# C++ Software Code for VCNL4020 and VCNL3020

This application note provides an overview of C++ software code that is available for the VCNL4020 and VCNL3020. There are three files included: **main.ccp**, the header file **VCNL40x0.h**, and the C-code file **VCNL40x0.ccp**.

To receive a copy of the complete software code along with the four text files that show screen results for the four applications listed below, send an e-mail to sensorstechsupport@vishay.com.

The **main.cpp** contains four examples:

1. main #1 Read proximity on demand and ambient light on demand in endless loop
2. main #2 Proximity measurement in self-timed mode with 4 measurements/s
   Read proximity value if ready with conversion, endless loop
3. main #3 Proximity measurement in self-timed mode with 31 measurements/s
   Interrupt waiting on proximity value > upper threshold limit
4. main #4 Proximity measurement and ambient light measurement in self-timed mode
   Proximity with 31 measurements/s, ambient light with 2 measurement/s
   Interrupt waiting on proximity value > upper threshold limit
5. main #5 Read proximity on demand in an endless loop

## main.cpp

The delivered **main.cpp** shows:

```
#define VERSION "\n Version: 1.2  01/2012\n"
#define MAIN4             // select MAIN1, MAIN2, MAIN3, MAIN4, or MAIN5
                          // * please note that MAIN1 and MAIN4 must be modified to be used with VCNL3020 *
                             (no ambient measurements) e.g. MAIN5
#define BAUD    115200    // increase up to 921600 for high speed communication (depends on terminal programm
and USB mode)

#include "mbed.h"
#include "VCNL40x0.h"

VCNL40x0 VCNL40x0_Device (p28, p27, VCNL40x0_ADDRESS);      // Define SDA, SCL pin and I2C address
DigitalOut mled0(LED1);                                     // LED #1
DigitalOut mled1(LED2);                                     // LED #2
DigitalOut mled2(LED3);                                     // LED #3
Serial pc(USBTX, USBRX);                                    // Tx, Rx USB transmissiondefine VERSION "
```

mbed.h takes care of all necessary I²C-Bus, port, and internal and peripheral handling.

This header file comes with the mbed board: (www.mbed.org).

The function of the LEDs is just to show the I²C transmissions and interrupt activation. More information about the microcontroller and the compiler can be found at the end of this document.

The main #4 is the most complex software code and is shown below.

For main #4, the proximity measurement speed is 31 measurements per second in self-timed mode. The ambient light measurement which is made at the same time has a rate of 2 measurements per second.

The interrupt is assigned to the proximity measurement and the upper threshold is set to 100 counts above the previously measured offset counts.

When the count exceeds the upper threshold and an interrupt is generated, a red indicator light is illuminated on the VCNL40x0 demo kit sensor board.

APPLICATION NOTE

# C++ Software Code for VCNL4020 and VCNL3020

## main.cpp [Example 4]

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// main #4
// Proximity Measurement and Ambient light Measurement in selftimed mode
// Proximity with 31 measurements/s, Ambient light with 2 measurement/s
// Interrupt waiting on proximity value > upper threshold limit
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
# ifdef MAIN4

int main() {
    unsigned int i=0;
    unsigned char ID=0;
    unsigned char Command=0;
    unsigned char Current=0;
    unsigned int  ProxiValue=0;
    unsigned int  SummeProxiValue=0;
    unsigned int  AverageProxiValue=0;
    unsigned int  AmbiValue=0;
    unsigned char InterruptStatus=0;
    unsigned char InterruptControl=0;

    pc.baud(BAUD);

    // print information on screen
    pc.printf("\n\n VCNL4020/3020 Proximity/Ambient Light Sensor");
    pc.printf("\n library tested with mbed LPC1768 (ARM Cortex-M3 core) on www.mbed.org");
    pc.printf(VERSION);
    pc.printf("\n Demonstration #4:");
    pc.printf("\n Proximity Measurement and Ambient light Measurement in selftimed mode");
    pc.printf("\n Proximity with 31 measurements/s, Ambient light with 2 measurement/s");
    pc.printf("\n Interrupt waiting on proximity value > upper threshold limit");

    VCNL40x0_Device.ReadID (&ID);                           // Read VCNL40x0 product ID revision register
    pc.printf("\n\n Product ID Revision Register: %d", ID);

    VCNL40x0_Device.SetCurrent (20);                        // Set current to 200mA
    VCNL40x0_Device.ReadCurrent (&Current);                 // Read back IR LED current
    pc.printf("\n IR LED Current: %d", Current);

    // stop all activities (necessary for changing proximity rate, see datasheet)
    VCNL40x0_Device.SetCommandRegister (COMMAND_ALL_DISABLE);

    // set proximity rate to 31/s
    VCNL40x0_Device.SetProximityRate (PROX_MEASUREMENT_RATE_31);

    // enable prox and ambi in selftimed mode
    VCNL40x0_Device.SetCommandRegister (COMMAND_PROX_ENABLE |
                                        COMMAND_AMBI_ENABLE |
                                        COMMAND_SELFTIMED_MODE_ENABLE);

    // set interrupt control for threshold
    VCNL40x0_Device.SetInterruptControl (INTERRUPT_THRES_SEL_PROX |
                                         INTERRUPT_THRES_ENABLE |
                                         INTERRUPT_COUNT_EXCEED_1);

    // set ambient light measurement parameter
    VCNL40x0_Device.SetAmbiConfiguration (AMBI_PARA_AVERAGE_32 |
                                          AMBI_PARA_AUTO_OFFSET_ENABLE |
                                          AMBI_PARA_MEAS_RATE_2);

    // measure average of prox value
    SummeProxiValue = 0;

    for (i=0; i<30; i++) {
        do {                                                // wait on prox data ready bit
            VCNL40x0_Device.ReadCommandRegister (&Command);     // read command register
        } while (!(Command & COMMAND_MASK_PROX_DATA_READY));    // prox data ready ?

        VCNL40x0_Device.ReadProxiValue (&ProxiValue);           // read prox value

        SummeProxiValue += ProxiValue;                          // Summary of all measured prox values
    }

    AverageProxiValue = SummeProxiValue/30;                     // calculate average

    VCNL40x0_Device.SetHighThreshold (AverageProxiValue+100);   // set upper threshold for interrupt
    pc.printf("\n Upper Threshold Value: %i cts\n\n", AverageProxiValue+100);

    wait_ms(2000);                                              // wait 2s (only for display)

// endless loop ////////////////////////////////////////////////////////////////////////////////
```

# C++ Software Code for VCNL4020 and VCNL3020

```
    while (1) {

        // wait on data ready bit
        do {
            VCNL40x0_Device.ReadCommandRegister (&Command); // read command register
        } while (!(Command & (COMMAND_MASK_PROX_DATA_READY | COMMAND_MASK_AMBI_DATA_READY))); // data ready ?

        // read interrupt status register
        VCNL40x0_Device.ReadInterruptStatus (&InterruptStatus);

        // check interrupt status for High Threshold
        if (InterruptStatus & INTERRUPT_MASK_STATUS_THRES_HI) {
            mled2 = 1;                                          // LED on, Interrupt
            VCNL40x0_Device.SetInterruptStatus (InterruptStatus);  // clear Interrupt Status
            mled2 = 0;                                          // LED off, Interrupt
        }

        // prox value ready for using
        if (Command & COMMAND_MASK_PROX_DATA_READY) {
            mled0 = 1;                                  // LED on, Prox Data Ready
            VCNL40x0_Device.ReadProxiValue (&ProxiValue);   // read prox value

            // print prox value and interrupt status on screen
            pc.printf("\nProxi: %5.0i cts \tInterruptStatus: %i", ProxiValue, InterruptStatus);

            mled0 = 0;                                  // LED off, Prox data Ready
        }

        // ambi value ready for using
        if (Command & COMMAND_MASK_AMBI_DATA_READY) {
            mled1 = 1;                                  // LED on, Ambi Data Ready
            VCNL40x0_Device.ReadAmbiValue (&AmbiValue);     // read ambi value

            // print ambi value and interrupt status on screen
            pc.printf("\n                                                  Ambi: %i", AmbiValue);

            mled1 = 0;                                  // LED off, Ambi Data Ready
        }
    }
}
# endif
```

## VCNL40x0.h

This **VCNL40x0.h** header file contains all VCNL40x0 register numbers and registers bit information:

```
#ifndef VCNL40x0_H
#define VCNL40x0_H

#include "mbed.h"

// Library for the Vishay Proximity/Ambient Light Sensor VCNL4020/3020
// The VCNL4x00 is a I2C digital Proximity and Ambient Light Sensor in a small SMD package

#define VCNL40x0_ADDRESS               (0x26) // 001 0011 shifted left 1 bit = 0x26

// registers
#define REGISTER_COMMAND               (0x80)
#define REGISTER_ID                    (0x81)
#define REGISTER_PROX_RATE             (0x82)
#define REGISTER_PROX_CURRENT          (0x83)
#define REGISTER_AMBI_PARAMETER        (0x84)
#define REGISTER_AMBI_VALUE            (0x85)
#define REGISTER_PROX_VALUE            (0x87)
#define REGISTER_INTERRUPT_CONTROL     (0x89)
#define REGISTER_INTERRUPT_LOW_THRES   (0x8a)
#define REGISTER_INTERRUPT_HIGH_THRES  (0x8c)
#define REGISTER_INTERRUPT_STATUS      (0x8e)
#define REGISTER_PROX_TIMING           (0xf9)
#define REGISTER_AMBI_IR_LIGHT_LEVEL   (0x90)   // This register is not intended to be use by customer

// Bits in Command register (0x80)
#define COMMAND_ALL_DISABLE            (0x00)
#define COMMAND_SELFTIMED_MODE_ENABLE  (0x01)
#define COMMAND_PROX_ENABLE            (0x02)
#define COMMAND_AMBI_ENABLE            (0x04)
#define COMMAND_PROX_ON_DEMAND         (0x08)
#define COMMAND_AMBI_ON_DEMAND         (0x10)
#define COMMAND_MASK_PROX_DATA_READY   (0x20)
#define COMMAND_MASK_AMBI_DATA_READY   (0x40)
#define COMMAND_MASK_LOCK              (0x80)
```

# C++ Software Code for VCNL4020 and VCNL3020

```
// Bits in Product ID Revision Register (0x81)
#define PRODUCT_MASK_REVISION_ID        (0x0f)
#define PRODUCT_MASK_PRODUCT_ID         (0xf0)

// Bits in Prox Measurement Rate register (0x82)
#define PROX_MEASUREMENT_RATE_2         (0x00)   // DEFAULT
#define PROX_MEASUREMENT_RATE_4         (0x01)
#define PROX_MEASUREMENT_RATE_8         (0x02)
#define PROX_MEASUREMENT_RATE_16        (0x03)
#define PROX_MEASUREMENT_RATE_31        (0x04)
#define PROX_MEASUREMENT_RATE_62        (0x05)
#define PROX_MEASUREMENT_RATE_125       (0x06)
#define PROX_MEASUREMENT_RATE_250       (0x07)
#define PROX_MASK_MEASUREMENT_RATE      (0x07)

// Bits in Proximity LED current setting (0x83)
#define PROX_MASK_LED_CURRENT           (0x3f)   // DEFAULT = 2
#define PROX_MASK_FUSE_PROG_ID          (0xc0)

// Bits in Ambient Light Parameter register (0x84)
#define AMBI_PARA_AVERAGE_1             (0x00)
#define AMBI_PARA_AVERAGE_2             (0x01)
#define AMBI_PARA_AVERAGE_4             (0x02)
#define AMBI_PARA_AVERAGE_8             (0x03)
#define AMBI_PARA_AVERAGE_16            (0x04)
#define AMBI_PARA_AVERAGE_32            (0x05)   // DEFAULT
#define AMBI_PARA_AVERAGE_64            (0x06)
#define AMBI_PARA_AVERAGE_128           (0x07)
#define AMBI_MASK_PARA_AVERAGE          (0x07)

#define AMBI_PARA_AUTO_OFFSET_ENABLE    (0x08)   // DEFAULT enable
#define AMBI_MASK_PARA_AUTO_OFFSET      (0x08)

#define AMBI_PARA_MEAS_RATE_1           (0x00)
#define AMBI_PARA_MEAS_RATE_2           (0x10)   // DEFAULT
#define AMBI_PARA_MEAS_RATE_3           (0x20)
#define AMBI_PARA_MEAS_RATE_4           (0x30)
#define AMBI_PARA_MEAS_RATE_5           (0x40)
#define AMBI_PARA_MEAS_RATE_6           (0x50)
#define AMBI_PARA_MEAS_RATE_8           (0x60)
#define AMBI_PARA_MEAS_RATE_10          (0x70)
#define AMBI_MASK_PARA_MEAS_RATE        (0x70)

#define AMBI_PARA_CONT_CONV_ENABLE      (0x80)
#define AMBI_MASK_PARA_CONT_CONV        (0x80)   // DEFAULT disable

// Bits in Interrupt Control Register (x89)
#define INTERRUPT_THRES_SEL_PROX        (0x00)
#define INTERRUPT_THRES_SEL_ALS         (0x01)

#define INTERRUPT_THRES_ENABLE          (0x02)

#define INTERRUPT_ALS_READY_ENABLE      (0x04)

#define INTERRUPT_PROX_READY_ENABLE     (0x08)

#define INTERRUPT_COUNT_EXCEED_1        (0x00)   // DEFAULT
#define INTERRUPT_COUNT_EXCEED_2        (0x20)
#define INTERRUPT_COUNT_EXCEED_4        (0x40)
#define INTERRUPT_COUNT_EXCEED_8        (0x60)
#define INTERRUPT_COUNT_EXCEED_16       (0x80)
#define INTERRUPT_COUNT_EXCEED_32       (0xa0)
#define INTERRUPT_COUNT_EXCEED_64       (0xc0)
#define INTERRUPT_COUNT_EXCEED_128      (0xe0)
#define INTERRUPT_MASK_COUNT_EXCEED     (0xe0)

// Bits in Interrupt Status Register (x8e)
#define INTERRUPT_STATUS_THRES_HI       (0x01)
#define INTERRUPT_STATUS_THRES_LO       (0x02)
#define INTERRUPT_STATUS_ALS_READY      (0x04)
#define INTERRUPT_STATUS_PROX_READY     (0x08)
#define INTERRUPT_MASK_STATUS_THRES_HI  (0x01)
#define INTERRUPT_MASK_THRES_LO         (0x02)
#define INTERRUPT_MASK_ALS_READY        (0x04)
#define INTERRUPT_MASK_PROX_READY       (0x08)
```

# C++ Software Code for VCNL4020 and VCNL3020

```
typedef enum {
    VCNL40x0_ERROR_OK = 0,              // Everything executed normally
    VCNL40x0_ERROR_I2CINIT,             // Unable to initialize I2C
    VCNL40x0_ERROR_I2CBUSY,             // I2C already in use
    VCNL40x0_ERROR_LAST
} VCNL40x0Error_e;


class VCNL40x0 {
public:
// Creates an instance of the class.
// Connect module at I2C address addr using I2C port pins sda and scl.

    VCNL40x0 (PinName sda, PinName scl, unsigned char addr);

// Destroys instance

    ~VCNL40x0();

// public functions

    VCNL40x0Error_e Init (void);

    VCNL40x0Error_e SetCommandRegister (unsigned char Command);
    VCNL40x0Error_e SetCurrent (unsigned char CurrentValue);
    VCNL40x0Error_e SetProximityRate (unsigned char ProximityRate);
    VCNL40x0Error_e SetAmbiConfiguration (unsigned char AmbiConfiguration);
    VCNL40x0Error_e SetLowThreshold (unsigned int LowThreshold);
    VCNL40x0Error_e SetHighThreshold (unsigned int HighThreshold);
    VCNL40x0Error_e SetInterruptControl (unsigned char InterruptControl);
    VCNL40x0Error_e SetInterruptStatus (unsigned char InterruptStatus);
    VCNL40x0Error_e SetModulatorTimingAdjustment (unsigned char ModulatorTimingAdjustment);

    VCNL40x0Error_e ReadID (unsigned char *ID);
    VCNL40x0Error_e ReadCurrent (unsigned char *CurrentValue);
    VCNL40x0Error_e ReadCommandRegister (unsigned char *Command);
    VCNL40x0Error_e ReadProxiValue (unsigned int *ProxiValue);
    VCNL40x0Error_e ReadAmbiValue (unsigned int *AmbiValue);
    VCNL40x0Error_e ReadInterruptStatus (unsigned char *InterruptStatus);
    VCNL40x0Error_e ReadInterruptControl (unsigned char *InterruptControl);

    VCNL40x0Error_e ReadProxiOnDemand (unsigned int *ProxiValue);
    VCNL40x0Error_e ReadAmbiOnDemand (unsigned int *AmbiValue);

private:
    I2C _i2c;
    int _addr;
    char _send[3];
    char _receive[2];

};

#endif
```

## VCNL40x0.cpp

This **VCNL40x0.cpp** file contains all definitions and public functions:

```
#include "VCNL40x0.h"

////////////////////////////////////////////////////////////////////////////////////////////////////////////////

VCNL40x0::VCNL40x0(PinName sda, PinName scl, unsigned char addr) : _i2c(sda, scl), _addr(addr) {
    _i2c.frequency(1000000);                                // set I2C frequency to 1MHz
}

VCNL40x0::~VCNL40x0() {
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////

VCNL40x0Error_e VCNL40x0::SetCommandRegister (unsigned char Command) {

    _send[0] = REGISTER_COMMAND;                            // VCNL40x0 Configuration register
    _send[1] = Command;
    _i2c.write(VCNL40x0_ADDRESS,_send, 2);                  // Write 2 bytes on I2C

    return VCNL40x0_ERROR_OK;
}
```

# C++ Software Code for VCNL4020 and VCNL3020

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

VCNL40x0Error_e VCNL40x0::ReadCommandRegister (unsigned char *Command) {

    _send[0] = REGISTER_COMMAND;                          // VCNL40x0 Configuration register
    _i2c.write(VCNL40x0_ADDRESS,_send, 1);                // Write 1 byte on I2C
    _i2c.read(VCNL40x0_ADDRESS+1,_receive, 1);            // Read 1 byte on I2C

    *Command = (unsigned char)(_receive[0]);

    return VCNL40x0_ERROR_OK;
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

VCNL40x0Error_e VCNL40x0::ReadID (unsigned char *ID) {

    _send[0] = REGISTER_ID;                               // VCNL40x0 product ID revision register
    _i2c.write(VCNL40x0_ADDRESS, _send, 1);               // Write 1 byte on I2C
    _i2c.read(VCNL40x0_ADDRESS+1, _receive, 1);           // Read 1 byte on I2C

    *ID = (unsigned char)(_receive[0]);

    return VCNL40x0_ERROR_OK;
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

VCNL40x0Error_e VCNL40x0::SetCurrent (unsigned char Current) {

    _send[0] = REGISTER_PROX_CURRENT;                     // VCNL40x0 IR LED Current register
    _send[1] = Current;
    _i2c.write(VCNL40x0_ADDRESS,_send, 2);                // Write 2 bytes on I2C

    return VCNL40x0_ERROR_OK;
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

VCNL40x0Error_e VCNL40x0::ReadCurrent (unsigned char *Current) {

    _send[0] = REGISTER_PROX_CURRENT;                     // VCNL40x0 IR LED current register
    _i2c.write(VCNL40x0_ADDRESS,_send, 1);                // Write 1 byte on I2C
    _i2c.read(VCNL40x0_ADDRESS+1,_receive, 1);            // Read 1 byte on I2C

    *Current = (unsigned char)(_receive[0]);

    return VCNL40x0_ERROR_OK;
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

VCNL40x0Error_e VCNL40x0::SetProximityRate (unsigned char ProximityRate) {

    _send[0] = REGISTER_PROX_RATE;                        // VCNL40x0 Proximity rate register
    _send[1] = ProximityRate;
    _i2c.write(VCNL40x0_ADDRESS,_send, 2);                // Write 2 bytes on I2C

    return VCNL40x0_ERROR_OK;
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

VCNL40x0Error_e VCNL40x0::SetAmbiConfiguration (unsigned char AmbiConfiguration) {

    _send[0] = REGISTER_AMBI_PARAMETER;                   // VCNL40x0 Ambient light configuration
    _send[1] = AmbiConfiguration;
    _i2c.write(VCNL40x0_ADDRESS,_send, 2);                // Write 2 bytes on I2C

    return VCNL40x0_ERROR_OK;
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

VCNL40x0Error_e VCNL40x0::SetInterruptControl (unsigned char InterruptControl) {

    _send[0] = REGISTER_INTERRUPT_CONTROL;                // VCNL40x0 Interrupt Control register
    _send[1] = InterruptControl;
    _i2c.write(VCNL40x0_ADDRESS,_send, 2);                // Write 2 bytes on I2C

    return VCNL40x0_ERROR_OK;
}
```

# C++ Software Code for VCNL4020 and VCNL3020

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

VCNL40x0Error_e VCNL40x0::ReadInterruptControl (unsigned char *InterruptControl) {

    _send[0] = REGISTER_INTERRUPT_CONTROL;                  // VCNL40x0 Interrupt Control register
    _i2c.write(VCNL40x0_ADDRESS,_send, 1);                  // Write 1 byte on I2C
    _i2c.read(VCNL40x0_ADDRESS+1,_receive, 1);              // Read 1 byte on I2C

    *InterruptControl = (unsigned char)(_receive[0]);

    return VCNL40x0_ERROR_OK;
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

VCNL40x0Error_e VCNL40x0::SetInterruptStatus (unsigned char InterruptStatus) {

    _send[0] = REGISTER_INTERRUPT_STATUS;                   // VCNL40x0 Interrupt Status register
    _send[1] = InterruptStatus;
    _i2c.write(VCNL40x0_ADDRESS,_send, 2);                  // Write 2 bytes on I2C

    return VCNL40x0_ERROR_OK;
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

VCNL40x0Error_e VCNL40x0::SetModulatorTimingAdjustment (unsigned char ModulatorTimingAdjustment) {

    _send[0] = REGISTER_PROX_TIMING;                        // VCNL40x0 Modulator Timing Adjustment register
    _send[1] = ModulatorTimingAdjustment;
    _i2c.write(VCNL40x0_ADDRESS,_send, 2);                  // Write 2 bytes on I2C

    return VCNL40x0_ERROR_OK;
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

VCNL40x0Error_e VCNL40x0::ReadInterruptStatus (unsigned char *InterruptStatus) {

    _send[0] = REGISTER_INTERRUPT_STATUS;                   // VCNL40x0 Interrupt Status register
    _i2c.write(VCNL40x0_ADDRESS,_send, 1);                  // Write 1 byte on I2C
    _i2c.read(VCNL40x0_ADDRESS+1,_receive, 1);              // Read 1 byte on I2C

    *InterruptStatus = (unsigned char)(_receive[0]);

    return VCNL40x0_ERROR_OK;
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

VCNL40x0Error_e VCNL40x0::ReadProxiValue (unsigned int *ProxiValue) {

    _send[0] = REGISTER_PROX_VALUE;                         // VCNL40x0 Proximity Value register
    _i2c.write(VCNL40x0_ADDRESS, _send, 1);                 // Write 1 byte on I2C
    _i2c.read(VCNL40x0_ADDRESS+1, _receive, 2);             // Read 2 bytes on I2C
    *ProxiValue = ((unsigned int)_receive[0] << 8 | (unsigned char)_receive[1]);

    return VCNL40x0_ERROR_OK;
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

VCNL40x0Error_e VCNL40x0::ReadAmbiValue (unsigned int *AmbiValue) {

    _send[0] = REGISTER_AMBI_VALUE;                          // VCNL40x0 Ambient Light Value register
    _i2c.write(VCNL40x0_ADDRESS, _send, 1);                  // Write 1 byte on I2C
    _i2c.read(VCNL40x0_ADDRESS+1, _receive, 2);              // Read 2 bytes on I2C
    *AmbiValue = ((unsigned int)_receive[0] << 8 | (unsigned char)_receive[1]);

    return VCNL40x0_ERROR_OK;
}
```

# C++ Software Code for VCNL4020 and VCNL3020

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

VCNL40x0Error_e VCNL40x0::SetLowThreshold (unsigned int LowThreshold) {

    unsigned char LoByte=0, HiByte=0;

    LoByte = (unsigned char)(LowThreshold & 0x00ff);
    HiByte = (unsigned char)((LowThreshold & 0xff00)>>8);

    _send[0] = REGISTER_INTERRUPT_LOW_THRES;                // VCNL40x0 Low Threshold Register, Hi Byte
    _send[1] = HiByte;
    _i2c.write(VCNL40x0_ADDRESS,_send, 2);                  // Write 2 bytes on I2C

    _send[0] = REGISTER_INTERRUPT_LOW_THRES+1;              // VCNL40x0 Low Threshold Register, Lo Byte
    _send[1] = LoByte;
    _i2c.write(VCNL40x0_ADDRESS,_send, 2);                  // Write 2 bytes on I2C

    return VCNL40x0_ERROR_OK;
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

VCNL40x0Error_e VCNL40x0::SetHighThreshold (unsigned int HighThreshold) {

    unsigned char LoByte=0, HiByte=0;

    LoByte = (unsigned char)(HighThreshold & 0x00ff);
    HiByte = (unsigned char)((HighThreshold & 0xff00)>>8);

    _send[0] = REGISTER_INTERRUPT_HIGH_THRES;               // VCNL40x0 High Threshold Register, Hi Byte
    _send[1] = HiByte;
    _i2c.write(VCNL40x0_ADDRESS,_send, 2);                  // Write 2 bytes on I2C

    _send[0] = REGISTER_INTERRUPT_HIGH_THRES+1;             // VCNL40x0 High Threshold Register, Lo Byte
    _send[1] = LoByte;
    _i2c.write(VCNL40x0_ADDRESS,_send, 2);                  // Write 2 bytes on I2C

    return VCNL40x0_ERROR_OK;
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

VCNL40x0Error_e VCNL40x0::ReadProxiOnDemand (unsigned int *ProxiValue) {

    unsigned char Command=0;

    // enable prox value on demand
    SetCommandRegister (COMMAND_PROX_ENABLE | COMMAND_PROX_ON_DEMAND);

    // wait on prox data ready bit
    do {
        ReadCommandRegister (&Command);                     // read command register
    } while (!(Command & COMMAND_MASK_PROX_DATA_READY));

    ReadProxiValue (ProxiValue);                            // read prox value

    SetCommandRegister (COMMAND_ALL_DISABLE);               // stop prox value on demand

    return VCNL40x0_ERROR_OK;
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

VCNL40x0Error_e VCNL40x0::ReadAmbiOnDemand (unsigned int *AmbiValue) {

    unsigned char Command=0;

    // enable ambi value on demand
    SetCommandRegister (COMMAND_PROX_ENABLE | COMMAND_AMBI_ON_DEMAND);

    // wait on ambi data ready bit
    do {
        ReadCommandRegister (&Command);                     // read command register
    } while (!(Command & COMMAND_MASK_AMBI_DATA_READY));

    ReadAmbiValue (AmbiValue);                              // read ambi value

    SetCommandRegister (COMMAND_ALL_DISABLE);               // stop ambi value on demand

    return VCNL40x0_ERROR_OK;
}
```

# C++ Software Code for VCNL4020 and VCNL3020

The five text files included within the zip show the results that would be seen on-screen when running the five separate listed applications.

These five examples are related to the “main #1” to “main #5” routines within the **main.ccp**:

### Logging_Main1.txt

In addition to the proximity on demand value, this “main #1” also shows the ambient light on demand value and calculated illuminance.

```
...

Proxi:  3012 cts        Ambi:   710 cts         Illuminance:  177.50 lx
Proxi:  2974 cts        Ambi:   709 cts         Illuminance:  177.25 lx
Proxi:  3003 cts        Ambi:   709 cts         Illuminance:  177.25 lx
Proxi:  3074 cts        Ambi:   708 cts         Illuminance:  177.00 lx

...
```

To get the illuminance value from the counts, just divide them by four, e.g.: 708 cts/4 = 177.00 lx.

### Logging_Main2.txt

This “main #2” shows some results of proximity measurement in self-timed mode with 4 measurements/s.

```
...

Proxi:  2458 cts
Proxi:  2390 cts
Proxi:  2444 cts

...
```

The endless loop enables evaluation of the offset counts within the used application, as well as testing for the amount of counts that will be read out with the object in a defined distance.

### Logging_Main3.txt

The “main #3” shows results of proximity measurement in self-timed mode with 31 measurements/s and also the corresponding interrupt status.

```
...

Proxi:  2930 cts        InterruptStatus: 0
Proxi:  3002 cts        InterruptStatus: 0
Proxi:  3043 cts        InterruptStatus: 0
Proxi:  3218 cts        InterruptStatus: 1
Proxi:  3257 cts        InterruptStatus: 1
Proxi:  3307 cts        InterruptStatus: 1

...
```

Within the line where the interrupt status changes from “0” to “1,” one can also see the change to 3218 counts from just 3043 which is more than 100 over the defined limit.

### Logging_Main4.txt

The “main #4” shows the results of parallel proximity measurements AND ambient light measurements in self-timed mode with 31 measurements/s for proximity and 2 measurements/s for ambient light.

The interrupt is assigned to proximity and a subroutine is averaging 30 samples to define currently available offset counts. The upper threshold is set at 100 counts higher than this averaged result.

Only this fourth example “Logging_Main4.txt” is shown here completely, and explained on the next page.

# C++ Software Code for VCNL4020 and VCNL3020

## Logging_Main4.txt

```
VCNL4020/3020 Proximity/Ambient Light Sensor
library tested with mbed LPC1768 (ARM Cortex-M3 core) on www.mbed.org
Version: 1.2  01/2012

Demonstration #4:
Proximity Measurement and Ambient light Measurement in selftimed mode
Proximity with 31 measurements/s, Ambient light with 2 measurement/s
Interrupt waiting on proximity value > upper threshold limit

Product ID Revision Register: 33
IR LED Current: 20
Upper Threshold Value: 2570 cts

Proxi:  2492 cts        InterruptStatus: 0
Proxi:  2416 cts        InterruptStatus: 0
Proxi:  2479 cts        InterruptStatus: 0
Proxi:  2475 cts        InterruptStatus: 0
Proxi:  2483 cts        InterruptStatus: 0
Proxi:  2487 cts        InterruptStatus: 0
Proxi:  2437 cts        InterruptStatus: 0
Proxi:  2446 cts        InterruptStatus: 0
Proxi:  2454 cts        InterruptStatus: 0
Proxi:  2479 cts        InterruptStatus: 0
Proxi:  2423 cts        InterruptStatus: 0

Proxi:  2483 cts        InterruptStatus: 0
Proxi:  2436 cts        InterruptStatus: 0
Proxi:  2407 cts        InterruptStatus: 0
Proxi:  2437 cts        InterruptStatus: 0
Proxi:  2407 cts        InterruptStatus: 0
Proxi:  2438 cts        InterruptStatus: 0
Proxi:  2451 cts        InterruptStatus: 0
Proxi:  2489 cts        InterruptStatus: 0
Proxi:  2432 cts        InterruptStatus: 0
Proxi:  2504 cts        InterruptStatus: 0
Proxi:  2504 cts        InterruptStatus: 0
Proxi:  2443 cts        InterruptStatus: 0
Proxi:  2424 cts        InterruptStatus: 0
Proxi:  2428 cts        InterruptStatus: 0
Proxi:  2481 cts        InterruptStatus: 0

Proxi:  2538 cts        InterruptStatus: 0
Proxi:  2435 cts        InterruptStatus: 0
Proxi:  2521 cts        InterruptStatus: 0
Proxi:  2482 cts        InterruptStatus: 0
Proxi:  2533 cts        InterruptStatus: 0
Proxi:  2523 cts        InterruptStatus: 0
Proxi:  2544 cts        InterruptStatus: 0
Proxi:  2575 cts        InterruptStatus: 1
Proxi:  2632 cts        InterruptStatus: 1
Proxi:  2658 cts        InterruptStatus: 1
Proxi:  2671 cts        InterruptStatus: 1
Proxi:  2780 cts        InterruptStatus: 1
Proxi:  2754 cts        InterruptStatus: 1
Proxi:  2808 cts        InterruptStatus: 1
Proxi:  2863 cts        InterruptStatus: 1
Proxi:  2848 cts        InterruptStatus: 1

Proxi:  2926 cts        InterruptStatus: 1
Proxi:  2889 cts        InterruptStatus: 1
```

From here the actual proximity offset count is printed as available with the used application. It is about 2470 (± 60 cts).

```
Ambi: 1118
```

For every 15/16 proximity values, one ambient result is visible.
This is due to programs
PROX_MEASUREMENT_RATE_31
and
AMBI_PARA_MEAS_RATE_2

```
Ambi: 1075
```

With the evaluated offset count of 2470 above (30 measurements averaged (see main.cpp)) and the definition of the high threshold to be at offset cts + 100 cts = 2570, at 2575 cts this threshold is exceeded and the interrupt is set.

What is also possible to see here:
The closer the hand, the higher the proximity counts AND the lower the ambient counts.
This is due to shadowing created by the light source straight above.

```
Ambi: 779
```

## Logging_Main5.txt

The “main #5” shows the proximity on demand values. It is a modified version of “main #1” in order to be used with the VCNL3020.

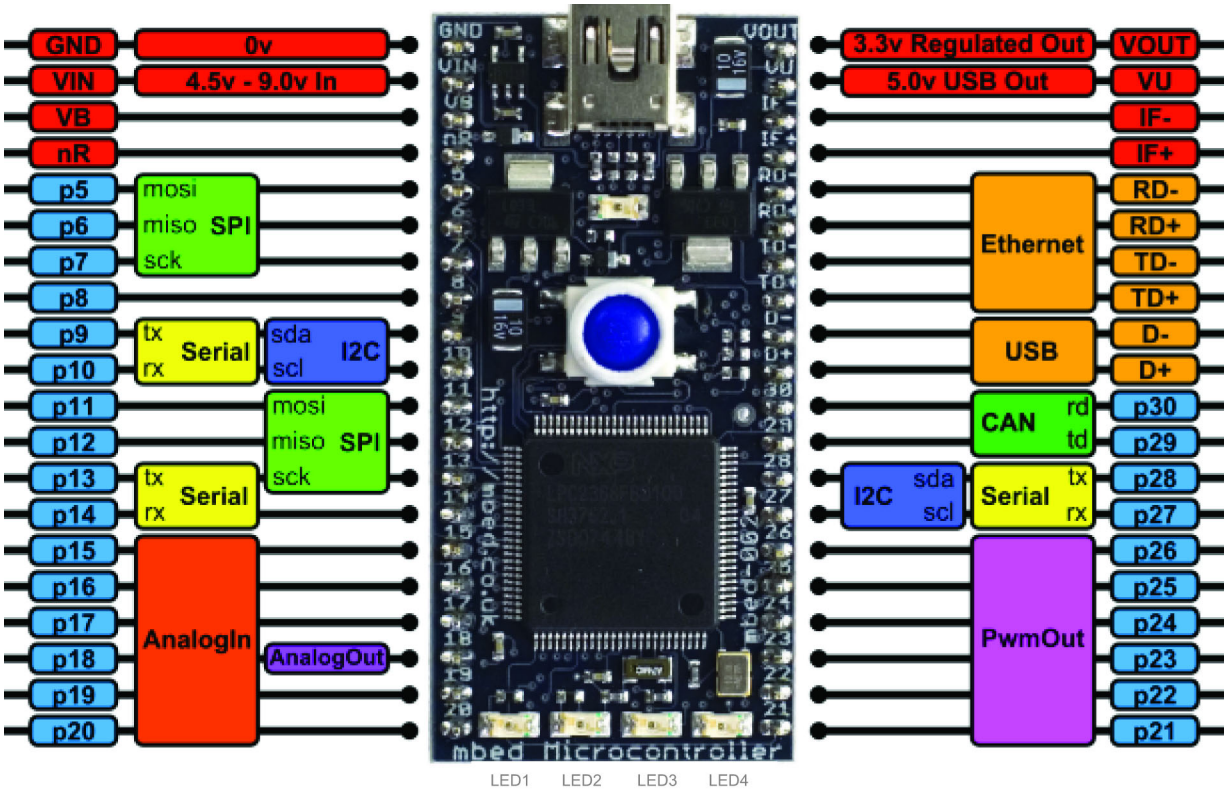
```
...

Proxi:  3012 cts
Proxi:  2974 cts
Proxi:  3003 cts
Proxi:  3074 cts

...
```

APPLICATION NOTE

# C++ Software Code for VCNL4020 and VCNL3020

The controller used for developing this code was an ARM Processor (NXP LPC1768), which is available on a so-called “mbed Board” and can be ordered here: http://mbed.org/handbook/order.

More useful information is available under: www.mbed.org.




**Microcontroller board: mbed NXP LPC1768**
**Rapid Prototyping for general microcontroller applications, Ethernet, USB and 32-bit ARM® Cortex™-M3 based designs**

The $I^2C$ bus of the VCNL40x0 demo board is connected towards P28 (SDA) and P27 (SCL), as shown within the mbed NXP LPC1768 board picture above.

The Interrupt pin is not routed from the VCNL40x0 towards the mbed board because the red LED at the VCNL40x0 demo board indicates this event.

LED0 indicates an available proximity measurement and LED1 an available ambient light measurement.

# C++ Software Code for VCNL4020 and VCNL3020


Author: Ds, version 1.2