Application Note

Using Sensors Interface of DA14585 IoT Multi Sensor Development Kit

AN-B-068

Abstract

The DA14585 IoT Multi Sensor DK (MSK) is designed to simplify the integration of new sensors with the Sensors Interface (SI) module. In the SI module, users can follow the pre-tested framework. Besides the SI module, DA14585 IoT MSK uses the Wakeup Adapter to handle interrupts from various sources. Therefore, the overall complexity of integrating different new sensors to DA14585 IoT MSK is reduced to minimum and developers' work gets simplified and accelerated. The app note provides step-by-step instructions on how to implement new sensor functionalities by using the SI module.

Contents

Figures

1 Terms and Definitions

DAL	Driver Adaptation Layer
SI	Sensors Interface
IMU	Inertial Measurement Unit
FIFO	First In First Out
MSK	Multi Sensor Development Kit

2 References

[1] UM-B-101, DA14585 IoT Multi Sensor Development Kit Developer's Guide, User Manual, Dialog Semiconductor.

[2] DA14585, Datasheet, Dialog Semiconductor.

3 Introduction

Sensors used in embedded systems are different from each other in many aspects. They measure different physical values and they produce different amount of data. The environmental sensors measure values every few seconds, while typical inertial measurement unit (IMU) sensors use millisecond time bases. IMU sensors need accurate interrupt driven service that read FIFO's, while light sensors can do their work with polling. Each sensor may use different communication protocols (I2C or SPI) and may have drivers from different manufactures. Sensors that use interrupts typically use different microcontroller pins for this purpose, but the application should work seamlessly when adding or removing such functionality. Implementing a system that integrates sensors without a common framework may be a very complex task that is prone to many types of bugs.

Dialog's Sensors Interface (SI, Figure 1) creates an abstraction layer that provides a clear path to integrate sensors into DA14585 IoT MSK. Driver Adaptation Layer (DAL) is a collection of hook functions that should be created for SI to interface with custom device drivers. In short, SI operates using DAL functions instead of operating directly with device drivers. For more information regarding the internal architecture of SI, please refer to *section 4.5* of [1].

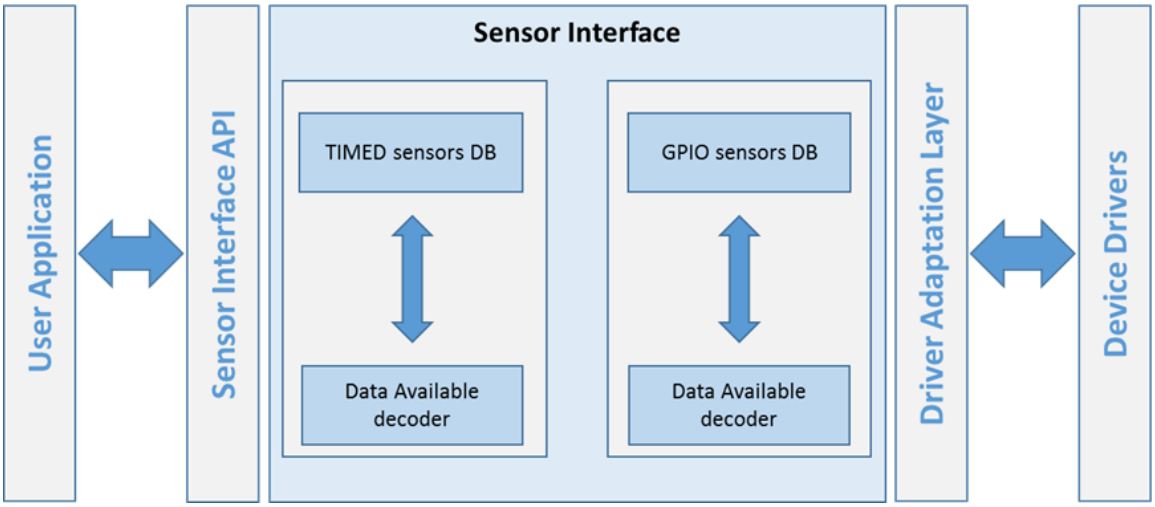

Figure 1: Sensors Interface Block Diagram

4 Wakeup Adapter

Wakeup adapter is the interrupt manager of IoT MSK and works in close coordination with SI. Full description of wakeup adapter is contained in *section 4.4* of [1] and the provided doxygen documentation. The most important function of the wakeup adapter module is `wkup_ad_register_gpio()` that gets the following parameters:

- `uint8_t sel_port`, the GPIO port that the interrupt applies to. For a list of available GPIO ports, please refer to DA14585 datasheet [2] and the `GPIO_PORT` enumerator in `gpio.h`
- `uint32_t sel_pins`, the pin that the interrupt applies to. Use macro `WKUPCT_PIN_SELECT(GPIO_PORT_x, GPIO_PIN_x)` to set this parameter
- `uint32_t pol_pins`, the active polarity of the interrupt, use macro `WKUPCT_PIN_POLARITY(GPIO_PORT_x, GPIO_PIN_x, (polarity_type_t)LOW/HIGH)`
- `void (*cb) (uint8_t, uint32_t)`, the callback function to be executed when the pin is asserted

- `Void (*cb_inv)(void)`, the callback function to be executed when the pin is deasserted (polarity is inversed)

The wakeup adapter is used by SI, but it may be used as a standalone module. The following snippet from IoT MSK shows how to use wakeup adapter with the button connected to P1.3.

```
void user_button_init(void)
{
   wkup_ad_register_gpio(GPIO_BUTTON_PORT,
   WKUPCT_PIN_SELECT(GPIO_BUTTON_PORT, GPIO_BUTTON_PIN),
   WKUPCT_PIN_POLARITY(GPIO_BUTTON_PORT, GPIO_BUTTON_PIN, polarity_type_t)LOW),
   user_on_button_press,
   user_on_button_release);
}
```

Function `user_on_button_press()` will be called when the button is pressed and function `user_on_button_release()` will be called when the button is released. GPIO_BUTTON_PORT and GPIO_BUTTON_PIN define the GPIO port and pin number that the button is connected to.

5 Sensors Interface Sensor Types

There are four sensor types supported by SI:

- Timed devices used by Environmental & Optical sensors of IoT MSK
- Interrupt devices used by Accelerometer-Gyroscope of IoT MSK
- Forced devices used by Magnetometer of IoT MSK in free running setup
- Forced with single shot devices. This is a hybrid method between Interrupt and Forced. It is used by Magnetometer of IoT MSK in single shot setup.

5.1 Common Sensors Interface Parameters

All sensor types are added in SI by a set of parameters that are defined by SI API. There is a common set of parameters and a set of parameters for each sensor type. All the parameters are defined inside C structure `si_config_t` contained in `sensors_interface_api.h`.

The SI common parameters are:

- `sensor_id`: each sensor should have a unique ID. This does not affect the operation but plays a significant role in debugging. For possible values or to add a new ID, users should refer to the values in `user_sensors.h`
- `operation_mode`: this parameter determines the sensor type and should take one of the following values:
  - `NO_OPERATION`
  - `INTERRUPT`
  - `TIMER`
  - `FORCED`
  - `FORCED_INTER_SNGL_SHOT`
- `set_sensor_config`: it is a user-defined callback function to setup the device in driver space
- `data_size`: it is the actual size of the data that are expected during a read operation
- `read_fn`: it is a user-defined callback function to read the actual data from the device in driver space

- pre_data_read_fn: it is a user-defined callback function that is executed before the normal read. Currently it is not used
- user_app_cb: it is a user-defined callback function to pass sensor data to application space

To initiate sensor registration, call API function si_register_sensor().

5.2 Parameters for Timed Devices

The parameters that are specific to the timed devices (Figure 2) are the following:

- operation_mode should be set to TIMER
- read_delay in milliseconds, which is the time needed between a force command being issued, telling the sensor to start a measurement, and the actual read of the measured data from the sensor. This means that the read_delay value should be larger than the maximum measurement time of the specific sensor
- force_read_fn: it holds the callback function that initializes the measurement procedure at read_delay time before the actual read
- periodic_read_interval in milliseconds is the actual sampling period

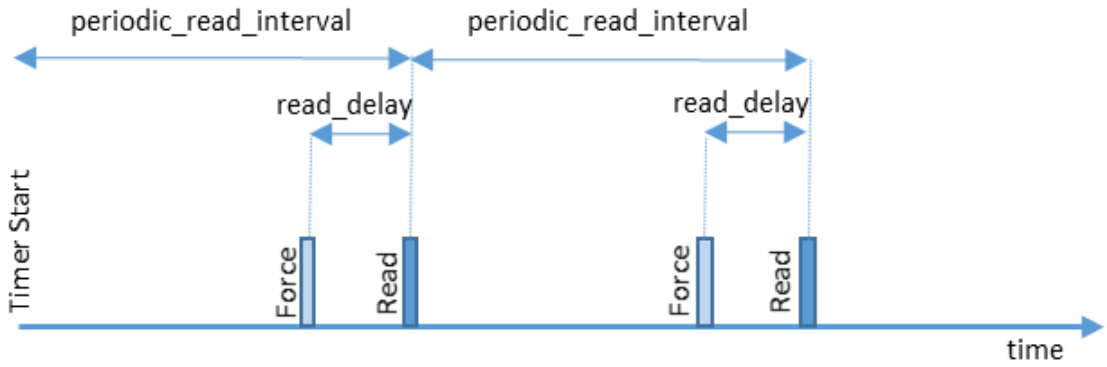

Figure 2: SI Timed Devices

5.3 Parameters for Interrupt Driven Devices

The parameters that are specific to the interrupt driven devices (Figure 3) are the following:

- operation_mode should be set to INTERRUPT
- sensor_port, the GPIO port that the interrupt applies to. For a list of available GPIO ports, please refer to DA14585 datasheet [2] and the GPIO_PORT enumerator in gpio.h
- sensor_pin, the GPIO pin of the sensor_port that the interrupt applies to. For a list of available GPIO pins, please refer to DA14585 datasheet [2] and GPIO_PIN enumerator in gpio.h
- irq_polarity, the interrupt active polarity that should be set to HIGH or LOW

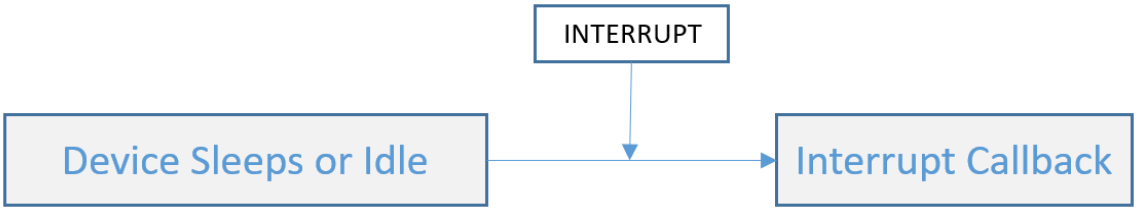

Figure 3: SI Interrupt Driven Devices

5.4 Parameters for Forced Read Devices Sensors Interface

The `operation_mode` of the forced read devices (Figure 4) should be set to `FORCED`. No other parameters are required.

A new measurement is initiated by a user command `SI_FORCE_TO_READ_CMD`, for example, `si_send_command(SI_FORCE_TO_READ_CMD, MAGNETOMETER)`. The sensor triggers an interrupt on measurement completion. To repeat the process users must reinitiate a "`SI_FORCE_TO_READ_CMD`".

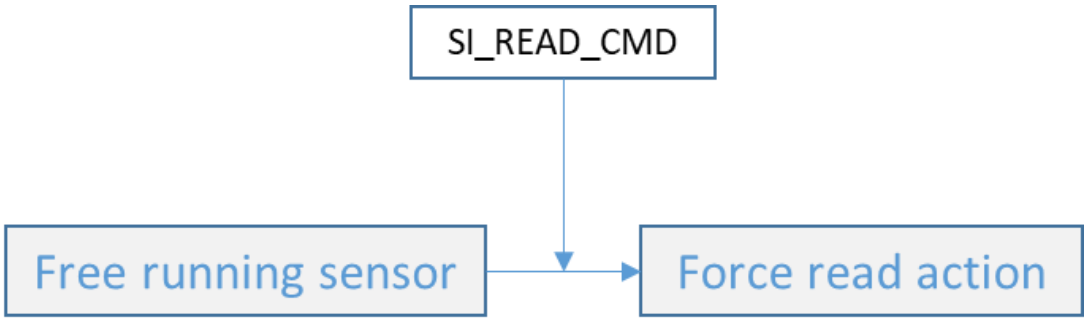

Figure 4: SI Forced Read Sequence

5.5 Parameters for Forced with Single Shot Devices

The parameters that are specific to the forced with single shot driven devices (Figure 5) are the following:

- `operation_mode` should be set to `FORCED_INTER_SNGL_SHOT`
- `sensor_port`, the GPIO port that the interrupt applies to, for a list of available GPIO ports refer to DA14585 datasheet [2] and `GPIO_PORT` enumerator in `gpio.h`
- `sensor_pin`, the GPIO pin of the `sensor_port` that the interrupt applies to, for a list of available GPIO pins refer to DA14585 datasheet [2] and `GPIO_PIN` enumerator in `gpio.h`
- `irq_polarity`, the interrupt active polarity that should be set `HIGH` or `LOW`
- `force_read_fn`, it holds the callback function that initializes the measurement procedure

A new measurement is initiated by a user command `SI_FORCE_TO_READ_CMD`, for example, `si_send_command(SI_FORCE_TO_READ_CMD, MAGNETOMETER)`. The sensor triggers an interrupt on measurement completion. To repeat the process users must reinitiate a "`SI_FORCE_TO_READ_CMD`".

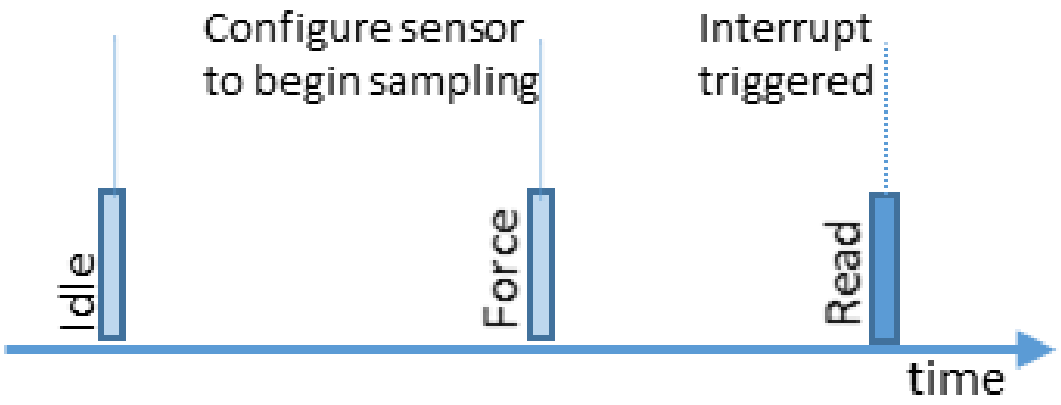

Figure 5: SI Forced with Single Shot Driven Devices

6 Adding a New Sensor

We will integrate a virtual sensor, named `MY_SENSOR`, into DA14585 IoT MSK using Sensors Interface to provide an example.

1. In `user_config_basic.h`, add `#define MY_SENSOR_AVAILABLE`. This should be used as a compilation switch for every `my_sensor` related code inside `user_sensors.c` and `user_sensors.h`.
2. In the folder `\projects\target_apps\common\src\drivers`, create a new folder `my_sensor` and place all the drivers provided by the manufacturer there. This folder should be added in the project include paths.
3. In the folder `projects\target_apps\common\src\driver_adaptation_layer`, create two new files, `my_sensor.c` and `my_sensor.h`. These files contain the data abstraction layer (DAL) functions that are the glue code between the Sensors Interface and peripheral specific driver functions. Usually silicon manufacturers provide SW drivers for their device, and these SW drivers should fit and adapt to what SI expects by using the DAL functions. In `my_sensor.c` and `my_sensor.h` the following new functions should be defined and declared:
   - `uint8_t my_sensor_setup(void)`:

     This function is responsible for setting up the sensor in the desired operation. The SI does not provide any means to pass device specific initialization parameters. Thus, the initialization parameters are passed through global structure objects. For more information please refer to `user_accel_setup` structure in DA14585 IoT MSK code
   - `void my_sensor_force_read(void)`:

     This function performs all required operations to put the device in forced mode
   - `void my_sensor_read(uint8_t *data_ptr, uint16_t *data_size)`:

     This function performs all required operations to retrieve data from the device. The address space to use is provided (allocated) from SI and must return the data size obtained from the driver
   - `uint8_t my_sensor_disable(void)`:

This function stops the device operation and puts it in minimum consumption mode. It should be added into `user_periph_sensors_suspend()` function in `user_sensors.c` by which sensors are disabled as a group.

4. `my_sensor.h` should be included in `user_sensors.h`
5. Create a new function `void user_my_sensor_data_cb (uint8_t *data_ptr, uint16_t *data_size)` in `user_sensors.c`. This function is called to handle the data in user space, and users should place all application specific code in this function.
6. Create a new function `void user_my_sensor_init(void)` in `user_sensors.c` and call it from `user_sensors_init()`. This function initializes all the SI specific parameters described in section 5.

```
void user_my_sensor_init(uint8_t operation_mode)
{
   si_config_t si_my_sensor_config;
   memset(&si_my_sensor_config, 0, sizeof(si_config_t));
   //ADD COMMON SI PARAMETERS
   si_my_sensor_config.operation_mode=<MY_SENSOR OPERATING_MODE>
   si_my_sensor_config.sensor_id = <SENSOR_ID>;
   si_my_sensor_config.user_app_cb = (read_data_buf_fptr_t)user_my_sensor_data_cb;
   si_my_sensor_config.set_sensor_config=my_sensor_setup;
   si_my_sensor_config.read_fn = (read_data_buf_fptr_t) my_sensor_read;
      si_my_sensor_config.data_size = <maximum data size expected from read operation>

   //ADD OPERATION MODE SPECIFIC PARAMETERS
   si_my_sensor_config.sensors_pin_conf.sensor_port=<INTERRUPT PORT>
   si_my_sensor_config.sensors_pin_conf.sensor_pin=<INTERRUPT PIN>
   si_my_sensor_config.sensors_pin_conf.irq_polarity=<INTERRUPT POLARITY>
   si_my_sensor_config.force_read_fn=<FORCE READ FUNCTION>

   //FINALLY REGISTER THE SENSOR
   si_register_sensor(si_my_sensor_config);
}
```

The DA14585 IoT MSK uses different SI operating modes for IoT MSK Sensors, and the code for the SI operating modes is constructed using the same procedures described in this application note. Users may refer to the source code in the `driver_adaptation_layer` folder and `user_sensors.c` for sensor specific examples.

Revision History

Revision	Date	Description
1.1	17-Jan-2022	Updated logo, disclaimer, copyright.
1.0	28-Jan-2019	Initial version.

Status Definitions

Status	Definition
DRAFT	The content of this document is under review and subject to formal approval, which may result in modifications or additions.
APPROVED or unmarked	The content of this document has been approved for publication.