VL53L0X
Time of Flight Distance Sensor on BeagleBone Green

CMPT433

Fall 2019

Written by:

Irene Abrea

Heidi Tong

Joanna Niemczyk

Table of Contents

1. Introduction

The VL53L0X is a “time of flight” sensor that measures distance by detecting how long light has taken to bounce back to the sensor. It contains a small invisible laser source and a matching sensor. The VL53L0X is good for determining distance of only the surface directly in front of it and can handle about 50mm to 1200mm of range distance. This guide will cover how to use the VL53L0X on the BeagleBone Green using I2C.

2. Required Parts

The following parts are required to connect the VL53L0X distance sensor to the BeagleBone Green:

- Breadboard
- VL53L0X Time of Flight Distance Sensor
- 4 male-to-female jumper wires

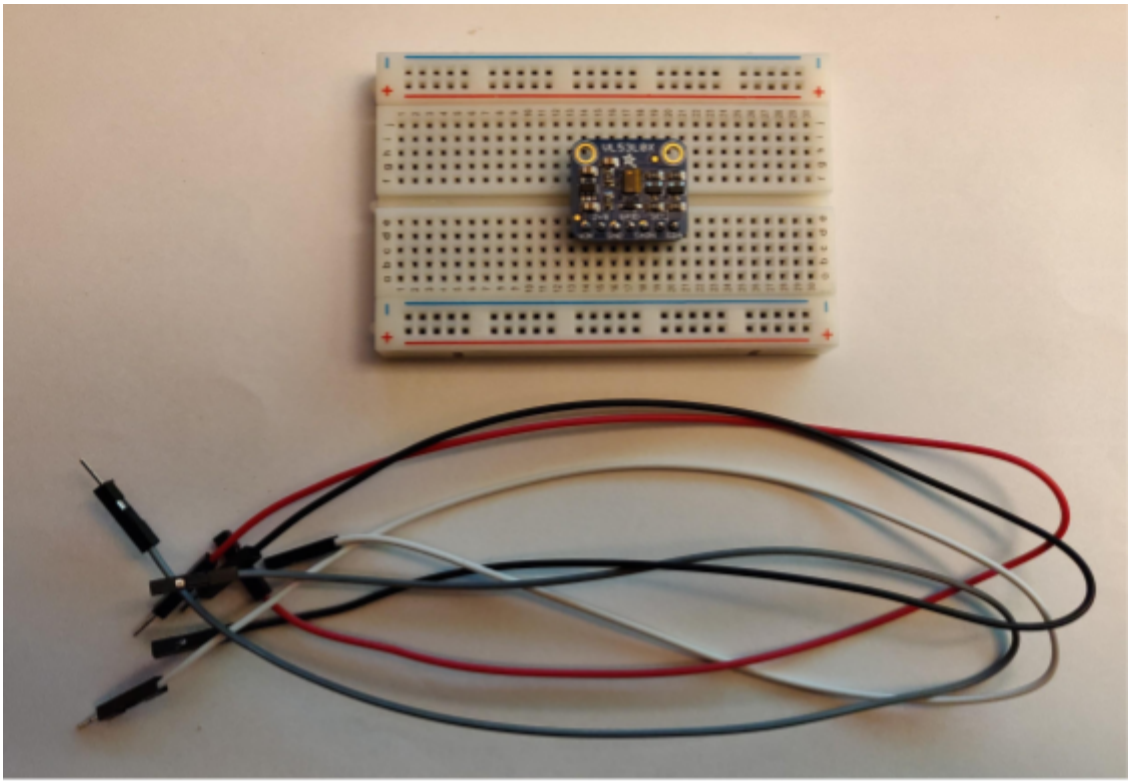

***Figure 1**: From top to bottom - breadboard, VL53L0X Distance Sensor, male-to-female jumper wires (x4)*

3. Connections and Setup

The BeagleBone Green supports three I2C buses numbered 0 to 2. In this guide, we will be using using I2C2. To setup the distance sensor, place it on the breadboard and make the following connections. Note that all the connections are made to the pins on the P9 header on the BeagleBone Green.

- Connect **VIN** to P9_3 (3.3V)
- Connect **GND** to P9_2 (GND)
- Connect the **SCL** pin to P9_19 (I2C2_SCL)
- Connect the **SDA** pin to P9_20 (I2C2_SDA)

The default I2C address for the distance sensor is **0x29**.

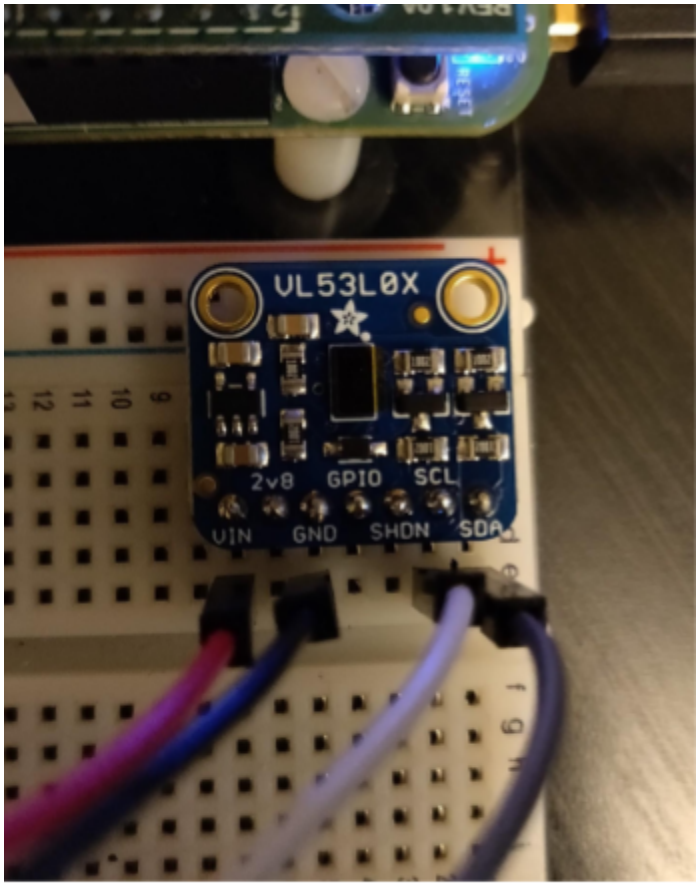

Figure 2: *Connections on distance sensor. From left to right - VIN (red), GND (black), SCL (white), SDA (gray)*

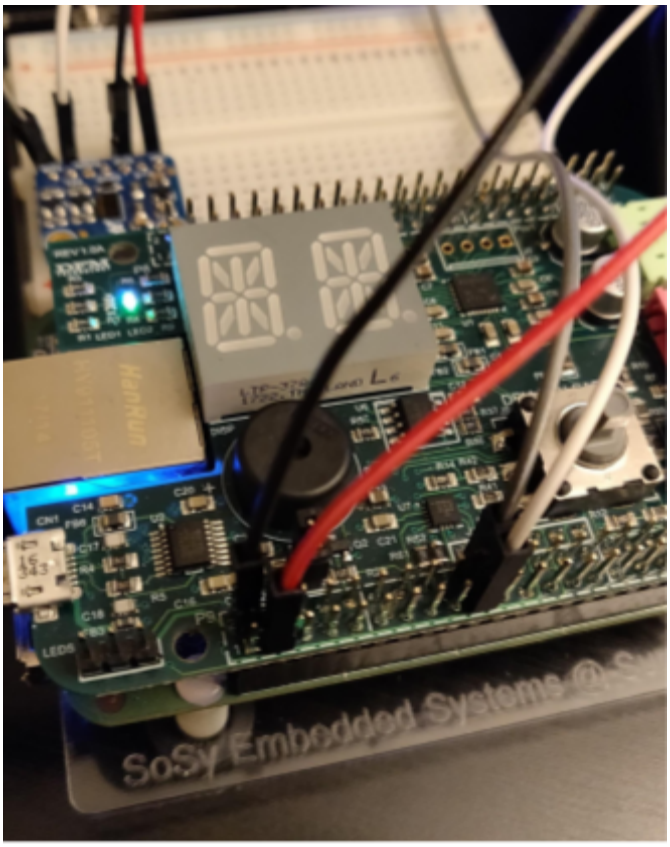

Figure 3: *Connections on BBG P9 header.*

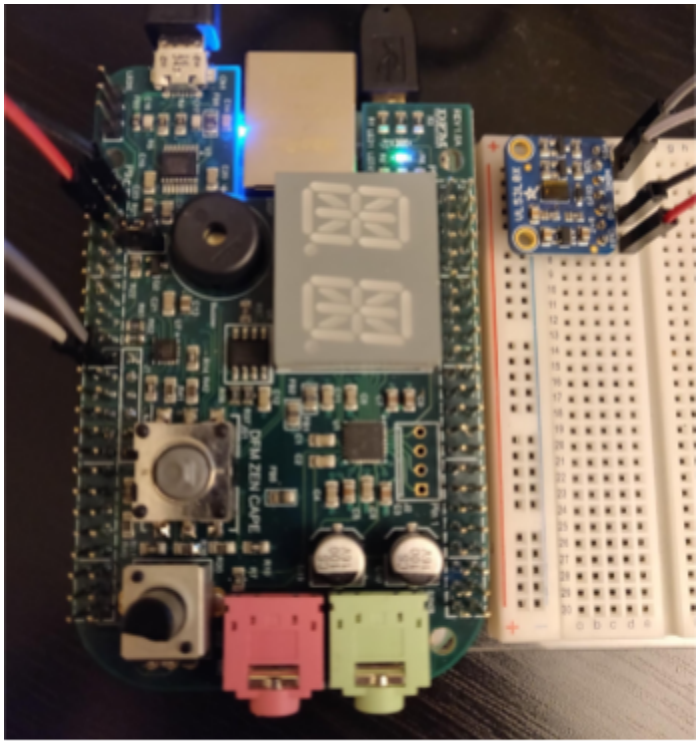

Figure 4: *Overview of full setup.*

4. Checking Distance Reading via Command Line

This guide assumes that you have previously followed the steps from Brian Fraser's I2C Guide, section 2.1 to enable the I2C buses.

1. Ensure that the I2C2 bus is enabled:

```
# i2cdetect -l
i2c-1    i2c             OMAP I2C adapter                I2C adapter
i2c-2    i2c             OMAP I2C adapter                I2C adapter
i2c-0    i2c             OMAP I2C adapter                I2C adapter
```

2. Display I2C devices on the I2C2 bus:
   `# i2cdetect -y -r 2`

   - Where `2` indicates the second I2C bus, Linux device `/dev/i2c-2`
   - If the distance sensor is properly wired and enabled, it should be detected on register **0x29**.
   - Sample output:

```
debian@beaglebone:/mnt/remote/myApps$ i2cdetect -y -r 2
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- UU -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- 29 -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- UU UU UU UU -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

"--" means no device found.
"##" means device at address ## detected (hex).
"UU" means in use by a driver (cape manager, HDMI, or another kernel device driver).

3. To display the internal memory of the distance sensor, run the command:
   `# i2cdump -y 2 0x29`

```
debian@beaglebone:/mnt/remote/myApps$ i2cdump -y 2 0x29
No size specified (using byte-data access)
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f    0123456789abcdef
00: 02 ff 01 00 00 00 00 00 00 00 00 00 00 00 00 00    ?.?.............
10: 00 00 01 40 40 06 bc 00 01 77 00 4c 00 0f 00 14    ..?@@??.?w.L.?.?
20: 00 00 00 00 00 00 00 00 00 e2 00 00 00 00 00 00    .........?......
30: 1a 04 03 03 03 44 00 00 00 00 00 00 00 00 00 00    ?????D..........
40: 82 80 07 17 00 00 2a 00 00 00 00 00 00 00 00 00    ????..*.........
50: 07 00 99 00 00 00 08 30 00 00 00 00 00 00 00 00    ?.?...?0........
60: 00 00 00 00 00 00 40 00 00 00 00 00 00 00 00 00    ......@.........
70: 05 02 85 00 00 00 00 00 21 00 0a 00 00 00 00 00    ???.....!.?.....
80: 00 00 00 00 01 00 01 00 24 00 29 00 00 00 00 00    ....?.?.$.).....
90: ff df f7 ff ff ff b7 ff fe ff 7f bf 97 ef bf 9f    .??...?.?.??????
a0: df 6f f7 ff ff ff ff ff f7 be df bf df ff ff ff    ?o?.....?????...
b0: ff ff bf ff df 0b b4 00 00 00 00 00 00 00 00 01    ..?.???........?
c0: ee aa 10 00 01 05 00 00 00 01 18 1f 01 00 01 00    ???.??...????.?.
d0: 00 0f 00 00 00 58 bc 00 02 ff ff ff ff ff 1f ff    .?...X?.?.....?.
e0: 00 00 00 00 00 01 00 00 00 00 00 00 00 26 01 26    .....?.......&?&
f0: 00 00 00 00 00 00 00 00 00 26 00 00 00 00 00 00    .........&......
```

   - The distance sensor reading is stored within 2 bytes of data with the Most Significant Bit (MSB) stored in register **0x1e** and the Least Significant Bit stored in **0x1f** (LSB).
   - When the sensor doesn't detect anything, the MSB and LSB registers can read 0x00 0x14 or 0x1f 0xfe by default. To filter out these values, we will accept values within a given range while reading the registers.
   - The sensor value can be converted to distance in mm by doing the following conversion, where MSB and LSB corresponds to the value from the respective registers:
     `uint16_t x = (MSB << 8) | (LSB);`

4. Troubleshooting
   - When checking if the I2C bus is enabled, if you get the output:

     `# i2cdetect -l`
     `-bash:i2cdetect:command not found`

     This means the I2C tools are not installed on the target. Follow Section 2.1 from Brian Fraser's I2C Guide.

5. Checking Distance Reading via C Code

5.1 Initialization

The following function initializes the distance sensor on I2C2. The function `DistanceSensor_configPins()` sets the pin configuration on P9 pins 19 and 20 to I2C. Note that this may not be necessary as these pins may be set to I2C by default.

```
/*  I2C device that the distance sensor is wired to */
#define I2C_LINUX_BUS2                              "/dev/i2c-2"
/*  Device address of the distance sensor */
#define DISTANCE_SENSOR_DEVICE_ADDRESS                0x29

static void DistanceSensor_configPins() {
   // Execute the shell command (output into pipe)
   FILE *pipe = popen("config-pin P9_19 i2c", "r");
   // Close pipe, check program's exit code
   int exitCode = WEXITSTATUS(pclose(pipe));
   if (exitCode != 0) {
       printf("Program failed: %d\n", exitCode);
   }

   pipe = popen("config-pin P9_20 i2c", "r");
   exitCode = WEXITSTATUS(pclose(pipe));
   if (exitCode != 0) {
       printf("Program failed: %d\n", exitCode);
   }
}

static int DistanceSensor_init() {
   DistanceSensor_configPins();
   int distanceSensorFD = open(I2C_LINUX_BUS2 , O_RDWR);
   int result = ioctl(distanceSensorFD, I2C_SLAVE,
DISTANCE_SENSOR_DEVICE_ADDRESS);
   if (result < 0) {
       perror("I2C: Unable to set I2C device to slave address.");
       exit(1);
   }

   return distanceSensorFD;
}
```

5.2 Configuring Distance Sensor to Read Continuously

By default the distance sensor is configured to take readings one at a time. Before reading the sensor values, we need to change the distance sensor's reading mode to enable continuous reading. The current reading mode is stored in register **0x00**. To allow continuous reading, we need to write **0x02** to this register. The list of modes and their corresponding hex values can be found in the [Adafruit VL53L0X library source code].

```
#define VL53L0X_REG_SYSRANGE_START              0x000
#define VL53L0X_REG_SYSRANGE_MODE_BACKTOBACK    0x02

static void DistanceSensor_setContinuous(int distanceSensorFD) {
   unsigned char buff[2];
   buff[0] = VL53L0X_REG_SYSRANGE_START;
   buff[1] = VL53L0X_REG_SYSRANGE_MODE_BACKTOBACK;
   int res = write(distanceSensorFD, buff, 2);
   if (res != 2) {
       perror("I2C: Unable to write i2c register.");
       exit(1);
   }
}
```

5.3 Reading Distance Sensor Output

The following function gets the sensor reading's MSB and LSB by reading from registers 0x1e and 0x1f, converts the value to distance, and returns it.

```
/*  Address of the distance sensor reading's MSB*/
#define DISTANCE_SENSOR_READING_MSB                         0x1e
/*  Total number of bytes to read,
   in our case we are reading two bytes,
   the MSB and LSB of the distance reading*/
#define NUM_BYTES_READ                                      2


static uint16_t DistanceSensor_readReg(int distanceSensorFD) {
   char values[NUM_BYTES_READ];

   unsigned char startRegAddr = DISTANCE_SENSOR_READING_MSB;
   for (int i = 0; i < NUM_BYTES_READ; i++) {
```

```
        unsigned char currAddr = startRegAddr + i;
        int res = write(distanceSensorFD, &(currAddr), sizeof(currAddr));
        if (res != sizeof(currAddr)) {
            perror("I2C: Unable to write to i2c register.");
            exit(1);
        }

        res = read(distanceSensorFD, &values[i], sizeof(*values));
        if (res != sizeof(*values)) {
            perror("I2C: Unable to read from i2c register");
            exit(1);
        }
    }

    // Convert the reading to distance
    uint16_t dist = (((uint16_t)(values[0]) << 8) | (uint16_t)(values[1]));

    return dist;
}
```

5.4 Main Program

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <linux/i2c.h>
#include <linux/i2c-dev.h>
#include <stdbool.h>
#include <stdint.h>


// Add the other functions above here...


int main() {
    // Initialize the I2C device
    int distanceSensorFD = DistanceSensor_init();

    // Set device to continuous ranging mode
    DistanceSensor_setContinuous(distanceSensorFD);
```

```
    while(true) {
        uint16_t dist = DistanceSensor_readReg(distanceSensorFD);

        if (dist > 20 && dist < 8190) {
            printf("Reading: %u mm\n", dist);
        } else {
            printf("Reading: Nothing detected\n");
        }

        // Wait 0.25 sec between each reading
        struct timespec reqDelay = {0, 250000000};
        nanosleep(&reqDelay, (struct timespec *) NULL);
    }

    return 0;
}
```