OPT3101 SDK Users Guide

Author: Alex Bhandari-Young – Texas Instruments Incorporated

Contents

Introduction

The OPT3101 device is a high-speed, high-resolution AFE for continuous-wave, time-of-flight based proximity sensing and range finding. The OPT3101SDK SDK provides pre-written functions for data capture, register read/write, and calibration of the device. OPT3101 Configurator tool is required to generate certain functions since the SDK's functions depend on how the OPT3101 system is configured. Additionally, the SDK can be used in calibration tool mode whereby the SDK will be loaded to an MSP430 Launchpad, wired to an OPT3101 board to test and calibrate and connected to a PC through USB. All calibration is printed to the COM port viewer on the PC.

Quick Start

The ***Calibration Tool Mode*** section covers the step-by-step to get the SDK up and running on a TI MSP430F5529 Launchpad. This serves as both the quick start as well as full instructions for running.

Calibration Overview

Background

As OPT3101 is an analog front-end (AFE) the optical emitter and photodiode components are placed on the printed circuit board (PCB) at the system level. OPT3101 is compatible with a wide variety of emitters and photodiodes that can be arranged in various configurations to support many different applications and use-cases. Due to this every OPT3101 system requires calibration to function correctly. The calibration is specific to the PCB design and layout, components used, enclosure, etc. Per-unit calibrations are required on every board and are done in the factory during production. Per-system calibrations are established on a small subset of boards during testing and then applied to all boards in production. See the How to set up and calibrate OPT3101 based systems document for more details on the calibration process.

Different types of calibration

OPT3101 supports crosstalk correction functions for improved performance

Mandatory for any use (testing, evaluation, production)

- Crosstalk single point
- Phase offset single point

Mandatory for typical production ready solution

- Crosstalk single point
- Phase offset single point
- Crosstalk over temperature
- Phase over temperature
- Phase over ambient

Using this SDK for calibration

The SDK can be used to perform calibration in the following ways ordered from least to most work for the end user.

- **Calibration tool mode on MSP430F5529 Launchpad** – SDK is preconfigured to run on an MSP430F5529 launchpad out of the box. SDK needs to be imported to code composer studio, configuration file from configurator tool is added to project, calibration step to run is selected in main.cpp, SDK is compiled in CCS and loaded to the MSP430, MS430 is wired to the I2C lines of the OPT3101 PCB, MSP430 USB is connected to PC, and calibration is controlled through the COM port terminal interface.
- **Calibration tool mode on another TI Launchpad –** The SDK code composer studio project is configured for the MSP430F5529 launchpad, but by changing CCS settings can be reconfigured to compile and run on another TI Launchpad.
- **Calibration tool mode on any host** – The SDK can run on any system that can compile C++ including microcontrollers (almost all support C++), Windows, Linux (including Raspberry Pi), etc. The same calibration tool mode can be used on any other these hosts with modifications to the hostController.cpp file which includes the functions that are host specific and would need to be updated. More details for this are in the SDK documentation.
- **SDK functions called as part of a larger code base** – The SDK is written so it's functions can be incorporated in any custom C++ system. This allows custom code with access to the SDK high and low level calibration functions, other OPT3101 functions used by the higher level functions, data capture functions, and register map and register read/write interface.

Calibration Tool Mode

This section covers the step-by-step to get the SDK up and running on a TI MSP430F5529 Launchpad. For those using a different host other than the Launchpad these instructions are still applicable, but the *Code Composer Studio IDE Setup* section should be skipped.

The SDK works on the MSP430F5529 Launchpad out of the box and can be compiled on other platforms with slight modification to the SDK. The same can be done on any other platform as well, as long the *hostController* is modified to have the reset, i2C communications and pause functions implemented for the host. Code composer studio software is required for compiling for the launchpad and the SDK includes a preconfigured CCS project that should compile out of the box.

This sections is divided into the following subsections

1. Setting up the calibration tool and getting it running
2. Basic testing and evaluation on a new design
3. System level coefficient determination for a new design for in-depth testing of the design and getting ready for production
4. Production calibration using the tool

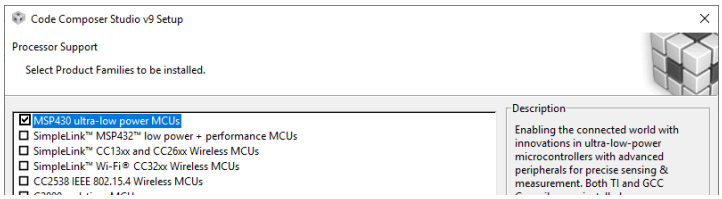
Setup

Steps to setup the SDK in calibration mode are as follows.

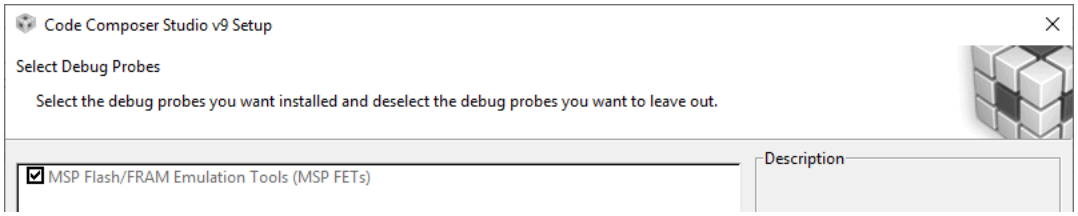
Code Composer Studio IDE Setup

1. Install Code Composer Studio (CSS) http://www.ti.com/tool/CCSTUDIO if not already installed. Installation instructions are here: https://software-dl.ti.com/ccs/esd/documents/users_guide/ccs_installation.html#installation-process

   Make sure to install with **MSP430 processor support** as shown below.

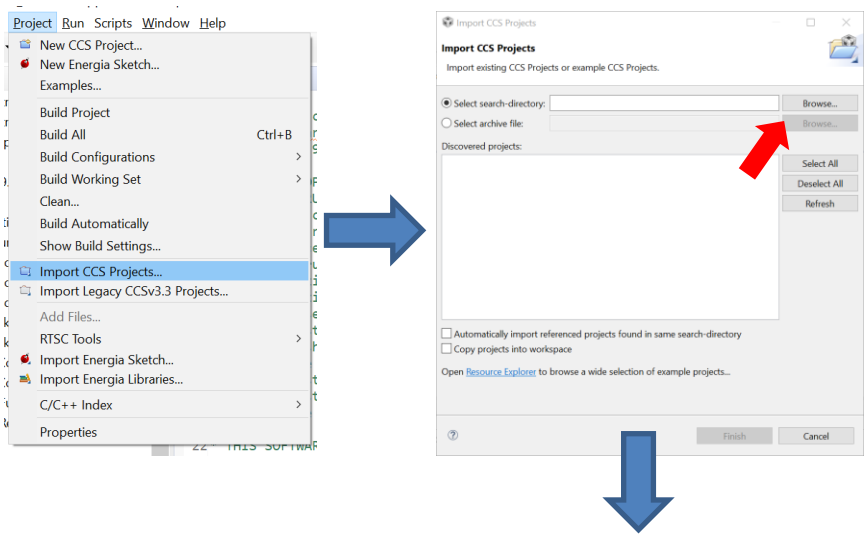
   

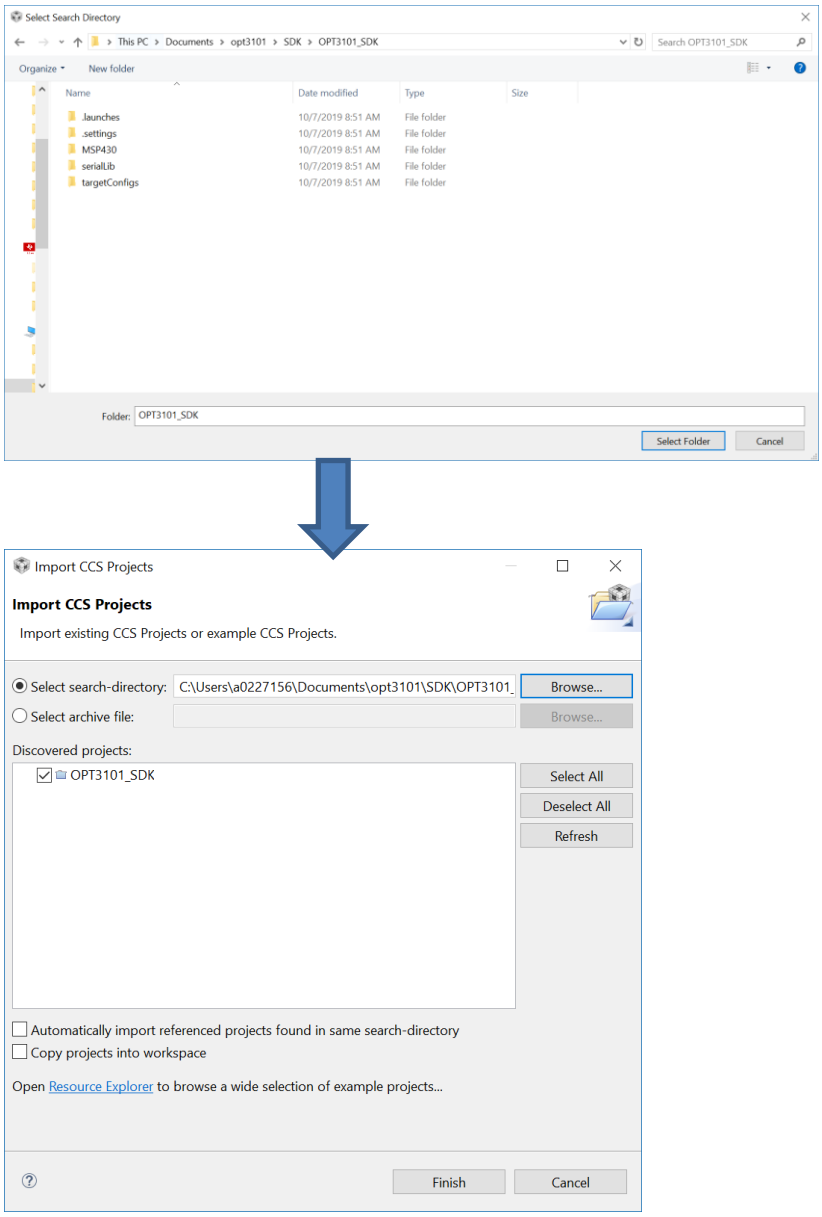
   Make sure to install MSP FETs support as shown below.

   This SDK is tested on **CCS version 9.1.0** so it is recommended to download this version. This SDK is tested on **MSP430 compiler version TI v18.12.3.LTS** so it is also recommended to install this compiler version.

2. Import the SDK project into code composer studio. Following the screenshots below. See here for more instructions: http://processors.wiki.ti.com/index.php/Importing_Projects_into_CCS#Importing_an_existing_project

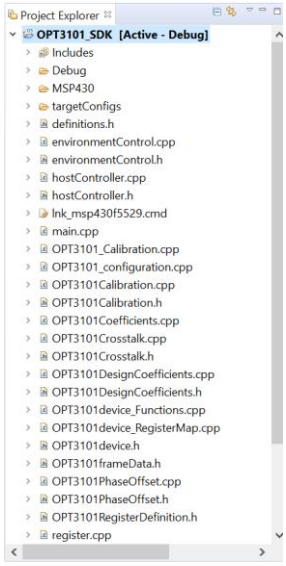
   



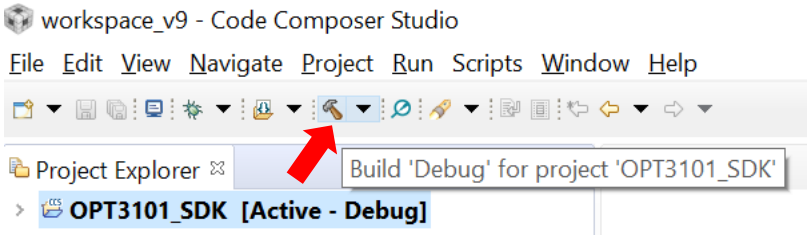
Then click finish.

Note: The SDK should be extracted from the zip folder it was downloaded from to be imported.

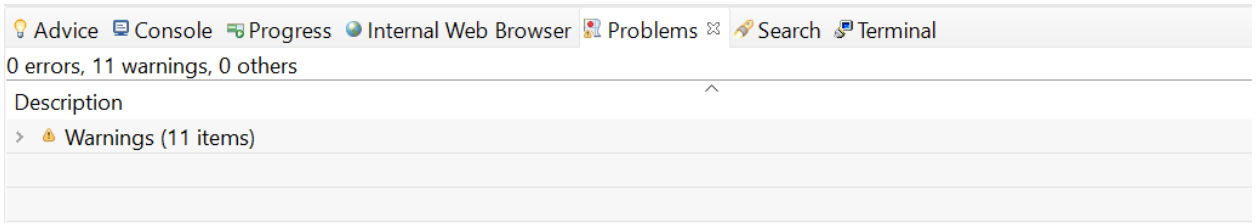
All files in CCS show up on the left side of the screen

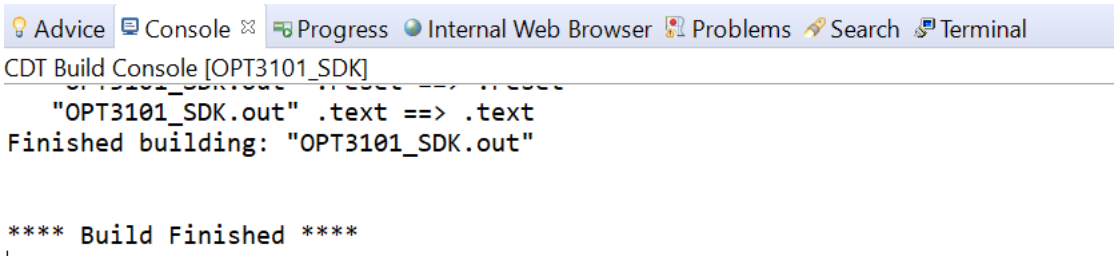


3. Compile and make sure you can compile without errors. To compile first open the console from View > Console. This will also you to see the build output messages. Then click the build button



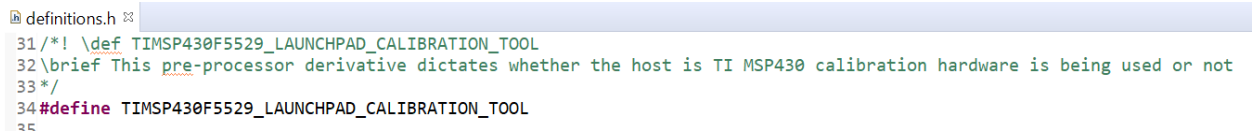
The process will likely take at least several minutes and up to ten minutes. Ensure there are no errors as shown below



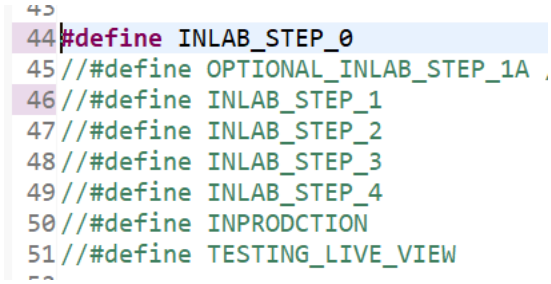


SDK setup

4. Use the OPT3101 Configurator tool to generate the correct configuration for your board. Replace the default OPT3101_configuration.cpp file with you generated one.
5. Make sure **#define TIMSP430F5529_LAUNCHPAD_CALIBRATION_TOOL** is in your definitions.h file to set the SDK in interactive mode for Calibration.



6. main.cpp also has several new **#defines** such as INLAB_STEP_1, INLAB_STEP_2, etc. These can be uncommented one by one to run the calibrationSession_firstTimeBringUp(), calibrationSession_perDesignCalibrationCrosstalkTemp(), and other functions. Select the appropriate one. For new designs step 0 will suffice to start.



7. Compile the project in code composer studio. The firmware can be transferred to the MSP430F5519 launchpad using the following command.
    1. MSP430Flasher.exe –w <firmware.txt> -v –e ERASE_MAIN

Once firmware is loaded on the MSP430F5529 launch pad, the Launchpad becomes a calibration hardware interface.

Additional details:

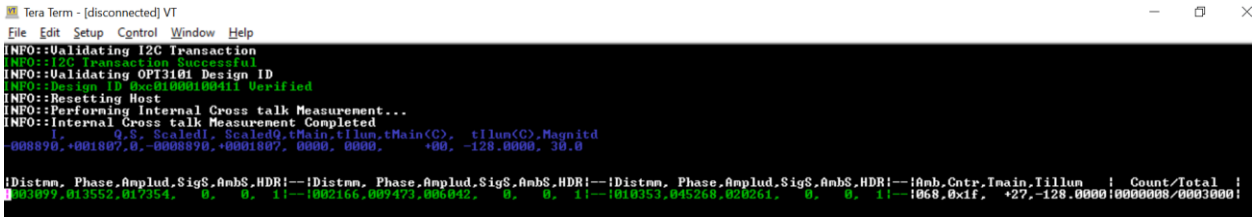
a. Download tool from http://www.ti.com/tool/MSP430-FLASHER and install to C:\ti\MSPFlasher_1.3.8
b. Open command prompt and run

cd C:\ti\MSPFlasher_1.3.8

MSP430Flasher.exe –w path_to_sdk \OPT3101_SDK\Debug\OPT3101_SDK.txt -v –e ERASE_MAIN

Where path_to_sdk needs to be filled in to match where code composer studio is storing the SDK project.

8. Wire the Launchpad to the OPT3101 using the pin mapping for the Launchpad shown below:
    1. P4.1 -> I2C Data
    2. P4.2 -> I2C Clock
    3. P2.0 → RSTZ_MS
    4. 5V, 3.3V Power and GND pins are marked on the Launchpad silkscreen.
9. Plug the Launchpad into a PC USB port. With the firmware loaded, the Launchpad will enumerate as a COM port on the host PC. A COMport terminal @9600 can be used to interact with the Launchpad hardware. Multi-color log message from calibration firmware can be viewed on the host and the firmware in some cases will wait from a key press from user to continue.



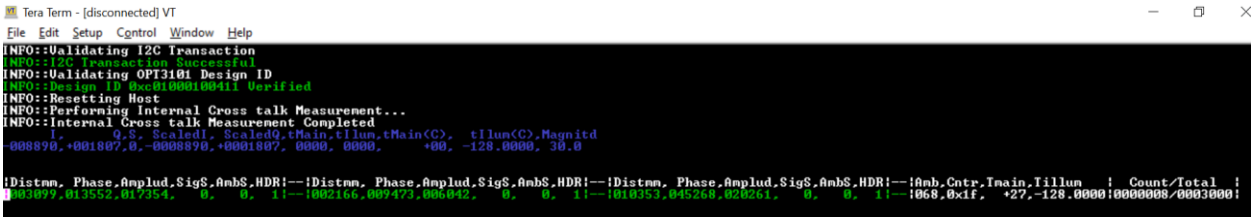
Basic testing

Step 0: Measure raw crosstalk on the board

IN_LAB_STEP_0 is for validating the quality of PCB design and layout and any optical or electrical shielding. Best raw crosstalk will be <200 codes. Okay crosstalk is <600 codes. Crosstalk greater than 843 codes requires using the force_scale feature of the OPT3101.

1. Enable IN_LAB_STEP_0, compile and program the launchpad
2. Mask the photodiode or use another method to ensure no light is entering the photodiode from the LED. See the how to calibrate doc mentioned in the introduction for more details here. Below is an example of masking the photodiode of the single channel OPT3101EVM with black electrical tape. See the [How to set up and calibrate OPT3101 based systems] document for more details on masking photodiode.

3. Plug in the launchpad and connect using a COM port terminal. Press enter on the keyboard to run the code on the device. Step 0 configures the device, corrects for electrical crosstalk, and runs a live view. The distance data is not useful without running step 1, but the amplitude measures the raw uncorrected crosstalk. Below the output is shown for a failing three channel board. You can see that the amplitude measured on all three channels is too high. This either means the photodiode is not masks correctly and light is leaking in (tape is not IR blocking,



Step 1A: Optional Additional Crosstalk Correction

Skip this step unless issues are seen with data output getting stuck due to high crosstalk on a single channel after running through step 1 and testing. Instructions for this step are in a separate document and included with the SDK files..

Step 1: Simple Bring-up

Simple bring-up corrects for crosstalk and phase offset. Ensure all channels have crosstalk less than 843 codes before running this step or force scale is used to reduce the effective crosstalk seen by the device. Crosstalk is correct with the photodiode masked. Phase offset allows the device to report the correct distance reading by setting the device at a known distance (defined in the C++ code) from a target.

1. The phase offset calibration in this step requires the target to be set an a specified distance. These distances are defined for each channel and HDR setting of the OPT3101 in the OPT3101Coefficients.cpp file as shown below.

OPT3101Coefficients.cpp

```
46
47 void environmentalController::manuallySetReferenceDistances(){
48     this->refDistancesInMM[0][0]=10;
49     this->refDistancesInMM[0][1]=20;
50     this->refDistancesInMM[1][0]=80;
51     this->refDistancesInMM[1][1]=120;
52     this->refDistancesInMM[2][0]=0;
53     this->refDistancesInMM[2][1]=0;
54 }
```

Channels and HDR settings are distinguished as *refDistancesInMM[TX channel][HDR setting]*

In this example the distances are defined for a single channel board using super HDR so there are 4 distances set for the increasing LED power and range. These values can be edited to match the configuration of the device. See the How to set up and calibrate OPT3101 based systems document for more details on phase offset correction and setting up the target at a known distance.

2. Enable IN_LAB_STEP_1, compile and program the launchpad
3. Plug in the launchpad and connect using a COM port terminal. Press enter on the keyboard to run the code on the device.
4. Device will prompt user to mask photodiode similar to step 0. After masking and pressing enter the crosstalk will be corrected.
5. Device will prompt user to set the OPT3101 facing a target at a known distance. See the How to set up and calibrate OPT3101 based systems document for details on setting up the target at a known distance.

System level coefficients

If you reach this point you have completed basic power up and testing on a new design. The following steps cover system level calibration on a subset of boards. These coefficients allow the device to give stable distance readings over changes in environmental temperature and infrared light.

In these steps a slope is determined to fit the change in a device reading (crosstalk or phase) with an environmental factor (temperature or ambient light). The data for this calculation is logged to the COM port terminal in csv format. The user is expected to take the output and paste in excel, plot in excel or another program and calculate the coefficients from the slope or slopes, and update the coefficients in the OPT3101Coefficients.cpp file.

The types of data to plot and the type of fit is shown below. See the How to set up and calibrate OPT3101 based systems document for more details.

- Illumination crosstalk temperature coefficient
    - Two simple y=mx+b slopes values. The offset b is ignored as it is covered by the single point crosstalk calibration step in simple bring-up and factory calibration steps.
        - I_PHASE illumination crosstalk reading vs temperature in codes
        - Q_PHASE illumination crosstalk reading vs temperature in codes
- Phase temperature coefficient
    - One simple y=mx+b slope value.

        - Phase reading vs temperature in codes
    - The offset b is ignored as it is covered by the single point phase offset calibration step in simple bring-up and factory calibration steps.
- Phase ambient coefficient
    - Unlike the other two coefficients, phase ambient is not a linear relationship so a piece-wise linear fit is required. This fit uses 4 slope values and 3 knee points that define where each slope is are used.
        - Each of the 4 slopes is phase reading vs ambient codes

Step 2: Crosstalk temperature coefficient

1. Set temperature sweep settings in definitions.h

definitions.h

```
63
64#define TEMP_CYCLE_DELAY_IN_SECONDS_BETWEEN_DATA_POINTS 1
65#define TEMP_CYCLE_TOTAL_NUMBER_OF_DATA_POINTS_PER_SETTING 1000
```

   This defines how long the device will wait between each data points and the total number of data points to take. This allows the data collection to be adapted to the heating/cooling rate and total time needed to log data.
2. Enable IN_LAB_STEP_2, compile and program the launchpad.
3. Place device in a thermal chamber.
4. Plug in the launchpad and connect using a COM port terminal. Press enter on the keyboard to run the code on the device.
5. Device will prompt to mask the photodiode as in the simple bring-up step.
6. Device will prompt to set the thermal chamber to 70 degrees. This is just a guideline. The displayed value does not affect data collected. It can be changed as shown below in the OPT3101_Calibration.cpp file if desired. Set the chamber temperature to 70 degrees.

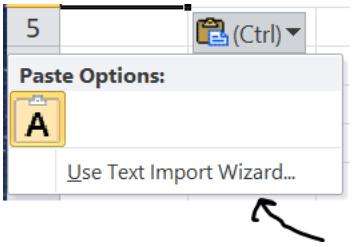
OPT3101_Calibration.cpp

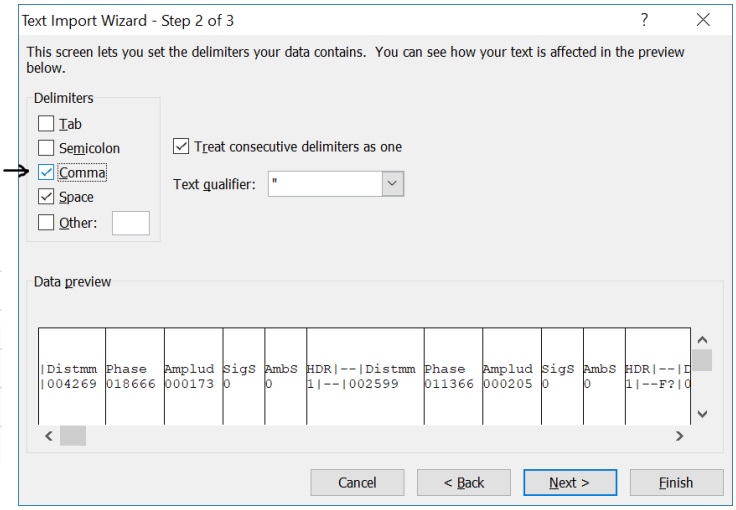
```
237void OPT3101::device::calibrationSession_perDesignCalibrationCrosstalkTemp() {
238    uint8_t c0, c1;
239    uint16_t count;
240    /// <b>Algorithm of the method is as follows</b>
241    OPT3101::crosstalkC illumXtalk; ///* Declares temporary variable of OPT3101::crosstalk(
242    this->reset(); ///* Resets the device calling OPT3101::device::reset method
243    this->initialize(); ///* Initializes the OPT3101 device by calling OPT3101::device::in
244    this->measureAndCorrectInternalCrosstalk(&this->calibration->internalCrosstalk[0]); //
245    envController.setTargetToInfinity_OR_coverPhotodiode(); ///* Calls the method environm
246    envController.setChamberTemperature(70); ///* Calls the method environmentalController
```

7. Once the temperature is reached turn off the chamber. The device should start to self-cool. Wait 1-2 minutes for the device to thermally stabilize. Then press enter to start data collection. The chamber door can be cracked open slightly to increase cooling rate before pressing enter if

chamber is too well insulated and faster cooling is required. This cooling step during data collection usually takes 15 minutes to an hour.

8. After collection finishes or devices reaches a stable near ambient temperature (~40 degrees or less is okay) copy the data and paste into excel. Note: Use the paste import wizard when pasting into excel. Select comma delimited.





9. Plot and find the slope for
    a. I_PHASE illumination crosstalk reading vs tmain
    b. Q_PHASE illumination crosstalk reading vs tmain

These slope coefficients are determined for each TX channel and HDR mode. All this data is outputted in one run.

10. Write the coefficients into OPT3101Coefficients.cpp as shown below. If a channel or HDR mode is not used it can be left at 0 to disable the calibration.

OPT3101Coefficients.cpp

```
29
30 void OPT3101::device::manuallySetIllumCrosstalkTempCoffs(){
31
32      ///* Units for coefficient: ScaledIorQ/tempRegister/MagnitudeCalc
33     this->calibration[0].illumCrosstalkTempCoff[0][0].coffI=0.124734811687;
34     this->calibration[0].illumCrosstalkTempCoff[0][0].coffQ=0.13592223628;
35     this->calibration[0].illumCrosstalkTempCoff[0][1].coffI=0.121257044071;
36     this->calibration[0].illumCrosstalkTempCoff[0][1].coffQ=0.0290795903941;
37     this->calibration[0].illumCrosstalkTempCoff[1][0].coffI=-0.000172933230344;
38     this->calibration[0].illumCrosstalkTempCoff[1][0].coffQ=-0.0163536237361;
39     this->calibration[0].illumCrosstalkTempCoff[1][1].coffI=-0.010450043412;
40     this->calibration[0].illumCrosstalkTempCoff[1][1].coffQ=-0.0910940906909;
41     this->calibration[0].illumCrosstalkTempCoff[2][0].coffI=0.0;
42     this->calibration[0].illumCrosstalkTempCoff[2][0].coffQ=0.0;
43     this->calibration[0].illumCrosstalkTempCoff[2][1].coffI=0.0;
44     this->calibration[0].illumCrosstalkTempCoff[2][1].coffQ=0.0;
45 }
46
```

11. It is suggested to test that the coefficients work. This can be done by running the live view with calibration loaded by enabling `#define TESTING_LIVE_VIEW`

Step 3: Phase temperature coefficient

Repeat the same steps from the above step using IN_LAB_STEP_3 instead of IN_LAB_STEP_2. Instead of masking the photodiode the device needs to be setup with a target at the same known distance from the phase offset step in simple bring-up. Since each TX channel and HDR setting can have a different target distance this step needs to be run TX channels * HDR settings per channel times. Each time it will prompt for setting target distance and then setting temperature. Each time data will be printed giving a number of outputs needing to be pasted to excel and plotted.

1. For each channel and HDR setting plot and find the slope for
    a. Phase reading vs tmain
2. Write the coefficients into OPT3101Coefficients.cpp as shown below. If a channel or HDR mode is not used it can be left at 0 to disable the calibration.

OPT3101Coefficients.cpp

```
53      this->refDistancesInMM[2][1]=0;
54  }
55
56  void OPT3101::device::manuallySetPhaseTempCoffs(){
57      ///* Units for coefficient: Phase/tempRegister
58      this->calibration[0].phaseTempCoff[0][0].coff=3.1183945334;
59      this->calibration[0].phaseTempCoff[0][1].coff=3.91217820975;
60      this->calibration[0].phaseTempCoff[1][0].coff=3.25081198763;
61      this->calibration[0].phaseTempCoff[1][1].coff=2.73744376613;
62      this->calibration[0].phaseTempCoff[2][0].coff=0.0;
63      this->calibration[0].phaseTempCoff[2][1].coff=0.0;
64      this->calibration[0].phaseTempCoff[0][0].istMainCoff=true;
65      this->calibration[0].phaseTempCoff[0][1].istMainCoff=true;
66      this->calibration[0].phaseTempCoff[1][0].istMainCoff=true;
67      this->calibration[0].phaseTempCoff[1][1].istMainCoff=true;
68      this->calibration[0].phaseTempCoff[2][0].istMainCoff=true;
69      this->calibration[0].phaseTempCoff[2][1].istMainCoff=true;
70  }
```

3. It is suggested to test that the coefficients work. This can be done by running the live view with calibration loaded by enabling `#define TESTING_LIVE_VIEW`

Step 4: Phase ambient coefficient

Repeat the same steps from the above step using IN_LAB_STEP_4 instead of IN_LAB_STEP_3. The device also needs to be setup with a target at the same known distance, but this calibration is only needed on one combination of TX channel/HDR setting since it is a characteristic of the photodiode. Device can be removed from the thermal chamber for this test. A strong halogen lamp is required to create the ambient light. See the how to calibrate doc for more details.

1. Update temperature sweep settings in definitions.h. These setting while called temperature are also used for the ambient test time between points and total number of points.

definitions.h

```
63
64  #define TEMP_CYCLE_DELAY_IN_SECONDS_BETWEEN_DATA_POINTS 1
65  #define TEMP_CYCLE_TOTAL_NUMBER_OF_DATA_POINTS_PER_SETTING 1000
```

2. Device will prompt to set distance. Set the target at this distance.
3. Press enter to start collecting data.
4. Use the halogen lamp to sweep the ambient levels. Starting far away and slowly moving closer allows the full range of ambient levels to be covered. See how to calibrate doc for more details.
5. Paste data into excel and save as a CSV.
6. Apply a piecewise linear fit to the data. The SDK files contain a python script that can be run to determine these coefficients. Instruction for using the script are shown below.

a. The script requires numpy, pandas, matplotlib (for generating output plot only), scipy, and pwlf libraries. Install if needed.
b. The data should first be plotted and 4 separate regions of the plot identified where 4 lines are be drawn to fit the data well. The 3 break points are manually selected using this method for the best fit and inputted into the python script as x0, x1, x2 as shown below.

```
16
17 ▼ def pwlFit(x,c0,c1,c2,c3):
18       x0=15
19       x1=40
20       x2=90
21       return np.piecewise(x,
```

c. The python script should be placed in the same directory as the data. If the data is named “phase_ambient.csv” the filename can be left as in. If another directory is required this can filename can be modified

```
 9
10   df = pd.read_csv("phase_ambient.csv")
11   dataX = df.ambient.values.astype(np.float)
12   dataY = (df.phase - df.phase.max()).values.astype(np.float)
13
```

d. After running the coefficients are displayed to the console. An output plot is also generated to visualize the fit.

7. Write the coefficients and split points into OPT3101Coefficients.cpp variables shown below.

```
72 void OPT3101::device::manuallySetPhaseAmbientCoffs() {
73     ///* <b>Warning:</b> User is expected to curve fit the phase amb
74     this->calibration[0].phaseAmbientCoff[0].coff[0] = 0.0; ///* Use
75     this->calibration[0].phaseAmbientCoff[0].coff[1] = 0.0; // Set t
76     this->calibration[0].phaseAmbientCoff[0].coff[2] = 0.0; // Set t
77     this->calibration[0].phaseAmbientCoff[0].coff[3] = 0.0; // Set t
78     this->calibration[0].phaseAmbientCoff[0].splitsReg[0] = 0; // Se
79     this->calibration[0].phaseAmbientCoff[0].splitsReg[1] = 0; // Se
80     this->calibration[0].phaseAmbientCoff[0].splitsReg[2] = 0; ///*
81 }
82
```

Production Calibration

Step 5: Production Calibration

Production calibration is similar to simple bring-up except that the all the system level coefficients are loaded to the device as well. Further after completing this step all calibration values will be written into the EEPROM connected to the OPT3101 if one exists and was configured using the configurator tool. This step is meant to be run in a factory setting during production. Modifications to the code will likely be necessary to streamline the process and integrate with the factory flow.