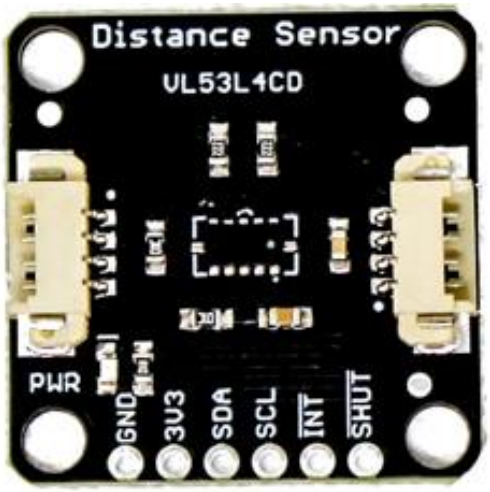
SmartElex Distance Sensor - 1.3 Meter, VL53L4CD

The VL53L4CD is Time Of Flight (ToF) sensors. Both uses a VCSEL (vertical cavity surface emitting laser) to emit a class 1 IR laser (940 nm) and time the reflection to the target. (You can't see the laser but cell phones can) What does all this mean? You can measure the distance to an object up to 4 meters away with millimeter resolution using the VL53L1X and up to 1.3 meters away with 1 millimeter resolution using the VL53L4CD!



We've found the precision of the VL53L1D sensor to be 1mm but the accuracy is around ±7mm. The minimum read distance of this sensor is 1cm (or 10mm).

Hardware Overview

First, let's check out some of the characteristics of the VL53L1X and VL53L4CD we're dealing with, so we know what to expect out of the board. Below is a comparison table for both sensors taken from the datasheet. Typically, the board is powered at **3.3V** via the connector.

Characteristic	VL53L1X	VL53L4CD
Operating Voltage	**2.6V to 3.5V**	
Power Consumption	20 mW @10Hz	-
Current Consumption	18mA	24mA
Measurement Range	~40mm to 4,000mm	1mm to 1300mm
Resolution	±1mm	
Light Source	Class 1 940nm VCSEL	
I²C Address	0x29	
Field of View	15° to 27°	18°
Max Read Rate	50Hz	100Hz

Pins

The following table lists all of the VL53L1X and VL53L4CD's pins and their functionality.

Pin	Description	Direction
GND	Ground	In
3.3V	Power	In
SDA	Data	In

SCL	Clock	In
INT	Interrupt, goes low when data is ready.	Out
SHUT	Shutdown, can be pulled low to put the IC in shutdown mode.	In

I²C

The I²C address for each sensor is **0x29** (7-bit unshifted) as stated earlier. You may notice that the datasheet and library use *0x52*, which is the address shifted.

LED

The onboard power LED (PWR) will light up when the board is powered. Exclusively for the VL53L4CD, this can be disabled by cutting the jumper labeled as **LED** on the back of the board.

Sensor and IR Laser

On the left side of each sensor IC is a single photon avalanche diode (SPAD) array. On the other side of each sensor IC is an invisible IR laser. The wavelength of the lasers found in the VL53L1X and VL53L4CD is 940nm and are classified as a Class 1 laser emitter. We found that the sensors worked best when left uncovered in your application to avoid crosstalk. If you do place a transparent material (material transmission should be greater than 85%) in front of the sensor, it is recommended to have an air gap that is as small as possible to avoid errors in sensor readings.

Note: While the IR laser is invisible to the human eye, you can view the laser at an angle using a camera. If you take out your smartphone and view the sensor through the camera, you can see the IR laser being emitted from the sensor!

Jumpers

The VL53L1X and VL53L4CD breakout boards include jumpers on the back of the board. If you need to disconnect any of the jumpers, they can be removed by cutting the traces on the corresponding jumpers highlighted below.

- **I2C** - By default, this 3-way jumper is closed by default. The 2.2kΩ pull-up resistors are attached to the I²C bus; if multiple sensors are connected to the bus

with the pull-up resistors enabled, the parallel equivalent resistance will create too strong of a pull-up for the bus to operate correctly. As a general rule of thumb, disable all but one pair of pull-up resistors if multiple devices are connected to the bus.
- **INT** - By default, this jumper is closed by default. This is connected to the 10kΩ pull-up resistor.
- **LED** - Exclusive to the VL53L4CD, this jumper is closed by default. Cutting this jumper will disable the PWR LED.

Arduino Library

First, you'll need the **SparkFun VL53L1X** Arduino library, which is an easy to use wrapper of ST's driver. This library was originally written for VL53L1X but it can also be used for the VL53L4CD. You can obtain these libraries through the Arduino Library Manager. Search for **Sparkfun VL53L1X Arduino Library** to install the latest version.

Before we get started developing a sketch, let's look at the available functions of the library.

- `boolean init();` --- Initialize the sensor
- `void startRanging();` --- Starts taking measurements.
- `void stopRanging();` --- Stops taking measurements.
- `bool checkForDataReady();` --- Checks if a measurement is ready.
- `void setTimingBudgetInMs(uint16_t timingBudget)` --- Set the timing budget for a measurement in ms. The timing budget is the amount of time over which a measurement is taken. This can be set to any of the following.
  - 15
  - 20
  - 33
  - 50
  - 100 (default)
  - 200
  - 500
- `uint16_t getTimingBudgetInMs();` --- Get's the current timing budget in ms.
- `void setDistanceModeLong();` --- Sets to 4M range.
- `void setDistanceModeShort();` --- Sets to 1.3M range
- `uint8_t getDistanceMode();` --- Returns 1 for short range, 2 for long range.
- `void setIntermeasurementPeriod(uint16_t intermeasurement);` --- Set's the period in between measurements. Must be greater than or equal to the timing budget. Default is 100 ms.
- `uint16_t getIntermeasurementPeriod();` --- Returns the intermeasurement period in ms.
- `bool checkBootState();` --- Checks whether the device has been booted. Returns true if the device has been booted.
- `uint16_t getSensorID();` --- Get the sensor ID, should be 0xEEAC.
- `uint16_t getDistance();` --- Returns the results from the last measurement, distance in mm

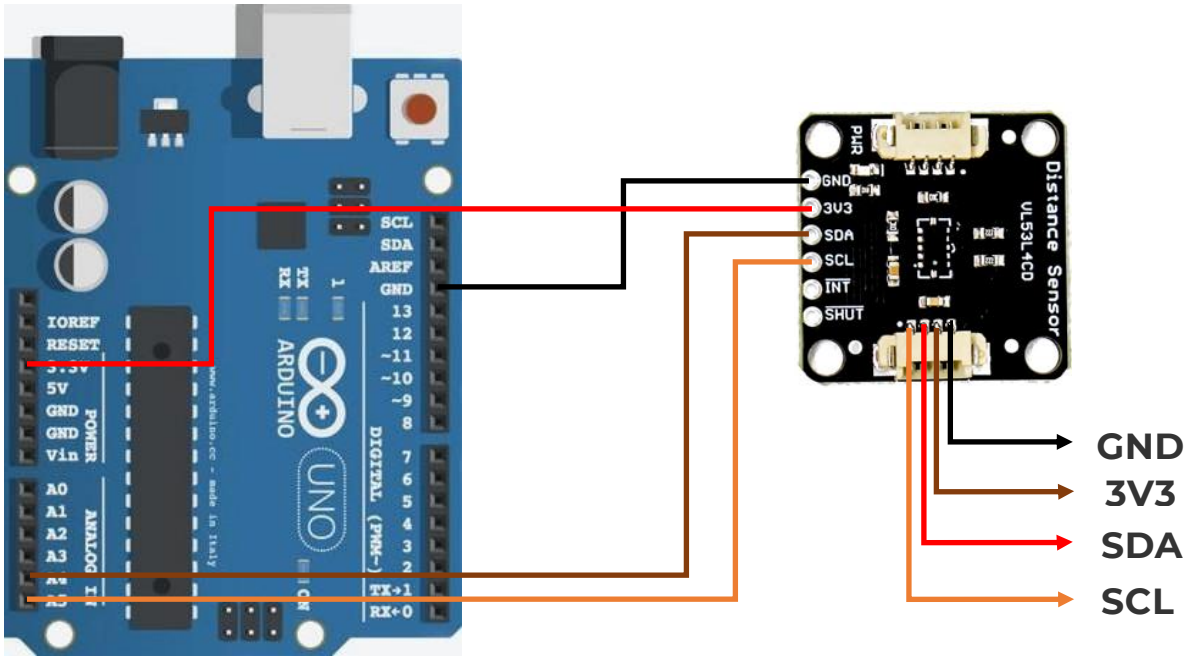
- **uint16_t getSignalPerSpad();** --- Returns the average signal rate per SPAD (The sensitive pads that detect light, the VL53L1X has a 16x16 array of these) in kcps/SPAD, or kilo counts per second per SPAD.
- **uint16_t getAmbientPerSpad();** --- Returns the ambient noise when not measuring a signal in kcps/SPAD.
- **uint16_t getSignalRate();** --- Returns the signal rate in kcps. All SPADs combined.
- **uint16_t getSpadNb();** --- Returns the current number of enabled SPADs
- **uint16_t getAmbientRate();** --- Returns the total ambinet rate in kcps. All SPADs combined.
- **uint8_t getRangeStatus();** --- Returns the range status, which can be any of the following.
    - **0**: No error
    - **1**: Signal fail
    - **2**: Sigma fail
    - **7**: Wrapped target fail
- **void setOffset(int16_t offset);** --- Manually set an offset for a measurement in mm.
- **int16_t getOffset();** --- Get the current offset in mm.
- **void setXTalk(uint16_t xTalk);** --- Manually set the value of crosstalk in counts per second (cps), which is interference from any sort of window in front of your sensor.
- **uint16_t getXTalk();** --- Returns the current crosstalk value in cps.
- **void setDistanceThreshold(uint16_t lowThresh, uint16_t hiThresh, uint8_t window);** --- Set bounds for the interrupt. lowThresh and hiThresh are the bounds of your interrupt while window decides when the interrupt should fire. The options for `window` are shown below.
    - **0**: Interrupt triggered on measured distance below lowThresh.
    - **1**: Interrupt triggered on measured distance above hiThresh.
    - **2**: Interrupt triggered on measured distance outside of bounds.
    - **3**: Interrupt triggered on measured distance inside of bounds.
- **uint16_t getDistanceThresholdWindow();** --- Returns distance threshold window option.
- **uint16_t getDistanceThresholdLow();** --- Returns lower bound in mm.
- **uint16_t getDistanceThresholdHigh();** --- Returns upper bound in mm
- **void setROI(uint16_t x, uint16_t y, uint8_t opticalCenter);** --- Set the height and width of the ROI in SPADs, lowest possible option is 4. The center of the ROI you set is based on the table below. Set the `opticalCenter` as the pad above and to the right of your exact center.

128	136	144	152	160	168	176	184	192	200	208	216	224	232	240	248
129	137	145	153	161	169	177	185	193	201	209	217	225	233	241	249
130	138	146	154	162	170	178	186	194	202	210	218	226	234	242	250
131	139	147	155	163	171	179	187	195	203	211	219	227	235	243	251
132	140	148	156	164	172	180	188	196	204	212	220	228	236	244	252
133	141	149	157	165	173	181	189	197	205	213	221	229	237	245	253

134	142	150	158	166	174	182	190	198	206	214	222	230	238	246	254
135	143	151	159	167	175	183	191	199	207	215	223	231	239	247	255
127	119	111	103	95	87	79	71	63	55	47	39	31	23	15	7
126	118	110	102	94	86	78	70	62	54	46	38	30	22	14	6
125	117	109	101	93	85	77	69	61	53	45	37	29	21	13	5
124	116	108	100	92	84	76	68	60	52	44	36	28	20	12	4
123	115	107	99	91	83	75	67	59	51	43	35	27	19	11	3
122	114	106	98	90	82	74	66	58	50	42	34	26	18	10	2
121	113	105	97	89	81	73	65	57	49	41	33	25	17	9	1
120	112	104	96	88	80	72	64	56	48	40	32	24	16	8	0

- **`uint16_t getROIX();`** --- Returns the width of the ROI in SPADs
- **`uint16_t getROIY();`** --- Returns the height of the ROI in SPADs
- **`void setSignalThreshold(uint16_t signalThreshold);`** --- Programs the necessary threshold to trigger a measurement. Default is 1024 kcps.
- **`uint16_t getSignalThreshold();`** --- Returns the signal threshold in kcps
- **`void setSigmaThreshold(uint16_t sigmaThreshold);`** --- Programs a new sigma threshold in mm. (default=15 mm)
- **`uint16_t getSigmaThreshold();`** --- Returns the current sigma threshold.
- **`void startTemperatureUpdate();`** --- Recalibrates the sensor for temperature changes. Run this any time the temperature has changed by more than 8°C
- **`void calibrateOffset(uint16_t targetDistanceInMm);`** --- Autocalibrate the offset by placing a target a known distance away from the sensor and passing this known distance into the function.
- **`void calibrateXTalk(uint16_t targetDistanceInMm);`** --- Autocalibrate the crosstalk by placing a target a known distance away from the sensor and passing this known distance into the function.

WIRING



Arduino	VL53L4CD
A5(SCL)	SCL
A4(SDA)	SDA
3.3V	3V3
GND	GND

Arduino Example Code

Now that we have our library installed and we understand the basic functions, let's run some examples for our distance sensor to see how it behaves.

Example 1 – Distance Array

To get started with the first example, open up **File** > **Examples** > **SparkFun VL53L1x 4M Laser Distance Sensor** > **Example1_DistanceArray**. In this example, we begin by creating a **SFEVL53L1X** object called distanceSensor with our wire port, Wire, and then our shutdown and interrupt pins. Then we initialize our sensor object in the setup() loop. The code to do this is shown below and is repeated in some form in all of the examples.Once we've initialized our sensor, we can start grabbing measurements from it. To do this, we send some configuration bytes to our sensor using distanceSensor.startRanging() to initiate the measurement. We then wait for data to become available and when it does, we read it in, convert it from millimeters to feet, and print it out over serial. The void loop() function that does this is shown below.

```
#include <Wire.h>
#include "SparkFun_VL53L1X.h" //Click here to get the library:
http://librarymanager/All#SparkFun_VL53L1X

//Optional interrupt and shutdown pins.
#define SHUTDOWN_PIN 2
#define INTERRUPT_PIN 3

SFEVL53L1X distanceSensor;
//Uncomment the following line to use the optional shutdown and interrupt pins.
//SFEVL53L1X distanceSensor(Wire, SHUTDOWN_PIN, INTERRUPT_PIN);

void setup(void)
{
  Wire.begin();

  Serial.begin(115200);
  Serial.println("VL53L1X Qwiic Test");

  if (distanceSensor.begin() != 0) //Begin returns 0 on a good init
  {
    Serial.println("Sensor failed to begin. Please check wiring. Freezing...");
    while (1)
      ;
  }
  Serial.println("Sensor online!");
}

void loop(void)
{
  distanceSensor.startRanging(); //Write configuration bytes to initiate measurement
  while (!distanceSensor.checkForDataReady())
```

```
  {
    delay(1);
  }
  int distance = distanceSensor.getDistance(); //Get the result of the measurement
from the sensor
  distanceSensor.clearInterrupt();
  distanceSensor.stopRanging();

  Serial.print("Distance(mm): ");
  Serial.print(distance);

  float distanceInches = distance * 0.0393701;
  float distanceFeet = distanceInches / 12.0;

  Serial.print("\tDistance(ft): ");
  Serial.print(distanceFeet, 2);

  Serial.println();
}
//////////////////////////////////////////END//////////////////////////////////////////
```

Opening your serial monitor to a baud rate of **9600** should show the distance between the sensor and the object it's pointed at in both millimeters and feet. The output should look something like the below image.