libvl53 -- a VL53L0x driver

Document Contents

1. REPOSITORY CONTENTS

This repository comprises the following:

- A redesign/rewrite of the published “API” for the VL53L0x ranging sensor
- An interactive command-line utility which, using libvl53, allows the user to setup all published device parameters and control ranging operations.

2. AFFILIATION AND LISCENSING

For the record, the author is not affiliated with ST Microelectronics (the Vendor) by employment, contract or otherwise. The author has no knowledge of the VL53L0x device or Vendor’s software beyond the contents of public domain works.

Whereas the libvl53 library is a derived work (including this document), the Vendor’s software license has application and is included within the software archive. See **`vl53/license-lib.txt`**.

The test utility is a new work and is provided under license. See **`testapp/license-app.txt`**.

3. REVISIONS

3.1 2020-05-21

First uploaded version.

3.2 2020-06-04

Upload this document as a pdf.

4. HISTORY

Initial work began by porting the Vendor's code to QNX Neutrino 6.5 on an X86 platform. This wasn't too difficult and the VL53L0x itself looked promising. However, navigating the API to get a deeper understanding of the device and its capabilities was challenging. I decided to begin simplifying the existing code and one thing led to another….

A driver with a smaller footprint was needed for the ultimate intended target – an ARM Cortex-M0. The driver offered here meets the constraint while retaining all functionality.

5. LIBRARY

5.1 Platform-Specific I²C Module

To use the driver, the application needs to supply the I²C interface functions declared in the file **`vl53/vl53_core.h`**.

The I²C functions are similar to those needed for the original Vendor's code.

There are sample files that have been used on different platforms:

- **`vl_app/i2c_372.c`** (SMB controller of Intel 82371 Southbridge)
- **`vl_app/i2c_686.c`** (SMB controller of VT82C686B South Bridge)
- **`vl_app/i2c_82x.c`** (interface to generic read/write functions, used on LPC82x CPU)

Achieving I²C communication with the device can be the hardest part of bringing up the software. It's easiest to start with the byte read function, and use the test utility in register access mode to confirm the device ident can be read (i.e., register 0xc0=0xee). Then supply the byte write function, and again use the test utility to confirm registers can be written and read back. The word and block access functions can be built on top of the byte functions (two of the sample files do this), or more efficient routines can be written.

5.2 Compile-Time Options

A few symbols exist to control the library footprint. They are defined in file **`vl53/vl53_plat.h`**.

The **`AllStrings`** symbol enables the capability to retrieve text names for errors, limits, sequence steps, etc. If this symbol is not defined, retrieval functions will return an empty string.

The **`AllFunctions`** symbol enables code which is unused but is included for reference.

The **`SigmaEst`** symbol enables calculation of sigma estimate and dmax. This is used for a software-derived limit check. If this symbol is undefined, the sigma limit check can still be enabled, but will have no effect.

The **`SigmaFloat`** symbol enables sigma and dmax calculation using floating point variables. If this symbol is not defined, fixed point integer calculations are used. The floating point and fixed point calculations yield the same results, aside from minor precision and rounding discrepancies.

5.3 Error Handling

This library uses an error reporting design very different from the vendor's original code.

Instead of every function returning an error status, an element of the driver structure records an error. The error element is set only by the driver, and cleared only by the application.

The approach within the library is simple:

- An error is set only when no other error has been recorded (the first error is latched).
- A device transaction is not attempted while an error is set.

The actions of the application are as follows:

- Several driver functions may be called without checking for errors.
- When an error is sensed, the application assumes the device and driver context are in an unknown condition.
- The application recovers from an error by clearing the error and reinitializing the device and driver.

5.4 Sigma Estimate

This section shows the sigma estimate calculation distilled from the rewrite. Type conversions and limit checks are removed for clarity.

Constants:

EffWidthPulse_cNs = 800	100ths of ns
EffWidthAmb_cNs = 600	100ths of ns
WidthVcsel_ps = 4700	pico sec (2.84 PLL clks)
DfltFinalRangeTimeUs = 25000.0	Default 25 ms integration time
Vol = 2.997	C, x 1e8 m/s
Tof_psPerMm = 6.6	6.6 psec/mm round trip

Calculate total xTalk CompRate for all SPADs:

```
sigRateXtalkComp_mcps =
        pRangeData->effSpadRtnCnt * VL_Cal( XTalkCompRateMcps )
```

Retrieve signal rates:

```
sigRateRtn_mcps = pRangeData->sigRateRtnMcps
sigRateTotal_mcps = sigRateRtn_mcps + sigRateXtalkComp_mcps
ambToSigRatio = pRangeData->sigRateAmbRtnMcps / sigRateTotal_mcps
```

Retrieve step timeouts:

```
preRangeUs = VL_Timing( preRangeTimeoutUs )
finalRangeUs = VL_Timing( finalRangeTimeoutUs )
```

Final range integration time:

```
finalRangeInteg_us = finalRangeUs + preRangeUs
```

Calculate pre-range and final range macro periods:

```
preRangeMclks =
        vl_calc_timeout_mclks(preRangeUs, VL_Timing( preRangePclks ) );
finalRangeMclks =
        vl_calc_timeout_mclks(finalRangeUs, VL_Timing( finalRangePclks ) );
```

Some intermediate calculations:

```
vcselWidth = ( VL_Timing( finalRangePclks ) == 8 ) ? 2 : 3
```

$$\text{peakVcselDuration\_us} = \frac{\text{vcselWidth} * 2048 * (\text{preRangeMclks} + \text{finalRangeMclks}) * \text{PllPeriod\_ps}}{1000000}$$

$$\text{vcselTotalEventsRtn} = \text{sigRateTotal\_mcps} * \text{peakVcselDuration\_us}$$

$$\text{tof\_ps} = \text{pRangeData}{->} \mathit{RangeMm} * \mathit{Tof\_psPerMm}$$

Calculate the Xtalk correction:

$$\text{xTalkCorrection} = \frac{\text{sigRateRtn\_mcps} - \text{sigRateXtalkComp\_mcps}}{\text{sigRateRtn\_mcps} + \text{sigRateXtalkComp\_mcps}}$$

Calculate pulse width factor:

$$\text{pwMult} = (1 + \frac{\text{tof\_ps}}{\text{WidthVcsel\_ps}} * (1 - \text{xTalkCorrection}))^2$$

Calculate the Sigma Estimate:

$$\text{sqr1} = (\text{pwMult} * \text{EffWidthPulse\_cNs})^2$$

$$\text{sqr2} = (\text{ambToSigRatio} * \text{EffWidthAmb\_cNs})^2$$

$$\text{sigmaEstRtn} = \frac{\sqrt{\text{sqr1} + \text{sqr2}}}{2.0 * \sqrt{\text{vcselTotalEventsRtn} * 12.0}} * \text{Vol}$$

$$\text{sigmaEstRef} = 1\text{mm} * \sqrt{\frac{\text{DfltFinalRangeTimeUs}}{\text{finalRangeInteg\_us}}}$$

(pre-range is included in final range)

$$\text{sigmaEst} = \sqrt{\text{sigmaEstRtn}^2 + \text{sigmaEstRef}^2}$$

The original Vendor's code contained vestiges hinting at some of the timing constants to be configurable. This has not been implemented. The algorithm cannot be assessed for correctness by the author, but it may serve a starting point for someone wishing to investigate further.

5.5 DMax Estimate

This section shows the sigma estimate calculation distilled from the rewrite. Type conversions and limit checks are removed for clarity. Some of the constants and variables from the previous sigma estimate calculations are referenced here.

Constants:

$$\text{SigmaLimit} = 18.0 \quad \text{mm}$$

$$\text{SigmaEstRef} = 1.0 \quad \text{mm}$$

Intermediate calculations:

$$\text{sigAt0mm} = \text{VL\_Cal}(\text{DmaxCalSigRateMcps}) * \text{VL\_Cal}(\text{DmaxCalRangeMm})^2$$

$$\text{p1} = (\text{pwMult} * \text{EffWidthPulse\_cNs})^2$$

$$\text{p2} = (\text{ambToSigRatio} * \text{EffWidthAmbDmax\_cNs})^2$$

dmax ambient width is different from sigma

$$\text{p3} = \left(\frac{\text{sigRateTotal\_mcps}}{\text{sigRateRtn\_mcps}} * \text{Vol} \right)^2$$

$$\text{p4} = 4 * 12 * (\text{SigmaLimit}^2 - \text{SigmaEstRef}^2)$$

Calculate dmax:

$$\text{minSigNeeded} = \frac{\text{p1} + \text{p2}}{\text{peakVcselDuration\_us}} * \frac{\text{p3}}{\text{p4}}$$

$$\text{dmaxDark} = \sqrt{\frac{\text{sigAt0mm}}{\text{SigRateLimit}}}$$

$$\text{dmaxAmb} = \sqrt{\frac{\text{sigAt0mm}}{\text{minSigNeeded}}}$$

The lesser of dmaxDark and dmaxAmb is returned.

6. TEST UTILITY

6.1 Command Synopsis

6.1.1 `init`

Initializes the driver and device. The application takes no action when started, so this command must be given before attempting to use the device.

6.1.2 `reset`

Restore the device to power-on condition.

6.1.3 `rev`

Show product and revision info available from the device.

6.1.4 `power`

Command variants:

- **`power`** - show current power mode.
- **`power [0|1]`** - set power mode to Standby|Idle.

6.1.5 `gpio`

Command variants:

`gpio` - show the current GPIO int function.

`gpio int [0-4]` - assign GPIO as int function.

`gpio drive [0|1]` - assign GPIO as static drive, low|high.

`gpio osc` - assign GPIO as oscillator output.

6.1.6 `meas`

Command variants:

`meas` - show current measurement parameters.

`meas set` - measurement parameter setup dialog. The value of each parameter is displayed followed by a prompt. use <Enter> to keep the same value, or enter a new value. For the limit checks, use 'y' or 'n' to enable/disable the individual checks.

6.1.7 `timing`

Command variants:

`timing` - show current timing parameters.

`timing set` - timing parameter setup dialog. The value of each parameter is displayed followed by a prompt. use <Enter> to keep the same value, or enter a new value. For the sequence steps, use 'y' or 'n' to enable/disable the individual steps.

6.1.8 `cal`

Command variants:

cal - show current calibration parameters.

cal set - calibration parameter setup dialog. . The value of each parameter is displayed followed by a prompt. use <Enter> to keep the same value, or enter a new value.

cal ref - perform reference calibration.

cal offset[=dist] - perform offset calibration. If the distance is not specified, 10cm is assumed.

cal xtalk[=dist] - perform xtalk calibration. If the distance is not specified, 40cm is assumed.

cal spads - regenerate reference SPAD map.

6.1.9 mode

Command variants:

mode def - set device parameters for default ranging mode.

mode acc - set device parameters for more accurate ranging.

mode far - set device parameters for greater distance ranging

mode fast - set device parameters for fast ranging.

6.1.10 range

Command variants:

range - perform one ranging measurement and show results.

range cont[=num] - perform continuous ranging measurements. If the count is not specified, 5 measurements are done.

range timed[=num] – timed ranging measurements. If the count is not specified, 5 measurements are done. One of the measurement parameters controls the inter-measurement interval.

6.1.11 reg

reg - enter register (and i/o) read/write mode. Once in this mode, either VL53 register space or ISA I/O space can be selected. A short help line is emitted when entering this mode.

6.1.12 debug

Command variants:

debug - show the current debug flags

debug [grants(hex)] - set debug grant flags

6.1.13 q

Exit the test utility.