#### 7.25.1.92 FIFO\_CFG

```
#define FIFO_CFG 0x76
```

Definition at line 138 of file [ICM20948\\_regs.h](#).

#### 7.25.1.93 FIFO\_COUNTH

```
#define FIFO_COUNTH 0x70
```

Definition at line 134 of file [ICM20948\\_regs.h](#).

#### 7.25.1.94 FIFO\_COUNTL

```
#define FIFO_COUNTL 0x71
```

Definition at line 135 of file [ICM20948\\_regs.h](#).

#### 7.25.1.95 FIFO\_EN\_1

```
#define FIFO_EN_1 0x66 /* SLV3..SLV0 FIFO_EN */
```

Definition at line 130 of file [ICM20948\\_regs.h](#).

#### 7.25.1.96 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](#).

#### 7.25.1.97 FIFO\_MODE

```
#define FIFO_MODE 0x69 /* FIFO_MODE[4:0] */
```

Definition at line 133 of file [ICM20948\\_regs.h](#).

#### 7.25.1.98 FIFO\_R\_W

```
#define FIFO_R_W 0x72
```

Definition at line 136 of file [ICM20948\\_regs.h](#).

#### 7.25.1.99 FIFO\_RST

```
#define FIFO_RST 0x68 /* FIFO_RESET[4:0] */
```

Definition at line 132 of file [ICM20948\\_regs.h](#).



**7.25.1.100 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](#).

**7.25.1.101 FSYNC\_INT\_MODE\_EN**

```
#define FSYNC_INT_MODE_EN BIT(2)
```

Definition at line 70 of file [ICM20948\\_regs.h](#).

**7.25.1.102 g16**

```
#define g16 3
```

Definition at line 329 of file [ICM20948\\_regs.h](#).

**7.25.1.103 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](#).

**7.25.1.104 g4**

```
#define g4 1
```

Definition at line 327 of file [ICM20948\\_regs.h](#).

**7.25.1.105 g8**

```
#define g8 2
```

Definition at line 328 of file [ICM20948\\_regs.h](#).

**7.25.1.106 GYRO\_AVGCFG\_MASK**

```
#define GYRO_AVGCFG_MASK (0x7u << GYRO_AVGCFG_SHIFT)
```

Definition at line 188 of file [ICM20948\\_regs.h](#).

**7.25.1.107 GYRO\_AVGCFG\_SHIFT**

```
#define GYRO_AVGCFG_SHIFT 0
```

Definition at line 187 of file [ICM20948\\_regs.h](#).

**7.25.1.108 GYRO\_BW\_MEDIUM**

```
#define GYRO_BW_MEDIUM 2 /* e.g., CFG 2/3 */
```

Definition at line 435 of file [ICM20948\\_regs.h](#).

**7.25.1.109 GYRO\_BW\_NARROW**

```
#define GYRO_BW_NARROW 3 /* e.g., CFG 4/5/6/7 */
```

Definition at line 436 of file [ICM20948\\_regs.h](#).

**7.25.1.110 GYRO\_BW\_ULTRA\_WIDE**

```
#define GYRO_BW_ULTRA_WIDE 0 /* bypass path */
```

Definition at line 433 of file [ICM20948\\_regs.h](#).

**7.25.1.111 GYRO\_BW\_WIDE**

```
#define GYRO_BW_WIDE 1 /* e.g., CFG 0/1 */
```

Definition at line 434 of file [ICM20948\\_regs.h](#).

**7.25.1.112 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](#).

**7.25.1.113 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](#).

**7.25.1.114 GYRO\_DLPF\_BASE\_HZ**

```
#define GYRO_DLPF_BASE_HZ 1100.0f
```

Definition at line 426 of file [ICM20948\\_regs.h](#).

**7.25.1.115 GYRO\_DLPF\_ODR\_HZ**

```
#define GYRO_DLPF_ODR_HZ(  
    rate_request)
```

**Value:**

```
(rate_request) /* symbolic; your driver computes DIV = round(1100/rate)-1 */
```

Definition at line 427 of file [ICM20948\\_regs.h](#).

**7.25.1.116 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](#).

**7.25.1.117 GYRO\_DLPFCFG\_1**

```
#define GYRO_DLPFCFG_1 1
```

Definition at line 416 of file [ICM20948\\_regs.h](#).

**7.25.1.118 GYRO\_DLPFCFG\_2**

```
#define GYRO_DLPFCFG_2 2
```

Definition at line 417 of file [ICM20948\\_regs.h](#).

**7.25.1.119 GYRO\_DLPFCFG\_3**

```
#define GYRO_DLPFCFG_3 3
```

Definition at line 418 of file [ICM20948\\_regs.h](#).

**7.25.1.120 GYRO\_DLPFCFG\_4**

```
#define GYRO_DLPFCFG_4 4
```

Definition at line 419 of file [ICM20948\\_regs.h](#).

**7.25.1.121 GYRO\_DLPFCFG\_5**

```
#define GYRO_DLPFCFG_5 5
```

Definition at line 420 of file [ICM20948\\_regs.h](#).

#### 7.25.1.122 GYRO\_DLPFCFG\_6

```
#define GYRO_DLPFCFG_6 6
```

Definition at line 421 of file [ICM20948\\_regs.h](#).

#### 7.25.1.123 GYRO\_DLPFCFG\_7

```
#define GYRO_DLPFCFG_7 7
```

Definition at line 422 of file [ICM20948\\_regs.h](#).

#### 7.25.1.124 GYRO\_DLPFCFG\_MASK

```
#define GYRO_DLPFCFG_MASK (0x7u << GYRO_DLPFCFG_SHIFT)
```

Definition at line 181 of file [ICM20948\\_regs.h](#).

#### 7.25.1.125 GYRO\_DLPFCFG\_SHIFT

```
#define GYRO_DLPFCFG_SHIFT 5
```

Definition at line 180 of file [ICM20948\\_regs.h](#).

#### 7.25.1.126 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](#).

#### 7.25.1.127 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](#).

#### 7.25.1.128 GYRO\_FCHOICE\_DLPF

```
#define GYRO_FCHOICE_DLPF 1
```

Definition at line 408 of file [ICM20948\\_regs.h](#).

**7.25.1.129 GYRO\_FS\_1000DPS**

```
#define GYRO_FS_1000DPS (2u << GYRO_FS_SEL_SHIFT)
```

Definition at line 178 of file [ICM20948\\_regs.h](#).

**7.25.1.130 GYRO\_FS\_2000DPS**

```
#define GYRO_FS_2000DPS (3u << GYRO_FS_SEL_SHIFT)
```

Definition at line 179 of file [ICM20948\\_regs.h](#).

**7.25.1.131 GYRO\_FS\_250DPS**

```
#define GYRO_FS_250DPS (0u << GYRO_FS_SEL_SHIFT)
```

Definition at line 176 of file [ICM20948\\_regs.h](#).

**7.25.1.132 GYRO\_FS\_500DPS**

```
#define GYRO_FS_500DPS (1u << GYRO_FS_SEL_SHIFT)
```

Definition at line 177 of file [ICM20948\\_regs.h](#).

**7.25.1.133 GYRO\_FS\_SEL\_MASK**

```
#define GYRO_FS_SEL_MASK (0x3u << GYRO_FS_SEL_SHIFT)
```

Definition at line 175 of file [ICM20948\\_regs.h](#).

**7.25.1.134 GYRO\_FS\_SEL\_SHIFT**

```
#define GYRO_FS_SEL_SHIFT 1
```

Definition at line 174 of file [ICM20948\\_regs.h](#).

**7.25.1.135 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](#).

**7.25.1.136 GYRO\_SMPLRT\_MIN\_HZ**

```
#define GYRO_SMPLRT_MIN_HZ 4.3f
```

Definition at line [425](#) of file [ICM20948\\_regs.h](#).

**7.25.1.137 GYRO\_XOUT\_H**

```
#define GYRO_XOUT_H 0x33
```

Definition at line [117](#) of file [ICM20948\\_regs.h](#).

**7.25.1.138 GYRO\_XOUT\_L**

```
#define GYRO_XOUT_L 0x34
```

Definition at line [118](#) of file [ICM20948\\_regs.h](#).

**7.25.1.139 GYRO\_YOUT\_H**

```
#define GYRO_YOUT_H 0x35
```

Definition at line [119](#) of file [ICM20948\\_regs.h](#).

**7.25.1.140 GYRO\_YOUT\_L**

```
#define GYRO_YOUT_L 0x36
```

Definition at line [120](#) of file [ICM20948\\_regs.h](#).

**7.25.1.141 GYRO\_ZOUT\_H**

```
#define GYRO_ZOUT_H 0x37
```

Definition at line [121](#) of file [ICM20948\\_regs.h](#).

**7.25.1.142 GYRO\_ZOUT\_L**

```
#define GYRO_ZOUT_L 0x38
```

Definition at line [122](#) of file [ICM20948\\_regs.h](#).

**7.25.1.143 HighAccuracy**

```
#define HighAccuracy 3
```

Definition at line [457](#) of file [ICM20948\\_regs.h](#).

**7.25.1.144 Hz10**

```
#define Hz10 3
```

Definition at line [443](#) of file [ICM20948\\_regs.h](#).

**7.25.1.145 Hz15**

```
#define Hz15 4
```

Definition at line [444](#) of file [ICM20948\\_regs.h](#).

**7.25.1.146 Hz2**

```
#define Hz2 0
```

- Friendly ODR presets you can map to dividers

Definition at line [440](#) of file [ICM20948\\_regs.h](#).

**7.25.1.147 Hz20**

```
#define Hz20 5
```

Definition at line [445](#) of file [ICM20948\\_regs.h](#).

**7.25.1.148 Hz25**

```
#define Hz25 6
```

Definition at line [446](#) of file [ICM20948\\_regs.h](#).

**7.25.1.149 Hz30**

```
#define Hz30 7
```

Definition at line [447](#) of file [ICM20948\\_regs.h](#).

**7.25.1.150 Hz6**

```
#define Hz6 1
```

Definition at line [441](#) of file [ICM20948\\_regs.h](#).

**7.25.1.151 Hz8**

```
#define Hz8 2
```

Definition at line 442 of file [ICM20948\\_regs.h](#).

**7.25.1.152 I2C\_MST\_CLK\_MASK**

```
#define I2C_MST_CLK_MASK 0x0F
```

Definition at line 253 of file [ICM20948\\_regs.h](#).

**7.25.1.153 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](#).

**7.25.1.154 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](#).

**7.25.1.155 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](#).

**7.25.1.156 I2C\_MST\_P\_NSR**

```
#define I2C_MST_P_NSR BIT(4)
```

Definition at line 252 of file [ICM20948\\_regs.h](#).

**7.25.1.157 I2C\_MST\_STATUS**

```
#define I2C_MST_STATUS 0x17 /* PASS_THROUGH, *_DONE, *_NACK, LOST_ARB */
```

Definition at line 86 of file [ICM20948\\_regs.h](#).



**7.25.1.158 I2C\_SLV0\_ADDR**

```
#define I2C_SLV0_ADDR 0x03 /* RNW + ID[6:0] */
```

Definition at line 263 of file [ICM20948\\_regs.h](#).

**7.25.1.159 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](#).

**7.25.1.160 I2C\_SLV0\_DELAY\_EN**

```
#define I2C_SLV0_DELAY_EN BIT(0)
```

Definition at line 261 of file [ICM20948\\_regs.h](#).

**7.25.1.161 I2C\_SLV0\_DO**

```
#define I2C_SLV0_DO 0x06
```

Definition at line 272 of file [ICM20948\\_regs.h](#).

**7.25.1.162 I2C\_SLV0\_REG**

```
#define I2C_SLV0_REG 0x04
```

Definition at line 265 of file [ICM20948\\_regs.h](#).

**7.25.1.163 I2C\_SLV1\_ADDR**

```
#define I2C_SLV1_ADDR 0x07
```

Definition at line 274 of file [ICM20948\\_regs.h](#).

**7.25.1.164 I2C\_SLV1\_CTRL**

```
#define I2C_SLV1_CTRL 0x09
```

Definition at line 276 of file [ICM20948\\_regs.h](#).

**7.25.1.165 I2C\_SLV1\_DELAY\_EN**

```
#define I2C_SLV1_DELAY_EN BIT(1)
```

Definition at line 260 of file [ICM20948\\_regs.h](#).

**7.25.1.166 I2C\_SLV1\_DO**

```
#define I2C_SLV1_DO 0x0A
```

Definition at line 277 of file [ICM20948\\_regs.h](#).

**7.25.1.167 I2C\_SLV1\_REG**

```
#define I2C_SLV1_REG 0x08
```

Definition at line 275 of file [ICM20948\\_regs.h](#).

**7.25.1.168 I2C\_SLV2\_ADDR**

```
#define I2C_SLV2_ADDR 0x0B
```

Definition at line 279 of file [ICM20948\\_regs.h](#).

**7.25.1.169 I2C\_SLV2\_CTRL**

```
#define I2C_SLV2_CTRL 0x0D
```

Definition at line 281 of file [ICM20948\\_regs.h](#).

**7.25.1.170 I2C\_SLV2\_DELAY\_EN**

```
#define I2C_SLV2_DELAY_EN BIT(2)
```

Definition at line 259 of file [ICM20948\\_regs.h](#).

**7.25.1.171 I2C\_SLV2\_DO**

```
#define I2C_SLV2_DO 0x0E
```

Definition at line 282 of file [ICM20948\\_regs.h](#).

**7.25.1.172 I2C\_SLV2\_REG**

```
#define I2C_SLV2_REG 0x0C
```

Definition at line 280 of file [ICM20948\\_regs.h](#).

**7.25.1.173 I2C\_SLV3\_ADDR**

```
#define I2C_SLV3_ADDR 0x0F
```

Definition at line 284 of file [ICM20948\\_regs.h](#).

**7.25.1.174 I2C\_SLV3\_CTRL**

```
#define I2C_SLV3_CTRL 0x11
```

Definition at line 286 of file [ICM20948\\_regs.h](#).

**7.25.1.175 I2C\_SLV3\_DELAY\_EN**

```
#define I2C_SLV3_DELAY_EN BIT(3)
```

Definition at line 258 of file [ICM20948\\_regs.h](#).

**7.25.1.176 I2C\_SLV3\_DO**

```
#define I2C_SLV3_DO 0x12
```

Definition at line 287 of file [ICM20948\\_regs.h](#).

**7.25.1.177 I2C\_SLV3\_REG**

```
#define I2C_SLV3_REG 0x10
```

Definition at line 285 of file [ICM20948\\_regs.h](#).

**7.25.1.178 I2C\_SLV4\_ADDR**

```
#define I2C_SLV4_ADDR 0x13
```

Definition at line 289 of file [ICM20948\\_regs.h](#).

**7.25.1.179 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](#).

**7.25.1.180 I2C\_SLV4\_DELAY\_EN**

```
#define I2C_SLV4_DELAY_EN BIT(4)
```

Definition at line 257 of file [ICM20948\\_regs.h](#).

**7.25.1.181 I2C\_SLV4\_DI**

```
#define I2C_SLV4_DI 0x17
```

Definition at line 294 of file [ICM20948\\_regs.h](#).

**7.25.1.182 I2C\_SLV4\_DLY\_MASK**

```
#define I2C_SLV4_DLY_MASK 0x1F
```

Definition at line 292 of file [ICM20948\\_regs.h](#).

**7.25.1.183 I2C\_SLV4\_DO**

```
#define I2C_SLV4_DO 0x16
```

Definition at line 293 of file [ICM20948\\_regs.h](#).

**7.25.1.184 I2C\_SLV4\_REG**

```
#define I2C_SLV4_REG 0x14
```

Definition at line 290 of file [ICM20948\\_regs.h](#).

**7.25.1.185 I2C\_SLVx\_BYTE\_SW**

```
#define I2C_SLVx_BYTE_SW BIT(6)
```

Definition at line 268 of file [ICM20948\\_regs.h](#).

**7.25.1.186 I2C\_SLVx\_EN**

```
#define I2C_SLVx_EN BIT(7)
```

Definition at line 267 of file [ICM20948\\_regs.h](#).

**7.25.1.187 I2C\_SLVx\_GRP**

```
#define I2C_SLVx_GRP BIT(4)
```

Definition at line 270 of file [ICM20948\\_regs.h](#).

**7.25.1.188 I2C\_SLVx LENG\_MASK**

```
#define I2C_SLVx LENG_MASK 0x0F
```

Definition at line 271 of file [ICM20948\\_regs.h](#).

**7.25.1.189 I2C\_SLVx\_REG\_DIS**

```
#define I2C_SLVx_REG_DIS BIT(5)
```

Definition at line 269 of file [ICM20948\\_regs.h](#).

**7.25.1.190 I2C\_SLVx\_RNW**

```
#define I2C_SLVx_RNW BIT(7)
```

Definition at line 264 of file [ICM20948\\_regs.h](#).

**7.25.1.191 INT1\_ACTL**

```
#define INT1_ACTL BIT(7) /* active low */
```

Definition at line 65 of file [ICM20948\\_regs.h](#).

**7.25.1.192 INT1\_LATCH\_INT\_EN**

```
#define INT1_LATCH_INT_EN BIT(5) /* latch until status read */
```

Definition at line 67 of file [ICM20948\\_regs.h](#).

**7.25.1.193 INT1\_OPEN**

```
#define INT1_OPEN BIT(6) /* open-drain */
```

Definition at line 66 of file [ICM20948\\_regs.h](#).

**7.25.1.194 INT\_ACTL\_FSYNC**

```
#define INT_ACTL_FSYNC BIT(3)
```

Definition at line 69 of file [ICM20948\\_regs.h](#).

**7.25.1.195 INT\_ANYRD\_2CLEAR**

```
#define INT_ANYRD_2CLEAR BIT(4)
```

Definition at line 68 of file [ICM20948\\_regs.h](#).

**7.25.1.196 INT\_DMP\_INT1\_EN**

```
#define INT_DMP_INT1_EN BIT(1)
```

Definition at line 77 of file [ICM20948\\_regs.h](#).

**7.25.1.197 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](#).

#### 7.25.1.198 INT\_ENABLE\_1

```
#define INT_ENABLE_1 0x11 /* RAW_DATA_0_RDY_EN */
```

Definition at line 80 of file [ICM20948\\_regs.h](#).

#### 7.25.1.199 INT\_ENABLE\_2

```
#define INT_ENABLE_2 0x12 /* FIFO_OVERFLOW_EN[4:0] */
```

Definition at line 83 of file [ICM20948\\_regs.h](#).

#### 7.25.1.200 INT\_ENABLE\_3

```
#define INT_ENABLE_3 0x13 /* FIFO_WM_EN[4:0] */
```

Definition at line 84 of file [ICM20948\\_regs.h](#).

#### 7.25.1.201 INT\_I2C\_MST\_INT\_EN

```
#define INT_I2C_MST_INT_EN BIT(0)
```

Definition at line 78 of file [ICM20948\\_regs.h](#).

#### 7.25.1.202 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](#).

#### 7.25.1.203 INT\_PLL\_RDY\_EN

```
#define INT_PLL_RDY_EN BIT(2)
```

Definition at line 76 of file [ICM20948\\_regs.h](#).

#### 7.25.1.204 INT\_RAW\_DATA\_0\_RDY\_EN

```
#define INT_RAW_DATA_0_RDY_EN BIT(0)
```

Definition at line 81 of file [ICM20948\\_regs.h](#).

**7.25.1.205 INT\_REG\_WOF\_EN**

```
#define INT_REG_WOF_EN BIT(7)
```

Definition at line 74 of file [ICM20948\\_regs.h](#).

**7.25.1.206 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](#).

**7.25.1.207 INT\_STATUS\_1**

```
#define INT_STATUS_1 0x1A /* RAW_DATA_0_RDY_INT */
```

Definition at line 102 of file [ICM20948\\_regs.h](#).

**7.25.1.208 INT\_STATUS\_2**

```
#define INT_STATUS_2 0x1B /* FIFO_OVERFLOW_INT[4:0] */
```

Definition at line 104 of file [ICM20948\\_regs.h](#).

**7.25.1.209 INT\_STATUS\_3**

```
#define INT_STATUS_3 0x1C /* FIFO_WM_INT[4:0] */
```

Definition at line 105 of file [ICM20948\\_regs.h](#).

**7.25.1.210 INT\_WOM\_INT\_EN**

```
#define INT_WOM_INT_EN BIT(3)
```

Definition at line 75 of file [ICM20948\\_regs.h](#).

**7.25.1.211 INTERNAL\_20MHZ**

```
#define INTERNAL_20MHZ 0x00 /* 0: Internal 20 MHz RC */
```

Definition at line 319 of file [ICM20948\\_regs.h](#).

#### 7.25.1.212 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](#).

#### 7.25.1.213 LP\_ACCEL\_CYCLE

```
#define LP_ACCEL_CYCLE BIT(5)
```

Definition at line 47 of file [ICM20948\\_regs.h](#).

#### 7.25.1.214 LP\_CONFIG

```
#define LP_CONFIG 0x05 /* I2C_MST_CYCLE ACCEL_CYCLE GYRO_CYCLE - */
```

Definition at line 45 of file [ICM20948\\_regs.h](#).

#### 7.25.1.215 LP\_GYRO\_CYCLE

```
#define LP_GYRO_CYCLE BIT(4)
```

Definition at line 48 of file [ICM20948\\_regs.h](#).

#### 7.25.1.216 LP\_I2C\_MST\_CYCLE

```
#define LP_I2C_MST_CYCLE BIT(6)
```

Definition at line 46 of file [ICM20948\\_regs.h](#).

#### 7.25.1.217 MOD\_CTRL\_USR

```
#define MOD_CTRL_USR 0x54 /* REG_LP_DMP_EN */
```

Definition at line 237 of file [ICM20948\\_regs.h](#).

#### 7.25.1.218 MST\_LOST\_ARB

```
#define MST_LOST_ARB BIT(5)
```

Definition at line 89 of file [ICM20948\\_regs.h](#).



**7.25.1.219 MST\_ODR\_CFG\_MASK**

```
#define MST_ODR_CFG_MASK 0x0F
```

Definition at line 248 of file [ICM20948\\_regs.h](#).

**7.25.1.220 MST\_PASS\_THROUGH**

```
#define MST_PASS_THROUGH BIT(7)
```

Definition at line 87 of file [ICM20948\\_regs.h](#).

**7.25.1.221 MST\_SLV0\_NACK**

```
#define MST_SLV0_NACK BIT(0)
```

Definition at line 94 of file [ICM20948\\_regs.h](#).

**7.25.1.222 MST\_SLV1\_NACK**

```
#define MST_SLV1_NACK BIT(1)
```

Definition at line 93 of file [ICM20948\\_regs.h](#).

**7.25.1.223 MST\_SLV2\_NACK**

```
#define MST_SLV2_NACK BIT(2)
```

Definition at line 92 of file [ICM20948\\_regs.h](#).

**7.25.1.224 MST\_SLV3\_NACK**

```
#define MST_SLV3_NACK BIT(3)
```

Definition at line 91 of file [ICM20948\\_regs.h](#).

**7.25.1.225 MST\_SLV4\_DONE**

```
#define MST_SLV4_DONE BIT(6)
```

Definition at line 88 of file [ICM20948\\_regs.h](#).

**7.25.1.226 MST\_SLV4\_NACK**

```
#define MST_SLV4_NACK BIT(4)
```

Definition at line 90 of file [ICM20948\\_regs.h](#).

**7.25.1.227 MULT\_MST\_EN**

```
#define MULT_MST_EN BIT(7)
```

Definition at line 251 of file [ICM20948\\_regs.h](#).

**7.25.1.228 ODR\_ALIGN\_EN**

```
#define ODR_ALIGN_EN 0x09 /* ODR_ALIGN_EN */
```

Definition at line 197 of file [ICM20948\\_regs.h](#).

**7.25.1.229 ODR\_ALIGN\_EN\_BIT**

```
#define ODR_ALIGN_EN_BIT BIT(0)
```

Definition at line 198 of file [ICM20948\\_regs.h](#).

**7.25.1.230 PWR\_CLKSEL\_AUTO**

```
#define PWR_CLKSEL_AUTO 0x01 /* typical: auto selects best source */
```

Definition at line 57 of file [ICM20948\\_regs.h](#).

**7.25.1.231 PWR\_CLKSEL\_INT\_20MHZ**

```
#define PWR_CLKSEL_INT_20MHZ 0x01
```

Definition at line 56 of file [ICM20948\\_regs.h](#).

**7.25.1.232 PWR\_CLKSEL\_MASK**

```
#define PWR_CLKSEL_MASK 0x07
```

Definition at line 55 of file [ICM20948\\_regs.h](#).

**7.25.1.233 PWR\_DEVICE\_RESET**

```
#define PWR_DEVICE_RESET BIT(7)
```

Definition at line 51 of file [ICM20948\\_regs.h](#).

**7.25.1.234 PWR\_DISABLE\_ACCEL**

```
#define PWR_DISABLE_ACCEL BIT(3)
```

Definition at line 60 of file [ICM20948\\_regs.h](#).

**7.25.1.235 PWR\_DISABLE\_GYRO**

```
#define PWR_DISABLE_GYRO BIT(0)
```

Definition at line 61 of file [ICM20948\\_regs.h](#).

**7.25.1.236 PWR\_LP\_EN**

```
#define PWR_LP_EN BIT(5)
```

Definition at line 53 of file [ICM20948\\_regs.h](#).

**7.25.1.237 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](#).

**7.25.1.238 PWR\_MGMT\_2**

```
#define PWR_MGMT_2 0x07 /* - DISABLE_ACCEL DISABLE_GYRO */
```

Definition at line 59 of file [ICM20948\\_regs.h](#).

**7.25.1.239 PWR\_SLEEP**

```
#define PWR_SLEEP BIT(6)
```

Definition at line 52 of file [ICM20948\\_regs.h](#).

**7.25.1.240 PWR\_TEMP\_DIS**

```
#define PWR_TEMP_DIS BIT(3)
```

Definition at line 54 of file [ICM20948\\_regs.h](#).

**7.25.1.241 REG\_BANK\_SEL**

```
#define REG_BANK_SEL 0x7F /* USER_BANK[1:0] */
```

Definition at line 140 of file [ICM20948\\_regs.h](#).

**7.25.1.242 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](#).

**7.25.1.243 REG\_BANK\_SEL\_USER\_BANK\_SHIFT**

```
#define REG_BANK_SEL_USER_BANK_SHIFT 4
```

Definition at line 302 of file [ICM20948\\_regs.h](#).

**7.25.1.244 REG\_LP\_DMP\_EN**

```
#define REG_LP_DMP_EN BIT(7)
```

Definition at line 238 of file [ICM20948\\_regs.h](#).

**7.25.1.245 Regular**

```
#define Regular 1
```

Definition at line 455 of file [ICM20948\\_regs.h](#).

**7.25.1.246 SELF\_TEST\_X\_ACCEL**

```
#define SELF_TEST_X_ACCEL 0x0E /* XA_ST_DATA[7:0] */
```

Definition at line 150 of file [ICM20948\\_regs.h](#).

**7.25.1.247 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](#).

**7.25.1.248 SELF\_TEST\_Y\_ACCEL**

```
#define SELF_TEST_Y_ACCEL 0x0F /* YA_ST_DATA[7:0] */
```

Definition at line 151 of file [ICM20948\\_regs.h](#).

**7.25.1.249 SELF\_TEST\_Y\_GYRO**

```
#define SELF_TEST_Y_GYRO 0x03 /* YG_ST_DATA[7:0] */
```

Definition at line 148 of file [ICM20948\\_regs.h](#).

**7.25.1.250 SELF\_TEST\_Z\_ACCEL**

```
#define SELF_TEST_Z_ACCEL 0x10 /* ZA_ST_DATA[7:0] */
```

Definition at line 152 of file [ICM20948\\_regs.h](#).

**7.25.1.251 SELF\_TEST\_Z\_GYRO**

```
#define SELF_TEST_Z_GYRO 0x04 /* ZG_ST_DATA[7:0] */
```

Definition at line 149 of file [ICM20948\\_regs.h](#).

**7.25.1.252 STS\_DMP\_INT1**

```
#define STS_DMP_INT1 BIT(1)
```

Definition at line 99 of file [ICM20948\\_regs.h](#).

**7.25.1.253 STS\_I2C\_MST\_INT**

```
#define STS_I2C_MST_INT BIT(0)
```

Definition at line 100 of file [ICM20948\\_regs.h](#).

**7.25.1.254 STS\_PLL\_RDY\_INT**

```
#define STS_PLL_RDY_INT BIT(2)
```

Definition at line 98 of file [ICM20948\\_regs.h](#).

**7.25.1.255 STS\_RAW\_DATA\_0\_RDY\_INT**

```
#define STS_RAW_DATA_0_RDY_INT BIT(0)
```

Definition at line 103 of file [ICM20948\\_regs.h](#).

**7.25.1.256 STS\_WOM\_INT**

```
#define STS_WOM_INT BIT(3)
```

Definition at line 97 of file [ICM20948\\_regs.h](#).

**7.25.1.257 TEMP\_CONFIG**

```
#define TEMP_CONFIG 0x53 /* TEMP_DLPFCFG[2:0] */
```

Definition at line 233 of file [ICM20948\\_regs.h](#).

**7.25.1.258 TEMP\_DLPFCFG\_MASK**

```
#define TEMP_DLPFCFG_MASK (0x7u << TEMP_DLPFCFG_SHIFT)
```

Definition at line 235 of file [ICM20948\\_regs.h](#).

**7.25.1.259 TEMP\_DLPFCFG\_SHIFT**

```
#define TEMP_DLPFCFG_SHIFT 5
```

Definition at line 234 of file [ICM20948\\_regs.h](#).

**7.25.1.260 TEMP\_OUT\_H**

```
#define TEMP_OUT_H 0x39
```

Definition at line 123 of file [ICM20948\\_regs.h](#).

**7.25.1.261 TEMP\_OUT\_L**

```
#define TEMP_OUT_L 0x3A
```

Definition at line 124 of file [ICM20948\\_regs.h](#).

**7.25.1.262 TIMEBASE\_CORRECTION\_PLL**

```
#define TIMEBASE_CORRECTION_PLL 0x28 /* TBC_PLL[7:0] */
```

Definition at line 161 of file [ICM20948\\_regs.h](#).

**7.25.1.263 USER\_BANK\_0**

```
#define USER_BANK_0 BANK0
```

Definition at line 304 of file [ICM20948\\_regs.h](#).

**7.25.1.264 USER\_BANK\_1**

```
#define USER_BANK_1 BANK1
```

Definition at line 305 of file [ICM20948\\_regs.h](#).

**7.25.1.265 USER\_BANK\_2**

```
#define USER_BANK_2 BANK2
```

Definition at line 306 of file [ICM20948\\_regs.h](#).

**7.25.1.266 USER\_BANK\_3**

```
#define USER_BANK_3 BANK3
```

Definition at line 307 of file [ICM20948\\_regs.h](#).

**7.25.1.267 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](#).

**7.25.1.268 USER\_CTRL\_DMP\_EN**

```
#define USER_CTRL_DMP_EN BIT(7)
```

Definition at line 37 of file [ICM20948\\_regs.h](#).

**7.25.1.269 USER\_CTRL\_DMP\_RST**

```
#define USER_CTRL_DMP_RST BIT(3)
```

Definition at line 41 of file [ICM20948\\_regs.h](#).

**7.25.1.270 USER\_CTRL\_FIFO\_EN**

```
#define USER_CTRL_FIFO_EN BIT(6)
```

Definition at line 38 of file [ICM20948\\_regs.h](#).

**7.25.1.271 USER\_CTRL\_I2C\_IF\_DIS**

```
#define USER_CTRL_I2C_IF_DIS BIT(4)
```

Definition at line 40 of file [ICM20948\\_regs.h](#).

**7.25.1.272 USER\_CTRL\_I2C\_MST\_EN**

```
#define USER_CTRL_I2C_MST_EN BIT(5)
```

Definition at line 39 of file [ICM20948\\_regs.h](#).

**7.25.1.273 USER\_CTRL\_I2C\_MST\_RST**

```
#define USER_CTRL_I2C_MST_RST BIT(1)
```

Definition at line 43 of file [ICM20948\\_regs.h](#).

**7.25.1.274 USER\_CTRL\_SRAM\_RST**

```
#define USER_CTRL_SRAM_RST BIT(2)
```

Definition at line 42 of file [ICM20948\\_regs.h](#).

**7.25.1.275 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](#).

**7.25.1.276 WHO\_AM\_I\_VAL**

```
#define WHO_AM_I_VAL 0xEA
```

Definition at line 33 of file [ICM20948\\_regs.h](#).

**7.25.1.277 WOF\_DEGLITCH\_EN**

```
#define WOF_DEGLITCH_EN BIT(6)
```

Definition at line 229 of file [ICM20948\\_regs.h](#).



**7.25.1.278 WOF\_EDGE\_INT**

```
#define WOF_EDGE_INT BIT(5)
```

Definition at line 230 of file [ICM20948\\_regs.h](#).

**7.25.1.279 XA\_OFFS\_H**

```
#define XA_OFFS_H 0x14 /* XA_OFFS[14:7] */
```

Definition at line 154 of file [ICM20948\\_regs.h](#).

**7.25.1.280 XA\_OFFS\_L**

```
#define XA_OFFS_L 0x15 /* XA_OFFS[6:0] */
```

Definition at line 155 of file [ICM20948\\_regs.h](#).

**7.25.1.281 XG\_OFFS\_USRH**

```
#define XG_OFFS_USRH 0x03
```

Definition at line 190 of file [ICM20948\\_regs.h](#).

**7.25.1.282 XG\_OFFS\_USRL**

```
#define XG_OFFS_USRL 0x04
```

Definition at line 191 of file [ICM20948\\_regs.h](#).

**7.25.1.283 XGYRO\_CTEN**

```
#define XGYRO_CTEN BIT(5)
```

Definition at line 184 of file [ICM20948\\_regs.h](#).

**7.25.1.284 YA\_OFFS\_H**

```
#define YA_OFFS_H 0x17 /* YA_OFFS[14:7] */
```

Definition at line 156 of file [ICM20948\\_regs.h](#).

**7.25.1.285 YA\_OFFS\_L**

```
#define YA_OFFS_L 0x18 /* YA_OFFS[6:0] */
```

Definition at line 157 of file [ICM20948\\_regs.h](#).

**7.25.1.286 YG\_OFFS\_USRH**

```
#define YG_OFFS_USRH 0x05
```

Definition at line 192 of file [ICM20948\\_regs.h](#).

**7.25.1.287 YG\_OFFS\_USRL**

```
#define YG_OFFS_USRL 0x06
```

Definition at line 193 of file [ICM20948\\_regs.h](#).

**7.25.1.288 YGYRO\_CTEN**

```
#define YGYRO_CTEN BIT(4)
```

Definition at line 185 of file [ICM20948\\_regs.h](#).

**7.25.1.289 ZA\_OFFS\_H**

```
#define ZA_OFFS_H 0x1A /* ZA_OFFS[14:7] */
```

Definition at line 158 of file [ICM20948\\_regs.h](#).

**7.25.1.290 ZA\_OFFS\_L**

```
#define ZA_OFFS_L 0x1B /* ZA_OFFS[6:0] */
```

Definition at line 159 of file [ICM20948\\_regs.h](#).

**7.25.1.291 ZG\_OFFS\_USRH**

```
#define ZG_OFFS_USRH 0x07
```

Definition at line 194 of file [ICM20948\\_regs.h](#).

**7.25.1.292 ZG\_OFFS\_USRL**

```
#define ZG_OFFS_USRL 0x08
```

Definition at line 195 of file [ICM20948\\_regs.h](#).

**7.25.1.293 ZGYRO\_CTEN**

```
#define ZGYRO_CTEN BIT(3)
```

Definition at line 186 of file [ICM20948\\_regs.h](#).

## 7.26 ICM20948\_reg.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 /** - ±2g / ±4g / ±8g / ±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 + SMP_LRT_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 */
```