XM125 I^{2}C Cargo Example Application

User Guide

XM125 I^{2}C Cargo Example Application

User Guide

Author: Acconeer AB

Version:a121-v1.12.0

Acconeer AB October 15, 2025

Contents

1 Acconeer SDK Documentation Overview

To better understand what SDK document to use, a summary of the documents are shown in the table below.

Table 1: SDK document overview.

Name	Description	When to use
RSS API documentation (html)		
rss_api	The complete C API documentation.	- RSS application implementation - Understanding RSS API functions
User guides (PDF)		
A121 Assembly Test	Describes the Acconeer assembly test functionality.	- Bring-up of HW/SW - Production test implementation
A121 Breathing Reference Application	Describes the functionality of the Breathing Reference Application.	- Working with the Breathing Reference Application
A121 Distance Detector	Describes usage and algorithms of the Distance Detector.	- Working with the Distance Detector
A121 SW Integration	Describes how to implement each integration function needed to use the Acconeer sensor.	- SW implementation of custom HW integration
A121 Presence Detector	Describes usage and algorithms of the Presence Detector.	- Working with the Presence Detector
A121 Smart Presence Reference Application	Describes the functionality of the Smart Presence Reference Application.	- Working with the Smart Presence Reference Application
A121 Sparse IQ Service	Describes usage of the Sparse IQ Service.	- Working with the Sparse IQ Service
A121 Tank Level Reference Application	Describes the functionality of the Tank Level Reference Application.	- Working with the Tank Level Reference Application
A121 Touchless Button Reference Application	Describes the functionality of the Touchless Button Reference Application.	- Working with the Touchless Button Reference Application
A121 Parking Reference Application	Describes the functionality of the Parking Reference Application.	- Working with the Parking Reference Application
A121 STM32CubeIDE	Describes the flow of taking an Acconeer SDK and integrate into STM32CubeIDE.	- Using STM32CubeIDE
A121 Raspberry Pi Software	Describes how to develop for Raspberry Pi.	- Working with Raspberry Pi
A121 Ripple	Describes how to develop for Ripple.	- Working with Ripple on Raspberry Pi
A121 ESP32 User Guide	Describes how to develop with A121 and ESP32 targets.	- Working with ESP32 targets
XM125 Software	Describes how to develop for XM125.	- Working with XM125
XM126 Software	Describes how to develop for XM126.	- Working with XM126
I2C Distance Detector	Describes the functionality of the I2C Distance Detector Application.	- Working with the I2C Distance Detector Application
I2C Presence Detector	Describes the functionality of the I2C Presence Detector Application.	- Working with the I2C Presence Detector Application
I2C Breathing Reference Application	Describes the functionality of the I2C Breathing Reference Application.	- Working with the I2C Breathing Reference Application
I2C Cargo Example Application	Describes the functionality of the I2C Cargo Example Application.	- Working with the I2C Cargo Example Application
A121 Radar Data and Control (PDF)		
A121 Radar Data and Control	Describes different aspects of the Acconeer offer, for example radar principles and how to configure	- To understand the Acconeer sensor - Use case evaluation
Readme (txt)		
README	Various target specific information and links	- After SDK download

2 I^{2}C Cargo Example Application

The I^{2}C Cargo Example Application is an application that implements the Acconeer Cargo Example Application with a register-based I^{2}C interface.

The functionality of the Cargo Example Application is described in Acconeer Docs.

Note: Some of the registers have a different scale in the I^{2}C Cargo Example Application. For example, millimeters can used instead of meters. All such deviations are clearly marked in each register's description. This is to make it easier to handle the register values as integers.

2.1 I^{2}C Address Configuration

The device has a configurable I^{2}C address. The address is selected depending on the state of the **I2C_ADDR** pin according to the following table:

Connected to GND	0x51
Not Connected	0x52
Connected to VIN	0x53

2.2 I2C Speed

The device supports I2C speed up to 100kbps in Standard Mode and up to 400kbps in Fast Mode.

2.3 Usage

The module must be ready before the host starts I^{2}C communication.

The module will enter ready state by following this procedure.

- Set **WAKE_UP** pin of the module HIGH.
- Wait for module to be ready, this is indicated by the **MCU_INT** pin being HIGH.
- Start I^{2}C communication.

The module will enter a low power state by following this procedure.

- Wait for module to be ready, this is indicated by the **MCU_INT** pin being HIGH.
- Set the **WAKE_UP** pin of the module LOW.
- Wait for ready signal, the **MCU_INT** pin, to become LOW.

2.3.1 Read App Status

The status of the module can be acquired by reading the *App Status* register, The most important bits are the **Busy** and **Error** bits.

The **Busy** bit must not be set when a new command is written. If any of the **Error** bits are set the module will not accept any commands except the **RESET_MODULE** command.

2.3.2 Writing a command

A command is written to the *Command* register. When a command is written the **Busy** bit in the *App Status* register is set and it will be cleared automatically when the command has finished.

2.3.3 Setup and Start Application

Before the module can perform utilization-level measurements and/or presence measurements, it must be configured. The following steps is an example of how this can be achieved.

Note: The configuration parameters can not be changed after a **APPLY_CONFIGURATION** command. If reconfiguration is needed the module must be restarted by writing **RESET_MODULE** to the *Command* register.

1. Power on module
2. Read *Application Status* register and verify that neither **Busy** nor **Error** bits are set.
3. Write configuration to configuration registers, for example the *Container size* register.

4. Write **APPLY_CONFIGURATION** to *Command* register.
5. Poll *Application Status* until **Busy** bit is cleared.
6. Verify that no **Error** bits are set in the *Application Status* register.
7. Write **MEASURE_UTILIZATION_LEVEL** (or **MEASURE_PRESENCE**) to *Command* register.
8. Poll *Application Status* until **Busy** bit is cleared.
9. Verify that no **Error** bits are set in the *Application Status* register.
10. Read *Result Header* register
    - If **UTILIZATION_LEVEL_VALID** or **PRESENCE_VALID** is set a measurement of that kind has successfully been made.
    - If **APP_ERROR** is set an error has occurred, restart module with the **RESET_MODULE** command.
    - If utilization level measurement was valid, details from the measurement can be read in the registers *Utilization Distance*, *Utilization Level (mm)* & *Utilization Level (%)*.
    - If presence measurement was valid, details from the measurement can be read in the registers *Presence Detected*, *Max Inter Presence Score* & *Max Intra Presence Score*.
11. Go to step 7. (Write command **MEASURE_UTILIZATION_LEVEL** or **MEASURE_PRESENCE**)

2.4 Advanced Usage

2.4.1 Debug UART logs

UART logging can be enabled on the DEBUG UART by writing **ENABLE_UART_LOGS** to the *Command* register.

The application configuration can be logged on the UART by writing **LOG_CONFIGURATION** to the *Command* register.

UART logging can be disabled by writing **DISABLE_UART_LOGS** to the *Command* register.

2.4.2 Reset Module

The module can be restarted by writing **RESET_MODULE** to the *Command* register.

After the restart the application must be configured again.

3 Register Protocol

3.1 I^{2}C Slave Address

The default slave address is 0x52.

3.2 Protocol Byte Order

Both register address, 16-bit, and register data, 32-bit, are sent in big endian byte order.

3.2.1 I^{2}C Write Register(s)

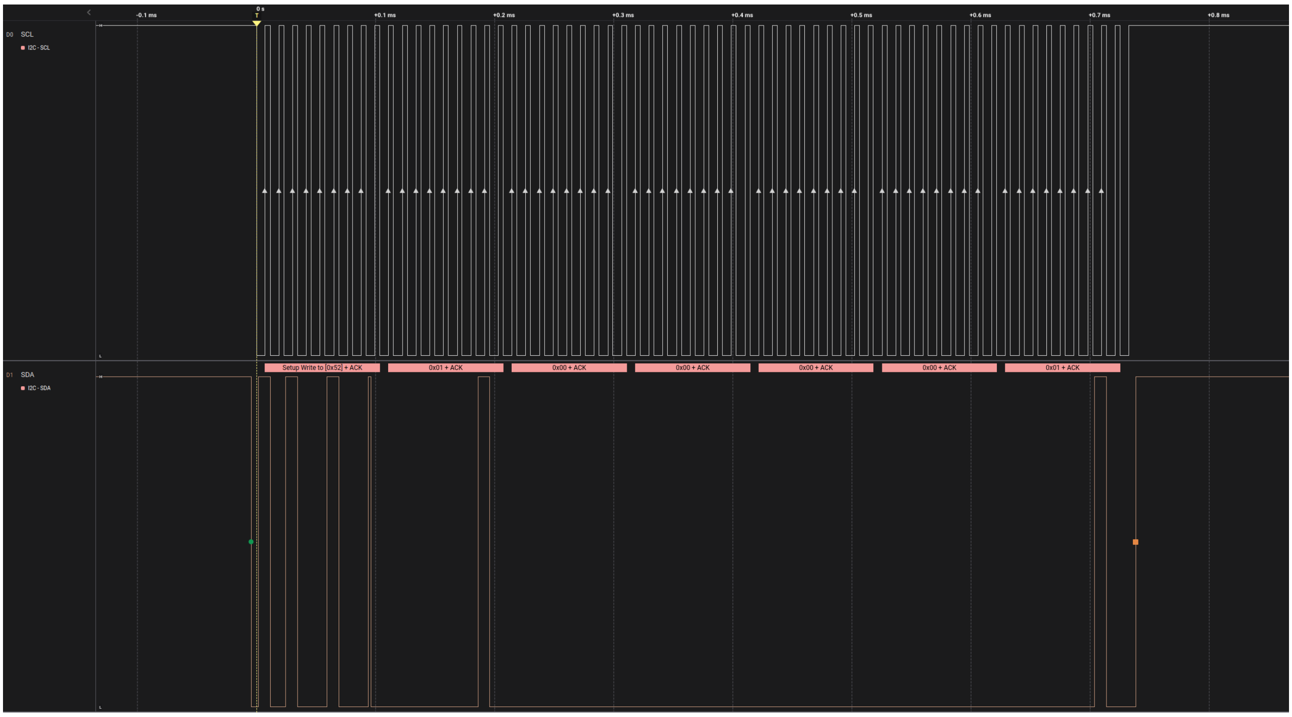
A write register operation consists of an I^{2}C write of two address bytes and four data bytes for each register to write. Several registers can be written in the same I^{2}C transaction, the register address will be incremented by one for each four data bytes.

Example 1: Writing six bytes will write one register, two address bytes and four data bytes.

Example 2: Writing 18 bytes will write four registers, two address bytes and 16 data bytes.

Example operation, write 0x11223344 to address 0x0025.

Description	Data
I^{2}C Start Condition	
Slave Address + Write	0x52 + W
Address to slave [15:8]	0x00
Address to slave [7:0]	0x25
Data to slave [31:24]	0x11
Data to slave [23:16]	0x22
Data to slave [15:8]	0x33
Data to slave [7:0]	0x44
I^{2}C Stop Condition	



Example Waveform: Write register with address 0x0100, the data sent from the master to the slave is 0x00000001

3.2.2 I^{2}C Read Register(s)

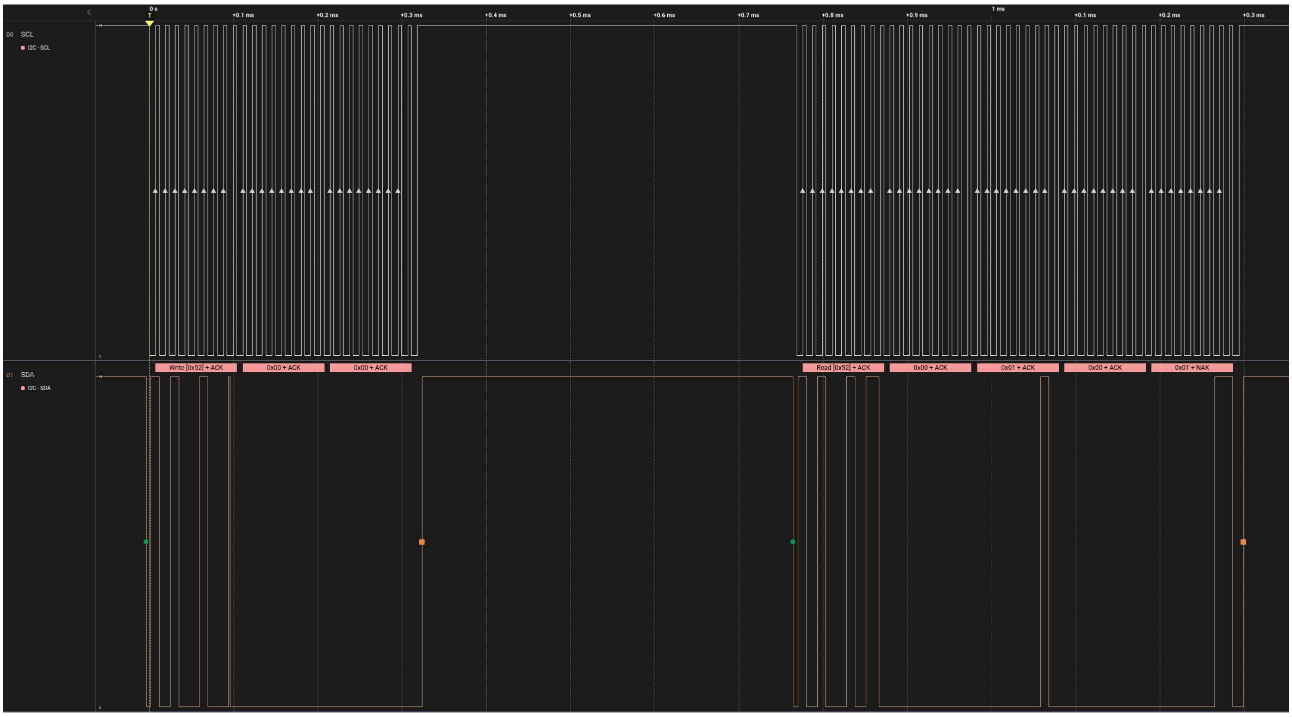
A read register operation consists of an I^{2}C write of two address bytes followed by an I^{2}C read of four data bytes for each register to read. Several registers can be read in the same I^{2}C transaction, the register address will be incremented by one for each four data bytes.

Example 1: Writing two bytes and reading four bytes will read one register.

Example 2: Writing two bytes and reading 16 bytes will read four registers.

Example operation, read 0x12345678 from address 0x0003.

Description	Data
I^{2}C Start Condition	
Slave Address + Write	0x52 + W
Address to slave [15:8]	0x00
Address to slave [7:0]	0x03
I^{2}C Stop Condition	
I^{2}C Start Condition	
Slave Address + Read	0x52 + R
Data from slave [31:24]	0x12
Data from slave [23:16]	0x34
Data from slave [15:8]	0x56
Data from slave [7:0]	0x78
I^{2}C Stop Condition	

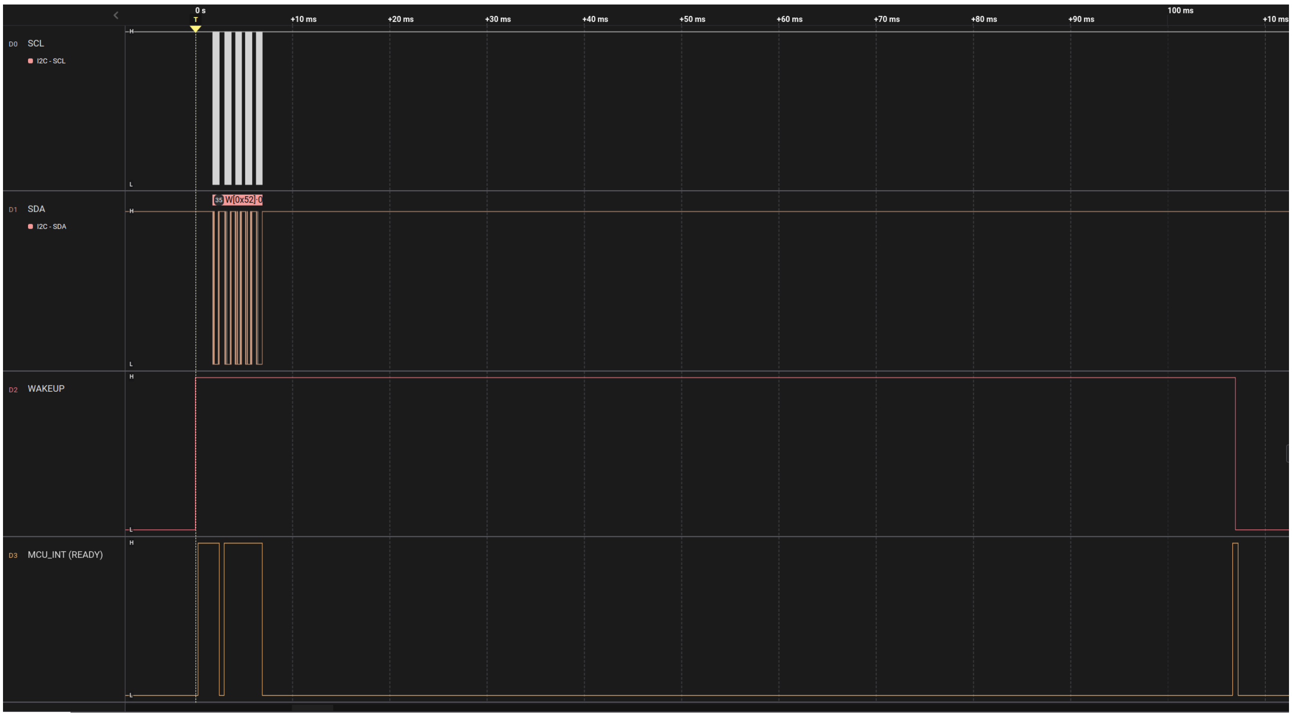


Example Waveform: Read register with address 0, the data sent from the slave to the master is 0x00010001

3.3 Register Protocol - Low Power Mode

3.3.1 I^{2}C Communication with Low Power Mode

Low power example



Low Power Example: Magnification of Wake up, Setup, & Power down

4 File Structure

The I^{2}C Cargo Example Application consists of the following files.

```
└── Src
    ├── applications
    │   └── i2c
    │       ├── acc_reg_protocol.c ...  A generic protocol handler
    │       │                           implementation
    │       ├── example_cargo_reg_protocol.c ...  The specific register
    │       │                                     protocol setup for the I2C
    │       │                                     Cargo Example Application
    │       ├── example_cargo_reg_protocol_access.c ...  The register read- and
    │       │                                            write access functions
    │       │                                            for the I2C Cargo Example
    │       │                                            Application
    │       ├── i2c_application_system_stm32.c ...  System functions, such as
    │       │                                       I2C handling, GPIO control
    │       │                                       and low power state
    │       └── i2c_example_cargo.c ...  The I2C Cargo Example
    │                                    Application
    └── use_cases
        └── example_apps
            └── example_cargo.c ...  The Cargo Example
                                     Application
└── Inc
    ├── acc_reg_protocol.h
    ├── example_cargo.h
    ├── example_cargo_reg_protocol.h
    ├── i2c_application_system.h
    └── i2c_example_cargo.h
```

5 Embedded Host Example

This is an example implementation of the host read and write register functions using the STM32 SDK.

5.1 Register Read/Write functions

```
#include <inttypes.h>
#include <stdbool.h>
#include <stdint.h>

#include "example_cargo_reg_protocol.h"

// Use 1000ms timeout
#define I2C_TIMEOUT_MS 1000

// The STM32 uses the i2c address shifted one position
// to the left (0x52 becomes 0xa4)
#define I2C_ADDR 0xa4

// The register address length is two bytes
#define REG_ADDRESS_LENGTH 2

// The register data length is four bytes
#define REG_DATA_LENGTH 4


/**
 * @brief Read register value over I2C
 *
 * @param[in] reg_addr The register address to read
 * @param[out] reg_data The read register data
 * @returns true if successful
 */
bool read_register(uint16_t reg_addr, uint32_t *reg_data)
{
    HAL_StatusTypeDef status = HAL_OK;

    uint8_t transmit_data[REG_ADDRESS_LENGTH];

    transmit_data[0] = (reg_addr >> 8) & 0xff;
    transmit_data[1] = (reg_addr >> 0) & 0xff;

    status = HAL_I2C_Master_Transmit(&STM32_I2C_HANDLE, I2C_ADDR,
                                      transmit_data, REG_ADDRESS_LENGTH,
                                      I2C_TIMEOUT_MS);
    if (status != HAL_OK)
    {
        return false;
    }

    uint8_t receive_data[REG_DATA_LENGTH];

    status = HAL_I2C_Master_Receive(&STM32_I2C_HANDLE, I2C_ADDR,
                                     receive_data, REG_DATA_LENGTH,
                                     I2C_TIMEOUT_MS);
    if (status != HAL_OK)
    {
        return false;
    }


    // Convert bytes to uint32_t
    uint32_t val = receive_data[0];
    val = val << 8;
    val |= receive_data[1];
    val = val << 8;
    val |= receive_data[2];
    val = val << 8;
    val |= receive_data[3];
    *reg_data = val;

    return true;
}


/**
 * @brief Write register value over I2C
 *
 * @param[in] reg_addr The register address to write
 * @param[in] reg_data The register data to write
 * @returns true if successful
 */
bool write_register(uint16_t reg_addr, uint32_t reg_data)
{
    HAL_StatusTypeDef status = HAL_OK;

    uint8_t transmit_data[REG_ADDRESS_LENGTH + REG_DATA_LENGTH];

    // Convert uint16_t address to bytes
    transmit_data[0] = (reg_addr >> 8) & 0xff;
    transmit_data[1] = (reg_addr >> 0) & 0xff;
    // Convert uint32_t reg_data to bytes
    transmit_data[2] = (reg_data >> 24) & 0xff;
    transmit_data[3] = (reg_data >> 16) & 0xff;
    transmit_data[4] = (reg_data >> 8) & 0xff;
    transmit_data[5] = (reg_data >> 0) & 0xff;

    status = HAL_I2C_Master_Transmit(&STM32_I2C_HANDLE, I2C_ADDR,
                                     transmit_data,
                                     REG_ADDRESS_LENGTH + REG_DATA_LENGTH,
                                     I2C_TIMEOUT_MS);
    if (status != HAL_OK)
    {
        return false;
    }

    return true;
}
```

5.2 Application setup functions

```
#include "example_cargo_reg_protocol.h"

/**
 * @brief Test if configuration of application is OK
 *
 * @returns true if successful
 */
bool configuration_ok(void)
{
    uint32_t status = 0
    if (!read_register(EXAMPLE_CARGO_REG_APPLICATION_STATUS_ADDRESS, &status
       ))
    {
        //ERROR
        return false;
    }

    uint32_t config_ok_mask =
         EXAMPLE_CARGO_REG_APPLICATION_STATUS_FIELD_RSS_REGISTER_OK_MASK |
         EXAMPLE_CARGO_REG_APPLICATION_STATUS_FIELD_SENSOR_CREATE_OK_MASK |
         EXAMPLE_CARGO_REG_APPLICATION_STATUS_FIELD_SENSOR_CALIBRATE_OK_MASK
             |
         EXAMPLE_CARGO_REG_APPLICATION_STATUS_FIELD_CARGO_CREATE_OK_MASK |
         EXAMPLE_CARGO_REG_APPLICATION_STATUS_FIELD_CARGO_CALIBRATE_OK_MASK
            |
         EXAMPLE_CARGO_REG_APPLICATION_STATUS_FIELD_CARGO_BUFFER_OK_MASK |
         EXAMPLE_CARGO_REG_APPLICATION_STATUS_FIELD_SENSOR_BUFFER_OK_MASK |
         EXAMPLE_CARGO_REG_APPLICATION_STATUS_FIELD_CONFIG_APPLY_OK_MASK;

   if (status != config_ok_mask)
   {
        //ERROR
       return false;
   }

   return true;
}
```

```
/**
 * @brief Wait for application not busy
 *
 * @returns true if successful
 */
bool wait_not_busy(void)
{
    uint32_t status = 0
    do
    {
        if (!read_register(EXAMPLE_CARGO_REG_APPLICATION_STATUS_ADDRESS, &
           status))
        {
            //ERROR
            return false;
        }
    } while((status & EXAMPLE_CARGO_REG_APPLICATION_STATUS_FIELD_BUSY_MASK)
       != 0);

    return true;
}


bool example_setup_and_start(void)
{
    // Set container size to 10ft
    if (!write_register(EXAMPLE_CARGO_REG_CONTAINER_SIZE_ADDRESS, 10U))
    {
        //ERROR
        return false;
    }
    // Activate presence (default off)
    if (!write_register(EXAMPLE_CARGO_REG_ACTIVATE_PRESENCE_ADDRESS, 1U))
    {
        //ERROR
        return false;
    }

    // Apply configuration
    if (!write_register(
            EXAMPLE_CARGO_REG_COMMAND_ADDRESS,
            EXAMPLE_CARGO_REG_COMMAND_ENUM_APPLY_CONFIGURATION))
    {
        //ERROR
        return false;
    }

    // Wait for the configuration to be done
    if (!wait_not_busy())
    {
        //ERROR
        return false;
    }

    // Test if configration of application was OK
    if (!configuration_ok())
    {
        //ERROR
        return false;
    }
    // Perform a utilization measurement
    if (!write_register(EXAMPLE_CARGO_REG_COMMAND_ADDRESS,
                        EXAMPLE_CARGO_REG_COMMAND_ENUM_MEASURE_UTILIZATION_LEVEL
                           ))
    {
        //ERROR
        return false;
    }

    // Wait for command be done
    if (!wait_not_busy())
    {
        //ERROR
        return false;
    }

    // Read cargo result header
    uint32_t result;
    if (!read_register(EXAMPLE_CARGO_REG_RESULT_HEADER_ADDRESS, &result))
    {
        //ERROR
        return false;
    }

    // Is utilization valid?

    bool utilization_valid = (result &
       EXAMPLE_CARGO_REG_RESULT_HEADER_FIELD_UTILIZATION_LEVEL_VALID_MASK)
       != 0;

    if (utilization_valid)
    {
        uint32_t utilization_distance_mm;
        if (read_register(EXAMPLE_CARGO_REG_UTILIZATION_DISTANCE_ADDRESS, &
           utilization_distance_mm))
        {
            printf("Distance (utilization): %" PRIu32 " mm\n",
               utilization_distance_mm);
        }
        else
        {
            //ERROR
            return false;
        }
    }

    return true;
}
```

6 Registers

6.1 Register Map

Address	Register Name	Type
0x0000	Version	Read Only
0x0001	Protocol Status	Read Only
0x0002	Measure Counter	Read Only
0x0003	Actual Presence Update Rate	Read Only
0x0004	Application Status	Read Only
0x0010	Container Size	Read / Write
0x0011	Activate Utilization Level	Read / Write
0x0012	Utilization Signal Quality	Read / Write
0x0013	Utilization Threshold Sensitivity	Read / Write
0x0014	Activate Presence	Read / Write
0x0015	Presence Update Rate	Read / Write
0x0016	Presence Sweeps Per Frame	Read / Write
0x0017	Presence Signal Quality	Read / Write
0x0018	Presence Inter Detection Threshold	Read / Write
0x0019	Presence Intra Detection Threshold	Read / Write
0x0020	Result Header	Read Only
0x0021	Utilization Distance	Read Only
0x0022	Utilization Level Mm	Read Only
0x0023	Utilization Level Percent	Read Only
0x0024	Presence Detected	Read Only
0x0025	Max Inter Presence Score	Read Only
0x0026	Max Intra Presence Score	Read Only
0x0030	Command	Write Only
0xffff	Application Id	Read Only

6.2 Register Descriptions

6.2.1 Version

Address	0x0000
Access	Read Only
Register Type	field
Description	Get the RSS version.

Bitfield	Pos	Width	Mask
MAJOR	16	16	0xffff0000
MINOR	8	8	0x0000ff00
PATCH	0	8	0x000000ff

MAJOR - Major version number

MINOR - Minor version number

PATCH - Patch version number

6.2.2 Protocol Status

Address	0x0001
Access	Read Only
Register Type	field
Description	Get protocol error flags.

Bitfield	Pos	Width	Mask
PROTOCOL_STATE_ERROR	0	1	0x00000001
PACKET_LENGTH_ERROR	1	1	0x00000002
ADDRESS_ERROR	2	1	0x00000004
WRITE_FAILED	3	1	0x00000008
WRITE_TO_READ_ONLY	4	1	0x00000010

PROTOCOL_STATE_ERROR - Protocol state error

PACKET_LENGTH_ERROR - Packet length error

ADDRESS_ERROR - Register address error

WRITE_FAILED - Write register failed

WRITE_TO_READ_ONLY - Write to read only register

6.2.3 Measure Counter

Address	0x0002
Access	Read Only
Register Type	uint
Description	Get the measure counter, the number of measurements performed since restart.

6.2.4 Actual Presence Update Rate

Address	0x0003
Access	Read Only
Register Type	uint
Unit	mHz
Description	Get the actual update rate (frame rate) of presence during a burst

6.2.5 Application Status

Address	0x0004
Access	Read Only
Register Type	field
Description	Get example app status flags.

Bitfield	Pos	Width	Mask
RSS_REGISTER_OK	0	1	0x00000001
SENSOR_CREATE_OK	1	1	0x00000002
SENSOR_CALIBRATE_OK	2	1	0x00000004
CARGO_CREATE_OK	3	1	0x00000008
CARGO_CALIBRATE_OK	4	1	0x00000010
SENSOR_BUFFER_OK	5	1	0x00000020
CARGO_BUFFER_OK	6	1	0x00000040
CONFIG_APPLY_OK	7	1	0x00000080
RSS_REGISTER_ERROR	8	1	0x00000100
SENSOR_CREATE_ERROR	10	1	0x00000400
SENSOR_CALIBRATE_ERROR	11	1	0x00000800
CARGO_CREATE_ERROR	12	1	0x00001000
CARGO_CALIBRATE_ERROR	13	1	0x00002000
SENSOR_BUFFER_ERROR	14	1	0x00004000
CARGO_BUFFER_ERROR	15	1	0x00008000
CONFIG_APPLY_ERROR	16	1	0x00010000
APPLICATION_ERROR	17	1	0x00020000

BUSY	18	1	0x00040000

RSS_REGISTER_OK - RSS register OK

SENSOR_CREATE_OK - Sensor create OK

SENSOR_CALIBRATE_OK - Sensor calibrate OK

CARGO_CREATE_OK - Cargo create OK

CARGO_CALIBRATE_OK - Cargo calibrate OK

SENSOR_BUFFER_OK - Memory allocation of sensor buffer OK

CARGO_BUFFER_OK - Memory allocation of cargo buffer OK

CONFIG_APPLY_OK - Cargo configuration apply OK

RSS_REGISTER_ERROR - RSS register error

SENSOR_CREATE_ERROR - Sensor create error

SENSOR_CALIBRATE_ERROR - Sensor calibrate error

CARGO_CREATE_ERROR - Cargo create error

CARGO_CALIBRATE_ERROR - Cargo calibrate error

SENSOR_BUFFER_ERROR - Memory allocation of sensor buffer error

CARGO_BUFFER_ERROR - Memory allocation of cargo buffer error

CONFIG_APPLY_ERROR - Cargo configuration apply error

APPLICATION_ERROR - Application error occured, restart necessary

BUSY - Cargo busy

6.2.6 Container Size

Address	0x0010
Access	Read / Write
Register Type	uint
Description	Size of the container. Valid values to write are 10U, 20U and 40U.

6.2.7 Activate Utilization Level

Address	0x0011
Access	Read / Write
Register Type	bool
Description	Whether to activate utilization level measurements. The command MEASURE_UTILIZATION_LEVEL cannot succeed if this register if false.

6.2.8 Utilization Signal Quality

Address	0x0012
Access	Read / Write
Register Type	uint
Description	Signal quality. This register is x1000 compared to the Cargo Example Application. For more information, see documentation about the Distance Detectors signal quality parameter.

6.2.9 Utilization Threshold Sensitivity

Address	0x0013
Access	Read / Write
Register Type	uint
Description	Threshold sensitivity. This register is x1000 compared to the Cargo Example Application. For more information, see documentation about the Distance Detectors threshold sensitivity parameter.

6.2.10 Activate Presence

Address	0x0014
Access	Read / Write
Register Type	bool
Description	Whether to activate presence measurements. The command MEASURE_PRESENCE cannot succeed if this register if false.

6.2.11 Presence Update Rate

Address	0x0015
Access	Read / Write
Register Type	uint
Unit	mHz
Description	The presence detector update rate (frame rate). This register is x1000 compared to the Cargo Example Application. For more information, see documentation about the Presence Detectors frame rate parameter.

6.2.12 Presence Sweeps Per Frame

Address	0x0016
Access	Read / Write
Register Type	uint
Description	The number of sweeps that will be captured in each frame (measurement). For more information, see documentation about the Presence Detectors sweeps_per_frame parameter.

6.2.13 Presence Signal Quality

Address	0x0017
Access	Read / Write
Register Type	uint
Description	Signal quality. This register is x1000 compared to the Cargo Example Application. For more information, see documentation about the Presence Detectors signal quality parameter.

6.2.14 Presence Inter Detection Threshold

Address	0x0018
Access	Read / Write
Register Type	uint
Description	This is the threshold for detecting slower movements between frames. This register is x1000 compared to the Cargo Example Application. For more information, see documentation about the Presence Detectors inter detection threshold parameter.

6.2.15 Presence Intra Detection Threshold

Address	0x0019
Access	Read / Write
Register Type	uint
Description	This is the threshold for detecting faster movements between frames. This register is x1000 compared to the Cargo Example Application. For more information, see documentation about the Presence Detectors intra detection threshold parameter.

6.2.16 Result Header

Address	0x0020
Access	Read Only
Register Type	field
Description	The result header for the cargo result.

Bitfield	**Pos**	**Width**	**Mask**
TEMPERATURE	0	16	0x0000ffff
UTILIZATION_LEVEL_VALID	17	1	0x00020000
PRESENCE_VALID	18	1	0x00040000

TEMPERATURE - Temperature in sensor (in degree Celsius) during the most recent measurement (presence/utilization). Note that it has poor absolute accuracy and should only be used for relative temperature measurements.

UTILIZATION_LEVEL_VALID - Whether utilization level results are valid. Utilization level results are found in the registers utilization_distance, utilization_level_mm and utilization_level_percent.

PRESENCE_VALID - Whether presence level results are valid. Presence results are found in the registers presence_detected, inter_presence_score and intra_presence_score.

6.2.17 Utilization Distance

Address	0x0021
Access	Read Only
Register Type	uint
Unit	mm
Description	The distance, in millimeters, to the detection.

6.2.18 Utilization Level Mm

Address	0x0022
Access	Read Only
Register Type	uint
Unit	mm
Description	The fill level in millimeters. Fill level is the distance from the detection to the back of the container.

6.2.19 Utilization Level Percent

Address	0x0023
Access	Read Only
Register Type	uint
Unit	%
Description	The fill level in percent. Fill level is the distance from the detection to the back of the container.

6.2.20 Presence Detected

Address	0x0024
Access	Read Only
Register Type	bool
Description	Whether presence was detected during the 5s presence burst

6.2.21 Max Inter Presence Score

Address	0x0025
Access	Read Only
Register Type	uint
Description	Inter presence score is a measure of the amount of slow motion detected. This register contains the maximum inter presence score during the 5s presence burst.

6.2.22 Max Intra Presence Score

Address	0x0026
Access	Read Only
Register Type	uint
Description	Intra presence score is measure of the amount of slow motion detected. This register contains the maximum intra presence score during the 5s presence burst.

6.2.23 Command

Address	0x0030
Access	Write Only
Register Type	enum
Description	Execute command.

Enum	Value
APPLY_CONFIGURATION	1
MEASURE_UTILIZATION_LEVEL	4
MEASURE_PRESENCE	5
ENABLE_UART_LOGS	32
DISABLE_UART_LOGS	33
LOG_CONFIGURATION	34
RESET_MODULE	1381192737

APPLY_CONFIGURATION - Apply the configuration

MEASURE_UTILIZATION_LEVEL - Do one utilization level measurement

MEASURE_PRESENCE - Do one 5s burst of presence measurements

ENABLE_UART_LOGS - DEBUG: Enable UART Logs

DISABLE_UART_LOGS - DEBUG: Disable UART Logs

LOG_CONFIGURATION - DEBUG: Print detector configuration to UART

RESET_MODULE - Reset module, needed to make a new configuration

6.2.24 Application Id

Address	0xffff
Access	Read Only
Register Type	enum

Description	The application id register.

Enum	Value
DISTANCE_DETECTOR	1
PRESENCE_DETECTOR	2
REF_APP_BREATHING	3
EXAMPLE_CARGO	4

DISTANCE_DETECTOR - Distance Detector Application

PRESENCE_DETECTOR - Presence Detector Application

REF_APP_BREATHING - Breathing Reference Application

EXAMPLE_CARGO - Cargo Example Application

7 Disclaimer

The information herein is believed to be correct as of the date issued. Acconeer AB (“Acconeer”) will not be responsible for damages of any nature resulting from the use or reliance upon the information contained herein. Acconeer makes no warranties, expressed or implied, of merchantability or fitness for a particular purpose or course of performance or usage of trade. Therefore, it is the user’s responsibility to thoroughly test the product in their particular application to determine its performance, efficacy and safety. Users should obtain the latest relevant information before placing orders.

Unless Acconeer has explicitly designated an individual Acconeer product as meeting the requirement of a particular industry standard, Acconeer is not responsible for any failure to meet such industry standard requirements.

Unless explicitly stated herein this document Acconeer has not performed any regulatory conformity test. It is the user’s responsibility to assure that necessary regulatory conditions are met and approvals have been obtained when using the product. Regardless of whether the product has passed any conformity test, this document does not constitute any regulatory approval of the user’s product or application using Acconeer’s product.

Nothing contained herein is to be considered as permission or a recommendation to infringe any patent or any other intellectual property right. No license, express or implied, to any intellectual property right is granted by Acconeer herein.

Acconeer reserves the right to at any time correct, change, amend, enhance, modify, and improve this document and/or Acconeer products without notice.

This document supersedes and replaces all information supplied prior to the publication hereof.