AS7050 Linux Sample Driver

Documentation
revision v1.0.0

Generated by Doxygen

Contents

1 Scope

This document describes the Linux Sample Driver for AS7050 and additional components which are used to show the usage of the driver. The sample driver requires some knowledge of the chiplibrary of the AS7050 which is part of it and implements chip specific details for accessing registers as well as configuring and executing measurements.

2 AS7050 Linux Kernel Driver

This driver is a sample implementation how the vital sensor AS7050 can be accessed in a Linux based environment. It is implemented as a kernel driver and fully integrates the chiplibrary of the sensor which is part the software support package of the AS7050. To provide the common interface of the chiplibrary an additional wrapper "AS7050 Chiplibrary Wrapper" is part of the driver which resides in the userspace. The wrapper simply translates chiplibrary related function calls to linux sys calls which are processed by the driver further on.

The wrapper implements the following functions of the chiplibrary and is fully compatible with the interface of the chiplibrary:

- **as7050_initialize**: Initializes the chiplibrary wrapper as well as the chiplibrary itself. Parameter 'p_interface↩ _descr' which defines the underlying interface for communication is ignored. The chiplibrary wrapper starts an additional thread for data processing from the underlying chiplibrary. This thread is responsible for calling the user callback (parameter 'p_callback') method to send data to the user.
- **as7050_shutdown**: Stops the thread for data processing, stops all internal actions of the chiplibrary and powers down the device.
- **as7050_set_reg_group**: Directly calls the underlying chiplibrary function to configures the sensor directly via register formated structure.
- **as7050_get_reg_group**: Directly calls the underlying chiplibrary function to reads the actual register data into a register group structure.
- **as7050_set_agc_config**: Directly calls the underlying chiplibrary and sets the configuration for auto-gain-control (AGC).
- **as7050_start_measurement**: Directly calls the underlying chiplibrary and starts a measurement.
- **as7050_stop_measurement**: Directly calls the underlying chiplibrary and stops a measurement.
- **as7050_write_register**: Directly calls the underlying chiplibrary and sets the value of a register.
- **as7050_read_register**: Directly calls the underlying chiplibrary and gets the value of a register.
- **as7050_get_version**: Requests the version information of the chiplibrary.
- **as7050_calculate_dac_reference_value**: Directly calls the underlying chiplibrary function to calculate DAC reference value

Building the sources of kernel driver

- copy driver sources from 'as7050_linux_reference/kernel' into kernel directory 'drivers/staging'
- adopt Kconfig file in directory drivers/staging/ and add 'source "drivers/staging/ams/Kconfig"'
- adopt Makefile in directory drivers/staging/ and add 'obj-y += ams/'
- configure the kernel to compile the driver by adding it to the kernel configuration

```
Linux Kernel Configuration
    Device Drivers --->
        [*] Staging drivers --->
            ams VitalSensors --->
                <M> ams AS7050 device driver support
```

Building the wrapper "AS7050 Chiplibrary Wrapper"

- call make in directory 'as7050_linux_reference/userspace/chiplib_wrapper'
- if cross compiling is needed, specify compiler and archive command e.g.: make CC=arm-linux-gnueabihf-cc AR=arm-linux-gnueabihf-ar

Dynamic loading the kernel driver

If the kernel driver is build as module and the sensor is properly connected, the driver can be loaded dynamically into the system. In this example, the sensor is connected at I2C bus number 4, with I2C address 0x55. The interrupt line is connected to GPIO 500.

```
user@dev:~$ insmod /lib/modules/.../kernel/drivers/staging/ams/as7050/ams-as7050-i2c.ko
user@dev:~$ echo as7050-i2c 0x55 > /sys/bus/i2c/devices/i2c-4/new_device
user@dev:~$ echo 500 > /sys/class/misc/as7050/irq
```

Check if the load of the kernel driver succeeded by calling 'dmesg' and check the command output. Example output:

```
[1645584.772566] misc as7050: (vital_sensor_misc_probe) Reading open firmware data failed. Using default
       values!
[1645584.772591] misc as7050: (vital_sensor_misc_probe) Allocating 256 byte for FIFO.
[1645584.772598] misc as7050: (vital_sensor_misc_probe) Requesting Enable GPIO for device failed! Device
       shall be enabled manually.
[1645584.772604] as7050-i2c 4-0055: (vital_sensor_i2c_probe) Successfully configured.
[1645584.772674] i2c i2c-4: new_device: Instantiated device as7050-i2c at 0x55
```

Device Tree support

If the kernel driver is compiled into the kernel, the following device tree syntax can be used to load the driver during boot up of the kernel.

```
&i2c4 {
    as7050: as7050@55 {
        compatible = "ams,as7050-i2c";
        reg = <0x55>;
        interrupt-parent = <&gpio>;
        interrupts = <500 2>;
        fifo_size = <256>;
    };
};
```

Accessing the kernel driver and use the chiplibrary wrapper

A project which shall use the chiplibrary in combination with the linux kernel driver shall link against the wrapper library. The wrapper shall be called in the same manner as known from the chiplibrary interface. But, the wrapper ignores the parameter 'p_interface_descr' of the function 'as7050_initialize' and instead opens the device exported by the kernel driver.

For correct usage, the files 'as7050_wrapper.h' and 'as7050_chiplib.h' needs to be included in the project. All files needed for the wrapper can be found in directory 'as7050_linux_reference/userspace/chiplib_wrapper'.

Interfaces of the kernel driver

The kernel driver is implemented as an I2C client device driver and fulfils the character device driver interface. Therfore, it offers the following methods for interacting with the userspace.

Method	Description
open	Opens the device node exported by the driver. See Linux sys calls for further documentation.
close	Closes the device node exported by the driver. See Linux sys calls for further documentation.
read	Reads actual measurement data from the device node. See Linux sys calls for further documentation.
poll	Poll and wait until new measurement data is available. See Linux sys calls for further documentation.
ioctl	Execute a sys call to call chiplibrary functions.

For interaction with the chiplibrary functionality the kernel driver implements the following sys calls via the function ioctl.

IOCTL	Chiplibrary function	Description
initialize	as7050_initialize	See documentation of chiplibrary.
shutdown	as7050_shutdown	See documentation of chiplibrary.
set_reg_group	as7050_set_reg_group	See documentation of chiplibrary.
get_reg_group	as7050_get_reg_group	See documentation of chiplibrary.
set_agc_config	as7050_set_agc_config	See documentation of chiplibrary.
get_agc_config	as7050_get_agc_config	See documentation of chiplibrary.
write_register	as7050_write_register	See documentation of chiplibrary.
read_register	as7050_read_register	See documentation of chiplibrary.
get_measurement_config	as7050_get_measurement_config	See documentation of chiplibrary.
start_measurement	as7050_start_measurement	See documentation of chiplibrary.
stop_measurement	as7050_stop_measurement	See documentation of chiplibrary.
get_version	-	Read actual version info of the driver.
get_fifo_size	-	Read actual configured FIFO size used by the driver to buffer sensor data.
get_dac_ref_val	as7050_calculate_dac_reference_↩ value	See documentation of chiplibrary.

Please see the documentation of the chiplibrary for further information of those functions. The specific driver sys calls and there parameters are defined in file 'as7050-fcntl.h'. An example how to setup an appropriate sys call can be found in the file 'as7050_wrapper.c' which is part of the wrapper library.

The kernel driver exports a device node to /dev/ with the name 'as7050'. For configuration of the driver it supports the access through sysfs. Therefore, the driver exports a sysfs node to /sys/class/misc/as7050 with additional properties. These are listed in the following table.

Property name	Description	Type of access
irq	Configure system specific GPIO connected to the IRQ line of the sensor.	read/write
fifo_size	Return actual configured FIFO size used to buffer sensor data.	read
version	Return actual version of the driver.	read

Verified kernels

The driver is verified against the following kernel versions:

- 4.19.x