A guide to using the VL53L1X ultra lite driver

Introduction

The purpose of this user manual is to describe the main functions of the VL53L1X ultra lite driver (ULD) to program the VL53L1X sensor for ranging distances and detecting objects.

The VL53L1X ULD is an optimized version of the initial VL53L1X driver. The table below indicates the main differences between the two drivers. For example, in the the VL53L1X ULD, the API contains only four files instead of 35, and the code footprint is much smaller (2.3 KB vs. 9 KB in Flash).

The VL53L1X is the third generation of long distance ranging, Time-of-Flight sensors. Please refer to the VL53L1X datasheet for more details.

Figure 1. VL53L1X sensor module

Table 1. Main differences between VL53L1X API and VL53L1X ULD API

	VL53L1X API	VL53L1X ULD API
Code footprint in Flash	9 KB	2.3 KB
Number of files	35	4
Timing budget (ms)	[20 - 500]	[15, 20, 33, 50, 100, 200, 500]
Fast ranging 100 Hz	Yes	No (66 Hz max.)
Dynamic SPAD selection (DSS) (1)	No	Yes

1. *DSS is an internal operation which consists of adapting dynamically the number of active SPADs according to the quantity of returned photons. This avoids saturating the SPAD receiver. DSS is executed at the beginning of each ranging operation.*

1 Acronyms and abbreviations

Table 2. Acronyms and abbreviations

Acronym/abbreviation	Definition
API	application programming interface
cps	counts per second
DSS	dynamic SPAD selection
FMT	final module test
FoV	field of view
I2C	inter-integrated circuit (serial bus)
IR	infrared radiation
IMP	inter-measurement period
kcps	kilo counts per second
NVM	non volatile memory
ROI	region of interest
SPAD	single photon avalanche diode
TB	timing budget
ToF	Time-of-Flight
ULD	ultra lite driver
VCSEL	vertical cavity surface-emitting laser
xcd	crosstalk calibration distance
xtalk	crosstalk

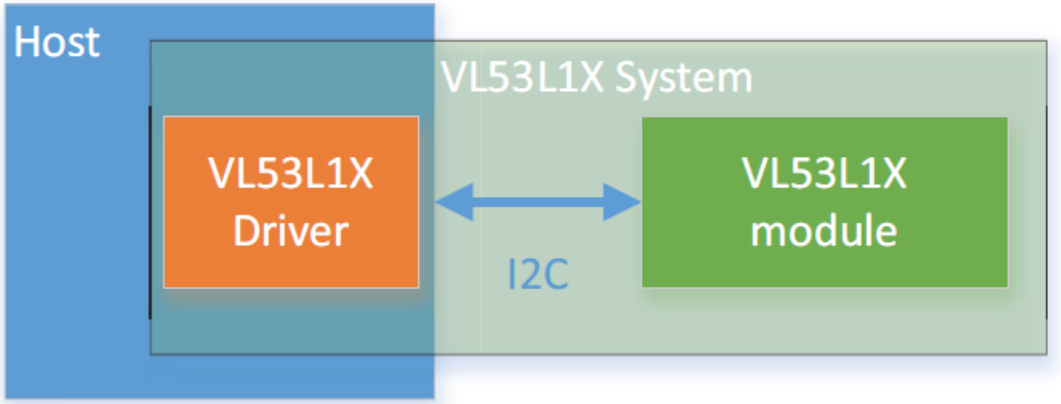
2 VL53L1X system overview

The VL53L1X system is composed of a hardware module and an ultra lite software driver (VL53L1X_ULD) running on a host (see figure below). The hardware module contains the ToF sensor.

ST delivers the software driver which is referred to in this document as "the driver".

This document describes the functions of the driver which are accessible to the host. These functions control the sensor and get the ranging data.

Figure 2. VL53L1X system overview



3 Ranging API functional description

This section gives a functional description of the ranging operations and describes the call flow to be followed to perform a ranging measurement using the VL53L1X sensor.

3.1 Distance ranging description

The sensor ranges continuously and autonomously with a programmable inter-measurement period. Autonomously means that ranging is made without involvement from the host. This allows the host to be in a Low-power state (energy saving). The host is only woken up upon measurement interrupts when ranging data are available. There is no need to issue a new start command to enable new ranging, however a clear interrupt is required after getting ranging data to enable the next interrupt event. Otherwise, the device ranges continuously and no interrupt event occurs to notify the user of the availability of new ranging data once the sensor finishes the ranging sequence.

Note: In this user manual, the term interrupt may designate a physical interrupt on the GPIO1 pin or a "software" interrupt which is a value change in the GPIO__TIO_HV_STATUS register.

3.2 Distance threshold detection description

In addition to the regular ranging capability, the sensor can be programmed to detect an object under certain predefined criteria by using the function VL53LX_SetDistanceThreshold (). The available detection condition use-cases include:

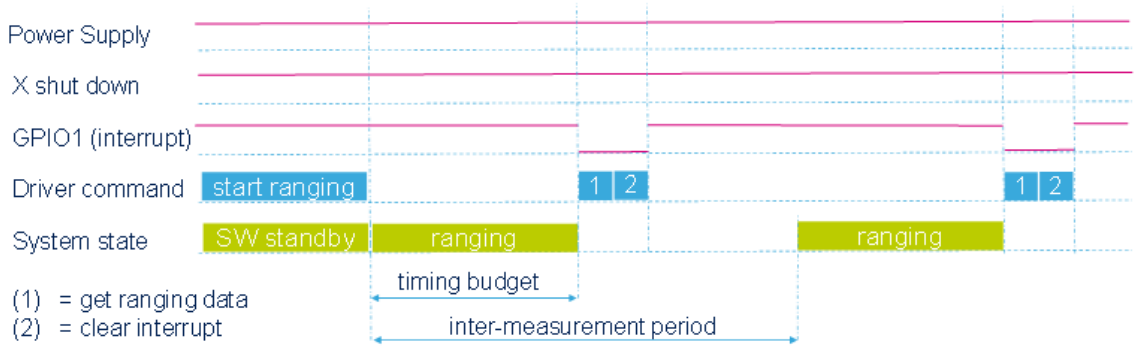
- Object under a certain distance
- Objet beyond a certain distance
- Object within a window limited by a near and far threshold
- Object out of a window limited by a near and far threshold

3.3 Timing considerations

The timing budget (TB) is the time required for the device to make one distance measurement. Increasing the TB improves the measurement reliability but also increases power consumption. So, there is a trade-off between measurement accuracy and power consumption. In the VL53L1X_ULD driver, the TB values available are [15, 20, 33, 50, 100, 200, 500 ms].

The inter-measurement period (IMP) is the time between two consecutive measurements. The IMP must be greater than or equal to the TB otherwise the actual IMP is double the expected value. Note that there is no automatic check in the VL53L1X_ULD driver, so the user has to verify this condition when setting the IMP.

Figure 3. Ranging sequence and timings



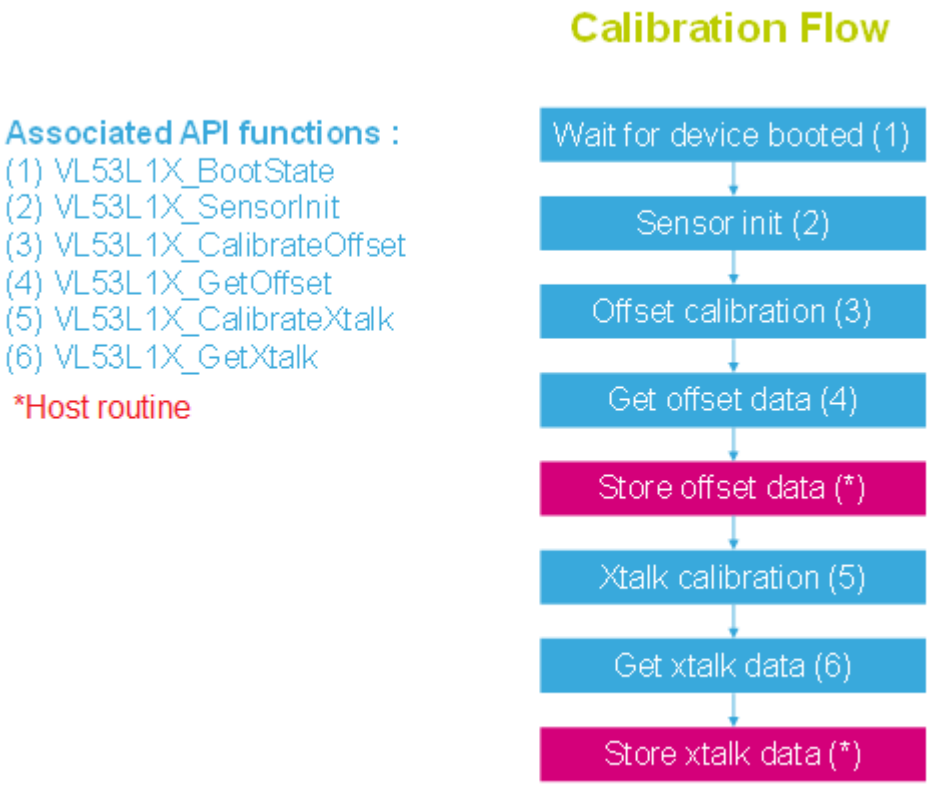
3.4 API function call flow

The VL53L1X_ULD driver is used in the following two use cases:

- Calibration flow is used to calibrate the sensor. Such calibration is done during manufacturing. The calibration data must be stored in the host sytem and sent to the sensor during the bootup stage.
- Ranging flow is used for distance measurements or object detection operations in the application software.

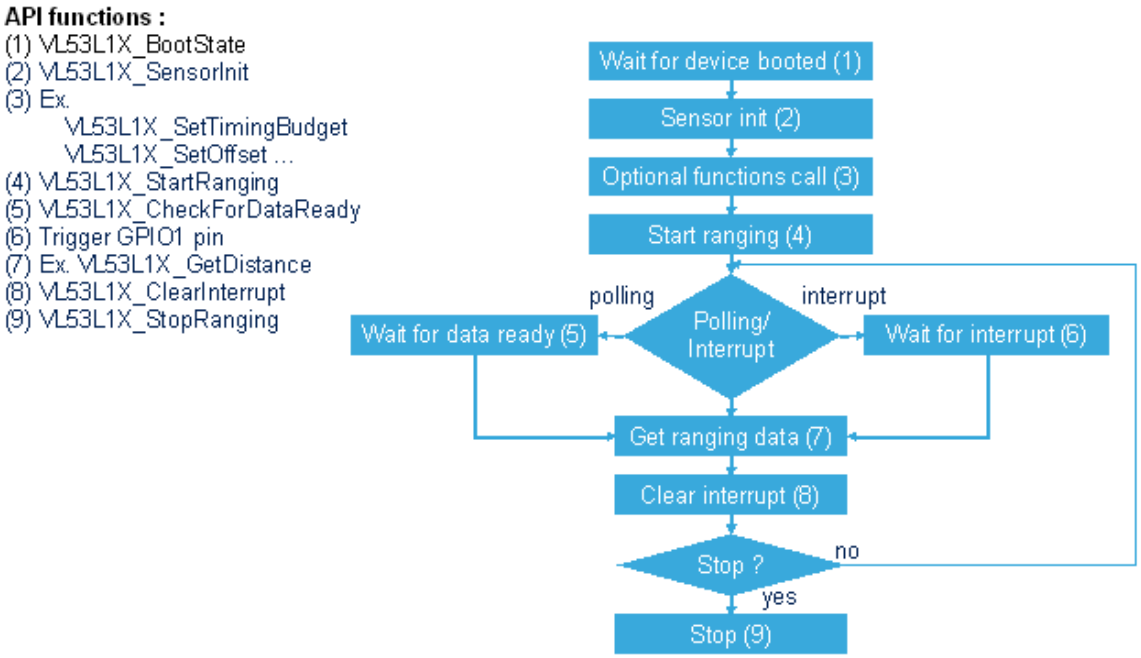
3.4.1 Calibration flow

Figure 4. Calibration flow



3.4.2 Ranging flow

Figure 5. Ranging flow



3.5 Mandatory functions

Mandatory functions include:

- VL53L1X_BootState
- VL53L1X_SensorInit
- VL53L1X_StartRanging
- VL53L1X_CheckForDataReady
- VL53L1X_GetDistance
- VL53L1X_ClearInterrupt
- VL53L1X_Stop

3.5.1 Sensor boot

The VL53L1X_BootState function is used to check that the sensor has booted. It is strongly recommended to ensure that the sensor finishes booting before the first I2C access.

3.5.2 Sensor init

The VL53L1X_SensorInit function is called once to initialize the sensor with a default configuration. With this default configuration, the sensor ranges at 10 Hz in Long-distance mode.

3.5.3 Start a measurement

The VL53L1X_StartRanging function is used to make a distance measurement.

3.5.4 Wait for notification ranging data to be ready

There are two ways to receive notification that ranging data are ready:

- The host can poll a register value change. This is done by using the VL53L1X_CheckForDataReady which returns “1” when new ranging data are ready.
- The host can wait for a physical interrupt event. The interrupt event can be triggered at the GPIO1 pin. The interrupt polarity can be set by the VL53L1X_SetInterruptPolarity function. This changes the interruption transition (from high to low or from low to high)

3.5.5 Get ranging data

Each type of ranging data has its own get ranging data functions. The VL53L1X_GetRangeStatus and VL53L1X_GetDistance are the main get ranging data functions. If required, other ranging data parameters can be read, for example, return signal, ambient rate, etc.

3.5.6 Clear interrupt

Use the VL53L1X_ClearInterrupt function to clear the interrupt. It is strongly recommanded to clear the interrupt to enable the next interrupt event when new ranging data are ready.

3.5.7 Stop the measurement

Ranging occurs continuously. To stop the current ranging operation, the user can use the stop command, VL53L1X_Stop (). If the stop command is issue during the ranging operation, the sensor completes the current ranging operation before stopping.

3.6 Optional functions

3.6.1 Setting distance mode

Two distance modes are available: Short and Long (default). Short mode has better ambient light immunity but the maximum distance measurable is limited to 1.3 m. Long distance mode ranges up to 4 m but is less performant under ambient light.

Use the function VL53L1X_SetDistanceMode (dev, DM) to set the distance mode. For Short mode, DM = 1 and for Long mode, DM = 2.

VL53L1X_GetDistanceMode returns the current distance mode.

3.6.2 Inter-measurement period

The inter-measurement period (IMP) is the time between two ranging operations (see Section 3.3 Timing considerations).

Use the VL53L1X_SetInterMeasurementInMs function to set the intermeasurement period in milliseconds. VL53L1X_GetIntermeasurementInMs returns the curent IMP. The default IMP is 100 ms.

The IMP must be greater than or equal to the TB otherwise the start of ranging is missed and the intermeaurement period is effectively doubled. The user has to verify the TB and the intermeasurement period as there is no automatic check in the API.

3.6.3 Setting the timing budget

The TB is the time required by the sensor to make one distance measurement (see Section 3.3 Timing considerations).

Use the VL53L1X_SetTimingBudget function to set the TB in milliseconds. The TB values available are [15, 20, 33, 50, 100, 200, 500]. This function must be called after VL53L1X_SetDistanceMode.

Note: 15 ms only works with Short distance mode. 100 ms is the default value.

The TB can be adjusted to improve the standard deviation (SD) of the measurement. Increasing the TB, decreases the SD but increases the power consumption. For example, VL53L1X_SetTimingBudgetInMs (dev, 50) sets the TB to 50 ms. VL5L31X_GetTimingBudget returns the current TB.

3.6.4 Setting threshold and detection mode

In addition to the regular ranging features, the sensor also has the capability to detect a target with predefined detection criteria. When the preconfigured criteria are matched the sensor raises an interrupt and returns the distance.

Use the VL53L1X_ERROR VL53L1X_SetDistanceThreshold (dev, ThreshLow, ThreshHigh, Window, IntOnNoTarget) to configure the detection criteria.

- ThreshLow is the low distance threshold in millimeters
- ThreshHigh is the high distance threshold in millimeters
- Window :
  - Window = 0: Below a certain distance
    - If object distance > ThreshLow or no object found: no report
    - If object distance < ThreshLow and object found: report
  - Window = 1: Beyond a certain distance
    - If object distance < ThreshHigh or no object found: no report
    - If object distance > ThreshHigh and object found: report
  - Window = 2: Out of distance range (min/max), "out of Window"
    - ThreshLow < object distance < ThreshHigh: no report
    - ThreshLow > object distance > ThreshHigh: report
  - Window = 3: Within the distance range (min/max), "inside Window"
    - ThreshLow > object distance > ThreshHigh: no report
    - ThreshLow < object distance < ThreshHigh: report
- IntOnNoTarget is no longer used, set to 1

To go back to the regular ranging mode, write 0x20 to the 0x46 register (8 bits).

3.6.5 Setting ROI

The receiving SPAD array of the sensor consists of 16x16 SPADs which cover the full FoV.

It is possible to program a smaller ROI, with a smaller number of SPADs, to reduce the FoV for applications which require a narrow FoV.

Use the function VL53L1X_SetROI (dev, X, Y).

- X is the ROI width from 4 to 16, if X < 4 the Firmware limits the width to 4
- Y is the ROI height from 4 to 16, if Y < 4 the Firmware limits the height to 4

VL53L1X_GetROI returns the current ROI.

Note: Lowering the number of SPADs limits the maximum ranging distance. Some experimentation is required.

3.7 Ranging example pseudo code

```
main (void ){
        /* Platform Initialization code here */
        ...
        /* Wait for device booted */
        While(state){
                Status = VL53L1X_BootState(dev, &state);
                HAL_delay(2);
        }
        /* Sensor Initialization */
        Status = VL53L1X_SensorInit()
        /* Modify the default configuration */
        Status = VL53L1X_SetInterMeasurementPeriod();
        Status = VL53l1X_SetOffset();
        /* enable the ranging*/
        Status = VL53L1X_StartRanging();
        /* ranging loop */
        While(1){
            While(dataReady==0){
                    Status = VL53L1X_CheckForDataReady(dev, &dataReady);
               }
               dataReady = 0;
               Status = VL53L1X_GetRangeStatus();
               Status = VL53L1X_GetDistance();
               Status = VL53L1X_ClearInterrupt();
        }
}
```

4 Calibration functions

To benefit from the full performance of the sensor, the VL53L1X_ULD driver includes two calibration functions (offset and crosstalk) which need to be run once at the production line. These calibration procedures have to be run to compensate for device-to-device variation (i.e. when the absolute measured distance is not the same from one device to another) and the presence of the cover glass that may affect the device ranging performances.

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

The calibration sequence order is important. Offset calibration should be run first followed by crosstalk calibration.

The two calibrations may be run sequentially one after the other, or may be run individually. When run individually, make sure the offset data have been programmed into the sensor before running crosstalk calibration.

4.1 Offset calibration

Use the VL53L1X_CalibrateOffset () function to run offset calibration.

VL53L1X_CalibrateOffset () finds the offset, applies the offset, and returns the offset correction value.

The offset correction value returned by VL53L1X_CalibrateOffset () must be stored in the host system and writen to the sensor during boot up with VL53L1X_SetOffset ().

4.1.1 Offset calibration procedure

Place a target, 17 % gray, at 140 mm from the sensor and call the VL53L1X_CalibrateOffset (dev, 140, &offset) function.

The calibration may take a few seconds. The offset correction is applied to the sensor at the end of calibration.

4.1.2 Getting offset calibration

The function VL53L1X_GetOffset () returns the current offset correction applied to the sensor.

A value of zero means no offset has been applied.

4.1.3 Setting offset calibration

The function VL53L1X_SetOffset () applies the offset value in millimeters to the sensor.

The user may use this function to apply, to the sensor, the offset found during calibration which is stored in the host system.

4.2 Crosstalk calibration

In imaging, crosstalk is defined as the amount of return signal received on the 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 purposes.

Depending on the cover glass quality, the amount of return signal may be significant and may affect sensor performance. The VL53L1X has a built-in correction that allows the user to compensate for this crosstalk phenomenon.

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

4.2.1 Crosstalk calibration function

Use the function VL53L1X_CalibrateXtalk () to perform crosstalk calibration.

The VL53L1X_ CalibrateXtalk () finds the crosstalk compensation value, applies the correction, and returns the crosstalk correction value.

The crosstalk compensation value returned by VL53L1X_CalibrateXtalk () must be stored in the host system and applied to the sensor during boot up using VL53L1X_SetXtalk ().

4.2.2 Crosstalk calibration procedure

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

Place a 17 % reflectance chart at the crosstalk calibration distance (xcd), then call the VL53l1X_CalibrateXtalk(dev, xcd, &xtalk) to perform the calibration. It may take a few seconds.

To characterize the xcd, refer to Section 4.2.3 Crosstalk calibration distance characterization.

4.2.3 Crosstalk calibration distance characterization

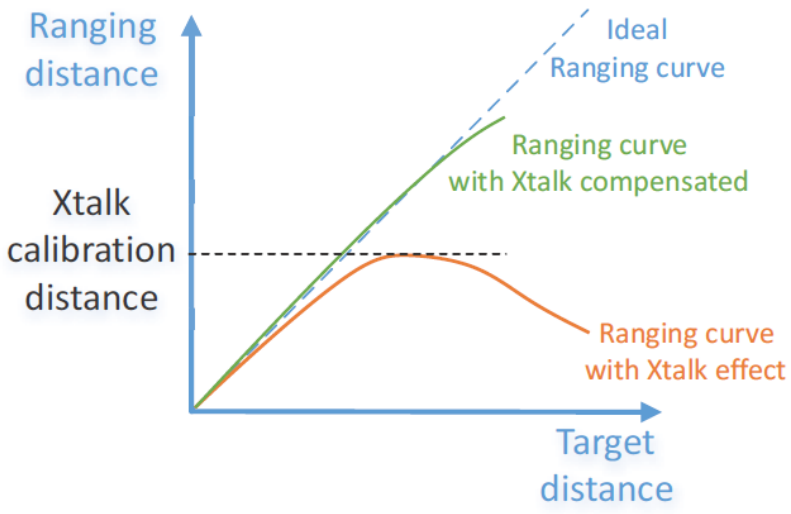
The crosstalk calibration distance needs to be characterized as it depends on the system environment which mainly includes:

- The cover glass material and optical properties
- The air gap value i.e. the distance between the sensor and the cover glass

Do a full sweep with the target from near to far, noting the resulting measurement. You will get a plot similar to Figure 6). At some point, the actual value and the measured value start to diverge. This is the crosstalk calibration distance.

The figure below shows the crosstalk effect on the ranging curve. From a given distance, the effect of crosstalk is predominant, and the sensor starts to under range.

Figure 6. VL53L1X crosstalk calibration distance definition



The crosstalk calibration distance corresponds to the maximum ranging distance achievable by the sensor when the cover glass is present.

The ranging curve with crosstalk corrected is the ranging result when crosstalk compensation is applied i.e. when crosstalk calibration is completed or after crosstalk calibration compensation is applied.

4.2.4 Getting crosstalk calibration data

The function VL53L1_GetXtalk () returns the current crosstalk value programmed in the sensor. Zero means there is no crosstalk compensation. The unit is cps (counts per second).

4.2.5 Setting crosstalk calibration data

The function VL53L1X_SetXtalk () applies the crosstalk value, in cps, to the sensor.

The user may use this function to apply, to the sensor, the crosstalk correction found during calibration and stored in the host system.

5 Range status interpretation

There are five range statuses: 0, 1, 2, 4, and 7. When the range status is 0, there is no error. Range status 1 and 2 are error warnings while range status 4 and 7 are errors.

When the range status is 1, there is a sigma failure. This means that the repeatability or standard deviation of the measurement is bad due to a decreasing signal noise ratio. Increasing the timing budget can improve the standard deviation and avoid a range status 1.

When the range status is 2, there is a signal failure. This means that the return signal is too week to return a good answer. The reason is because the target is too far, or the target is not reflective enough, or the target is too small. Increasing the timing buget might help, but there may simply be no target available.

When the range status is 4, the sensor is "out of bounds". This means that the sensor is ranging in a “non-appropriated” zone and the measured result may be inconsistent. This status is considered as a warning but, in general, it happens when a target is at the maximum distance possible from the sensor, i.e. around 5 m. However, this is only for very bright targets.

Range status 7 is called "wraparound". This situation may occur when the target is very reflective and the distance to the target/sensor is longer than the physical limited distance measurable by the sensor. Such distances include approximately 5 m when the senor is in Long distance mode and approximately 1.3 m when the sensor is in Short distance mode. Example: a traffic sign located at 6 m can be seen by the sensor and returns a range of 1 m. This is due to “radar aliasing”: if only an approximate distance is required, we may add 6 m to the distance returned. However, that is a very approximate estimation.

6 Reducing the driver size to its absolute minimum

Although ST have attempted to create a small driver, it might not be small enough for some applications. To create the smallest possible driver, try the following approach:

- Use the current driver and configure the parameters as required. Start ranging (to show that it works).
- When satisfied that ranging works, dump the contents of the sensor from the location 0x2D through to 0x87.
- Replace the default register settings in the the array VL51L1X_DEFAULT_CONFIGURATION[] which is located in the file VL53L1X_api.c.
- This makes your settings the default settings
- Your main routine then needs only to call the the VL53L1X_SensorInit() function and VL53L1X_StartRanging().
- Most other functions can then be removed

7 Function descriptions

In the descriptions below, most functions return an error status. 0 is successful. Most values are unsigned 16-bit integers except where noted. Most ‘Set’ functions are paired with a corresponding ‘Get’ function.

7.1 GetSWVersion

```
VL53L1X_ERROR VL53L1X_GetSWVersion(VL53L1X_Version_t *pVersion);
```

This function returns the SW driver version.

7.2 SetI2CAddress

```
VL53L1X_ERROR VL53L1X_SetI2CAddress(uint16_t, uint8_t new_address);
```

This function sets the sensor I2C address used for a multiple device application. The default address for the sensor at boot up is 0x52.

7.3 SensorInit

```
VL53L1X_ERROR VL53L1X_SensorInit(uint16_t dev);
```

This function loads the 45 bytes of configuration values which initialize the sensor.

7.4 VL53L1X ClearInterrupt

```
VL53L1X_ERROR VL53L1X_ClearInterrupt(uint16_t dev);
```

This function clears the interrupt to be called after a ranging data reading, to arm the interrupt for the next data ready event.

7.5 SetInterruptPolarity

```
VL53L1X_ERROR VL53L1X_SetInterruptPolarity(uint16_t dev,
    uint8_t IntPol);
```

This function programs the interrupt polarity, 1 = active high (default), 0 = active low.

7.6 GetInterruptPolarity

```
VL53L1X_ERROR VL53L1X_GetInterruptPolarity(uint16_t dev,
    uint8_t *pIntPol);
```

This function returns the current interrupt polarity, 1 = active high (default), 0 = active low.

7.7 StartRanging

```
VL53L1X_ERROR VL53L1X_StartRanging(uint16_t dev);
```

This function starts the ranging distance operation which is continuous. The clear interrupt has to be done after each "get data" to allow the interrupt to be raised when the next data are ready. 1 = active high (default), 0 = active low. If required, use SetInterruptPolarity() to change the interrupt polarity.

7.8 StopRanging

```
VL53L1X_ERROR VL53L1X_StopRanging(uint16_t dev);
```

This function stops the ranging.

7.9 CheckForDataReady

```
VL53L1X_ERROR VL53L1X_CheckForDataReady(uint16_t dev,
    uint8_t *isDataReady);
```

This function checks if the new ranging data are available by polling the dedicated register. Data are not ready when "isDataReady == 0". Data are ready when "isDataReady==1".

7.10 SetTimingBudgetInMs

```
VL53L1X_ERROR VL53L1X_SetTimingBudgetInMs(uint16_t dev,
    uint16_t TimingBudgetInMs);
```

This function programs the timing budget in ms. The predefined values are 15, 20, 50, 100, 200, and 500. This function must be called after the VL53L1X_SetDistanceMode.

7.11 GetTimingBudgetInMs

```
VL53L1X_ERROR VL53L1X_GetTimingBudgetInMs(uint16_t dev,
    uint16_t *pTimingBudgetInMs);
```

This function returns the current timing budget in ms.

7.12 SetDistanceMode

```
VL53L1X_ERROR VL53L1X_SetDistanceMode(uint16_t dev,
    uint16_t DistanceMode);
```

This function programs the distance mode (1 = Short, 2 = Long). Short mode maximum distance is limited to 1.3 m but results in a better ambient immunity. Long mode can range up to 4 m in the dark with a timing budget of 200 ms.

7.13 GetDistanceMode

```
VL53L1X_ERROR VL53L1X_GetDistanceMode(uint16_t dev,
    uint16_t *pDistanceMode);
```

This function returns the current distance mode (1 = Short, 2 = Long).

7.14 SetInterMeasurementInMs

```
VL53L1X_ERROR VL53L1X_SetInterMeasurementInMs(uint16_t dev,
    uint32_t InterMeasurementInMs);
```

This function programs the intermeasurement period (IMP) in ms. The IMP must be greater than or equal to the timing budget. This condition is not checked by the API, so the customer must check this condition.

7.15 GetInterMeasurementInMs

```
VL53L1X_ERROR VL53L1X_GetInterMeasurementInMs(uint16_t dev,
    uint16_t * pIM);
```

This function returns the intermeasurement period in ms.

7.16 BootState

```
VL53L1X_ERROR VL53L1X_BootState(uint16_t dev, uint8_t *state);
```

This function returns the Boot state of the device (1 = booted, 0 = not booted).

7.17 GetSensorId

```
VL53L1X_ERROR VL53L1X_GetSensorId(uint16_t dev, uint16_t *id);
```

This function returns the sensor ID which must be 0xEEAC.

7.18 GetDistance

```
VL53L1X_ERROR VL53L1X_GetDistance(uint16_t dev,
    uint16_t *distance);
```

This function returns the distance measured by the sensor in mm.

7.19 GetSignalPerSpad

```
VL53L1X_ERROR VL53L1X_GetSignalPerSpad(uint16_t dev,
    uint16_t *signalPerSp);
```

This function gives the returned signal per SPAD in kcps/SPAD.

7.20 GetAmbientPerSpad

```
VL53L1X_ERROR VL53L1X_GetAmbientPerSpad(uint16_t dev,
    uint16_t *amb);
```

This function returns the ambient per SPAD in kcps/SPAD.

7.21 GetSignalRate

```
VL53L1X_ERROR VL53L1X_GetSignalRate(uint16_t dev,
    uint16_t *signalRate);
```

This function gives the returned signal in kcps.

7.22 GetSpadNb

```
VL53L1X_ERROR VL53L1X_GetSpadNb(uint16_t dev, uint16_t *spNb);
```

This function returns the current number of enabled SPADs.

7.23 GetAmbientRate

```
VL53L1X_ERROR VL53L1X_GetAmbientRate(uint16_t dev,
    uint16_t *ambRate);
```

This function returns the ambient rate in kcps.

7.24 VL53L1X_GetRangeStatus

```
VL53L1X_ERROR VL53L1X_GetRangeStatus(uint16_t dev,
    uint8_t *rangeStatus);
```

This function returns the ranging status error where 0 = no error, 1 = sigma failure, 2 = signal failure, 4 = sensor out-of-bounds, and 7 = wraparound).

7.25 SetOffset

```
VL53L1X_ERROR VL53L1X_SetOffset(uint16_t dev,
    int16_t OffsetValue);
```

This function programs the offset correction in mm where OffsetValue = the offset correction value to program in mm.

7.26 GetOffset

```
VL53L1X_ERROR VL53L1X_GetOffset(uint16_t dev, int16_t *Offset);
```

This function returns the programmed offset correction value in mm.

7.27 SetXtalk

```
VL53L1X_ERROR VL53L1X_SetXtalk(uint16_t dev, uint16_t XtalkValue);
```

This function programs the crosstalk correction value in cps. This is the number of photons reflected back from the cover glass in cps.

7.28 GetXtalk

```
VL53L1X_ERROR VL53L1X_GetXtalk(uint16_t dev, uint16_t *Xtalk);
```

This function returns the current programmed crosstalk correction value in cps.

7.29 SetDistanceThreshold

```
VL53L1X_ERROR VL53L1X_SetDistanceThreshold(uint16_t dev,
    uint16_t ThreshLow,
    uint16_t ThreshHigh, uint8_t Window,
    uint8_t IntOnNoTarget);
```

This function programs the threshold detection mode. For example:

VL53L1X_SetDistanceThreshold(dev,100,300,0,1): below 100

VL53L1X_SetDistanceThreshold(dev,100,300,1,1): above 300

VL53L1X_SetDistanceThreshold(dev,100,300,2,1): out-of-window

VL53L1X_SetDistanceThreshold(dev,100,300,3,1): in window

Where:

- dev is the device address
- ThreshLow(in mm) is the threshold under which the device raises an interrupt if window = 0
- ThreshHigh(in mm) is the threshold above which the device raises an interrupt if window = 1
- IntOnNoTarget is the window detection mode where 0 = below, 1 = above, 2 = out, and 3 = in
- IntOnNoTarget = 1 (no longer used so just use 1)

7.30 GetDistanceThresholdWindow

```
VL53L1X_ERROR VL53L1X_GetDistanceThresholdWindow(uint16_t dev,
    uint16_t *window);
```

This function returns the window detection mode where 0 = below, 1 = above, 2 = out and 3 = in.

7.31 GetDistanceThresholdLow

```
VL53L1X_ERROR VL53L1X_GetDistanceThresholdLow(uint16_t dev,
    uint16_t *low);
```

This function returns the low threshold in mm.

7.32 GetDistanceThresholdHigh

```
VL53L1X_ERROR VL53L1X_GetDistanceThresholdHigh(uint16_t dev,
    uint16_t *high);
```

This function returns the high threshold in mm.

7.33 SetROI

```
VL53L1X_ERROR VL53L1X_SetROI(uint16_t dev, uint16_t X,
    uint16_t Y);
```

This function programs the ROI, the position of which is centered about the optical center. The smallest acceptable ROI size is 4. X is the ROI width and Y is the ROI height.

7.34 GetROI_XY

```
VL53L1X_ERROR VL53L1X_GetROI_XY(uint16_t dev, uint16_t *ROI_X,
    uint16_t *ROI_Y);
```

This function returns the ROI width (X) and height (Y).

7.35 SetROICenter

```
VL53L1X_ERROR VL53L1X_SetROICenter(uint16_t dev, uint8_t ROICenter);
```

This function programs the new user ROI center. Please note that there is no check in this function. If the ROI center vs the ROI size is outside the border, the ranging function returns an error #13.

7.36 GetROICenter

```
VL53L1X_ERROR VL53L1X_GetROICenter(uint16_t dev, uint8_t *ROICenter);
```

This function returns the current user ROI center.

7.37 SetSignalThreshold

```
VL53L1X_ERROR VL53L1X_SetSignalThreshold(uint16_t dev,
    uint16_t signal);
```

This function programs a new signal threshold in kcps where the default is 1024 kcps.

7.38 GetSignalThreshold

```
VL53L1X_ERROR VL53L1X_GetSignalThreshold(uint16_t dev,
    uint16_t *signal);
```

This function returns the current signal threshold in kcps.

7.39 SetSigmaThreshold

```
VL53L1X_ERROR VL53L1X_SetSigmaThreshold(uint16_t dev,
    uint16_t sigma);
```

This function programs a new sigma threshold in mm. The default value is 15 mm.

7.40 GetSigmaThreshold

```
VL53L1X_ERROR VL53L1X_GetSigmaThreshold(uint16_t dev,
    uint16_t *signal);
```

This function returns the current sigma threshold in mm.

7.41 StartTemperatureUpdate

```
VL53L1X_ERROR VL53L1X_StartTemperatureUpdate(uint16_t dev);
```

This function performs the temperature calibration.

If the sensor has been stopped for a long time, it is recommended to perform the temperature update prior to restarting the ranging.

By default, the sensor can adequately handle any temperature change as long as it is running, but if the sensor is stopped for an extended period of time, a temperature compensation is advised.

8 API structure

The API is composed of four files:

- VL53L1X_api.c (mandatory)
- VL53L1X_api.h (mandatory)
- VL53L1X_calibration.c (optional)
- VL53L1X_calibration.h (optional)

The VL53L1X_api.c and VL53L1X_api.h files contain all the required functions to initialize the sensor, to change the default settings, to perform ranging, and to get ranging data.

The VL53L1X_calibration.c and VL53L1X_calibration.h files contain two calibration functions and can be excluded from the software project if no device-to-device calibrations are required to optimize the code size.

Revision history

Table 3. Document revision history

Date	Version	Changes
06-Dec-2018	1	Initial release
09-May-2019	2	Updated Table 1. Main differences between VL53L1X API and VL53L1X ULD API. Section 7 Function descriptions: updated "dev" type from a structure to a simple "uint16_t" which is the device I2C address. Added Section 7.35 SetROICenter and Section 7.36 GetROICenter.

Contents