HAL Software Integration

User Guide

HAL Software Integration

User Guide

Author: Acconeer AB

Version:a111-v2.15.2

Acconeer AB August 18, 2023

Contents

1 HAL Software Integration

1.1 Introduction

This document aims to provide help and support for customers that wish to integrate A111 towards an MCU or processor that is not included in Acconeer's module offering. This document covers the software needed to integrate Acconeer's libraries with MCU specific drivers. Hardware integration is covered in the document "Hardware and physical integration guideline" that is available for download at the developer site.

The A111 pulse coherent radar sensor is dependent on a software stack running on a host MCU. The host software handles low-level configuration of the radar sensor and pre-processing of radar data. All low-level sensor configurations are uploaded to the sensor at startup, meaning that no firmware or configuration parameters are stored in the sensor permanently.

Acconeer have selected a few MCUs to verify our software against for each release. Some of them have also been included in different versions of our EVKs or modules.

1.2 Acconeer EVK and Modules

Acconeer has designed EVKs and Modules based on the following MCU/Processors. All of which can be bought on Digi-Key.

- Raspberry Pi 3, 3+ and 4: Our standard EVK (XC/XR112), running Raspbian.
- Atmel ATSAME70Q21A-AN: An ARM Cortex M7 MCU that we have chosen to use in our high performance module XM112.
- Nordic Semiconductors nRF52840: An ARM Cortex M4 MCU that is featured in our IoT module XM122, designed for Low Cost and Low Power.
- STMicroelectronics STM32G071RB: An ARM Cortex M0 MCU that is featured in our entry module XM132, designed to be Low Cost, Low Power and integration-ready.

Documentation, including schematics and BOM, for above HW is available from Acconeer web site.

1.3 Acconeer Software Delivery

Acconeer provides software deliveries to the EVKs and Modules on our developer site. It contains example programs and fully implemented HAL for respective hardware for an easy start of development.

Acconeer also provides an SDK for ARM Cortex M4 and M7 based MCUs. The SDK contains the libraries for respective MCU and example programs to show how to use the different services in Radar System Software (RSS). Acconeer also provides a reference implementation of the Hardware Abstraction Layer (HAL) for MCUs from ST Microelectronics and this implementation can be used for a quick start with STM32CubeIDE.

Instructions on how to integrate STM32 MCUs from ST Microelectronics and set up a project in STM32CubeIDE can be found on the developer site. For other MCUs a Hardware Abstraction Layer (HAL) integration must be written according to the guidelines in this document.

1.4 HW Integration Overview

Hardware integration is covered in the document "Hardware and physical integration guideline" that is available for download at the developer site. The purpose of this section is only to define the pin names and connections necessary for the software integration.

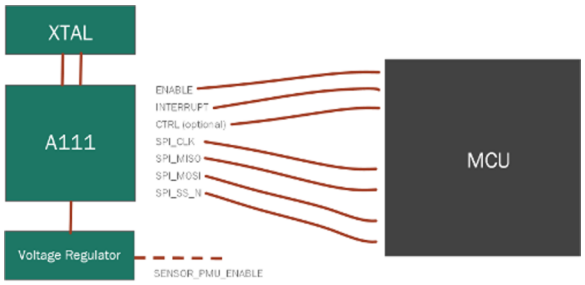

Figure 1: Pin names and connections between A111 and MCU

1.4.1 GPIO Control Signals

The Acconeer Sensor SW package uses two GPIO control signals A111_ENABLE and A111_INTERRUPT. A111_ENABLE signal is used to turn the A111 sensor on and off. The A111_ENABLE signal is active high. The A111_INTERRUPT signal is used to signal the host MCU that the internal buffer memory contains new measurement data ready to be transferred via the SPI interface. The A111_INTERRUPT signal is active high.

1.4.2 SPI Bus

The A111 communicates with the host MCU via a 4-line SPI interface. The maximum supported SPI clock frequency is 50MHz. The A111 is an SPI slave device. The SPI signals are named A111_SPI_CLK, A111_SPI_MOSI, A111_SPI_MISO, and A111_SPI_SS_N. Note that the slave select signal, A111_SPI_SS_N, is active low and often controlled by a GPIO on the MCU.

1.4.3 Power Control

Some hardware designs include a Power Management Unit (PMU) or a power switch that allows the MCU to completely shut off the power to the A111 sensor. In examples for software integration we use a GPIO pin with the name SENSOR_PMU_ENABLE to control the power to the radar sensor.

1.4.4 Crystal

The A111 radar sensor needs a crystal or an external clock source for clock reference. The frequency of the crystal has to be specified in the HAL integration.

1.5 Prototype Integrations

For prototype integrations with an MCU development board we recommend using the “SparkFun Pulsed Radar Breakout - A111”. It is a breakout board with an A111 radar sensor, crystal and a level shifter that makes it easy to connect the A111 to an MCU board with I/O-pins that operates at 3.3V. Note that some Sparkfun breakout boards have the Vcc pin labeled 5V. This pin should be connected to 3.3V if you have an MCU with 3.3V logic on the I/O pins.

1.6 Software Integration

The Acconeer software delivery consists of an SDK with pre-compiled RSS libraries, headers and example programs to show how to use the different services and detectors.

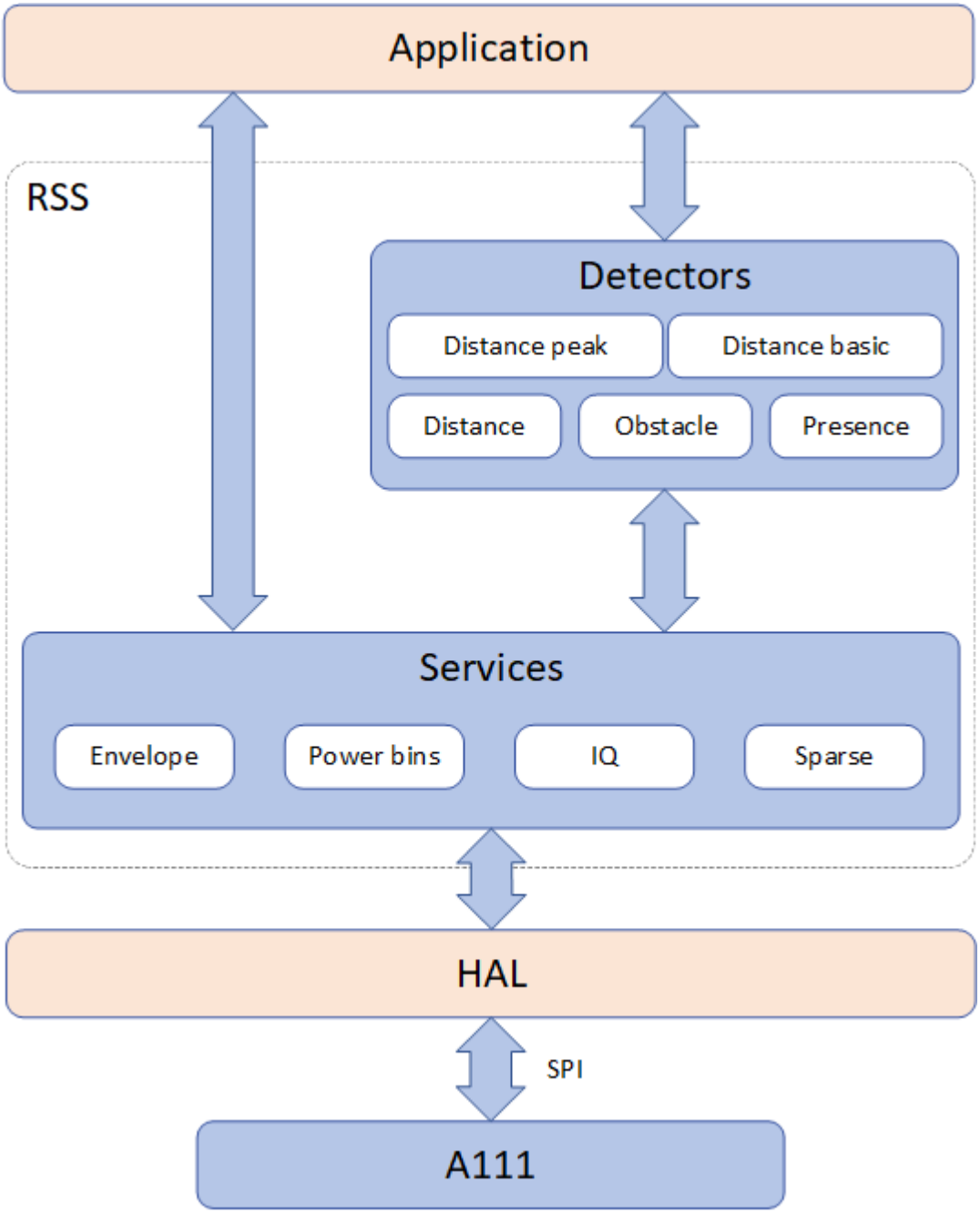

Figure 2: Overview of Radar System Software

Radar System Software (RSS) is the software that will help you interact with the A111 radar sensor. To properly function, this software utilizes MCU specific functions to handle memory and communication with the sensor. Each integration of the sensor to a hardware requires a specific pin configuration and different drivers from the MCU. This is solved by a user-written Hardware Abstraction Layer (HAL). The HAL is a glue layer between RSS and the MCU drivers.

RSS must be activated before it is used, and a HAL struct is passed as an argument to the function acc_rss_activate. The HAL struct contains properties and function pointers that RSS use in its communication with the hardware. The HAL struct and the function pointer types are declared in the header file acc_hal_definitions.h.

Acconeer provide fully verified implementations of a HAL for our EVKs and Modules in the respective software deliveries. Reference implementations are also provided for STM32 MCUs in the Cortex M4 and Cortex M7 packages, see for example the file acc_hal_integration_stm32cube_sparkfun_a111.c

The following sub-chapters describe how to set the properties and write the functions that are included in the HAL struct.

1.6.1 The sensor_count Property

The sensor count is the maximal sensor ID that the integration layer supports and must not exceed ACC_SENSOR_ID_MAX.

1.6.2 The max_spi_transfer_size Property

The max_spi_transfer_size should be set to the maximum buffer size that can be transferred over the SPI bus. If there is no restriction, the limit should be set to SIZE_MAX.

1.6.3 Power-on Function

The power-on function is called when the sensor needs to be enabled. It takes sensor_id as an argument and it returns void. The power to the A111 sensor is preferably turned on and A111_SPI_SS_N set to high before creating the service, but can optionally be done here.

The function should do the following:

- Prerequisite: The sensor is powered on and A111_SPI_SS_N is set to high
- Set A111_ENABLE High
- Wait 2 ms for sensor crystal to stabilize
- Clear pending interrupts

1.6.4 Power-off Function

The power-off function is called when the sensor should be disabled. The function is not optional to implement since it can be called several times during sensor calibration.

The function should do the following:

- Set A111_ENABLE Low
- Optional: The power to the sensor is turned off and all SPI signals as well as INTERRUPT and CTRL are set to low to avoid current leakage.
- Wait 2 ms

1.6.5 Hibernate-enter Function

The hibernate-enter function shall be implemented if the application wants to use the power save mode hibernate. A prerequisite is that the GPIO CTRL is connected on the sensor.

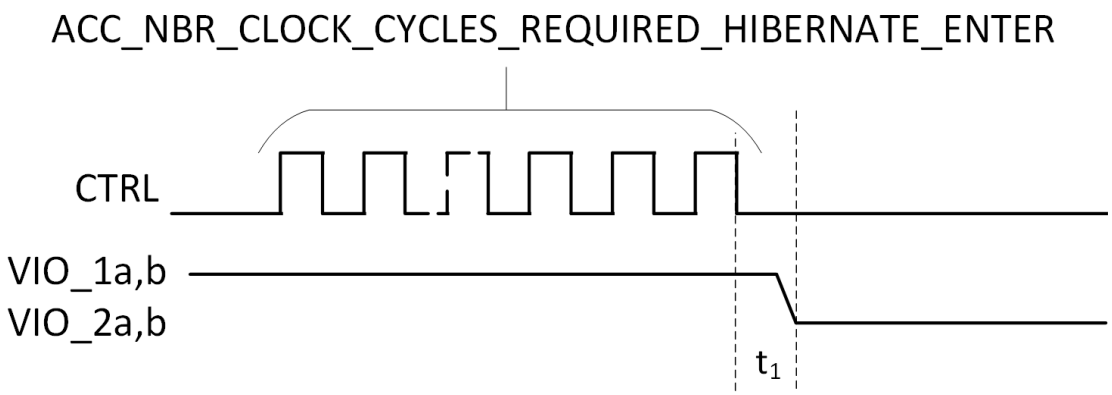

Figure 3: Timing diagram of CTRL signal to enter Hibernate state

The function should do the following:

- GPIO CTRL should be set and cleared ACC_NBR_CLOCK_CYCLES_REQUIRED_HIBERNATE_ENTER number of times.

- Optional power save by disabling VIO_1 and VIO_2: required time before disable VIO_1 and VIO_2 after entering Hibernate, t1: 0 ns. If this is done, make sure that VIO_1 and VIO_2 are enabled again in the Power-on Function.

XM122 supports power save mode hibernate and can be used as a reference on how to implement the hibernate HAL functions.

1.6.6 Hibernate-exit Function

The hibernate-exit function shall be implemented if the application wants to use the power save mode hibernate. A prerequisite is that the GPIO CTRL is connected on the sensor.

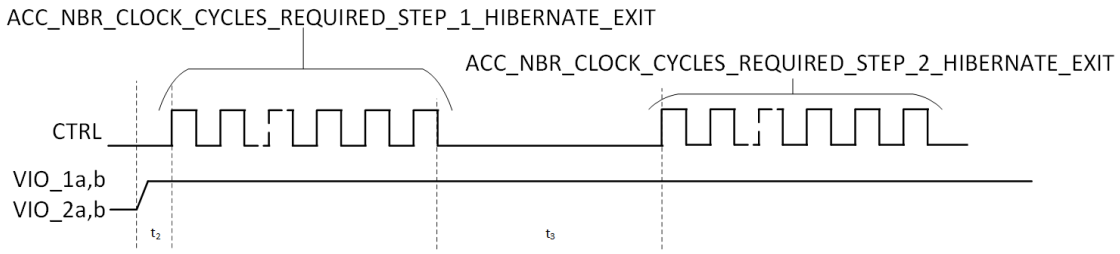

Figure 4: Timing diagram of CTRL signal to exit Hibernate state

The function should do the following:

- Optional power save by disabling VIO_1 and VIO_2: required time after disable VIO_1 and VIO_2 before exit Hibernate, t2: 0 ns
- GPIO CTRL should be set and cleared ACC_NBR_CLOCK_CYCLES_REQUIRED_STEP_1_HIBERNATE_EXIT number of times.
- sleep ACC_WAIT_TIME_HIBERNATE_EXIT_MS, t3: 2 ms (absolute waiting time depends on crystal and tuning capacitor)
- GPIO CTRL should be set and cleared ACC_NBR_CLOCK_CYCLES_REQUIRED_STEP_2_HIBERNATE_EXIT number of times.

XM122 supports power save mode hibernate and can be used as a reference on how to implement the hibernate HAL functions.

1.6.7 Sensor Transfer Function

The sensor transfer function is called by RSS to transfer data to and from the radar sensor over the SPI bus. The maximum buffer size can be limited by setting the property max_spi_transfer_size in the HAL struct. The buffer size is always a multiple of two and the buffer is guaranteed to be 16 bit aligned even though the transfer function is defined with an 8-bit array. It is beneficial from performance perspective to utilize DMA if available and as big buffer transfer size as possible.

The function should do the following:

- Set A111_SPI_SS_N Low
- Send and receive SPI buffer
- Set A111_SPI_SS_N High

1.6.8 Wait for Interrupt Function

This function shall wait at most timeout_ms for the interrupt to become active and then return true. It may return true immediately if an interrupt has occurred since last call to this function. If a timeout occurred and the function waited more than timeout_ms for the interrupt to become active it shall return false.

A simple and portable way of implementing the wait for interrupt function is to use polling. The function should then repeatedly do the following until the interrupt pin is high or a timeout has occurred:

- Check the A111_INTERUPT pin and return true if interrupt pin is HIGH
- Sleep a short time

- Return false if timeout
- Start over and check the interrupt pin again.

While polling is great for maximum portability an optimized implementation may take advantage of HW interrupt support in the MCU to reduce latency and power consumption.

1.6.9 Mem-alloc function

Return pointer to memory, NULL is seen as failure. Allocated memory should be naturally aligned.

1.6.10 Mem-free function

Free memory which is previously allocated.

1.6.11 Get Time Function

The get time function should return the current time in milliseconds. The value is primarily used in log messages and does not have to return the correct wall clock time. Many implementations return the number of milliseconds since power-on of the MCU.

1.6.12 Log Function

The purpose of the log function is to format log messages and to print them to e.g. the console or debug UART.

The function below shows an example of how the log formatting can be implemented:

```
#define LOG_BUFFER_MAX_SIZE 150
#define LOG_FORMAT "%02u:%02u:%02u.%03u (%c) (%s): %s\n"

void acc_hal_integration_log(acc_log_level_t level, const char *module,
    const char *format, ...)
{
    char    log_buffer[LOG_BUFFER_MAX_SIZE];
    va_list ap;

    va_start(ap, format);

    int ret = vsnprintf(log_buffer, LOG_BUFFER_MAX_SIZE, format, ap);
    if (ret >= LOG_BUFFER_MAX_SIZE)
    {
        log_buffer[LOG_BUFFER_MAX_SIZE - 4] = '.';
        log_buffer[LOG_BUFFER_MAX_SIZE - 3] = '.';
        log_buffer[LOG_BUFFER_MAX_SIZE - 2] = '.';
        log_buffer[LOG_BUFFER_MAX_SIZE - 1] = 0;
    }

    uint32_t time_ms = acc_hal_integration_get_current_time();
    char     level_ch;

    unsigned int timestamp    = time_ms;
    unsigned int hours        = timestamp / 1000 / 60 / 60;
    unsigned int minutes      = timestamp / 1000 / 60 % 60;
    unsigned int seconds      = timestamp / 1000 % 60;
    unsigned int milliseconds = timestamp % 1000;

    level_ch = (level <= ACC_LOG_LEVEL_DEBUG) ? "EWIVD"[level] : '?';

    printf(LOG_FORMAT, hours, minutes, seconds, milliseconds, level_ch,
        module, log_buffer);

    va_end(ap);
}
```

1.6.13 Log Level Configuration

The log.log_level property should be set in the HAL struct to restrict the number of log messages generated by RSS.

Supported log levels:

```
typedef enum
{
    ACC_LOG_LEVEL_ERROR,
    ACC_LOG_LEVEL_WARNING,
    ACC_LOG_LEVEL_INFO,
    ACC_LOG_LEVEL_VERBOSE,
    ACC_LOG_LEVEL_DEBUG
} acc_log_level_enum_t;
```

1.6.14 16-bit Sensor Transfer Function

The 16-bit sensor transfer function is optional and can be implemented on supported integrations to optimize the SPI data transfer time. It will do the same as the normal transfer function except that it will transfer data over SPI to the sensor in 16 bit chunks.

1.7 References, List of Documentation

- Hardware and physical integration guideline
- STM32CubeIDE User Guide

All Documents referred to can be found on the developer site.

2 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.