UM2356
User Manual

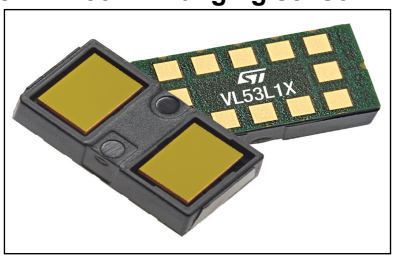
VL53L1X API User Manual

Introduction

VL53L1X is a long-distance ranging time-of-flight sensor.

The purpose of this User Manual is to describe the set of functions to call in order to get ranging data using the VL53L1X driver

Figure 1. VL53L1X ranging sensor module



References

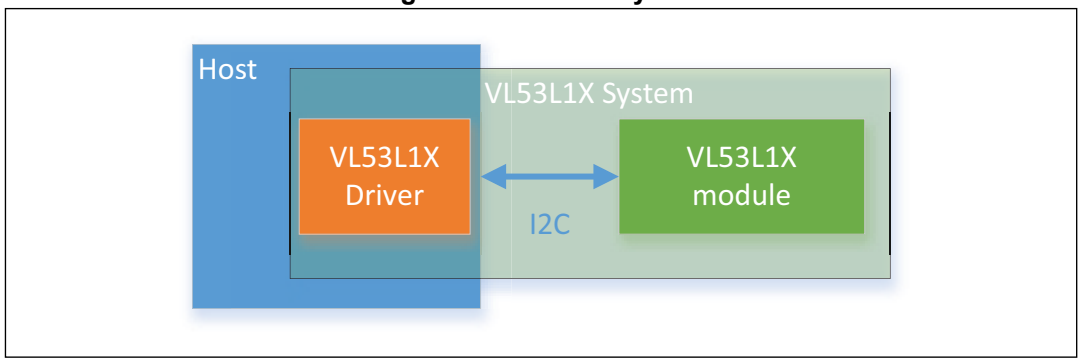
1. VL53L1X Datasheet (DocID029632)

Contents

1 VL53L1X System overview

VL53L1X system is composed of the VL53L1X module and a driver running on the host.

Figure 2. VL53L1X System



ST delivers a software driver, referred to as “Driver” in this document.

This document describes the driver functions accessible to the Host, to control the device and get the ranging data.

The driver is an implementation of a set of functions required to use the VL53L1X device. It does minimal assumptions on the OS integration and services. As such, sequencing of actions, execution/threading model, platform adaptation, and device structure allocation are not part of the driver implementation but are left open to the integrator.

The sequencing of the function calls must follow a set of rules, defined in this document.

2 Ranging API functions description

This section give a functional description of the ranging and describes the API call flow that should be followed to perform a ranging measurement using the VL53L1X.

2.1 Autonomous Ranging description

The sensor performs the ranging continuously and autonomously with a programmable inter-measurement period.

Ranging is done without involvement from the host allowing the host to be in a low-power state. Host is only waken up upon measurement interrupts when a ranging is available.

It is possible to set a threshold of distance and/or signal detection criteria, then an interrupt is raised when the criteria is met.

2.2 Timing considerations

Timing budget is defined as the programmed time needed by the sensor to perform and report a ranging measurement data. During this time, the VCSEL is pulsed. An interrupt is raised or the date ready register is updated at the end of the timing budget.

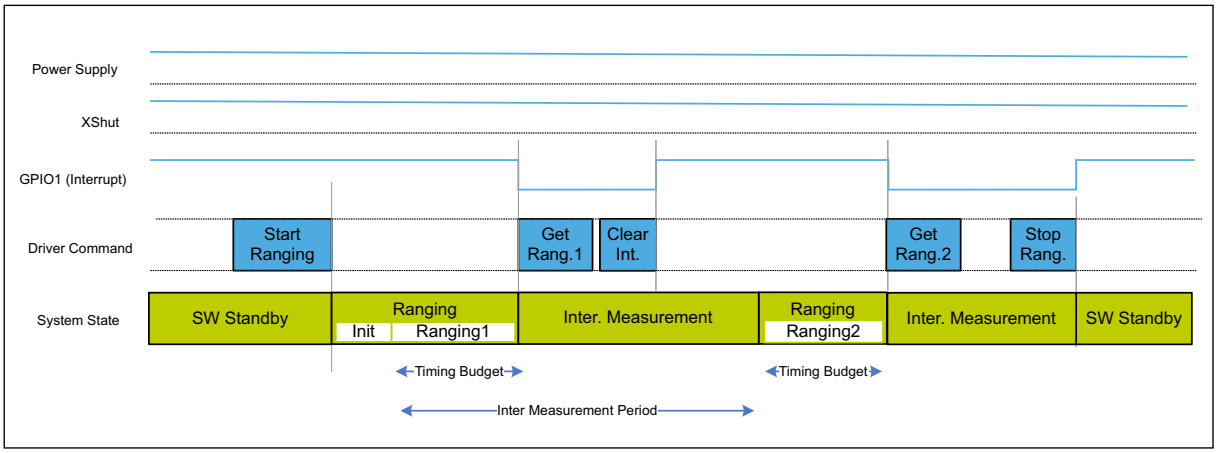
Inter-measurement period is defined as the programmed time between two consecutive measurements.

Figure 3: VL53L1X Autonomous Ranging sequence and timings shows the timing budget and inter-measurement period.

Host can change the default timing budget and inter-measurement period by using a dedicated driver function described in *Chapter 2.5.2: Timing Budget and Inter-measurement period on page 10*.

Host can decide to change timing budget to improve ranging accuracy or max distance limits.

Figure 3. VL53L1X Autonomous Ranging sequence and timings



2.3 API functions call flow

The VL53L1X driver is used in two use cases:

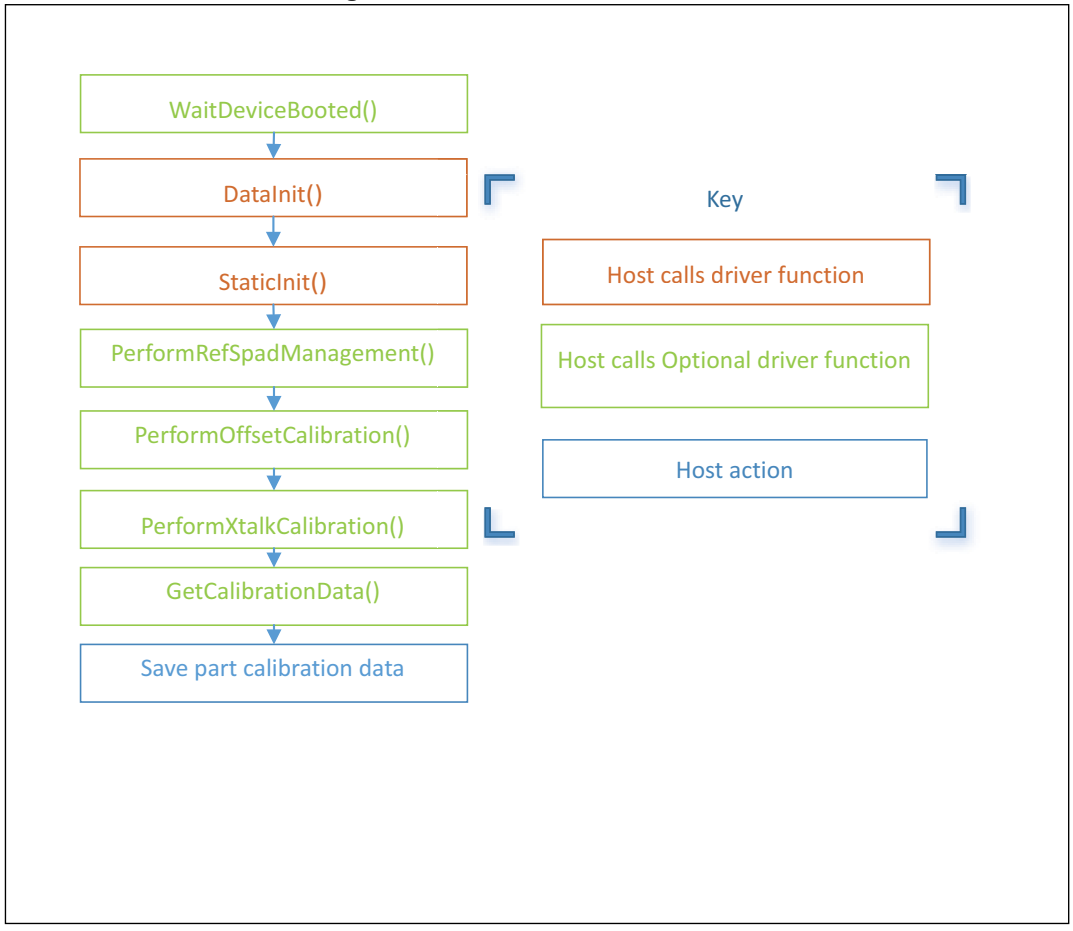
- Calibration flow used for device calibration.
- Ranging flow used at user applications level.

2.3.1 Calibration flow

Calibration flow is described in *Figure 4: VL53L1X Calibration flow on page 6*.

All API functions for calibration are described in *Chapter 3: Calibration functions on page 21*

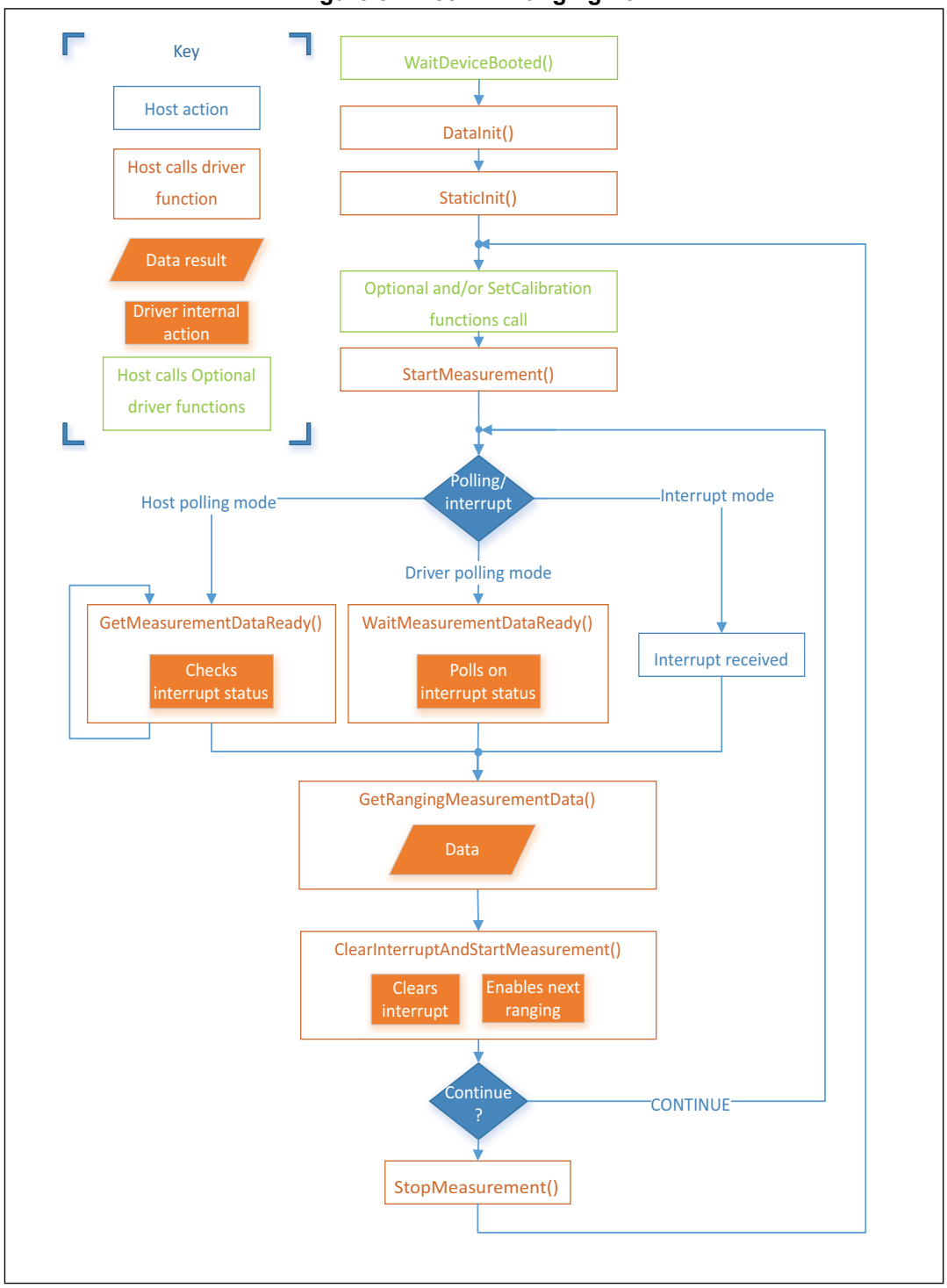
Figure 4. VL53L1X Calibration flow



2.3.2 Ranging flow

Ranging flow is described in *Figure 5: VL53L1X Ranging flow on page 7*

Figure 5. VL53L1X Ranging flow



2.4 Mandatory ranging functions

The following section shows the API functions required to perform the system initialization, before starting a measurement.

2.4.1 Data init

VL53L1_DataInit() function is called one time, and it performs the device initialization.

To be called **once and only once** after device is brought out of reset.

2.4.2 Static Init

VL53L1_StaticInit() function allows to load device settings specific for a given use case.

2.4.3 Start a measurement

VL53L1_StartMeasurement() function must be called to start a measurement.

2.4.4 Waiting for a result: polling or interrupt

There are 3 ways to know that a ranging data is available:

1. **The host can call a polling function to wait until a ranging data is available,**

The function *VL53L1_WaitMeasurementDataReady()* is polling on the device interrupt status until a ranging data is ready.

This function is blocking any other operation on the Host as long as it is not completed, because an internal polling is performed.

2. **The host can poll on a function to ask if a the ranging data is available,**

Host can poll on the function *VL53L1_GetMeasurementDataReady()* to know if a new ranging data is ready.

This function is not blocking. It is the preferred and recommended method if the sensor is used in polling mode.

3. **The host can wait for a physical interrup**t.

An alternative and preferred way to get the ranging status is to use the physical interrupt output: By default, GPIO1 is pulled down when a new ranging data is ready.

This pin is an output pin only, there is no input interrupt pin on this device.

2.4.5 Get measurement

Vl53L1_GetRangingMeasurementData() can be used to get a ranging data.

When calling this function to get the device ranging results, a structure called **VL53L1_RangingMeasurementData_t** is returned.

This structure is described in *Chapter 2.6: RangingMeasurementData structure*.

2.4.6 Clear source of interrupt

Interrupt must be cleared by calling driver function, after reading ranging data, *VL53L1_ClearInterruptAndStartMeasurement()*.

In order to get consistent results, it is mandatory to call this function after getting the ranging measurement.

If this function is not called, the next ranging will be started anyway and the results will be updated. But the data ready status flag will not be updated, and the physical interrupt pin will not be cleared.

2.4.7 Stop a measurements

Host can decide to stop the measurement by calling *VL53L1_StopMeasurement()* function.

If the stop request occurs during a range measurement, then the measurement is aborted immediately.

2.5 Optional driver functions

2.5.1 Wait for boot

VL53L1_WaitDeviceBooted() function allows to ensure that the device is booted and ready.

This function is optional. It is a blocking function because there is an internal polling. This function should not be blocking for more than 4ms, assuming 400kHz I2C and 2ms latency per transaction.

2.5.2 Timing Budget and Inter-measurement period

Timing budget is the time required by the sensor to perform one range measurement.

VL53L1_SetMeasurementTimingBudgetMicroSeconds() is the function to be used.

The minimum and maximum timing budgets are [20ms, 1000ms]

Example of use:

status = VL53L1_SetMeasurementTimingBudgetMicroSeconds(&VL53L1Dev, 66000); sets the timing budget to 66ms.

The function *VL53L1_GetMeasurementTimingBudgetMicroSeconds()* allows to get the programmed timing budget.

Inter-measurement period is the delay between two ranging operations.

An inter-measurement period can be programmable. When a ranging completes, the device waits for the end of the programmed inter-measurement period before resuming the next ranging. In the inter-measurement period, the sensor is in a low-power state.

VL53L1_SetInterMeasurementPeriodMilliSeconds() is the function to be used.

Example of use:

status = VL53L1_SetInterMeasurementPeriodMilliSeconds(&VL53L1Dev, 1000); sets the inter-measurement period to 1s.

The function *VL53L1_GetInterMeasurementPeriodMilliSeconds()* allows to get the programmed inter-measurement period.

Note: If the inter-measurement period is shorter than the timing budget, once the device completed the ranging, the next ranging starts immediately.

Note: The timing budget and inter-measurement period should not be called when the sensor is ranging. User has to stop the ranging, change these parameters and re-start ranging.

2.5.3 Distance mode

A function is provided to optimize the internal settings and tunings to get the best ranging performances depending on the ranging distance required by the application and the ambient light conditions.

The benefit of changing the distance mode is detailed in *Table 1: Distance modes*

Table 1. Distance modes

Possible Distance mode	Maximum distance	Benefit / comments
Short	up to 1.3m	Better Ambient immunity
Medium	up to 3 m	
Long (default)	up to 4 m	Maximum distance

The function to be use is *VL53L1_SetDistanceMode()*.

User can call *VL53L1_GetDistanceMode()* to get the programmed distance mode.

2.5.4 Limit checks setting

Driver uses 2 parameters to qualify the ranging measurement: signal and sigma.

If signal or sigma are outside the limits, the ranging is flagged as invalid (RangeStatus is different than zero)

Applicable limits are:

- Sigma: VL53L1_CHECKENABLE_SIGMA_FINAL_RANGE

Sigma is expressed in mm and is the estimation of the standard deviation of the measurement.

- Signal: *VL53L1_CHECKENABLE_SIGNAL_RATE_FINAL_RANGE*

Signal rate measurement, expressed in MCPS, represents the amplitude of the signal reflected from the target and detected by the device.

Table 2 gives the default limit states and values.

Table 2. Default limit states and values

Limit ID	Default limit state	Default limit value	Associated *RangeStatus*
Sigma	Enabled	15mm	1
Signal	Enabled	1Mcps	2

If user disables the limit checks, ranging values will no more be filtered and possible wrong measurement will be returned by the sensor. In this case, RangeStatus 1 and 2 will never be reported.

Changing limits default settings should be done with care, as the side effects can be important.

The limits change effects on standard deviation and max ranging distance are described in *Table 3*.

Table 3. Signal and Sigma limits change effects

Limit ID	Action	Effect on standard deviation	Effect on max ranging distance
Sigma	Increase limit	-	+
	Decrease limit	+	-
Signal	Increase limit	+	-
	Decrease limit	-	+

Use *VL53L1_SetLimitCheckEnable()* and *VL53L1_GetLimitCheckEnable()* to enable/disable a limit.

The limit value is set using *VL53L1_SetLimitCheckValue()* and *VL53L1_GetLimitCheckValue()*.

Example of use to enable signal check and set the limit to 0.4MCps:

Status = VL53L1_SetLimitCheckEnable(&VL53L1Dev,
VL53L1_CHECKENABLE_SIGNAL_RATE_FINAL_RANGE, 1);

Status = VL53L1_SetLimitCheckValue(&VL53L1Dev,
*VL53L1_CHECKENABLE_SIGNAL_RATE_FINAL_RANGE, 0.40*65536);*

2.5.5 Thresholds

The device can be configured to operate in distance or/and signal threshold detection mode. The ranging data is reported to the host when the pre-configured criteria is matched.

Detection mode

Detection mode allows to select the filtering conditions:

- 0: No filter (default value, standard ranging mode),
- 1: Filter on distance criteria only,

Distance detection mode, based on thresholds:

Distance detection mode (through CrossMode parameter) defines the distance criteria:

- 0: Below a certain distance, "Threshold low"
  - If object distance > Distance Low or no object found: no report
  - If object distance < Distance Low and object found: report
- 1: Beyond a certain distance, "Threshold high"
  - If object distance < Distance High or no object found: no report
  - If object distance > Distance High and object found: report
- 2: Out of a distance range (min/max), "out of Window"
  - Distance Low < detected distance < Distance High: no report
  - Distance Low > detected distance > Distance High: report
- 3: Within a distance range (min/max), "inside Window"
  - Distance Low > detected distance > Distance High: no report
  - Distance Low < detected distance < Distance High: report

Distance Low is the minimum configured distance in millimeter.

Distance High is the maximum configured distance in millimeter.

No Target

This is an alternate detection mode. In the standard use case If no target is detected no ranging is reported. Using no target detection mode (setting IntrNoTarget to 1) allows to generate an interrupt when no target is present.

API function

VL53L1_SetThresholdConfig() is the function to be used.

The VL53L1_DetectionConfig_t structure contains all parameters to be set.

Example of use:

```
detectionConfig.DetectionMode = 1;

detectionConfig.Distance.CrossMode = 3;

detectionConfig.IntrNoTarget = 0;
```

detectionConfig.Distance.High = 1000;

detectionConfig.Distance.Low = 100;

status = VL53L1_SetThresholdConfig(&VL53L1Dev, &detectionConfig);

to program the device to report ranging only when an object is detected within 10 cm and 1m (in this example).

The function *VL53L1_GetThresholdConfig()* allows to get the programmed report threshold configuration.

2.5.6 Region of Interest (ROI) setting

The receiving SPAD array of the sensor is including 16x16 SPADs which cover the full field of view (FoV). It is possible to program a smaller ROI, with a smaller number of SPAD, in order to reduce the FoV.

To set a Region Of Interest (ROI) different than the default 16x16 one, user can call *VL53L1_SetUserROI()* function

ROI is a square or rectangle defined by 2 corners: Top Left and Bottom Right.

4 coordinates are used to localize these 2 corners on the full SPAD array:

- TopLeftX
- TopLeftY
- BotRightX
- BotRightY

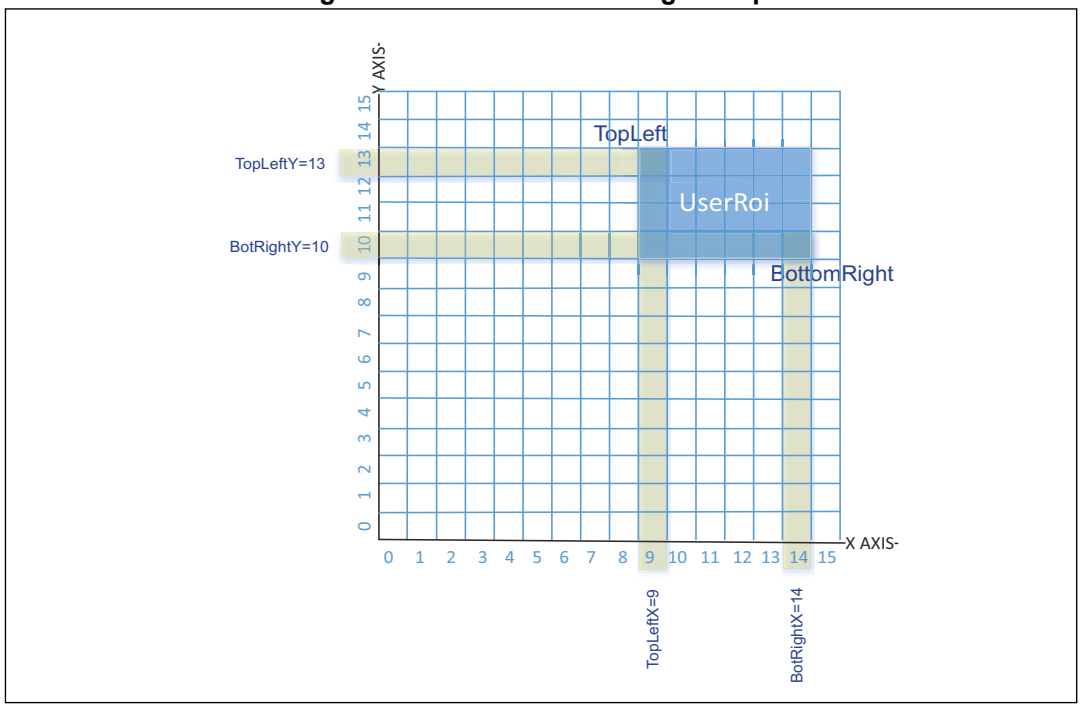
These coordinates are part of a structure of type VL53L1_UserRoi_t.

User has to define the ROI coordinates values in the structure, and call the driver function to apply the ROI change.

Minimum ROI size is 4x4.

An example of ROI setting is given in *Figure 6: VL53L1X ROI setting example*

Figure 6. VL53L1X ROI setting example



VL53L1_UserRoi_t structure contains the coordinate of a ROI:

- TopLeftX: 8 bit integer that gives the top Left x coordinate [0;15]
- TopLeftY: 8 bit integer that gives the top Left y coordinate [0;15]
- BotRightX: 8 bit integer that gives the bottom Right x coordinate [0;15]
- BotRightY: 8 bit integer that gives the bottom Right x coordinate [0;15]

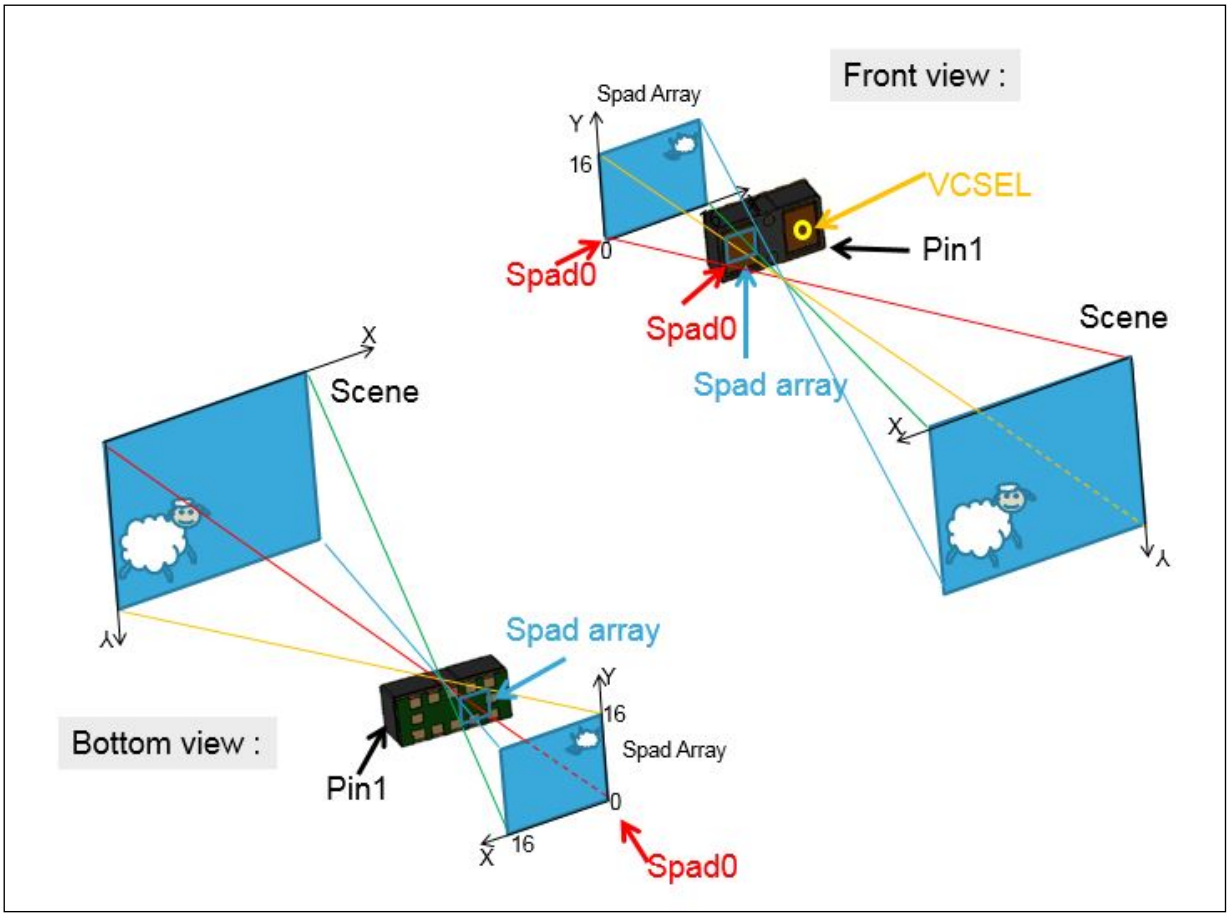
Example of use to set one ROI (based on *Figure 6: VL53L1X ROI setting example*):

```
VL53L1_UserRoi_t roiConfig;
roiConfig.TopLeftX  = 9;
roiConfig.TopLeftY  = 13;
roiConfig.BotRightX = 14;
roiConfig.BotRightY = 10;
status = VL53L1_SetUserROI(&VL53L1Dev, &roiConfig);
```

2.5.7 Spad array coordinates versus scene

Figure 7: VL53L1X coordinates vs Scene shows the coordinates of an object in the spad array compared to the location in the field of view.

Figure 7. VL53L1X coordinates vs Scene



2.5.8 Optical center coordinates

Due to assembly tolerances, the optical center of the device can vary. The optical center of the device is measured for each part. The optical center coordinates are stored in the device NVM.

User has access to the optical center coordinates by calling *VL53L1_GetCalibrationData()*. The returned structure `VL53L1_CalibrationData_t` contains another structure of type VL53L1_optical_centre_t which contains the 2 coordinates (expressed in SPAD number):

- x_centre
- y_centre

Host can use these two coordinates to better align the ROI to the optical center.

2.5.9 VDDIO configuration

As described in the datasheet, user can select 2 modes for VDDIO value: 1V8 or 2V8 modes.

The selection of the mode is done directly in the code though a compilation key called `USE_I2C_2V8k`.

If this compilation key is defined, the system will go in 2V8 mode, otherwise, it will be kept in the default 1V8 mode.

2.6 RangingMeasurementData structure

VL53L1_RangingMeasurementData_t structure is composed of:

- ***TimeStamp***; Not implemented, please ignore it.
- ***StreamCount***; this 8 bit integer gives a counter incremented at each range. The value is first starting at 0, incrementing to 255, and then incrementing from 128 to 255.
- ***RangingQualityLevel***: Not implemented, please ignore it.
- ***SignalRateRtnMegaCps***: this value is the return signal rate in MegaCountPer Second (MCPS), this is a 16.16 fix point value.To obtain real value it should be divided by 65536.
- ***AmbientRateRtnMegaCps***: this value is the return ambient rate (in MCPS), this is a 16.16 fix point value, which is effectively a measure of the infrared light. To obtain real value it should be divided by 65536.
- ***EffectiveSpadRtnCount*** : 16 bit integer that returns the effective SPAD count for the current ranging. To obtain real value it should be divided by 256.
- ***SigmaMilliMeter*** : this 16.16 fix point value is an estimation of the standard deviation of the current ranging, expressed in millimeter. To obtain real value it should be divided by 65536.
- ***RangeMilliMeter***; is a 16 bit integer giving the range distance in millimeter.
- ***RangeFractionalPart***: Not implemented, please ignore it.
- ***RangeStatus***: this is a 8 bit integer giving the range status for the current measurement. Value = 0 means ranging is valid (refer to *Table 4: Range Status*).

Table 4. Range Status

value	RangeStatus String	Comment
0	VL53L1_RANGESTATUS_RANGE_VALID	Ranging measurement is valid.
1	VL53L1_RANGESTATUS_SIGMA_FAIL	Raised if Sigma estimator check is above the internal defined threshold.
2	VL53L1_RANGESTATUS_SIGNAL_FAIL	Raised if Signal value is below the internal defined threshold.
4	VL53L1_RANGESTATUS_OUTOFBOUNDS_FAIL	Raised when phase is out of bounds.
5	VL53L1_RANGESTATUS_HARDWARE_FAIL	Raised in case of HW or VCSEL failure.
7	VL53L1_RANGESTATUS_WRAP_TARGET_FAIL	Wrapped target, not matching phases.
8	VL53L1_RANGESTATUS_PROCESSING_FAIL	Internal algorithm underflow or overflow
14	VL53L1_RANGESTATUS_RANGE_INVALID	The reported range is invalid.

3 Calibration functions

In order to get benefit of full performance of the device, VL53L1X driver includes calibration functions to be run once at customer production line.

Calibration procedures have to be run to compensate the part to part parameters and the presence of the cover glass that may affect the device performances.

Calibration data stored in the host have to be loaded in VL53L1X at each startup using a dedicated driver function.

3 calibrations are needed : RefSPAD, offset and crosstalk.

The order the calibration functions are called does matter : RefSPAD first, offset second and crosstalk third.

The 3 calibration functions can be done sequentially one after other, or individually. When run individually, the previous step data have to be loaded before running current calibration.

3.1 RefSPAD calibration

The number of SPAD is calibrated during Final Module Test at ST. This part to part value is stored into NVM and automatically loaded in the device during boot.

This calibration allows to adjust the number of SPAD to optimize the device dynamic.

However, adding a cover glass on top of the module may affect this calibration. We recommend that the customer performs again this calibration in the final product application.

The same algorithm running at FMT is applied when this function is called: the algorithm searches through the 3 possible types of SPAD: 1 (non attenuated SPAD), 2 (SPAD attenuated by a factor 5) and 3 (SPAD attenuated by a factor 10).

The number and type of SPAD is selected to avoid internal signal saturation.

3.1.1 RefSPAD calibration function

A dedicated function is available for this operation :
VL53L1_PerformRefSpadManagement(&VL53L1Dev)

Note: This function must be called first in the calibration procedure.

3.1.2 RefSPAD calibration procedure

User has to ensure that there is no target closer than 5 cm from the sensor during the calibration.

It is preferred to perform this calibration in the low IR light condition (indoor).

Time to perform this calibration is only few milliseconds.

The *VL53L1_PerformRefSpadManagement* function has to be called after *VL53L1_DataInit()* and *VL53L1_StaticInit()* functions are called. Refer to *Figure 4: VL53L1X Calibration flow on page 6*

When the calibration function is called, the RefSPAD calibration is performed and the RefSPAD new parameters are applied at the end.

3.1.3 Getting RefSPAD calibration results

The function *VL53L1_GetCalibrationData()* allows to get all calibration data. The returned structure *VL53L1_CalibrationData_t* contains in addition a child substructure called *VL53L1_customer_nvm_managed_t* which contains the 8 RefSPAD calibration parameters:

- *ref_spad_man__num_requested_ref_spads* : this value is between 5 and 44. It gives the number of SPAD selected.
- *ref_spad_man__ref_location* : this value can be 1 (non attenuated SPAD), 2 (SPAD attenuated by a factor 5) or 3 (SPAD attenuated by a factor 10)
- 6 additional parameters gives the good spad maps for the location selected.

After the factory calibration, these calibration data have to be stored in the host memory to be loaded at each device start up to avoid re-performing the calibration. Either the user

store the entire structure *VL53L1_CalibrationData_t* or decide to store only the 8 parameters to save memory space.

3.1.4 Setting RefSPAD calibration data

At each start up, after a hard reset, the user can load the RefSPAD calibration data from the host memory. The *VL53L1_SetCalibrationData()* has to be called after *VL53L1_DataInit()* and *VL53L1_StaticInit()* functions are called. Refer to *Figure 5: VL53L1X Ranging flow on page 7*

If the user has optimized the calibration data storage during the calibration, it is recommend to get the entire calibration structure by calling*VL53L1_GetCalibrationData()*, modify the 8 parameters described in the previous section and call *VL53L1_SetCalibrationData()*

3.2 Offset calibration

Soldering the device on the customer board or adding a cover glass can introduce an offset in the ranging distance. This part to part offset has to be measured and compensated during the offset calibration.

3.2.1 Offset calibration function

A dedicated function is available for this operation:

VL53L1_PerformOffsetSimpleCalibration(&VL53L1Dev, CalDistanceMilliMeter)

The argument of the function is the offset calibration distance in millimeters.

Note: Offset calibration has to be performed before the crosstalk calibration and after RefSPAD optimization is done (calibration done or RefSPAD parameters loaded)

3.2.2 Offset calibration procedure

Customer has to use a calibrated chart, placed at a given distance (*CalDistanceMilliMeter*) to perform the offset calibration.

Detail of the recommended setup is given in *Table 5: Offset calibration setup*.

Table 5. Offset calibration setup

Chart	Chart Distance (`CalDistanceMilliMeter`)	Ambient conditions
Grey target (17% reflectance at 940nm)	Recommended value: 140mm	Dark (no IR contribution)

When the calibration function is called, the offset calibration is performed and the offset correction is applied at the end.

3.2.3 Getting offset calibration results

The function *VL53L1_GetCalibrationData()* allows to get all calibration data. The returned structure VL53L1_CalibrationData_t contains in addition a child substructure called VL53L1_customer_nvm_managed_t which contains the main offset calibration result:

- algo__part_to_part_range_offset_mm

3.2.4 Setting offset calibration data

Customer can load the offset calibration data after *VL53L1_DataInit()* and *VL53L1_StaticInit()* functions are called, by using *VL53L1_SetCalibrationData()*

Better is to call *VL53L1_GetCalibrationData()* , modify the parameter described in previous sections (algo__part_to_part_range_offset_mm) and call *VL53L1_SetCalibrationData()*

3.3 Crosstalk calibration

Crosstalk (xtalk) is defined as the amount of return signal received on sensing array which is due to VCSEL light reflection inside the protective window (cover glass) added on top of the module for aesthetic and protective reasons.

Depending on the cover glass quality, the amount of the return signal can be consequent and affects the sensor performances. VL53L1X has a built-in correction that allows to compensate this crosstalk phenomenon.

Crosstalk calibration is used to estimate the amount of the correction needed to compensate the effect of a cover glass added on top of the module.

3.3.1 Cross talk calibration function

A dedicated function is available for this operation :
VL53L1_PerformSingleTargetXTalkCalibration(&VL53L1Dev, XtalkCalDistance);

One argument of the function is the xtalk calibration distance in millimeters.

Note: This function must be called in third position in the calibration flow, after the offset compensation is done (calibration done or offset parameters loaded).

3.3.2 Cross talk calibration procedure

Cross talk calibration should be conducted in a dark environment, with no IR contribution.

When the calibration function is called, the crosstalk calibration is performed and the crosstalk correction is applied at the end.

Table 6. Xtalk calibration setup

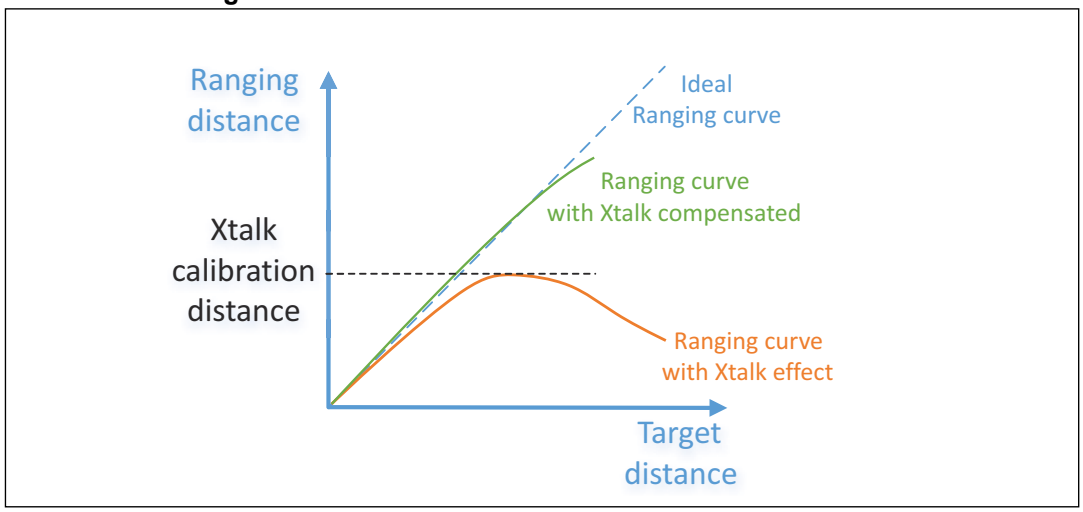
Chart	Chart Distance (`XtalkCalDistance`)	Ambient conditions
Grey target (17% reflectance at 940nm)	As defined in *Chapter 3.3.3*	Dark (no IR contribution)

3.3.3 Xtalk Calibration distance characterization

Xtalk calibration distance needs to be characterized by the user, as it depends on the system environment, mainly the cover glass material and optical properties, and the air gap value (distance between the sensor and the cover glass).

Figure 8: VL53L1X Xtalk calibration distance definition shows the xtalk effect on the ranging curve: from a given distance, the effect of the xtalk is predominant, and the sensor starts to under-range.

Figure 8. VL53L1X Xtalk calibration distance definition



The xtalk calibration distance corresponds to the maximum ranging distance achievable reported by the sensor when the cover glass is present. Refer to *Figure 8: VL53L1X Xtalk calibration distance definition*.

This ranging distance is one argument of the xtalk calibration driver function.

The ranging curve with xtalk corrected is the ranging results when xtalk compensation is applied (when xtalk calibration is completed or after xtalk calibration data are loaded).

3.3.4 Getting Cross talk calibration results

The function *VL53L1_GetCalibrationData()* allows to get all calibration data. The returned structure VL53L1_CalibrationData_t contains in addition a child substructure called VL53L1_customer_nvm_managed_t which contains the xtalk calibration result:

- algo__crosstalk_compensation_plane_offset_kcps

3.3.5 Setting Cross talk calibration data

Customer can load the cross talk calibration data after *VL53L1_DataInit()* and *VL53L1_StaticInit()* functions are called, by using *VL53L1_SetCalibrationData()*

It is recommended to call *VL53L1_GetCalibrationData()*, modify the algo__crosstalk_compensation_plane_offset_kcps parameter in the VL53L1_customer_nvm_managed_t substructure then call *VL53L1_SetCalibrationData()* to apply the xtalk compensation.

3.3.6 Enable/Disable cross talk compensation

The function to call to enable or disable the cross talk compensation is: *VL53L1_SetXTalkCompensationEnable()*.

VL53L1_SetXTalkCompensationEnable(&VL53L1Dev, 0); // disable the crosstalk compensation.

V53L1_SetXTalkCompensationEnable(&VL53L1Dev, 1); // enable the crosstalk compensation.

Note: This function does not perform any xtalk calibration or data loading, it just enables the xtalk compensation. It has to be called before starting ranging, with the optional function calls.

4 Driver errors and warnings

Driver error is reported when any driver function is called. Possible values for driver errors are described in *Table 7*.

Please note that warning are there to inform user that some parameters are not optimized. The warnings are not blocking for the Host.

Table 7. Bare driver errors and warnings description

Error value	API error string	Occurrence
0	VL53L1_ERROR_NONE	No error
-1	VL53L1_ERROR_CALIBRATION_WARNING	Invalid calibration data
-4	VL53L1_ERROR_INVALID_PARAMS	Invalid parameter is set in a function
-5	VL53L1_ERROR_NOT_SUPPORTED	Requested parameter is not supported in the programmed configuration
-6	VL53L1_ERROR_RANGE_ERROR	Interrupt status is incorrect
-7	VL53L1_ERROR_TIME_OUT	Ranging is aborted due to timeout
-8	VL53L1_ERROR_MODE_NOT_SUPPORTED	Requested mode is not supported
-10	VL53L1_ERROR_CALIBRATION_WARNING	Supplied buffer is larger than I2C supports
-14	VL53L1_ERROR_INVALID_COMMAND	Command is invalid in current mode
-16	VL53L1_ERROR_REF_SPAD_INIT	An error occurred during Reference SPADs calibration
-22	VL53L1_ERROR_XTALK_EXTRACTION_FAIL	Thrown when xtalk calibration function has no successful samples to compute the xtalk. In this case there is not enough information to generate new xtalk parm info. The function will exit and leave the current xtalk parameters unaltered
-23	VL53L1_ERROR_XTALK_EXTRACTION_SIGMA_LIMIT_FAIL	Thrown when xtalk calibration function has found that the sigma estimate is above the maximal limit allowed. In this case the xtalk sample is too noisy for measurement. The function will exit and leave the current xtalk parameters unaltered
-24	VL53L1_ERROR_OFFSET_CAL_NO_SAMPLE_FAIL	Thrown when offset calibration function has found no valid ranging.
-28	VL53L1_WARNING_REF_SPAD_CHAR_NOT_ENOUGH_SPADS	Thrown if there are less than 5 good SPADs are available. Ensure calibration setup is in line with ST recommendations.
-29	VL53L1_WARNING_REF_SPAD_CHAR_RATE_TOO_HIGH	Thrown if the final reference rate is greater than the upper reference rate limit - default is 40 Mcps. Ensure calibration setup is in line with ST recommendations.

Table 7. Bare driver errors and warnings description (continued)

Error value	API error string	Occurrence
-30	VL53L1_WARNING_REF_SPAD_CHAR_RATE_TOO_LOW	Thrown if the final reference rate is less than the lower reference rate limit - default is 10 Mcps. Ensure calibration setup is in line with ST recommendations.
-31	VL53L1_WARNING_OFFSET_CAL_MISSING_SAMPLES	Thrown if there is less than the requested number of valid samples. Ensure offset calibration setup is in line with ST recommendations.
-32	VL53L1_WARNING_OFFSET_CAL_SIGMA_TOO_HIGH	Thrown if the offset calibration range sigma estimate is too high. Ensure offset calibration setup is in line with ST recommendations.
-33	VL53L1_WARNING_OFFSET_CAL_RATE_TOO_HIGH	Thrown when signal rate is greater than a limit. Sensor is saturating. Ensure offset calibration setup is in line with ST recommendations.
-34	VL53L1_WARNING_OFFSET_CAL_SPAD_COUNT_TOO_LOW	Thrown when not enough SPADS can be used. Ensure offset calibration setup is in line with ST recommendations.
-41	VL53L1_ERROR_NOT_IMPLEMENTED	Function called is not implemented

5 Acronyms and abbreviations

Table 8. Acronyms and abbreviations

Acronym/ abbreviation	Definition
I2C	Inter-integrated circuit (serial bus)
NVM	Non volatile memory
SPAD	Single photon avalanche diode
VCSEL	Vertical cavity surface emitting laser
FMT	Final Module Test
API	Application Programming Interface

6 Revision history

Table 9. Document revision history

Date	Revision	Changes
Feb-22-2018	1	Initial release.