AN4478
Application note

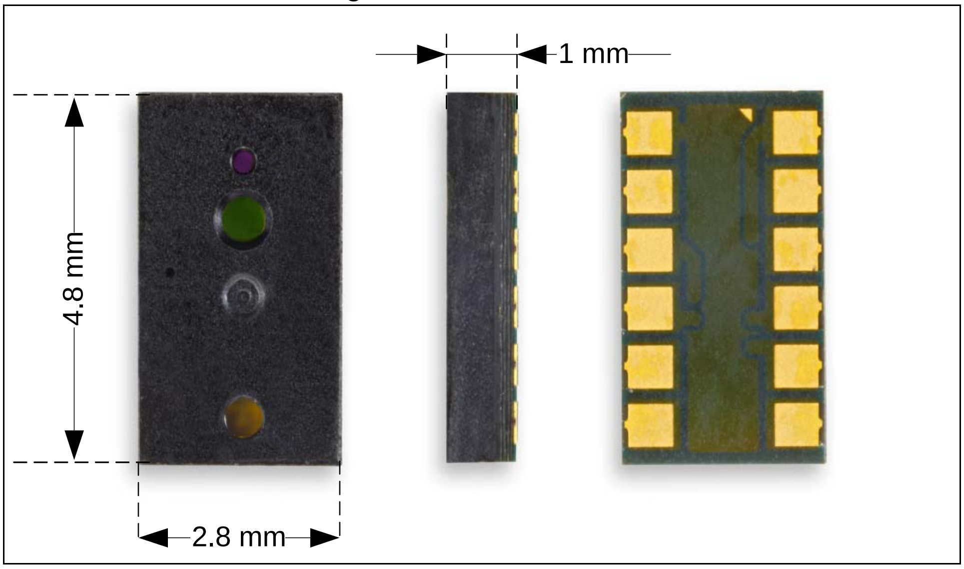
Using multiple VL6180Xs in a single design

By Darin Winterton

Introduction

The purpose of this document is to give an example of how multiple VL6180X devices can be placed on a board design while only using a single I2C interface to interact with the devices.

Figure 1. VL6180X device



Contents

1 Purpose

The purpose of this document is to give an example of how multiple VL6180X devices can be placed on a board design while only using a single I2C interface to interact with the device. The VL6180X is a proximity sensor that uses a time of flight technology to get distance, and the control and output interface is an I2C interface. Each VL6180X device has both a reset pin and an interrupt pin, which will be used in this design setup for using multiple devices.

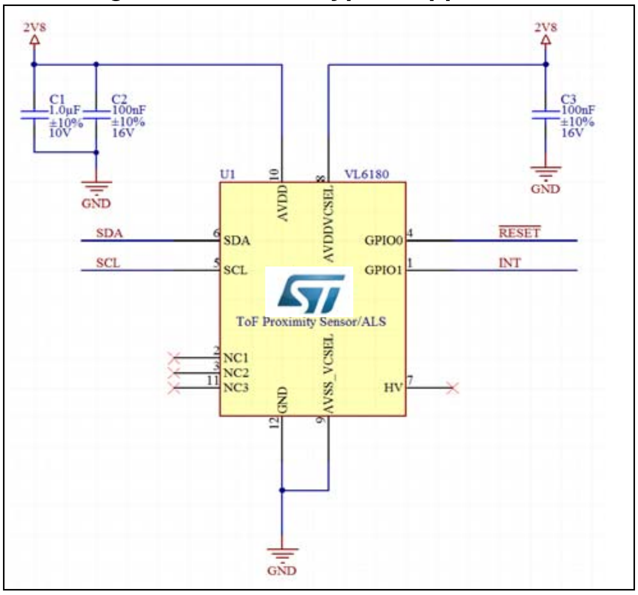
The uses for multiple VL6180Xs on a single design can include the following:

1. Distance sensing in multiple directions
2. Robotics with multiple sensors
3. Gesture Detection
4. ….

2 Multiple VL6180X operation steps and schematic suggestions

1. *Figure 2* is a typical example schematic using a VL6180X device. Since the VL6180X can have the I2C device address changed by doing an I2C write once it is booted, a separate reset pin would be needed to each VL6180X used in a design. Each device is then taken out of reset one at a time, and then the I2C Device Address is changed to a new unique address. This can be done by using multiple GPIO pins from the microprocessor on the board.

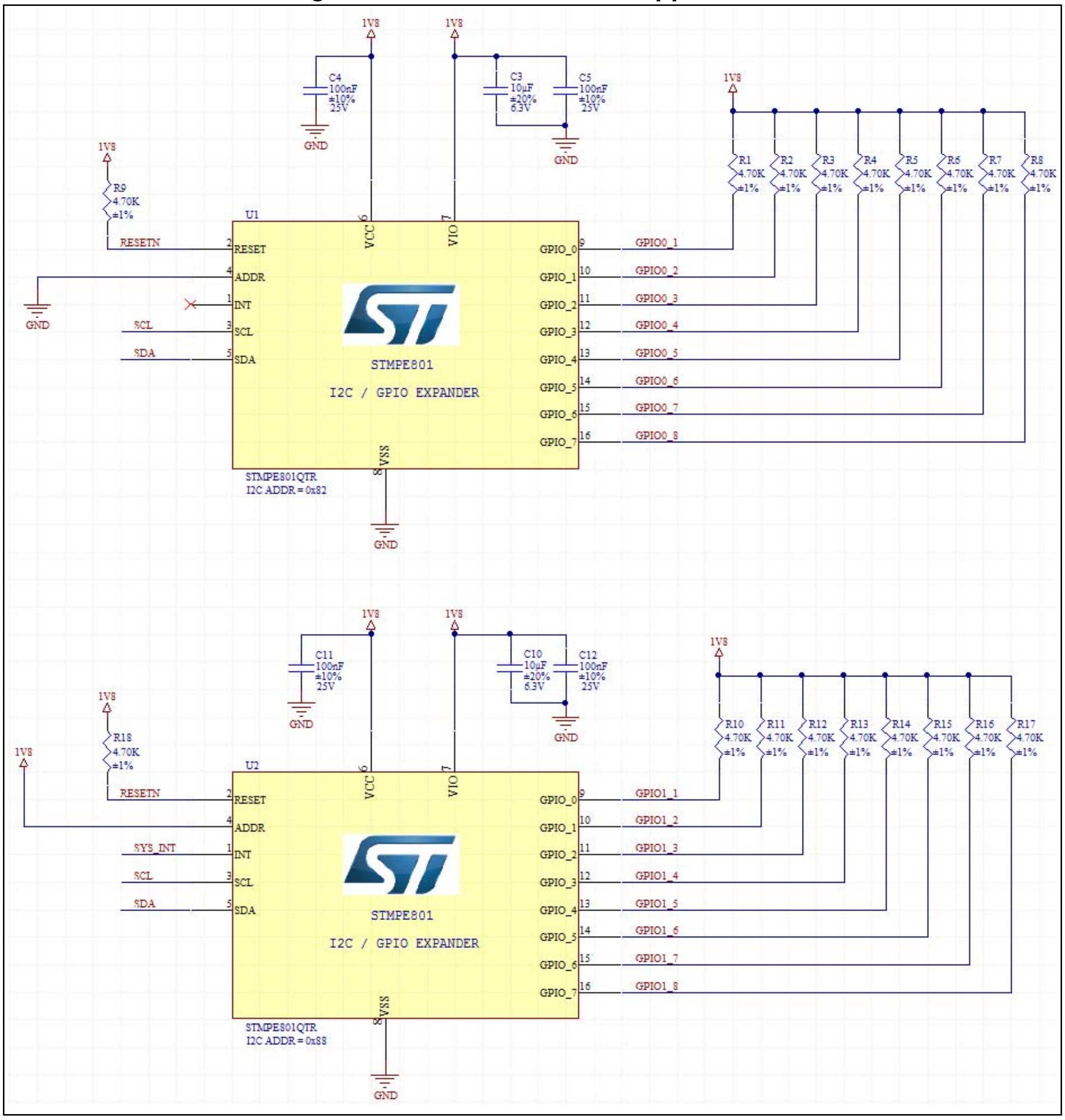
Figure 2. VL6180X typical application



2. For a board with multiple VL6180Xs used in the design, and there are a limited number of GPIO pins available on the microprocessor, the following are the recommended options for the use of these GPIO pins available.
   a) If only 1 GPIO pin is available from the micro-processor, then this pin would have to be used as a Reset pin for an I2C GPIO expander, and the interrupts from the VL6180Xs could not be used. The GPIO expander would control the resetting of the VL6180X devices, and polling of each VL6180X would have to be done to get any data needed. A second GPIO expander could be used for the interrupts, but the microprocessor would have to poll the U2 device to see if an interrupt has been triggered.
   b) If 2 GPIOs are available with multiple VL6180X devices, then two I2C GPIO expanders would be used, where the first would be for resetting the devices and the second would be used for the interrupts. An example schematic is shown in section 3.
   c) If (x) VL6180X devices are being used, and (x+1) GPIOs are available, then it is recommended to use a GPIO expander for the resetting of the VL6180X devices, and connect each individual interrupt to the microprocessor. Since the interrupt of each VL6180X is going directly to the microprocessor, time is saved when an interrupt is triggered directly to the microprocessor instead of going through a GPIO expander IC.
   d) If (x) VL6180X devices are being used, and (2x) GPIOs are available, then no GPIO expander chips are needed, and all reset and interrupts pins would be connected directly to the microprocessor.
3. If the options from section 2 recommends a I2C GPIO expander IC, *Figure 3* is an example schematic showing how to implement I2C GPIO expanders.

   This reference design has 8 VL6180X devices connected to the same I2C interface, where the I2C GPIO exander U1 is controlling the reset pins to each VL6180X and U2 is handling the Interrupts from the VL6180X devices. With this design below, only 2 GPIO pins are needed on the microprocessor, which are for the interrupt from U2 and the reset pins for both U1 and U2.

Figure 3. I2C GPIO extender application



4. The process to initialize each VL6180X is shown below

   a) Take VL6180X number 1 out of reset by either direct a direct GPIO from the microprocessor or by giving an I2C command to U1 to bring GPIO0_1 high.

   b) Write to the VL6180X just taken out of reset to the address 0x0212 with the value of the new I2C device address. The default value is 0x52, so for this example, the address of the first device will be changed to 0x80 (0x82 for device 2, 0x84 for device 3, …)

   c) To ensure it is working correctly, it is suggested that you now do a read of the device ID of the VL6180X with its new I2C device address written in step 3.b.

   d) If the device is now communicating on its new I2C device address, then repeat the steps 4.a to 4.c for each additional VL6180X attached to the device. In this case shown in step 3, GPIO0_x will each be brought high one at a time until all 8 devices are initialized.

5. After all of the VL6180X devices are initialized, the microprocessor can talk to each device individually and can configure them for operation. The configuration will include setting the convergence time, interrupts, inter-measurement time, Threshholds for the interrupts, etc.
6. To configure the Interrupts, the VL6180X device can give interrupts is various ways, including the following:
   a) Interrupts Disabled
   b) Interrupt when a measurement is completed
   c) Interrupt when a measurement is below a certain threshold of distance (ie. Distance less than 5cm)
   d) Interrupt when a measurement is above a a certain threshold of distance (ie. Distance greater than 10cm)
   e) Interrupt when a distance measured is out of a certain window (ie. Distance less than a Low Threshold or the Distance is greater than a High Threshold)
7. With the example of using the I2C GPIO Expander for the Interrupts, each VL6180X could be configured to give an interrupt when a distance is less than 5cm and to run in continuous measurement mode. If any device is showing a distance of less than 5cm, then an interrupt will go off for that device, which sets off the interrupt either directly to the microprocessor or from the U2 GPIO Expander. If the GPIO Expander is being used, the microprocessor would then poll U2 to see which VL6180X is triggering the interrupt. Next, the microprocessor would communicate directly to the VL6180X with the interrupt that is showing a distance less than 5cm. When the microprocessor has the information needed, it would then clear the interrupts on the VL6180X and the I2C GPIO expander U2 (if used).

3 Example source code

1. Function to see if a VL6180X can be detected on a certain I2C Device Address.

```
/*******************************************************************************
* Function name        : DetectVL6180X
* Description          : DetectVL6180X
* Input                : I2CDevAddr
* Output               : None
* Return               : Failure
*******************************************************************************/
u8 DetectVL6180X (u8 I2CDevAddr)
{
u8 i, DeviceID;
for (i=0; i<10; i++)
{
    DeviceID = I2C_VL6180XReadByte(I2CDevAddr, 0x00);
    Delay(20);
    if (DeviceID == 0xB4)
    {
            return 0;
    }
}
return 1;
}
```

2. Function to Change the I2C Device address

```
/*******************************************************************************
* Function name        : ChangeVL6180XI2CAddr
* Description          : ChangeVL6180XI2CAddr
* Input                : CurrentI2CDevAddr, NewI2CDevAddr
* Output               : None
* Return               : Failure
*******************************************************************************/
u8 ChangeVL6180XI2CAddr (u8 CurrentI2CDevAddr, u8 NewI2CDevAddr)
{
u8 Failure;
Failure = I2C_VL6180XByteWrite (CurrentI2CDevAddr, 0x0212, NewI2CDevAddr / 2);
if (Failure != 0x00)
{
    return 1;
}
else
{
    return 0;
}
 }
```

4 Revision history

Table 1. Document revision history

Date	Revision	Changes
14-May-2014	1	Initial release.