# UM2039
# User Manual

## World smallest Time-of-Flight ranging and gesture detection sensor Application Programming Interface

## Introduction

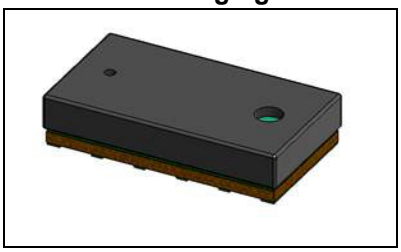
VL53L0X is a ranging and gesture detection sensor.

The purpose of this User Manual is to describe the Application Programming Interface (API), and the calibration procedures from a user perspective.

The API is a turnkey solution. It consists of a set of C functions which enables fast development of end user applications, without the complication of direct multiple register access. The API is structured in a way that it can be compiled on any kind of platform through a well isolated platform layer.

The API package allows the user to take full benefit of VL53L0X capabilities

**Figure 1. VL53L0X ranging sensor module**



**References**

1. VL53L0X Datasheet (DS11555)

# Contents

# 1 Overview

VL53L0X API is based on Photonic Abstraction Layer (PAL) specification. API is defined as the implementation of the PAL.

The API exposes high level functions to be used by the customer application to control the device.

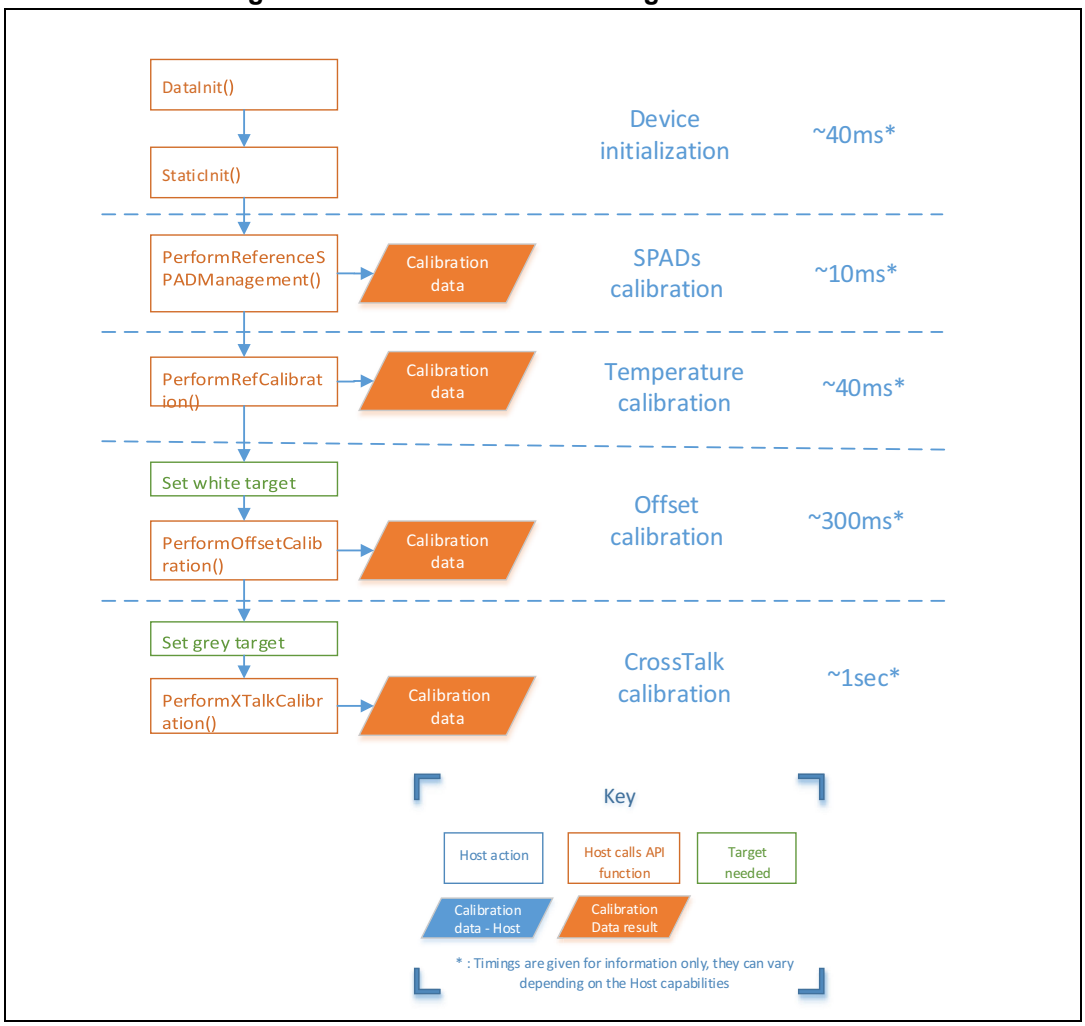
*Note: Full API documentation is available, as part of the API package, in chm and pdf formats.*

# 2 Initial customer manufacturing calibration

There is an initial, once only, calibration step required that should be applied at customer level during the manufacturing process. This flow takes into account all parameters (cover glass, temperature & voltage) from the application.

The customer manufacturing calibration flow is described in *Figure 2* below.

**Figure 2. Customer manufacturing calibration flow**




The following sections detail the API function calls required to perform the initial system calibration.

## 2.1 Data init

`VL53L0X_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.

## 2.2 Static Init

*VL53L0X_StaticInit()* function allows to load device settings specific for a given use case.

## 2.3 Reference SPADs calibration

In order to optimize the dynamic of the system, the reference SPADs have to be calibrated.

This step is performed on the bare modules during Final Module Test at STMicroelectronics, and the calibration data (SPAD numbers and type) are stored into the device NVM.

In case a cover glass is used on top of VL53L0X, the reference SPADs have to be re-calibrated by the customer.

Reference SPAD calibration needs to be done only once during the initial manufacturing calibration, calibration data should then be stored on the Host.

When calibration is performed and calibration data is available on the Host, the data can be loaded without re-performing the calibration.

These functions can be called after *VL53L0X_StaticInit()*. It has to be done before Ref Calibration, *VL53L0X_PerformRefCalibration()*.

### 2.3.1 Reference SPADs calibration procedure

No particular conditions have to be used. The calibration does not require specific target or lighting conditions.
The following procedure has to be performed:

- Call *VL53L0X_PerformRefSpadManagement()*
  - This function outputs the number and type of reference SPADs to be used.
  - At the end of this function, the reference SPADs number and type is programmed in the device.
- Host has to store these 2 values. See *Section 4.4.1* for loading calibration data from Host.

*Note: If a highly reflective target is covering the VL53L0X module during reference SPAD calibration, too much signal will be received on the reference array and the calibration may fail, reporting a '-50' status code. In this case, user has to remove the target away from device*

## 2.4 Ref (temperature) calibration

Ref calibration is the calibration of two parameters (VHV and phase cal) which are temperature dependent. These two parameters are used to set the device sensitivity.

Ref calibration allows the adjustment of the device sensitivity when temperature varies.

Ref calibration must be performed during initial manufacturing calibration, it should be performed again when temperature varies more than 8degC compared to the initial calibration temperature.

If temperature does not vary, the ref calibration data can be loaded without re-performing the calibration procedure.

### 2.4.1 Ref calibration procedure

User has two options:

1. Perform the calibration after `VL53L0X_PerformRefSPADManagement`, by calling `VL53L0X_PerformRefCalibration()`.
2. If user wants to improve the boot time, they can load only the calibration parameters after `VL53L0X_PerformRefSPADManagement` by using `VL53L0X_SetRefCalibration()`. This assumes that user has previously performed a calibration and stored the two parameters in the Host memory. See *Section 4.4.2*.

There is no specific setup needed to perform ref calibration. It has only to be done before offset and cross-talk calibrations, after Reference SPADs management and before the first ranging is performed.

## 2.5 Offset calibration

Range offset is performed during Final Module Test at STMicroelectronics, and the offset is stored into the device NVM.

For some cases, it can appear that the value programmed in the NVM is not correct. This can happen when the customer is using a cover window. In this case, ranging can be affected by an offset, due to the cover window and so the customer should perform a new offset calibration on its manufacturing line.

### 2.5.1 Offset calibration procedure

- Recommendation is to use a white (88%reflectance) target at 100mm, in a dark environment. Target distance can be changed depending on customer's constraints, but it has to be chosen in the linear part of the ranging curve.
- Both Reference SPADs and Ref calibrations have to be performed before calling offset calibration.
- A dedicated API function has to be called to compute the offset: `VL53L0X_PerformOffsetCalibration()`.
- The output of this function is the offset calibration value, in micrometers.
- Offset calibration value has to be stored into Host memory. See *Section 4.4.3* for loading calibration data from Host.

## 2.6 Cross-talk calibration

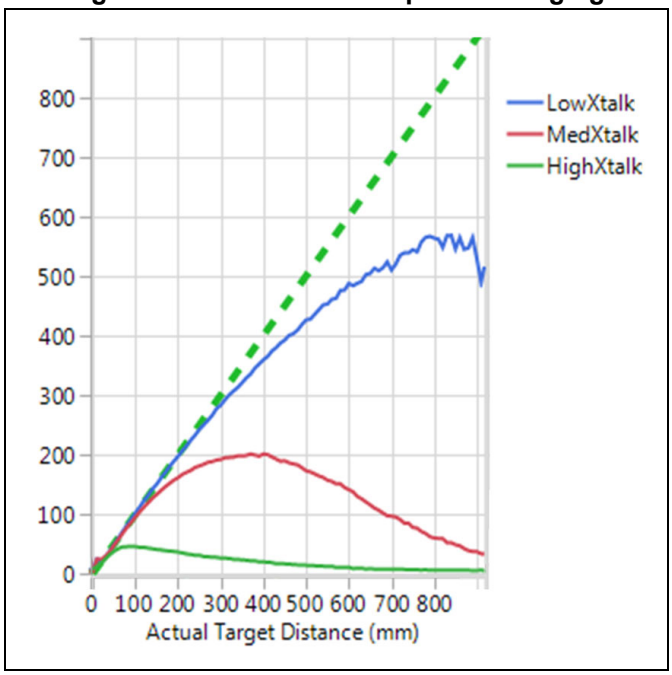
This section presents the effect of the cover window on ranging, and proposes a method for cross-talk calibration.

### 2.6.1 Cover window impact on ranging

The ranging performance is dependent on the quality of the cover window. *Figure 3* shows the cover window impact on ranging with low, medium and high cross-talk.

The ideal curve (with no cover window) is the green dotted line.

**Figure 3. Cover window impact on ranging**




The cross-talk correction is basically a weighted gain applied to the ranging data, based on a calibration result.

Correcting low cross-talk is easier than correcting high cross-talk.

### 2.6.2 Cross-talk calibration distance

The calibration distance is dependent on the quality of the cover window. Low cross-talk or high cross-talk calibration cannot be performed at the same distance.

The starting point of the valid distance to perform cross-talk calibration is when the actual signal starts to deviate from the ideal curve.

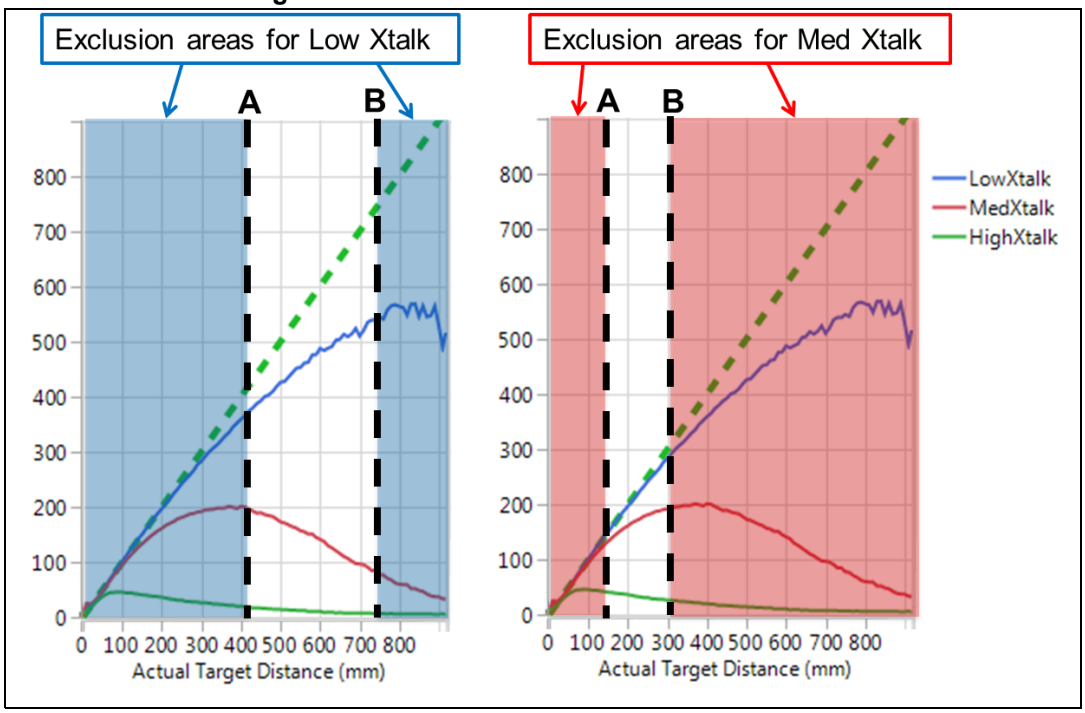
If the cross-talk calibration is performed in the linear area of the ranging curve, the correction factor will be too low, and the correction will have almost no effect.

The valid distance ends when the signal is starting to be too low (ranging distance starting to decrease).

*Figure 4: Cross-talk calibration valid distances* gives an example of exclusion areas where the cross-talk correction should not be performed.

In this figure, the valid distance for cross-talk calibration is from point A to point B.

**Figure 4. Cross-talk calibration valid distances**




### 2.6.3 Cross-talk calibration procedure

The following procedure has to be performed:

- Perform Offset calibration, refer to *Section 2.5: Offset calibration*
- Choose the calibration distance, based on a ranging with cover window to be used, as described in *Section 2.6.2: Cross-talk calibration distance*.
- Use a grey 17% reflectance target.
- Call the API calibration function: `VL53L0X_PerformXTalkCalibration()`.
- The input of this function is the calibration distance in millimeters.
- The output is the cross-talk factor. This has to be stored on Host memory.
- The function applies and enables the cross-talk correction.
- Store the cross-talk factor on the Host memory. See *Section 4.4.4* for loading calibration data from Host.

*Note: The API function `VL53L0X_PerformXTalkCalibration()` performs several measurements, means and computation. Nothing has to be done on the Host side, except calling this function, all is performed by the API.*

# 3 Range profile phases

This section describes the 3 phases that are required to perform the first range measurement after reset, using the VL53L0X.

There are 3 phases:

- Initialization & load calibration data
- Ranging
- Digital housekeeping - see *Section 6.2: Limit settings*.

## 3.1 Initialization/calibration phase

The initialization/calibration flow is described in *Figure 5: Initialization flow on page 11*.

All initialization functions are defined in *Section 4: System initialization/calibration on page 13*.

initialization/calibration phase should be run only after reset or system/setup change.

## 3.2 Ranging phase

The ranging flow is described in *Figure 6: VL53L0X API ranging flow on page 12*.

The first range measurement (after reset) has to be preceded by an initialization and calibration flow.

Basic functions used in the ranging flow are described in *Section 5: Ranging on page 15*.

**Figure 5. Initialization flow**

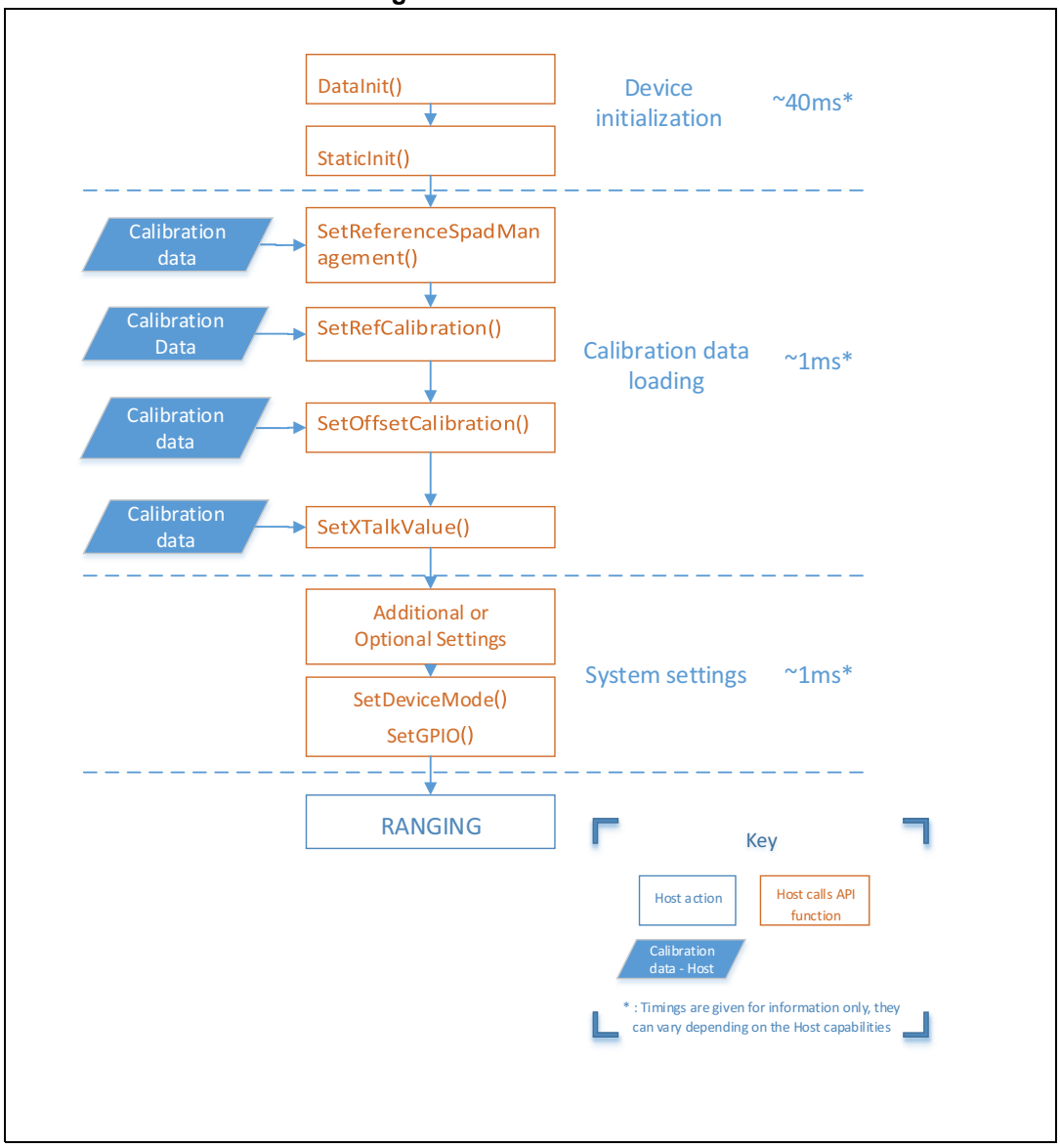
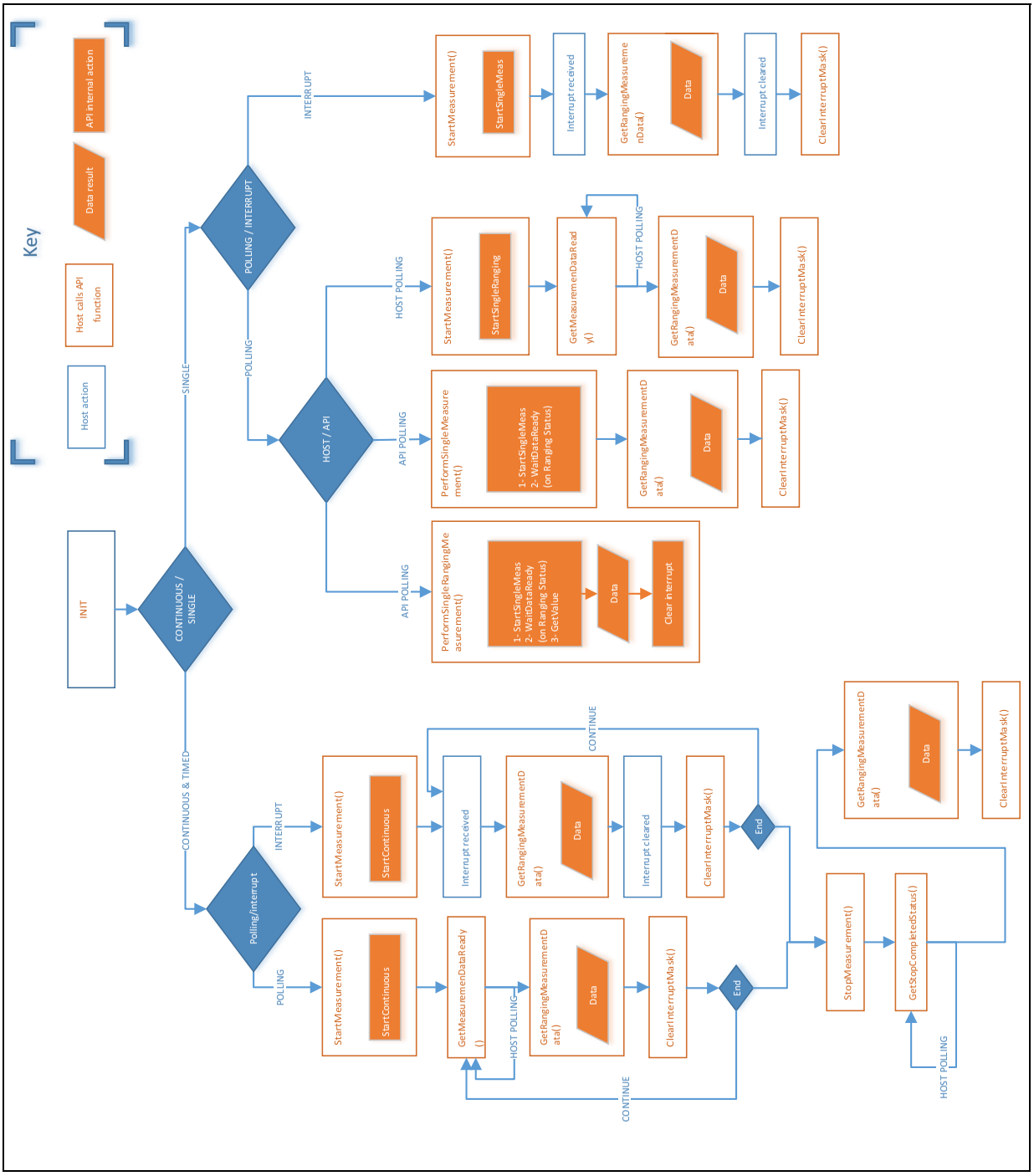


**Figure 6. VL53L0X API ranging flow**



# 4 System initialization/calibration

The following section shows the API function calls required to perform the system initialization, before starting the first range measurement after reset.

## 4.1 2V8 mode

IOVDD supply of VL53L0X sensor can be powered from 1V6 to 3V5:

- Standard mode (default) is used to supply from 1V6 to 1V9
- 2V8 mode is used to supply from 2V6 to 3V5

Originally, a dedicated function was used to program the sensor in 2V8 mode. However, the sensor also supports the supply up to 3V5 in standard mode. Therefore, the function has been removed. The standard mode can be used for all IOVDD power supply from 1V6 to 3V5.

## 4.2 Data init

*VL53L0X_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.

## 4.3 Static Init

*VL53L0X_StaticInit()* function allows to load device settings specific for a given use case.

## 4.4 Load calibration data

### 4.4.1 Reference SPADs

*VL53L0X_SetReferenceSpads()* and *VL53L0X_GetReferenceSpads()* can be used to set/get the number and type of reference SPADs.

If the calibration has not been performed (using *VL53L0X_PerformRefSpadManagement()*), or if the Host has not programmed the number and type of SPADs (using *VL53L0X_SetReferenceSpads()*), *VL53L0X_GetReferenceSpads()* will return the number and type of reference SPADs programmed into the device NVM.

### 4.4.2 Ref calibration

Load calibration parameters by using *VL53L0X_SetRefCalibration()*. This assumes that user has previously performed a calibration and stored the two parameters in the Host memory.

### 4.4.3 Offset calibration

Host has to load the offset calibration data at each startup of the device.

The offset value expressed in micrometers has to be set in the device using API function: *VL53L0X_SetOffsetCalibrationDataMicroMeter()*

### 4.4.4 Cross-talk correction

In a standard usage, the Host will have to program the cross-talk correction factor and enable the cross-talk correction.

The correction factor is the result of the calibration and has to be stored in the Host.

Cross-talk correction value can be set using the API function *VL53L0X_SetXTalkCompensationRateMegaCps()*, and is enabled using *VL53L0X_SetXTalkCompensationEnable()*.

## 4.5 Device mode

*VL53L0X_SetDeviceMode()* selects one of the following modes of operation:

- Single Ranging
- Continuous Ranging
- Continuous Timed Ranging

*VL53L0X_GetDeviceMode()* is used to know which mode is actually programmed.

These modes are described in the VL53L0X datasheet.

## 4.6 Polling and interrupt mode

Once a measurement is ready, the Host either receives an interrupt or poll on a measurement status.

*VL53L0X_SetGPIOConfig()* function configures the system interrupt mode.

Interrupt options and modes of operation are described in a *Section 5: Ranging*.

# 5 Ranging

## 5.1 Start a measurement

The API allows to get a measurement in different ways, depending on the user requirements:

- Start measurement only
- Start measurement and wait for data ready
- Start measurement, wait for data ready and report the data

### 5.1.1 Start measurement only

*VL53L0X_StartMeasurement()* function must be called to start a measurement. The device will start a measurement using the chosen mode (single or continuous)

### 5.1.2 Start measurement and wait for data ready

*VL53L0X_PerformSingleMeasurement()* function starts a measurement and waits for data ready, by polling on the ranging status or on the interrupt status.

The 2 following API functions are called internally:

- *VL53L0X_StartMeasurement()*
- *VL53L0X_GetMeasurementDataReady()*

### 5.1.3 Start measurement, wait for data ready and report the data

*VL53L0X_PerformSingleRangingMeasurement()* function starts a measurement, waits for data ready (by polling on the ranging status or on the interrupt status) and reports the data. This function also clears the interrupt after the measurement.

The 3 following API functions are called internally:

- *VL53L0X_PerformSingleMeasurement()*
- *VL53L0X_GetRangingMeasurementData()*
- *VL53L0X_ClearInterruptMask()*

## 5.2 Stop a measurement

In continuous mode, Host can stop the measurement by calling *VL53L0X_StopMeasurement()* function.

If the stop request occurs during a range measurement, then the measurement is completed before stopping. If it occurs during the inter-measurement period then the stop command takes immediate effect.

If the user wants to call an additional API function after the stop command (for example ref temperature calibration if required - see *Section 2.4*), they have to ensure that the current ranging measurement is finished first.

Therefore it is recommended to call *VL53L0X_GetStopCompletedStatus()* and poll this function to ensure that the ranging measurement is completed, before calling additional API functions.

## 5.3 Get a result

### 5.3.1 Host polling to get the result status

`VL53L0X_GetMeasurementDataReady()` function allows the Host to get a status on the ongoing measurement.

### 5.3.2 Get measurement

`VL53L0X_GetRangingMeasurementData()` function returns the ranging data.

The function returns a buffer which contains the following:

RangeMilliMeter
RangeDMaxMilliMeter:
SignalRateRtnMegaCps
AmbientRateRtnMegaCps
EffectiveSpadRtnCount
RangeStatus: Refer to *Table 1*

**Table 1. Range Status**

| RangeStatus value | RangeStatus String | Comment |
|---|---|---|
| 0 | Range Valid | Ranging measurement is valid |
| 1 | Sigma Fail | Sigma fail will trigger particularly in ambient light, when the amount of ambient light is adding too much noise onto the ranging measurement. |
| 2 | Signal Fail | Signal fail will trigger when the return signal is too low to give enough confidence on the range measured. The limit will be given by either the signal limit or the RIT (Range Ignore Threshold). |
| 3 | Min Range Fail | Not enabled as default. |
| 4 | Phase Fail | Phase fail will trigger when wraparound conditions are detected or when noise on signal is too high. |
| 5 | Hardware Fail | Hardware Fail will trigger if a VCSEL failure, or VHV fail are detected. |
| 255 | No Update | This error should not trigger. |

Ranging status string is available by calling `VL53L0X_GetRangeStatusString()` function

*Note: `VL53L0X_GetDeviceErrorStatus()` function shall be called only for debug purposes. It should not be used, unless requested by ST to customer.*

*Note: SignalRateRtnMegaCps includes the crosstalk correction (if the correction is enabled). If the user wants to get the signal rate without crosstalk correction, they can call `VL53L0X_GetTotalSignalRate()` function.*

# 6 API useful additional functions

## 6.1 Overall timing budget

The timing budget is the time allocated by the user to perform one range measurement. The API takes care of splitting this timing budget into dedicated sub steps in the ranging process.

The user only has to set the overall timing budget in micro seconds, and the function allocates the timings internally. A check is performed to know which part of the scheduler is enabled or disabled, in order to maximize Final Range timing budget.

*VL53L0X_SetMeasurementTimingBudgetMicroSeconds()* and *VL53L0X_GetMeasurementTimingBudgetMicroSeconds()*

The default timing budget value is 33ms, while the minimum is 20ms.

Example of use:

*Status = VL53L0X_SetMeasurementTimingBudgetMicroSeconds(pMyDevice,66000)* sets the overall timing budget to 66ms.

*Note: Increasing the timing budget increases the range measurement accuracy.*

*That is: x N on timing budget => standard deviation / square root of N.*

*For example is the timing budget is increased by a factor of x 2, then the range measurement standard deviation decreases by square root of 2.*

## 6.2 Limit settings

User can enable/disable limit checks and values.

Disabling or relaxing these limits can allow longer ranging, in this case, standard deviation will increase and measurement outliers will be received by the Host.

Applicable limits are:

- Sigma:

  *VL53L0X_CHECKENABLE_SIGMA_FINAL_RANGE*

Sigma is the time difference (shift) between the reference and return SPAD arrays. As Sigma represents time of flight and this translates to distance, this parameter is expressed in mm.

- Return Signal Rate

  *VL53L0X_CHECKENABLE_SIGNAL_RATE_FINAL_RANGE*

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

- Range Ignore Threshold

  *VL53L0X_CHECKENABLE_RANGE_IGNORE_THRESHOLD*

Signal rate minimum threshold. Measurements with signal rates below this value are ignored. This ensures that false measurements are not made due to reflection from the housing.

Use *VL53L0X_SetLimitCheckEnable()* and *VL53L0X_GetLimitCheckEnable()* to enable/disable a limit.

The limit value is set using *VL53L0X_SetLimitCheckValue()* and *VL53L0X_GetLimitCheckValue()*.

Two additional functions give access to the current value and state against which the limit is compared:

*VL53L0X_GetLimitCheckCurrent()* and *VL53L0X_GetLimitCheckStatus()*

*Table 2* gives the default limit states and values.

**Table 2. Default limit states and values**

| Limit ID | Default limit state | Default limit value |
|---|---|---|
| Sigma | Enabled | 18mm |
| Return Signal | Enabled | 0.25Mcps |
| Range Ignore Threshold | Disabled | N x Xtalk Mcps/spad<br>where N = 1.5 by default |

Example of use:

```
Status = VL53L0X_SetLimitCheckEnable(pMyDevice,
VL53L0X_CHECKENABLE_SIGNAL_RATE_FINAL_RANGE, 1);
Status = VL53L0X_SetLimitCheckValue(pMyDevice,
VL53L0X_CHECKENABLE_SIGNAL_RATE_FINAL_RANGE, 0.40*65536);
```

The current sigma value (the actual one, not the limit) can be accessed by calling:

```
Status = VL53L0X_GetLimitCheckCurrent(pMyDevice,
VL53L0X_CHECKENABLE_SIGMA_FINAL_RANGE, &Sigma);
```

## 6.3 Timed ranging

When the ranging mode is set to timed ranging, user has to define the period of time between two consecutive measurements.

*VL53L0X_SetInterMeasurementPeriodMilliSeconds()* and *VL53L0X_GetInterMeasurementPeriodMilliSeconds()*

## 6.4 API versions and product revision

User can get the API version and the PAL specification version.

*VL53L0X_GetVersion()*

*VL53L0X_GetPalSpecVersion()*

*VL53L0X_GetProductRevision()* functions returns the device cut ID.

## 6.5 API state and API error

*VL53L0X_GetPalState()* function gives the state of the API. All different states are given in *Table 3*.

*VL53L0X_GetPalStateString()* function returns the status string.

**Table 3. API state description**

| API state value | API state string | Comment |
|---|---|---|
| 0 | POWERDOWN state | Device is in HW reset |
| 1 | Wait for StaticInit State | Device is initialized and wait for static initialization |
| 2 | STANDBY State | Device is in low power Standby mode |
| 3 | IDLE State | Not used |
| 4 | RUNNING State | Device is performing measurement |
| 98 | UNKNOWN State | Device is in unknown state and needs to be rebooted |
| 99 | ERROR State | Device is in error state and needs to be rebooted |

An API error code is reported when any API function is called. Possible values for API errors are described in *Table 4*.

*VL53L0X_GetPalErrorString()* function returns the error string.

**Table 4. API error values and error strings description**

| API error value | API error string | Occurrence | Possible root cause |
|---|---|---|---|
| 0 | No Error | - | - |
| -1 | Calibration warning error | Not implemented, cannot happen | N/A |
| -2 | Min clipped error | Not implemented, cannot happen | N/A |
| -3 | Undefined error | Not implemented, cannot happen | N/A |
| -4 | Invalid parameters error | Parameter passed is invalid or out of range | Wrong API programming (wrong setting used), or I2C transaction corrupted |
| -5 | Not supported error | Function is not supported in current mode or configuration | Wrong API programming (non implemented function called), or I2C transaction corrupted |
| -6 | Range error | Device reports a ranging error interrupt status | Wrong API programming (syntax error), or no target present during offset calibration, or I2C transaction corrupted |
| -7 | Time out error | Aborted due to time out | Device is functionally failing, or I2C transaction corrupted |
| -8 | Mode not supported error | Requested mode is not supported by the device | Wrong API programming (non existent mode called), or I2C transaction corrupted |
| -9 | Buffer too small | Cannot happen | N/A |

**Table 4. API error values and error strings description (continued)**

| API error value | API error string | Occurrence | Possible root cause |
|---|---|---|---|
| -10 | GPIO not existing | User tried to set up a non-existing GPIO pin | Wrong API programming (wrong setting used), or I2C transaction corrupted |
| -11 | GPIO functionality not supported | Unsupported GPIO functionality | Wrong API programming (wrong setting used), or I2C transaction corrupted |
| -20 | Control Interface Error | Error reported from IO functions (comm error) | I2C comm error |
| -30 | Invalid Command Error | Cannot happen | N/A |
| -40 | Division by zero Error | Cannot occur in cut1.1 | N/A |
| -50 | Reference SPAD Init Error | Error during reference SPAD initialization | Bad NVM programming, module aperture blocked with high reflective target, I2C transaction corrupted (wrong programming) |
| -99 | Not implemented error | Not implemented | N/A |

## 6.6 I2C device address

User can change the I2C address of the device.

*VL53L0X_SetDeviceAddress()*

## 6.7 Reset

This functions resets the device and waits for the boot up.

*VL53L0X_ResetDevice()*

## 6.8 Interrupt settings

Use *VL53L0X_SetGpioConfig()* and *VL53L0X_GetGpioConfig()* to set/get the functionality of the interrupt.

Options are:

- No Interrupt
- Level Low (value < thresh_low)
- Level High (value > thresh_high)
- Out Of Window (value < thresh_low OR value > thresh_high)

There is a dedicated procedure embedded in the API for when the interrupt threshold is programmed to be larger than 254mm and also set to continuous or continuous timed mode. In this case some specific tuning parameters are loaded by the API at each ranging start which will introduce a delay of a few milliseconds (depending on the host I2C performance) to the very first ranging measurement.

In single ranging mode, the maximum programmable threshold is 254mm.

The interrupt threshold behaviour is described in *Table 5*.

**Table 5. Interrupt threshold behaviour**

| Interrupt options | Ranging distance | Threshold value | Ranging mode | GPIO state |
|---|---|---|---|---|
| Level Low | > thresh_low | - | all | no interrupt |
| | < thresh_low | thresh_low < 254mm | all | interrupt at GPIO |
| | < thresh_low | thresh_low > 254mm | continuous or continuous timed | interrupt at GPIO |
| | < thresh_low | thresh_low > 254mm | single | no interrupt (limitation) |
| Level High | < thresh_high | - | all | no interrupt |
| | > thresh_high | thresh_high < 254mm | all | interrupt at GPIO |
| | > thresh_high | thresh_high > 254mm | continuous or continuous timed | interrupt at GPIO |
| | > thresh_high | thresh_high > 254mm | single | no interrupt (limitation) |

`VL53L0X_SetInterruptThresholds()` and `VL53L0X_GetInterruptThresholds()` allow to set/get the interrupt thresholds.

*After reading a measurement, host has to clear the interrupt by using the following function.*

`VL53L0_ClearInterruptMask()`

`VL53L0X_ClearInterruptMask()` and `VL53L0X_GetInterruptMaskStatus()` allow to set/get the interrupt.

Example code is provided within the API release to help the implementation of interrupt settings on the host.

*Note: In Table 5, ranging mode all = continuous, continuous timed and single ranging*

*Note: There is no interrupt generated if no target is detected in threshold high mode (with any threshold value).*

# 7 Example API range profiles

There are 4 range profiles available via API example code.

**Table 6. Example API range profiles**

| | Timing budget | Typical max range | Typical application |
|---|---|---|---|
| Default mode | 30ms | 1.2m (white target) | standard |
| High accuracy | 200ms | 1.2m (white target) | precise measurement |
| Long range | 33ms | 2m (white target) | long ranging, only for dark conditions |
| High Speed | 20ms | 1.2m (white target) | high speed where accuracy is not priority |

## 7.1 High accuracy

The following settings have to be applied, before ranging:

```
if (Status == VL53L0_ERROR_NONE) {
    Status = VL53L0_SetLimitCheckValue(pMyDevice,
    VL53L0_CHECKENABLE_SIGNAL_RATE_FINAL_RANGE,
    (FixPoint1616_t)(0.25*65536));
    }
if (Status == VL53L0_ERROR_NONE) {
    Status = VL53L0_SetLimitCheckValue(pMyDevice,
    VL53L0_CHECKENABLE_SIGMA_FINAL_RANGE,
    (FixPoint1616_t)(18*65536));
    }
if (Status == VL53L0_ERROR_NONE) {
    Status =
    VL53L0_SetMeasurementTimingBudgetMicroSeconds(pMyDevice,
    200000);
    }
```

## 7.2 Long range

The following setting have to be applied, before ranging:

```
if (Status == VL53L0_ERROR_NONE) {
    Status = VL53L0_SetLimitCheckValue(pMyDevice,
    VL53L0_CHECKENABLE_SIGNAL_RATE_FINAL_RANGE,
    (FixPoint1616_t)(0.1*65536));
    }
```

```
if (Status == VL53L0_ERROR_NONE) {
    Status = VL53L0_SetLimitCheckValue(pMyDevice,
    VL53L0_CHECKENABLE_SIGMA_FINAL_RANGE,
    (FixPoint1616_t)(60*65536));
    }
if (Status == VL53L0_ERROR_NONE) {
    Status =
    VL53L0_SetMeasurementTimingBudgetMicroSeconds(pMyDevice,
    33000);
    }
if (Status == VL53L0_ERROR_NONE) {
    Status = VL53L0_SetVcselPulsePeriod(pMyDevice,
    VL53L0_VCSEL_PERIOD_PRE_RANGE, 18);
    }
if (Status == VL53L0_ERROR_NONE) {
    Status = VL53L0_SetVcselPulsePeriod(pMyDevice,
    VL53L0_VCSEL_PERIOD_FINAL_RANGE, 14);
    }
```

## 7.3 High speed

The following setting have to be applied, before ranging:

```
if (Status == VL53L0_ERROR_NONE) {
    Status = VL53L0_SetLimitCheckValue(pMyDevice,
    VL53L0_CHECKENABLE_SIGNAL_RATE_FINAL_RANGE,
    (FixPoint1616_t)(0.25*65536));
    }
if (Status == VL53L0_ERROR_NONE) {
    Status = VL53L0_SetLimitCheckValue(pMyDevice,
    VL53L0_CHECKENABLE_SIGMA_FINAL_RANGE,
    (FixPoint1616_t)(32*65536));
    }
if (Status == VL53L0_ERROR_NONE) {
    Status =
    VL53L0_SetMeasurementTimingBudgetMicroSeconds(pMyDevice,
    20000);
    }
```

# 8 Acronyms and abbreviations

**Table 7. 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 |
| PAL | Photonic Abstraction Layer |
| API | Application Program Interface |
| FMT | Final Module Test |
| XTALK | Cross-talk |

# 9 Revision history

Table 8. Document revision history

| Date | Revision | Changes |
|---|---|---|
| 02-Jun-2016 | 1 | Initial release. |
| 14-Nov-2022 | 2 | Added *Chapter 4.1: 2V8 mode*. |

**IMPORTANT NOTICE – PLEASE READ CAREFULLY**

STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST’s terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of Purchasers’ products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.