ISL29177 Android Driver – Driver Notes

Purpose

This is a supplementary document for understanding the isl29177 android driver.

Audience

This document is intended to help understand different pieces of code from the driver that performs specific tasks. Having a fundamental knowledge of Linux driver model would be helpful in completely understanding this document. But certain parts of this document consists of flows which might be understood by all readers.

Driver Registration

The following piece of code does the i2c driver registration for the isl29177 sensor. The structure ***isl29177_driver*** consists of references to the required callbacks supported by the driver which are necessary for the operation of the driver.

```
static struct i2c_driver isl29177_driver = {
        .driver   = { .name = "isl29177"},
        .id_table = isl_device_ids,
        .probe    = isl29177_probe,
        .remove   = isl29177_remove,
        .suspend  = isl29177_suspend,
        .resume   = isl29177_resume,
};

static int __init isl29177_init(void)
{
        return i2c_add_driver(&isl29177_driver);
}


static void __exit isl29177_exit(void)
{

        i2c_del_driver(&isl29177_driver);
}
```

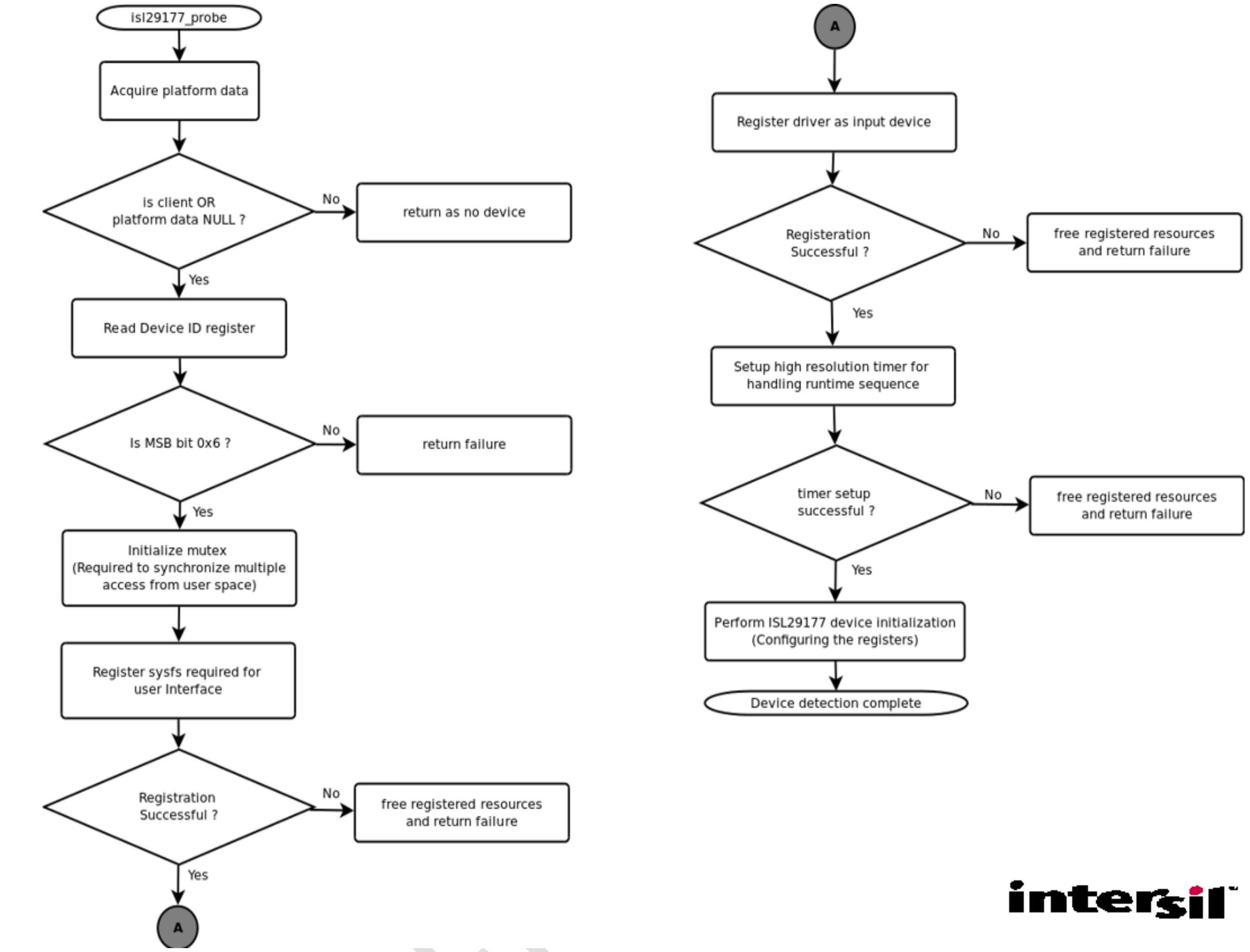
Reference	Description
.driver.name (“isl29177”)	Name by which the driver would be identified in the Linux kernel
isl_device_ids	A table of i2c addresses or devices that would be supported by this driver
isl29177_probe	Called when i2c core finds a matching device
isl29177_remove	Called when this driver is removed from kernel
isl29177_suspend	Called when the android system goes to suspend mode. Any handling of sensor that needs to be done during suspend would be added here
isl29177_resume	Called when the android system wakes up from sleep. Any handling of sensor that needs to be done after resume of the system would be added here

Device Detection

The platform information or board file for target platform or its device tree (used to describe the devices associated with the platform) contains the information on presence of isl29177 sensor with I2C slave address 0x44h being connected.

Once the Linux I2C core driver probes the I2C bus and finds this device, any I2C driver which has registered the device id for isl29177 would be notified about its presence and the driver would now be provided with access to the device.

The function isl29177_probe is called to notify the device presence. The following flow diagram summarizes the activities done in this function.


isl29177_probe
Acquire platform data
is client OR platform data NULL ?
No
return as no device
Yes
Read Device ID register
Is MSB bit 0x6 ?
No
return failure
Yes
Initialize mutex (Required to synchronize multiple access from user space)
Register sysfs required for user Interface
Registration Successful ?
No
free registered resources and return failure
Yes
A
A
Register driver as input device
Registeration Successful ?
No
free registered resources and return failure
Yes
Setup high resolution timer for handling runtime sequence
timer setup successful ?
No
free registered resources and return failure
Yes
Perform ISL29177 device initialization (Configuring the registers)
Device detection complete
intersil

The following helper functions are called during device detection to perform the specified tasks

Function	Description
setup_debugfs	Register the user interface support in driver
setup_input_device	Register driver as an input device driver in order to report proximity events to userspace
setup_hrtimer	Register the high resolution timer handler which would handle the runtime sequence

Device Initialization

The function isl29177_initialize implements the device initialization for the ISL29177 sensor device. It primarily consists of the register writes to the device and a call to offset_adjust to do the initial compensation for the cross-talk.

Below table indicates the register writes and their description in order of initialization

REG	VAL	Description
0x09	0x38	Reset sensor device
0x09	0x89	Enable test mode
0x02	0x20	Set high offset
0x03	0x02	Set interrupt persistence
0x04	0x25	Set the interrupt high threshold defined by PROX_HI_THRESHOLD in isl29177.h header
0x05	0x15	Set the interrupt low threshold defined by PROX_LO_THRESHOLD in isl29177.h header
0x0C	0x00	Disable residue reading
0x0F	0x40	Enable OTP
0x01	0x80	Enable prox sensing

Register Read / Write API

The following two functions are used in the driver to provide access to a specific set of bits or the complete register in the ISL29177 register space

1. isl_read_field(unsigned char reg, unsigned char mask, unsigned char *val)
2. isl_write_field(unsigned char reg, unsigned char mask, unsigned char val)

The usage of these A Pis are well documented in the driver source code.

Runtime Sequence

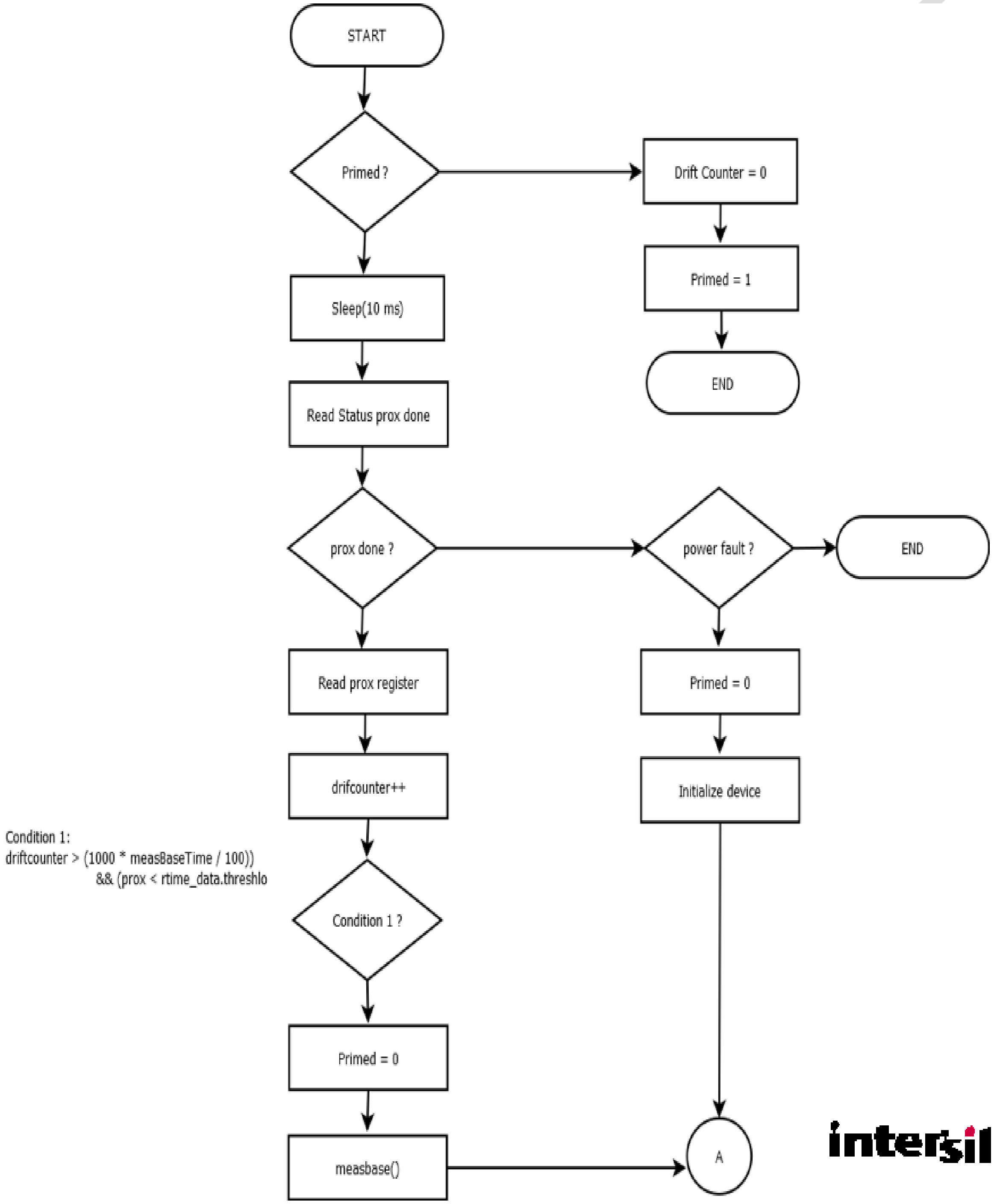
Currently the driver performs updates to the proximity and other associated data at the interval of 100ms. The driver had already registered a high resolution timer during device detection. High resolution timer (hrtimer) would trigger a software interrupt at defined intervals of 100ms. The hrtimer handler registered is ***isl29177_hrtimer_handler.*** This function internally schedules the thread ***sensor_irq_thread*** which does the actual work required to be done during runtime.

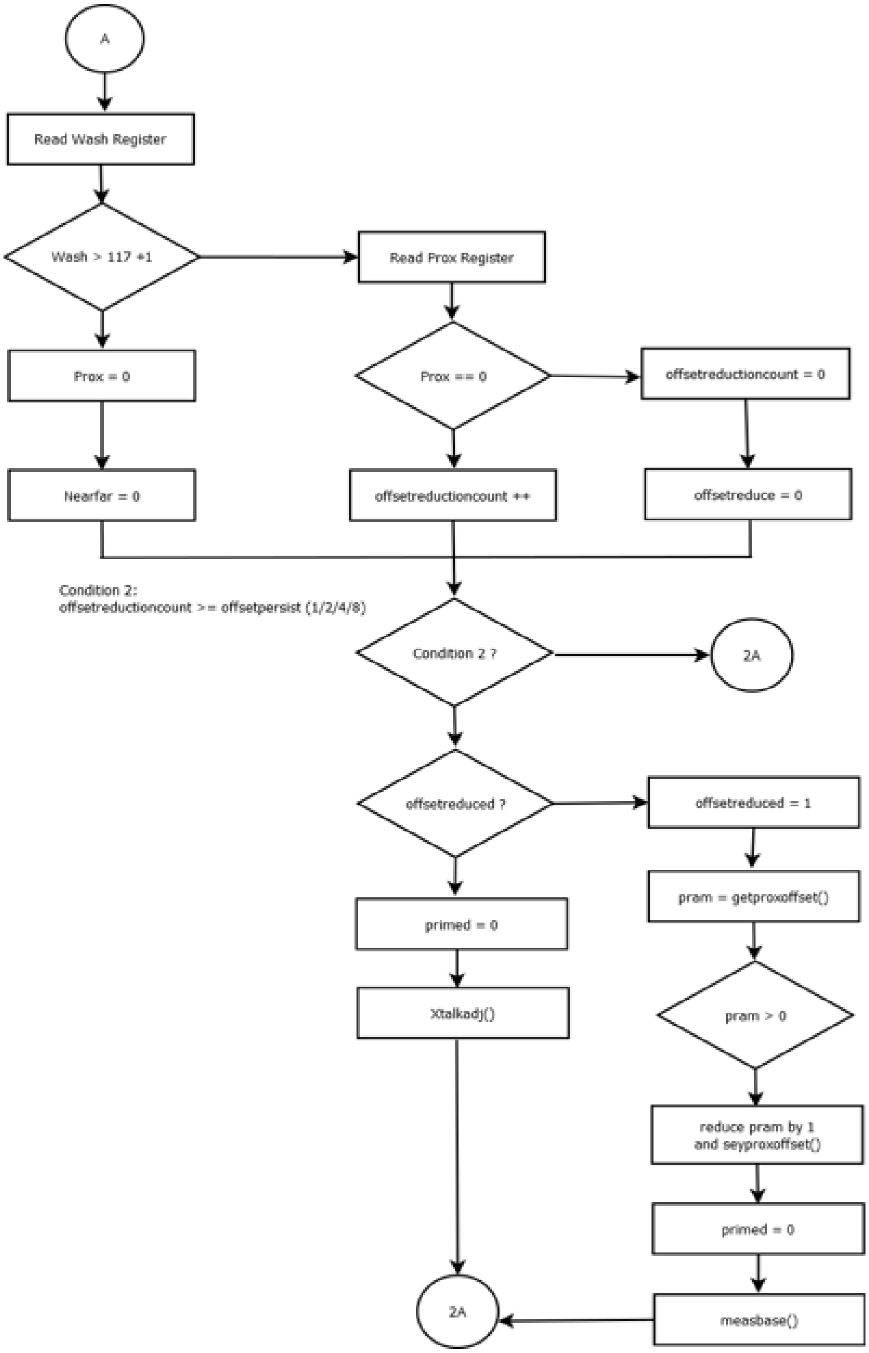
The function Xtalkadjuct is called when offset is reduced for fixed number of times.

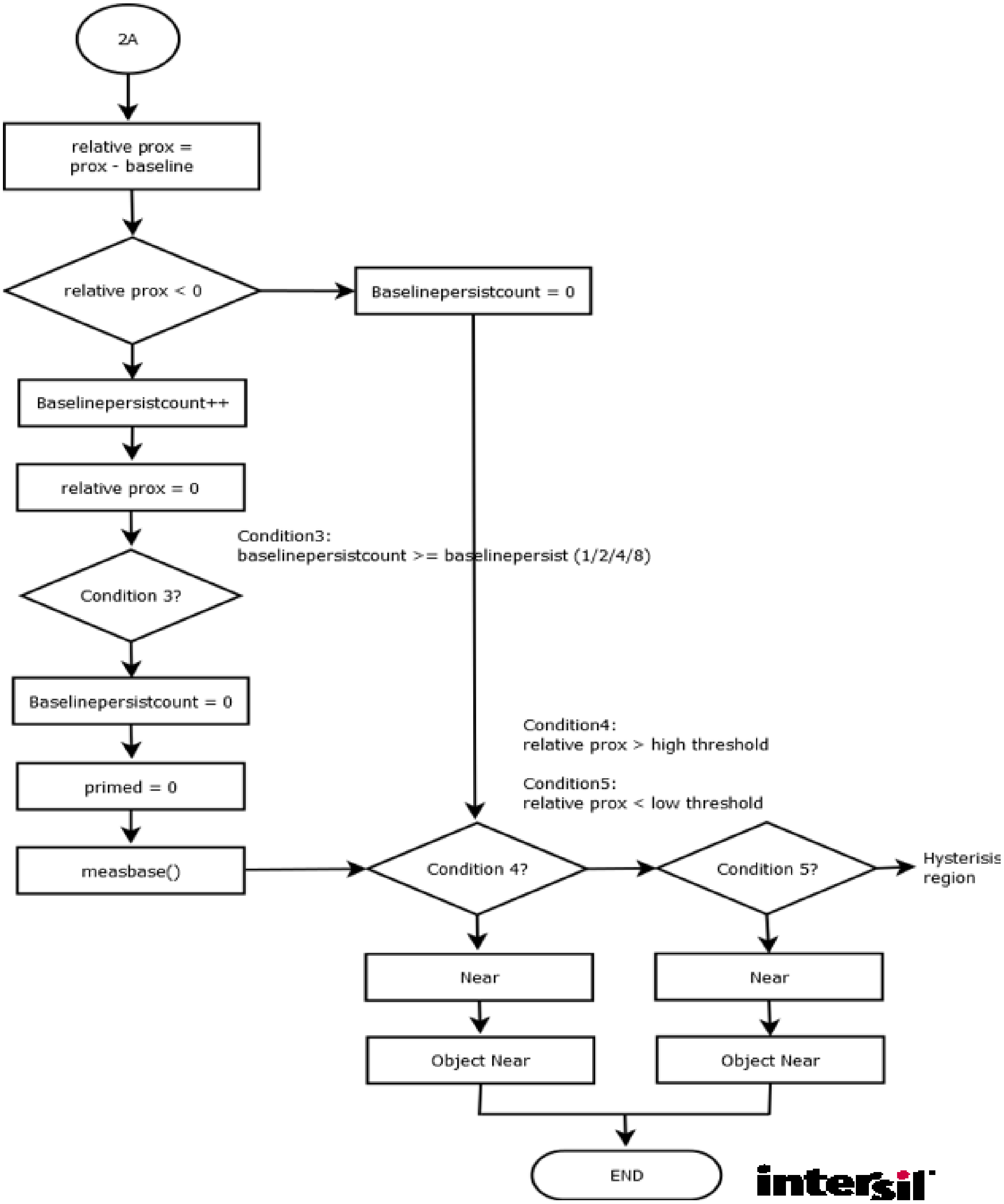
The function measbase which is used to calculate net baseline value is called when proximity value is below baseline for fixed number of times

The below flowchart represents the task done in the function ***sensor_irq_thread (Runtime)***
We divide the **runtime sequence** function in 2 following subsequences:

1. Read prox value from the register and remeasure the baseline if required
2. Washout detection and tracking
   2.1 Adjustment for high offset
   2.2 measure base if require



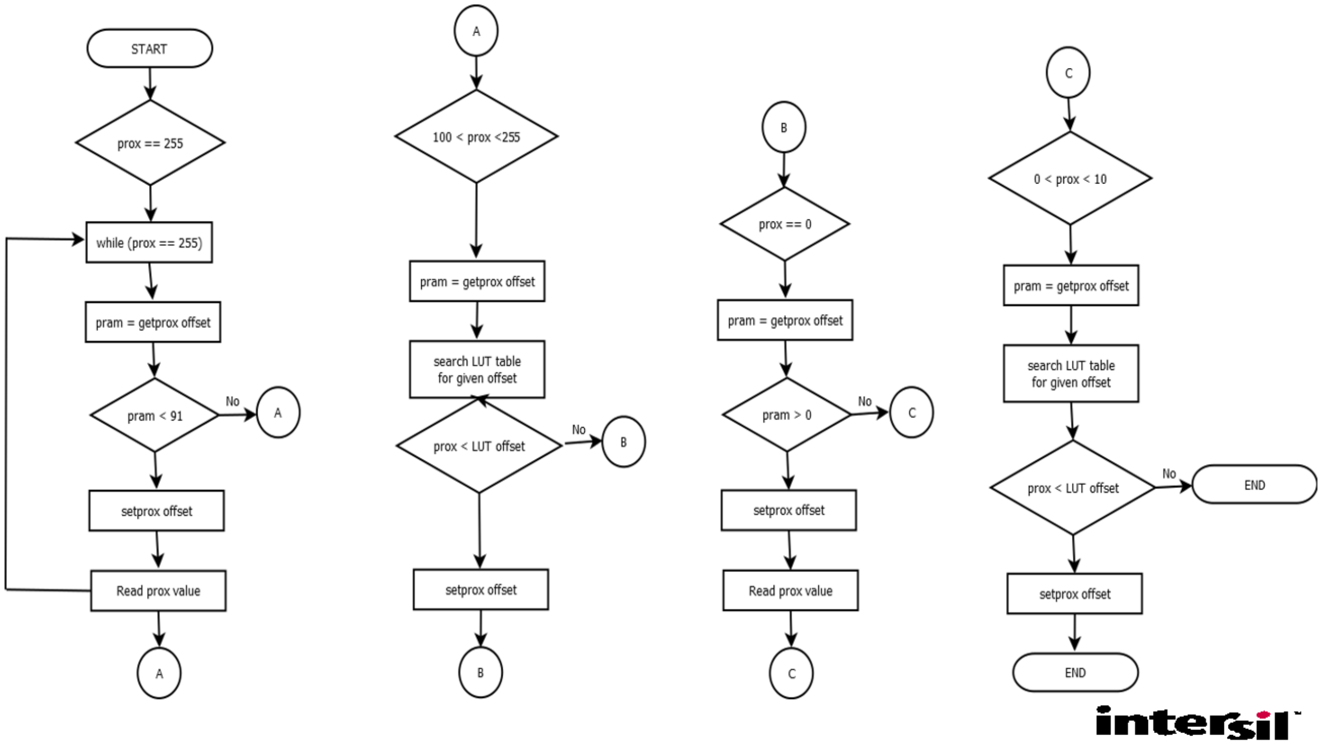

A
Read Wash Register
Wash > 117 +1
Read Prox Register
Prox = 0
Prox == 0
offsetreductioncount = 0
Nearfar = 0
offsetreductioncount ++
offsetreduce = 0
Condition 2:
offsetreductioncount >= offsetpersist (1/2/4/8)
Condition 2 ?
2A
offsetreduced ?
offsetreduced = 1
pram = getproxoffset()
primed = 0
Xtalkadj()
pram > 0
reduce pram by 1
and seyproxoffset()
primed = 0
2A
measbase()


2A
relative prox =
prox - baseline
relative prox < 0
Baselinepersistcount = 0
Baselinepersistcount++
relative prox = 0
Condition3:
baselinepersistcount >= baselinepersist (1/2/4/8)
Condition 3?
Baselinepersistcount = 0
primed = 0
measbase()
Condition4:
relative prox > high threshold
Condition5:
relative prox < low threshold
Condition 4?
Condition 5?
Hysterisis
region
Near
Near
Object Near
Object Near
END
intersil

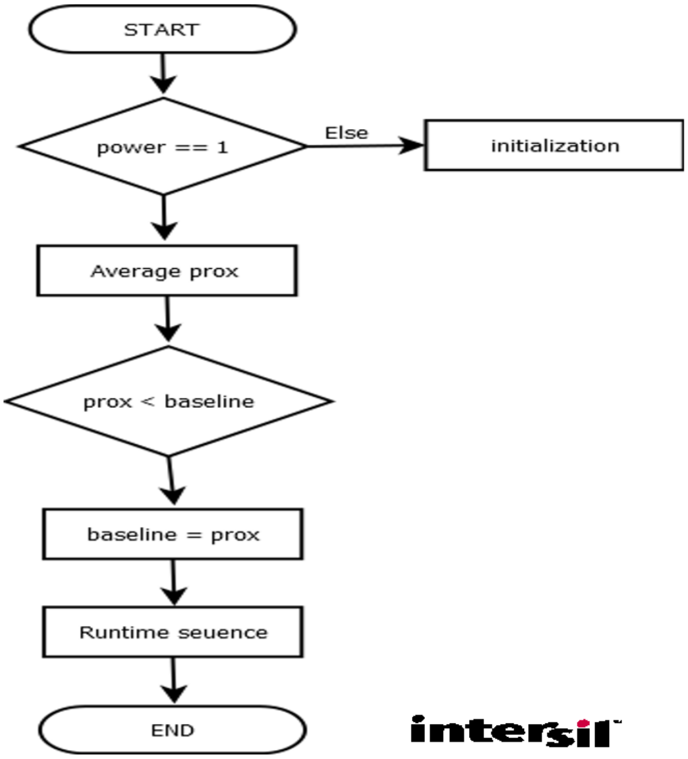
Xtalk Adjusment

The following proximity ranges are handle to compensate the offset value and range with the help of getproxoffset and setproxoffset functions. These function are basically used to get and set the proximity offset and range value respectively by taking the reference of **Look up table**.

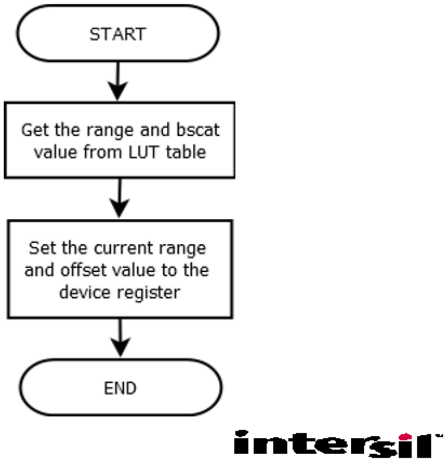
3. prox = FULL SCALE RANGE
4. 100 < prox < 255
5. 0 < prox < 10
6. Prox = 0



measBase
Function are mainly used to calculate the baseline again and again when we have to require to calculate the baseline we select the minimum prox value which we read from the sensor device and make it as a current baseline

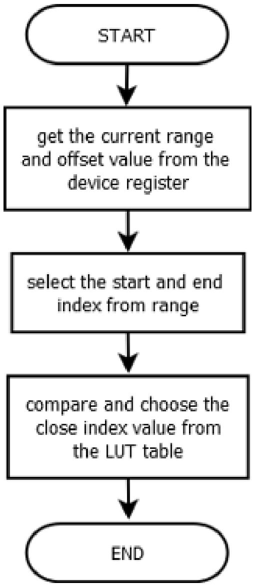


SetProx Offset
Function setproxoffset are mainly used to get the current range, offset prox and scattering value from the sensor device.


START
Get the range and bscat value from LUT table
Set the current range and offset value to the device register
END
intersil

GetProx Offset

Function getproxoffset are mainly used to get the current range, offset prox and scattering value with the reference of LUT index number from the sensor device.


START
get the current range and offset value from the device register
select the start and end index from range
compare and choose the close index value from the LUT table
END

User Interface

The driver implements the user interface using the “linux sysfs” infrastructure.

The driver provides access to three files from the userspace

I. enable
II. debug
III. reg_map

IMPORTANT NOTE: To know how to use the user interface please refer to the android driver integration guide provided by Intersil.

Enable

This sysfs file allows user to enable or disable the proximity conversion and runtime operations.

The following methods implement the read and write implementations of the enable sysfs file

1. enable_show – callback when user reads from this file
2. enable_store – callback when user writes to this file

Debug
This sysfs file allows user to get user friendly view of data related to sensor and also configure some of the parameters.

The following methods implement the read and write implementations of the debug sysfs file

1. show_log – callback when the user reads from this file
2. cmd_hndlr – callback when the user writes to this file

Reg_map
This sysfs file allows user to dump entire register of the ISL29177 sensor device as well as to write to individual registers of the device.

The following methods implement the read and write implementations of the debug sysfs file

1. reg_dump_show – callback when the user read from this file
2. reg_write – callback when the user writes to this file

IMPORTANT NOTE: If the end user has the information on the register details of the device, he / she can debug the device behavior at register level using the reg_map interface.