Real-Time Solar Power Forecasting Using a Cloud Motion Vector System

A Major Qualifying Project submitted to the faculty of Worcester Polytechnic Institute in partial fulfilment of the requirements for the degree of Bachelor of Science in cooperation with Eversource Energy.

Natale, Stephen
Sauter, Evan
Ferreira, Jonathan
Lewis, Timothy

Under the advisement of Professor Maqsood Ali Mughal

December 17, 2021

Abstract

A cloud motion vector system that was designed previously by WPI students was analyzed and improved. In the process, many expensive components were made obsolete, and additional features were introduced. A custom PCB was designed to facilitate the use of RJ45 connectors and ethernet cables to connect the sensors to the microcontroller, resulting in a much easier setup, transportation, and storage of the CMVS. Large steps were made towards the creation of a commercially-viable product, but there is still much testing that needs to be done to ensure stable and accurate operation.

Acknowledgements

Special thanks to Professor Maqsood Mughal for his thoughtful advisement and guidance, and for his continued dedication to advancing renewable energy research. I would also like to express deep and heartfelt gratitude to professor Isabella Stroe for her leadership of the Renewable Energy Materials Scholars STEM program, which has provided me with helpful resources and mentorship throughout my studies at WPI. Thanks also to Eversource Energy for taking interest in and sponsoring this project, which has enabled a great deal of additional opportunity and flexibility while conducting research, designing, and improving the cloud motion vector system that is the heart of this project.

Contents

Table of Contents i

List of Figures . . . iii

Introduction 1

Background 1

Growth of Photovoltaics . . . 1

Photovoltaic Limitations . . . 2

Cloud Motion Vector System . . . 3

Original Design . . . 3

Physical System Design . . . 4

Methodology 6

Identifying Design Aspects in Need of Improvement . . . 6

Planning Changes . . . 6

Accuracy Testing . . . 7

Results 8

Improvements in MATLAB Code . . . 8

Microcontroller and Firmware Improvements . . . 9

PCB Design and Connectors . . . 10

Conclusion 14

Future Work 14

References 15

Appendix A: MATLAB Code Listings 16

calculate lux.m . . . 16

ensure valid sample.m . . . 16

figure setup.m . . . 17

get cmv direction.m . . . 18

get cmv speed.m . . . 19

get csd.m . . . 20

get matrix.m . . . 22

get norm.m 22
get quadrant.m 22
get resultant vec.m 23
get sample range.m 23
get vid.m 23
main.m 24
read cmv data.m 30
Sensor.m 31
text wrapper.m 32
thing speak test.m 32

Appendix B: Arduino Code Listings **35**
cmvs sw wifi.ino 36
constants.h 37
secrets.h 38
led codes.h 39
led codes.cpp 40
tsl softwire.h 42
tsl softwire.cpp 44
wifi setup.h 49
wifi setup.cpp 50

Appendix C: Datasheets **52**
Arduino Nano 33 IoT 53
TSL2591 Light-to-Digital Converter 71
TCA9548A Low-Voltage 8-Channel I2C Switch with Reset 104

List of Figures

1 States with the highest percent of net power generated from solar sources. (Feldman, Wu, and Margolis 2021) 2

2 PV system pricing by year of installation. (Feldman, Wu, and Margolis 2021) 3

3 (Moreira et al. 2021) light sensor data in CSV form. 4

4 (Moreira et al. 2021) Method 1 – linear cloud edge. 4

5 (Moreira et al. 2021) Method 2 – image edge detection. 4

6 (Moreira et al. 2021) Old CMVS design set up for testing. 5

7 (Moreira et al. 2021) Method 1 – linear cloud edge. Shaded areas indicate the direction of the cloud 5

8 (Moreira et al. 2021) Old CMVS design as stored. Directly soldered connections resulted in significant tangling. 5

9 (Moreira et al. 2021) Old CMVS design block diagram 5

10 In the new implementation of the firmware, data for all nine sensors is uploaded as a string to a single ThingSpeak channel (see `thing_speak_test.m` for an example of how this code is imported into MATLAB). One benefit of using a string is that the data can be padded with spaces so that columns are separated and can more easily be read. 8

11 Improved Histogram to show both methods at once and show projection lines. As r-axis is intended to show probability it has been normalized to one. Compare with Figure 4 and Figure 5 to see that the same data is shown. 9

12 Block diagram of RJ45 connector implementation. Compare with Figure 9 to see the complete simplification of the system. 10

13 The old (a) and new(b) versions of the TSL2591 breakout boards made by Adafruit 11

14 PCB designs for new CMVS system 12

15 Indoor and outdoor testing arrangements 13

Introduction

Photovoltaic (PV) systems are continually being built to replace traditional generators of power, and an increased reliance on solar energy will come with new challenges to ensure continuous service even in times of low sunlight.

As the installation cost of PV components continues to drop and the need to adopt renewable energy becomes more pressing, solar panels will become a crucial part of power grid infrastructure. Large-scale deployments of solar arrays will begin to bear more of the load that is currently generated from nonrenewable sources like natural gas or coal, and entire cities and towns will rely upon solar energy for their power needs.

The adoption of new power-generating infrastructure will come hand and hand with the adoption of a new set of problems. PV systems require direct sunlight to maximize their power output, and as a result, the shadows cast by clouds can result in sudden drops of output power. These drops could result in interruptions of service if they occur during times of high or peak demand.

Generators that utilize traditional sources of energy can be implemented to cover the loss of output associated with blocked sunlight, but these generators are not able to start instantly – it is necessary to begin the startup process a short period of time prior to when the generator needs to bear the load on the grid.

This research aims to continue the development of an inexpensive and portable cloud motion vector system (CMVS) that can provide real-time information about local cloud movements by utilizing a cluster of light sensors and a microcontroller. The CMVS is aimed to take the place of pyranometer-based nowcasting systems, which come at a greater expense and slower update speeds.

Background

Growth of Photovoltaics

In the Fall 2021 Solar Industry Update, Feldman, Wu, and Margolis (2021) found that the median installation cost of utility-scale photovoltaic (UPV) projects decreased by 74% between 2012 and 2020, dropping to a cost of \$1.42/W_{AC} (\$1.05/W_{DC}) and continues to trend downward (Fig. 2). Feldman, Wu, and Margolis also found that five states in the United States – California, Massachusetts, Hawaii, Nevada, and Vermont – generated between 14.9% and 24.3% of their total net generation of power from solar sources, with a vast majority being generated via photovoltaic (PV) technology (Fig. 1.

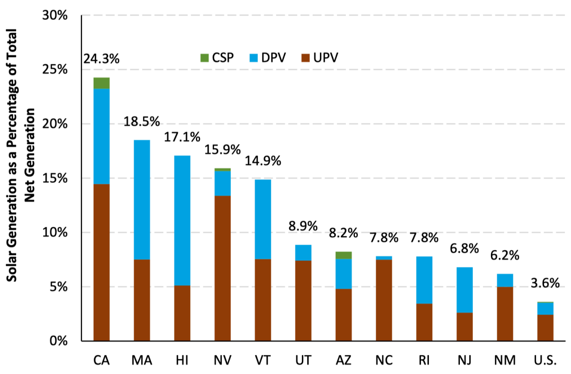


Figure 1: States with the highest percent of net power generated from solar sources. (Feldman, Wu, and Margolis 2021)

Photovoltaic Limitations

As falling costs lead to increased integration of large-scale PV systems into the power grid, reliance on solar power will increase. While this is an encouraging trend in terms of slowing climate change, the limitations of PV generation will become a much more significant issue as it continues to replace non-renewable sources.

The biggest disadvantage PV systems have over other methods of generation is that their performance is dependent on unobstructed access to sunlight. A cloud passing over a PV array could cause a significant drop in output, which would be especially problematic during times of peak demand, as it might require grid operators to shed some of the load if demand exceeds supply.

Load shedding can take the form of brownouts or rolling blackouts and is done intentionally to prevent damage to the grid. When considering PV generation, a secondary and sunlight-independent source of power is necessary to ensure that drops of power from cloud cover do not necessitate load shedding. Two common alternative sources of power are:

1. **Grid storage batteries** store excess power during times of low demand and then output power during times of high demand. Batteries a quick source of dispatchable power and do not need much time in advance to cover excessive demand. The downsides of battery storage are high cost and limitations on capacity.

2. **Standby generators** create power using traditional nonrenewable sources during times of high demand and are in a mostly idle state otherwise. Standby generators are based on mature technology and are more likely to already be in place. The most pertinent downside to standby generators is that they are relatively slow to start, but they also suffer from maintenance costs and pollute the environment.

In the near-future it stands to reason that standby generators will frequently be chosen over battery banks, especially for

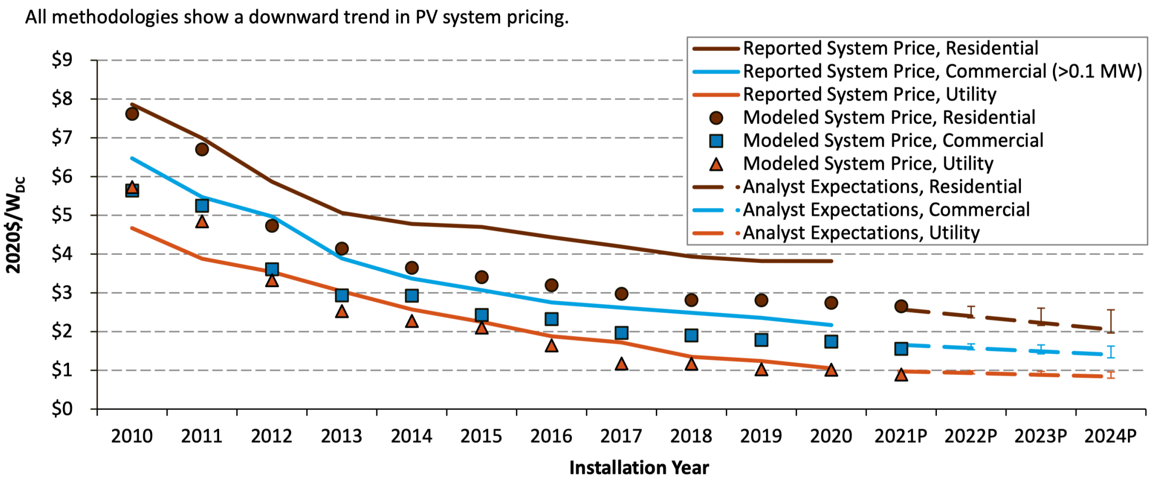


Figure 2: PV system pricing by year of installation. (Feldman, Wu, and Margolis 2021)

large-scale generation needs. As standby generators take a short amount of time to transition from idle to active, it is necessary to anticipate and predict drops in PV supply. If predictions are able to accurately assess losses of power output, unnecessary standby activations could also potentially be prevented. Minimizing unneeded activations would serve to reduce wear and tear and by extension maintenance costs.

Cloud Motion Vector System

A cluster of ambient light sensors could potentially serve as an inexpensive way to monitor local irradiance conditions in real-time. Strategically placed, this cluster of sensors could be controlled by an algorithm that utilizes the relative position of the sensors to each other along with their readings to calculate the direction and speed of a cloud passing overhead. This information could then be uploaded so that the information could be accessed remotely.

This system of detection, referred to hereafter as a cloud motion vector system (CMVS), is a design that has been worked on by previous teams of students at Worcester Polytechnic Institute. The current team set out to improve that design, put it through on-site testing at an Eversource location, and investigate the commercial viability of the CMVS design. As of writing this report, the project is still in progress.

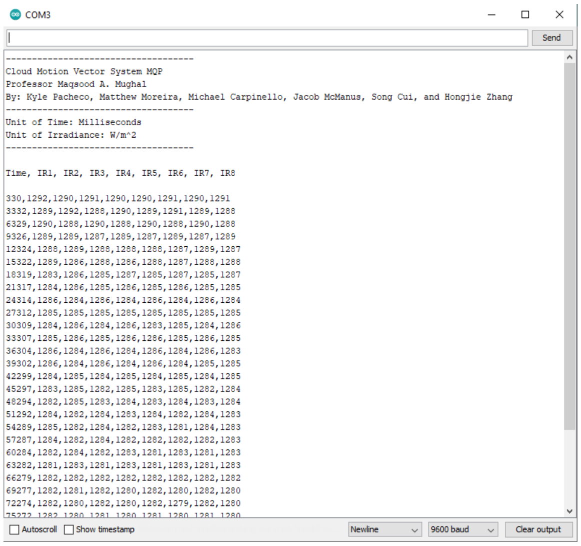
Original Design

Firmware and Algorithm

Moreira et al. (2021) stated that two methods were utilized to predict cloud motion: The linear cloud edge method (Bosch and

Kleissl 2013), and the image edge detection (Gonzalez and Woods 2008).

That data that was input to the algorithm was taken from the serial output of the microcontroller and saved in the form of a CSV file with the data from each sensor being set into individual columns.



```
------------------------------------
Cloud Motion Vector System MQP
Professor Maqsood A. Mughal
By: Kyle Pacheco, Matthew Moreira, Michael Carpinello, Jacob McManus, Song Cui, and Hongjie Zhang
------------------------------------
Unit of Time: Milliseconds
Unit of Irradiance: W/m^2
------------------------------------


Time, IR1, IR2, IR3, IR4, IR5, IR6, IR7, IR8

330,1292,1290,1291,1290,1290,1291,1290,1291
3332,1289,1292,1288,1290,1289,1291,1289,1288
6329,1290,1288,1290,1288,1290,1288,1290,1288
9326,1289,1289,1287,1289,1287,1289,1287,1289
12324,1288,1289,1288,1288,1288,1287,1289,1287
15322,1289,1286,1288,1286,1288,1287,1288,1288
18319,1283,1286,1285,1287,1285,1287,1285,1287
21317,1284,1286,1285,1286,1285,1286,1285,1285
24314,1286,1284,1286,1284,1286,1284,1286,1284
27312,1285,1285,1285,1285,1285,1285,1285,1285
30309,1284,1286,1284,1286,1283,1285,1284,1286
33307,1285,1286,1285,1286,1286,1285,1286,1285
36304,1286,1284,1286,1284,1286,1284,1286,1283
39302,1286,1284,1286,1284,1286,1284,1285,1285
42299,1284,1285,1284,1285,1284,1285,1284,1285
45297,1283,1285,1282,1285,1283,1285,1282,1284
48294,1282,1285,1283,1284,1283,1284,1283,1284
51292,1284,1282,1284,1283,1284,1282,1284,1283
54289,1285,1282,1284,1282,1283,1281,1284,1283
57287,1284,1282,1284,1282,1282,1282,1282,1283
60284,1282,1284,1282,1283,1281,1283,1281,1283
63282,1281,1283,1281,1283,1281,1283,1281,1283
66279,1282,1282,1282,1282,1282,1282,1282,1282
69277,1282,1281,1282,1280,1282,1280,1282,1280
72274,1282,1280,1282,1280,1282,1279,1282,1280
75272,1282,1280,1281,1280,1281,1280,1281,1280
```

Figure 3: (Moreira et al. 2021) light sensor data in CSV form.

These methods produced the histograms shown in Fig. 4 and Fig. 5 to show probabilities of cloud motion. The shaded areas of these figures are representative of the clouds direction of movement.

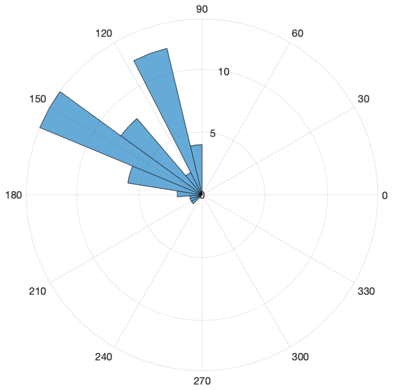


Figure 4: (Moreira et al. 2021) Method 1 – linear cloud edge.

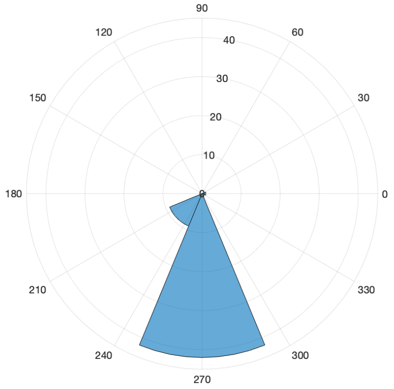


Figure 5: (Moreira et al. 2021) Method 2 – image edge detection.

Physical System Design

The physical design consisted of nine TSL2591ambient light sensors (ALS) – one centered, eight arranged in a circle with a radius of ten feet, and each sensor placed 45°apart (Fig. 6). Two TCA9548AI2C multiplexers[1] were used in conjunction with an Arduino Uno Rev.3 to connect to

[1] Multiplexers were included due to the TSL2591ALS components all sharing a fixed I2C address

and get readings from the sensors.

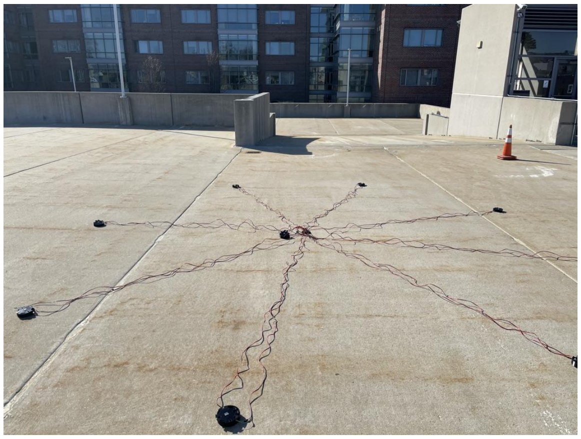


Figure 6: (Moreira et al. 2021) Old CMVS design set up for testing.

Connections were made by directly soldering 18-AWG stranded wire to the leads of the sensors and to other wires (Fig. 7). This style of connection resulted in significant tangling when the array was in storage, as can be seen in Fig. 8.

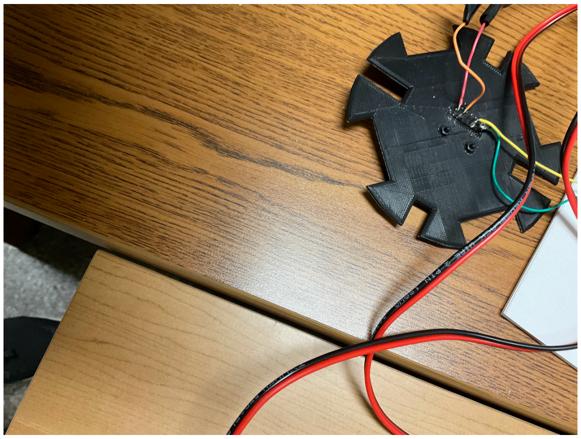


Figure 7: (Moreira et al. 2021) Method 1 – linear cloud edge. Shaded areas indicate the direction of the cloud

A BeagleBone Black was then connected to the Arduino and to a computer via USB, and also connected to the internet via an ethernet cable. The BeagleBone Black was used to monitor the serial stream of readings from the Arduino and store the results into a CSV file on the computer. The data readings were also uploaded to ThingSpeak – an IoT subscription service offered by MathWorks. The block diagram for this design can be seen in Fig. 9.

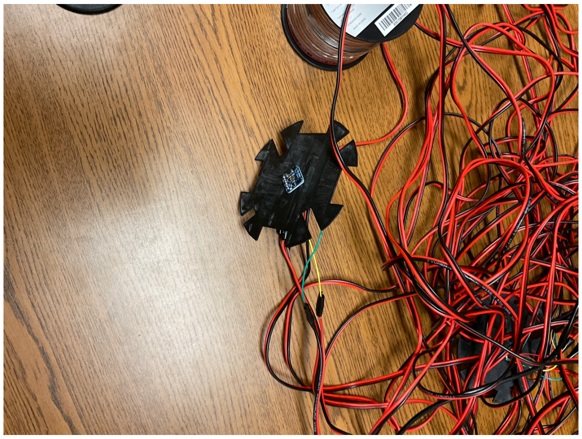


Figure 8: (Moreira et al. 2021) Old CMVS design as stored. Directly soldered connections resulted in significant tangling.

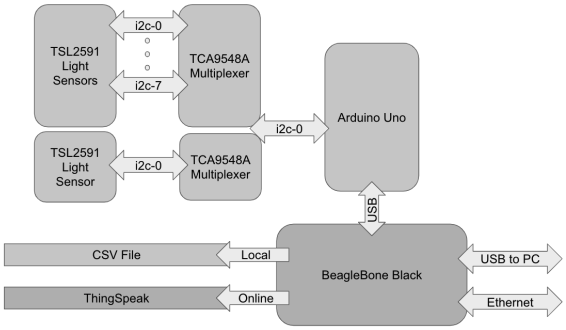


Figure 9: (Moreira et al. 2021) Old CMVS design block diagram

Methodology

Identifying Design Aspects in Need of Improvement

The old system was taken out and studied to find problems that might make it more difficult to set up, test, maintain, and by extension commercialize. The areas which needed improvement were noted as follows:

1. MATLAB scripts
   (a) Code mostly undocumented or underdocumented.
   (b) Variables and functions not descriptively named.
   (c) Two histogram algorithms producing different results.
   (d) Histograms are not labeled and do not show numerical values for speed and direction.
2. Physical System Design
   (a) Soldering headers directly to wires created a system that was difficult to transport, put into storage, or remove from storage without it becoming severely tangled.
   (b) Only one TCA9548A multiplexer is needed to allow for 9 TSL2591 sensors to be used in the system[2].
   (c) The BeagleBone Black appeared to not really have a purpose, as it was also connected to a computer.

Other general inconsistencies were noted in what was written by Moreira et al. (2021) and the functionality of the code. Noticing this, it was deemed to be highly important to clarify the firmware and the algorithm so that it could more easily be examined to assure correct operation.

Planning Changes

A plan of action was made to create a CMVS that could easily be transported and maintained. The plan consisted of the following steps.

[2] TCA9548A multiplexers were used because the TSL2591 have a fixed I2C address of 0x29, meaning that only one sensor could be on the I2C bus at a time. The multiplexers themselves had a range of addresses available – 0x70 - 0x77 – and support 8 I2C devices each. Therefore, the SDA and SCL bus from the Arduino should be able to be connected to both a single TSL2591 and a TCA9548A, which would be connected to the eight other sensors.

1. MATLAB scripts
    (a) Create a GitHub repository of the codebase so that all changes can be tracked.

    (b) Refactor and document code to increase legibility and maintainability.

    (c) Restyle the histograms so that they clearly convey information about the movement of clouds.

2. Physical System Design
    (a) Utilize connectors instead of directly soldering wires to headers so that CMVS can be dismantled for easier storage or transport.

    (b) Utilize cables or wire sleeves to prevent connections to the outer radius of sensors from tangling.

    (c) Adjust firmware and utilize a WiFi-capable microcontroller so that sensor data can be uploaded to ThingSpeak without the need for the CMVS to be connected to neither a BeagleBone Black nor a computer. As the BegaleBone Black is the most expensive individual component, this would make the CMVS significantly cheaper.

    (d) Attempt to implement I2C protocols within the firmware rather than using hardware TCA9548A multiplexers, which would further reduce the complexity and cost of the system.

    (e) Create a custom centralized PCB so that the center is compact and secure.

Accuracy Testing

Regardless of design, an important step on the path to commercial viability would be to test the system for accuracy. The old design, while flawed, had a good framework that could be built on top of rather than scrapping completely. With a plan to upgrade the design, the best course of action was deemed to be to make small changes whenever possible and then test the system to see if it produces the same or highly similar results to the old design. Working in a step-by-step fashion would ensure that redesigning the system would not compound with the flaws that were already present.

Results

Improvements in MATLAB Code

The MATLAB code was refactored to have less repetitious code, more descriptive variables and functions, and a more consistent style (e.g. functions are written in snake case, variables are written in camel case)[3]. Additional features were added, including a progress bar that displays during lengthy processes and input/output selection prompts.

The histograms were significantly improved in a number of ways:

1. The output image consists of three tiles: A large one that contains both methods for easy comparison between the two, and then two smaller ones that show the methods individually so they can be viewed in isolation.

2. The r-axis is now normalized to fall between zero and one, as the bins of the histogram are intended to represent the probability that the cloud is moving in a certain direction.

3. The calculated speed and direction of the cloud is now included as text in the bottom right.

4. The output image is titled so that an observer is more aware of what the

[3] Full listings of the MATLAB code can be seen in Appendix A.

```
0x56f1,    0x4014,    0x38df,    0x6181,    0x4ceb,    0x2c14,    0x4446,      0x3098,    0x3fb2
0x56fd,    0x4024,    0x38e6,    0x6194,    0x4b18,    0x2c15,    0x4453,      0x30a6,    0x3fb7
0x561c,    0x4033,    0x38fb,    0x61a7,    0x4cba,    0x2c1e,    0x4463,      0x309f,    0x3fb0
0x5702,    0x4012,    0x38ee,    0x61bd,     0x8d3,    0x2c1d,    0x4458,      0x30a4,    0x3fc3
0x5706,    0x4021,    0x38f5,    0x61a6,    0x4b2c,    0x2c1e,    0x445c,      0x30ab,    0x3fb1
0x570a,    0x4020,    0x38ee,    0x6187,     0x9f2,    0x2c1a,    0x4451,      0x30a6,    0x3fc4
0x571e,    0x4011,    0x38ec,    0x6198,    0x4b8a,    0x2c29,    0x445b,      0x30ae,    0x3fc8
0x5713,    0x4022,    0x3812,    0x61b1,     0x8e8,    0x2c27,    0x4462,      0x30b5,    0x3fca
0x5703,    0x4020,    0x39fc,    0x6194,     0x952,    0x2c1d,    0x445e,      0x30b5,    0x3fd8
0x571a,    0x4032,    0x384e,    0x619a,     0x815,    0x2c2f,    0x449f,      0x30af,    0x3fe5
```

Figure 10: In the new implementation of the firmware, data for all nine sensors is uploaded as a string to a single ThingSpeak channel (see `thing_speak_test.m` for an example of how this code is imported into MATLAB). One benefit of using a string is that the data can be padded with spaces so that columns are separated and can more easily be read.

displayed data is intended to represent.

The function calculate_lux.m was written to offload resource-heavy floating-point arithmetic from the microcontroller, if so desired. The CMVS needs to be set to collect luminance data for this function to work (see the !mode! variable in cmvs_sw_wifi.ino and the available modes in constants.h).

Microcontroller and Firmware Improvements

Connectivity

With the use of a WiFi-capable Arduino Nano 33 IoT in place of the Arduino Uno Rev3 in the original design, the system was able to directly connect to the internet and no longer needed to be connected to a BeagleBone Black or a computer to operate. Additionally, there is a ThingSpeak Arduino library that allowed the uploads to be made directly from the Arduino Nano 33 IoT. With this new code, the Python and Bash scripts used in the original design were also no longer necessary. wifi_setup.cpp can be viewed to see how the connectivity to WiFi and ThingSpeak is implemented.

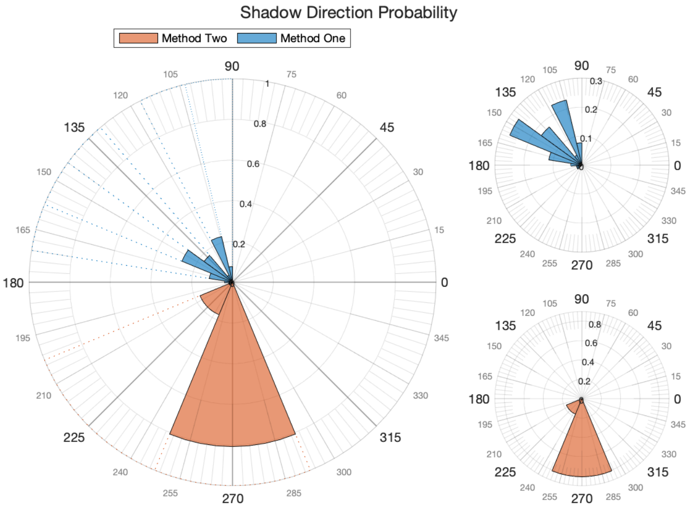


Figure 11: Improved Histogram to show both methods at once and show projection lines. As r-axis is intended to show probability it has been normalized to one. Compare with Fig. 4 and Fig. 5 to see that the same data is shown.

I2C Implementation

The firmware was also updated to use the SoftWire Arduino library to communicate with the TSL2591 sensors using a software I2C implementation rather than a hardware one.[4] This change resulted in the obsolescence of the multiplexers in the design, as the SDA and SCL pins could now be reassigned to any digital I/O pins on the Arduino Nano 33 IoT. This implementation can be viewed in `tsl_softwire.cpp`.

Observing the new block diagram of Fig. 12, it can be seen that the removal of several components has resulted in a significant simplification of the new design.

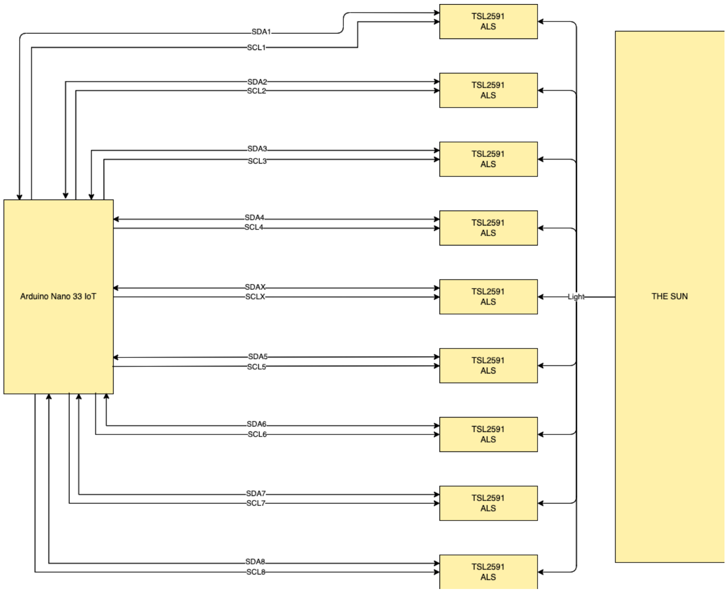


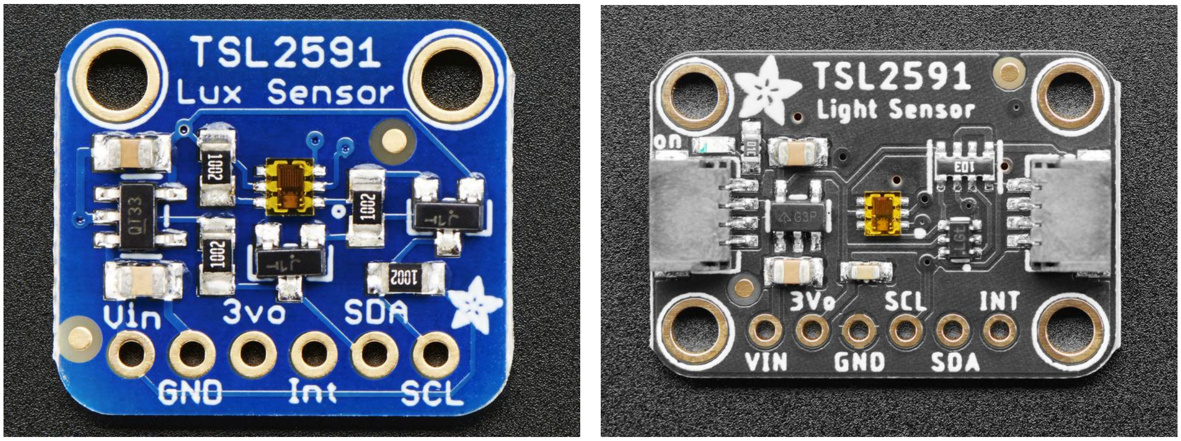
Figure 12: Block diagram of RJ45 connector implementation. Compare with Fig. 9 to see the complete simplification of the system.

PCB Design and Connectors

A printed circuit board was designed to use RJ45 connectors and ethernet cables to interface the Arduino Nano 33 IoT with the TSL2591 sensors. Ethernet cables are a good choice for this application as they are inexpensive, require no preparation or assembly to use, and they provide shielding from electrical interference if grounded. This is something that may want to be considered in the future if issues are seen in making data reads from the sensors, as the I2C protocol is not designed to be used over long distances.

Fig. 14a and Fig. 14b show the first version of the custom PCB. This board was

[4] Firmware code listings can be seen in Appendix B



(a)

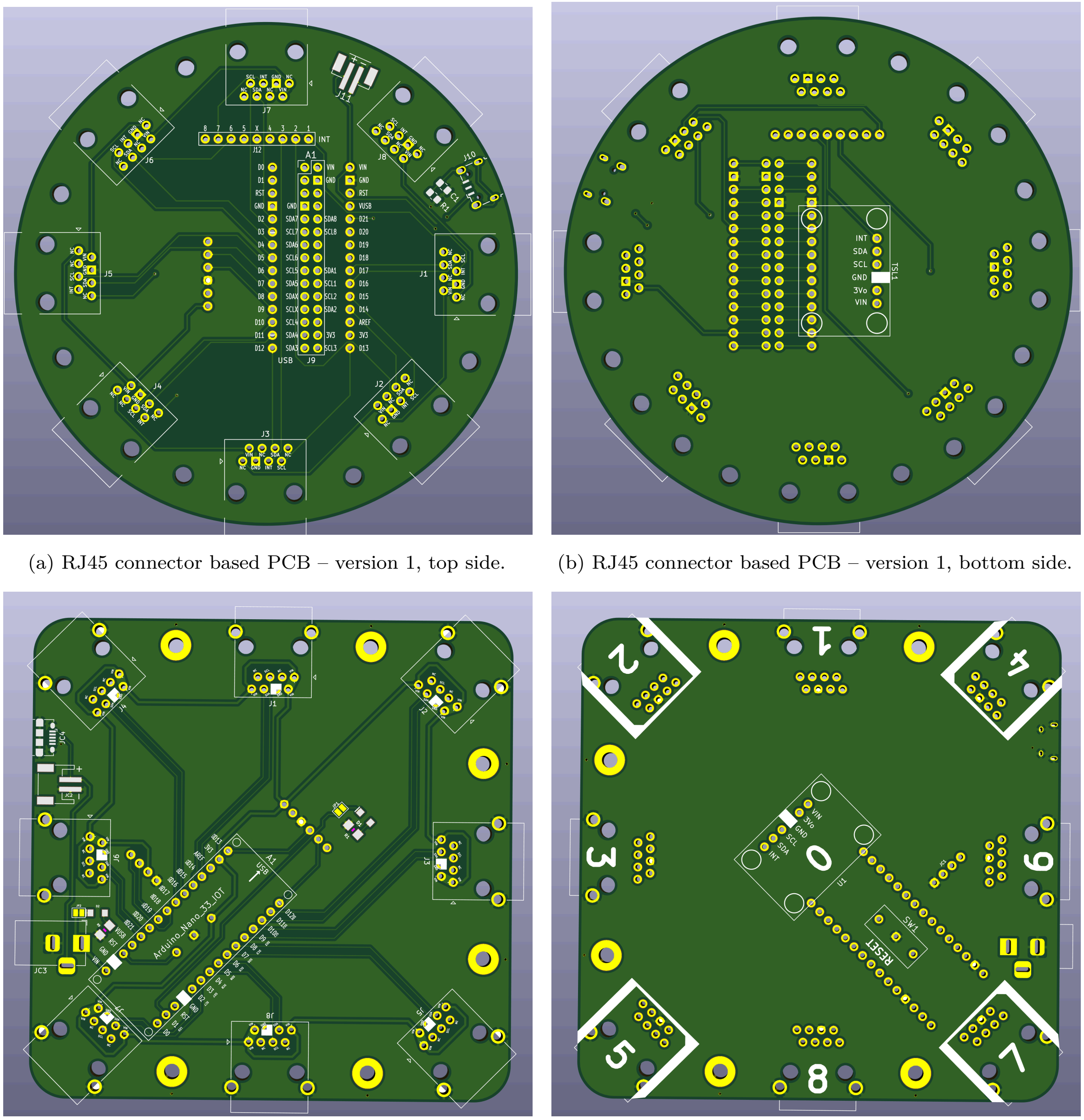
(b)

Figure 13: The old (a) and new(b) versions of the TSL2591 breakout boards made by Adafruit

designed and ordered to ensure correct operation and find unforeseen issues before the next version was designed. It should be noted that the first version uses a mixture of TSL2591 breakout boards, resulting in differences in how the RJ45 connectors are routed to the Arduino Nano 33 IoT. This choice was made due to the limited availability of the newer breakout boards (see Fig. 13).

The first version was tested and found to be somewhat crowded in layout, as it was difficult to connect the Arduino Nano 33 IoT to a computer using the USB port.The second version, which can be seen in Fig. 14c and Fig. 14d, is designed to only use the newer version of the breakout board, is more spacious, and contains additional options for power sources, external LED indicators, and an easily accessible reset switch. Fig. 15a and Fig. 15b show testing setups using version one of the PCB. The indoor setup used short ethernet cables to allow for one-person testing of the algorithm, as it is easier to block light from the sensors when they are closer together.

The outdoor setup was used to test how the system worked at the PV site located at WPI. This testing is what produced the data shown in Fig. 10, and the numerical annotations shown in brackets correspond to the columns of the output.



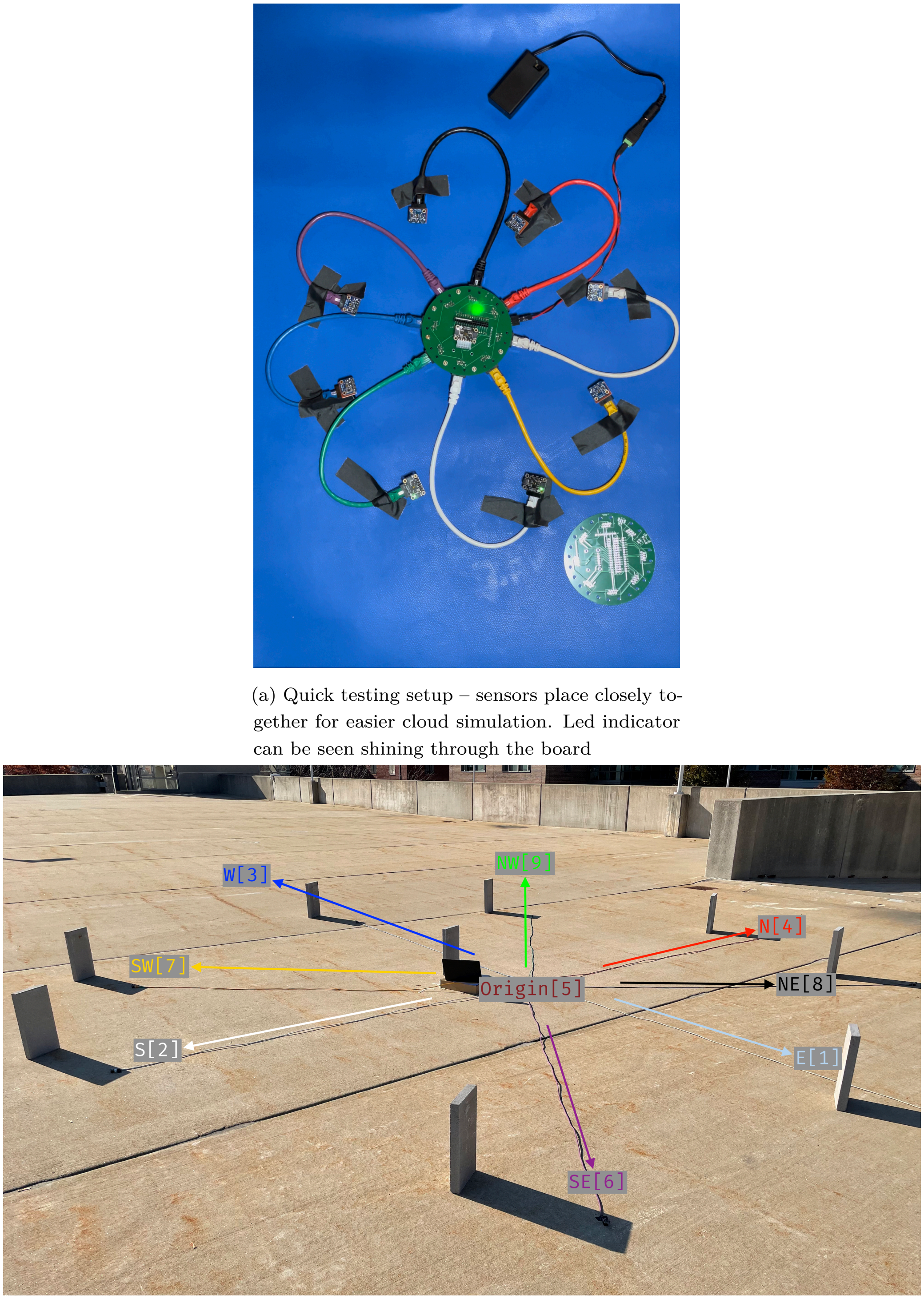
(a) RJ45 connector based PCB – version 1, top side.

(b) RJ45 connector based PCB – version 1, bottom side.

(c) RJ45 connector based PCB – version 2, top side.

(d) RJ45 connector based PCB – version 2, bottom side.

Figure 14: PCB designs for new CMVS system



(a) Quick testing setup – sensors place closely together for easier cloud simulation. Led indicator can be seen shining through the board

(b) Field testing setup – sensors place in ten-foot radius, with concrete blocks being used to simulate clouds.

Figure 15: Indoor and outdoor testing arrangements

Conclusion

The design of the array made by the research group last year contained serious flaws but provided a solid framework to build upon. Through use of this framework, a much improved design was implemented without loss of functionality. This newer design does not use a BeagleBone Black, nor any multiplexers, making the hardware layout cheaper and simpler. The simplification of the hardware design comes at a cost of increased complexity of the firmware. This is due to the software implementation of the I2C protocol. However, the design could easily be implemented to use only a single multiplexer to keep the firmware simple. Retaining a hardware implementation of I2C would likely increase portability of the design so that it can easily be used with non-arduino microcontrollers.

Future Work

Issues are still present in the MATLAB algorithms, as they were not adjusted. The two methods frequently produce predictions that are at odds with each other, so an important next step to take would be to perform further testing to find which algorithm is producing more accurate results, and if that algorithm reliably produces accurate results. One possible idea to facillitate robust testing of the system would be to design a piece of software that simulates changing light conditions. Taking this action would ensure that the system functions without error in most scenarios, and it would help to identify bugs that only occur in niche conditions.

References

[BK13] Juan Luis Bosch and Jan Kleissl. "Cloud motion vectors from a network of ground sensors in a solar power plant". In: *Solar Energy* 95 (2013), pp. 13–20.

[FWM21] David Feldman, Kevin Wu, and Robert Margolis. "H1 2021 Solar Industry Update". In: (July 2021). DOI: 10.2172/1808491. URL: https://www.osti.gov/biblio/1808491.

[GW08] Rafael C. Gonzalez and Richard E. Woods. *Digital image processing*. Upper Saddle River, N.J.: Prentice Hall, 2008, pp. 706–714. ISBN: 9780131687288 013168728X 9780135052679 013505267X. URL: http://www.amazon.com/Digital-Image-Processing-3rd-Edition/dp/013168728X.

[Mor+21] Matthew C. Moreira et al. *Cloud Motion Vector System to Monitor and Predict Output Power of a Photovoltaic System in Real Time*. Tech. rep. 100 Institute Road, Worcester MA 01609-2280 USA: Worcester Polytechnic Institute, Mar. 2021.

Appendix A: MATLAB Code Listings

The code in this section represents a refactored and more robust version of the MATLAB scripts implemented by Moreira et al. (2021). Additional work needs to be completed to ensure the validity of the algorithms that are in use to calculate cloud speed and cloud direction. Note that not all functions here are currently implemented in !main.m!, as some are auxiliary functions meant for testing or obsolete functions that are being included as a reference.

calculate lux.m

```
function [lux, luxIfBright, luxIfDim] = calculate_lux(Tsl, lum)
lumsize = size(lum);
lumel = numel(lum);
lum = uint32(lum(:))';
lum = struct(full=bitshift(lum, -16), ir=bitand(lum, 0x0000FFFF));

% Note: This algorithm is based on preliminary coefficients
% provided by AMS and may need to be updated in the future
countsPerLux = (Tsl.aTime * Tsl.aGain) / Tsl.DF;
luxIfBright   = (Tsl.COEFA * cast(lum.ir, 'single') - Tsl.COEFB * cast(lum.full,
  'single')) / countsPerLux;
luxIfDim      = (Tsl.COEFC * cast(lum.ir, 'single') - Tsl.COEFD * cast(lum.full,
  'single')) / countsPerLux;
lux       = max([luxIfBright ; luxIfDim ; zeros(1, lumel)]);

% Report overflows as -1

lux(lum.ir == 0xFFFF) = -1;
if sum(lux(lux == -1) > 0)
  warning("Overflow Detected");
end
lux = reshape(lux, lumsize);
end
```

ensure valid sample.m

```
function [DATA_WINDOW, FILTER_WINDOW, I_DATA_END, I_DATA_START] =
  ensure_valid_sample(DATA_WINDOW, data, FILTER_WINDOW, I_DATA_END, I_DATA_START, I_DELTA)
if DATA_WINDOW > length(data)
    DATA_WINDOW = length(data);
    dataWindowWarn = sprintf("DATA_WINDOW exceeds length of data and has been trimmed");
else
    dataWindowWarn = sprintf('');
end
```

```
if FILTER_WINDOW > length(data)
    FILTER_WINDOW = length(data);
    filterWindowWarn = sprintf("FILTER_WINDOW exceeds length of data and has been trimmed");
else
    filterWindowWarn = sprintf('');
end
if I_DATA_END > length(data)
    I_DATA_END = length(data);
    I_DATA_START = max(I_DATA_END - I_DELTA, 1);
    dataEndWarn = sprintf("I_DATA_END exceeds length of data. Range parameters have been changed");
else
    dataEndWarn = sprintf('');
end
if ~ismissing([dataWindowWarn, filterWindowWarn, dataEndWarn])
    warning('\n\t%s\n\t%s\n\t%s', dataWindowWarn, filterWindowWarn, dataEndWarn);
end
end
```

figure setup.m

```
function figure_setup()
  SENSOR_STRINGS = ['NW';' N';'NE'
                    ' W';' O';' E'
                    'SW';' S';'SE'];
  FIGURE_STRINGS  = ["data"
                    "dataSample"
                    "smoothSample"
                    "smoothSampleNorm"];
  SCALE = 150;
  FMT = struct;
  FMT.COLORORDER = colororder;
  FMT.AX.FontSize = 15;
  FMT.AX.FontWeight = 'bold';
  FMT.AX.YLim = [0 inf];
  FMT.POLAX.FontSize = 10;
  FMT.POLAX.FontWeight = 'normal';
  FMT.POLAX.ThetaTick = 0:15:360;
  FMT.POLAX.TickLabelInterpreter = 'tex';
  FMT.RTICKSET = @() set(gca, ...
    RTickLabel=strcat("\fontsize{7}", string(gca().RTickLabel)));

  BIN_EDGES = pi/8:pi/4:2*pi;
  POLAR_ORDER = [9 6 4 5 8 2 1 3];

  tickLabel = string(FMT.POLAX.ThetaTick);
  tickInc = round((length(FMT.POLAX.ThetaTick) - 1) / length(BIN_EDGES));
  idx = ~ismember(tickLabel(1:end-1), tickLabel(1:tickInc:end-1));
  tickLabel(idx) = strcat("\color{gray}\fontsize{7}", tickLabel(idx));
  % FMT.POLAX_DOTS = @(pax, rmax) polarplot(deg2rad(pax.BinEdges), ...
  %   repelem(rmax, length(pax.BinEdges)));
  FMT.POLAX.ThetaTickLabel = tickLabel;
  FMT.POLAX.TickLength = [0.2 0] ;
  FMT.POLAX.ThetaMinorTick = 'on' ;
  TILE.ROWS = 2;
  TILE.COLS = 3;
  TILE.LARGE_SPAN = [2 2];
  TILE.POS(1) = 1;
  TILE.POS(2) = 3;
  TILE.POS(3) = 6;
  FMT.TLO.Padding = 'compact';
  FMT.TLO.TileSpacing = 'compact';

  FMT.FIG.Units = 'Normalized';
  FMT.FIG.Visible = false;
  % FMT.FIG.OuterPosition = [0, 0.04, 0.25, 0.25];
  save figure_setup;
end
```

get cmv direction.m

```
function cmvDirection = get_cmv_direction(angleRad, histPlot, algorithm)
%% This function gets the CMV direction using algorithm
% (1) without 2 * pi wraparound or
% (2) with 2 * pi wraparound
[max_row, ~] = size(angleRad);
[~, edge_idx] = max(histPlot.Values);
edgeVals = histPlot.BinEdges;
% [~, edge_idx] = max(histo.Values);
% edgeVals = histo.BinEdges;
lowerBound = edgeVals(edge_idx);
upperBound = edgeVals(edge_idx + 1);
directionTemp = zeros(1, 1);
cnt = 1;

switch algorithm
    case 1
        % Get final CMV direction if Algorithm 2.1
        for id = 1:max_row
            if angleRad(id) > lowerBound && angleRad(id) < upperBound
                directionTemp(cnt) = angleRad(id);
                cnt = cnt + 1;
            end
        end

    case 2
        % Get final CMV direction if Algorithm 2.2
        for id = 1:max_row
            if angleRad(id) < 0.3926991
                angleRad(id) = angleRad(id) + 2 * pi;
            end

            if angleRad(id) > lowerBound && angleRad(id) < upperBound
                directionTemp(cnt) = angleRad(id);
                cnt = cnt + 1;
            end
        end
```

```
end

cmvDirection = rad2deg(mean(directionTemp));
if cmvDirection < 0
    cmvDirection = cmvDirection + 360;
elseif cmvDirection > 360
    cmvDirection = cmvDirection - 360;
end

end
```

get cmv speed.m

```
function cmvSpeed = get_cmv_speed(cmvDirection, dipLocArr)
% This function receives the CMV direction and calculates the cloud shadow
% speed from the local minima locations
theta = deg2rad(cmvDirection);

dipLocArr(dipLocArr == 0) = NaN;
v = zeros(3, 1);

% fullArr = repelem(nan, 9);
% fullArr(1:length(dipLocArr)) = dipLocArr;
fullArr = dipLocArr;
% Initialize variables
deltaT1 = 0;
deltaT2 = 0;
deltaT3 = 0;
deltaT4 = 0;
deltaT5 = 0;
%CMV_direction = ~45 degrees
if theta >= 0.3926991 && theta < 1.178097
        deltaT1 = fullArr(02) - fullArr(04); %sqrt(2) m
        deltaT2 = fullArr(03) - fullArr(07); %2m
        deltaT3 = fullArr(06) - fullArr(08); %sqrt(2) m
        deltaT4 = fullArr(03) - fullArr(05); %1m
        deltaT5 = fullArr(05) - fullArr(07); %1m
%CMV_direction = ~90 degrees
elseif theta >= 1.178097 && theta < 1.9634954
        deltaT1 = fullArr(01) - fullArr(07); %sqrt(2) m
        deltaT2 = fullArr(02) - fullArr(08); %2m
        deltaT3 = fullArr(03) - fullArr(09); %sqrt(2) m
%CMV_direction = ~135 degrees
elseif theta >= 1.9634954 && theta < 2.7488936
        deltaT1 = fullArr(02) - fullArr(06); %sqrt(2) m
        deltaT2 = fullArr(01) - fullArr(09); %2m
        deltaT3 = fullArr(04) - fullArr(08); %sqrt(2) m
        deltaT4 = fullArr(01) - fullArr(05); %1m
        deltaT5 = fullArr(05) - fullArr(09); %1m
%CMV_direction = ~180 degrees
elseif theta >= 2.7488936 && theta < 3.5342917
        deltaT1 = fullArr(01) - fullArr(03); %sqrt(2) m
        deltaT2 = fullArr(04) - fullArr(06); %2m
        deltaT3 = fullArr(07) - fullArr(09); %sqrt(2) m
```

```
%CMV_direction = ~225 degrees
elseif theta >= 3.5342917 && theta < 4.3196899
        deltaT1 = fullArr(04) - fullArr(02); %sqrt(2) m
        deltaT2 = fullArr(07) - fullArr(03); %2m
        deltaT3 = fullArr(08) - fullArr(06); %sqrt(2) m
        deltaT4 = fullArr(07) - fullArr(05); %1m
        deltaT5 = fullArr(05) - fullArr(03); %1m
%CMV_direction = ~270 degrees
elseif theta >= 4.3196899 && theta < 5.1050881
        deltaT1 = fullArr(07) - fullArr(01); %sqrt(2) m
        deltaT2 = fullArr(08) - fullArr(02); %2m
        deltaT3 = fullArr(09) - fullArr(03); %sqrt(2) m
%CMV_direction = ~315 degrees
elseif theta >= 5.1050881 && theta < 5.8904862
        deltaT1 = fullArr(06) - fullArr(02); %sqrt(2) m
        deltaT2 = fullArr(09) - fullArr(01); %2m
        deltaT3 = fullArr(08) - fullArr(04); %sqrt(2) m
        deltaT4 = fullArr(09) - fullArr(05); %1m
        deltaT5 = fullArr(05) - fullArr(01); %1m
%CMV_direction = ~360 degrees
elseif theta >= 5.8904862 && theta < 6.6758844
        deltaT1 = fullArr(03) - fullArr(01); %sqrt(2) m
        deltaT2 = fullArr(06) - fullArr(04); %2m
        deltaT3 = fullArr(09) - fullArr(07); %sqrt(2) m
end

v(1) = abs(sqrt(2) / deltaT1);
v(2) = abs(2 / deltaT2);
v(3) = abs(sqrt(2) / deltaT3);
v(4) = abs(1 / deltaT4);
v(5) = abs(1 / deltaT5);

for id = 1:5
    if isinf(v(id))
        v(id) = nan;
    end
end

cmvSpeed = mean(v, 'omitnan') / 150 * 1000; %meters per second
end
```

get csd.m

```
function outputArray = get_csd(magnitude, theta, threshold)
%% This function gets the raw magnitude and theta converts to a corrected array
[nRows, nCols, nPages] = size(magnitude);

%% Find average angles
angleArray = zeros(nPages, 1);   %initialize list of angles
for iPage = 1:nPages
    thetaAvg = 0; %initialize variable
    cnt = 0;
    for iRow = 1:nRows
        for iCol = 1:nCols
```

```
            if magnitude(iRow, iCol, iPage) > threshold
                thetaAvg = thetaAvg + deg2rad(theta(iRow, iCol, iPage));            %get a running tally of angles
                cnt = cnt + 1;
            end
        end
    end

    thetaAvg = thetaAvg / cnt;                                              %get average angle
    angleArray(iPage, 1) = thetaAvg;
end

%% Find the first non - NaN element's sign
testArray = angleArray;
cnt = 0;
for idx = 1:numel(testArray)
    if ~isnan(testArray(idx))
        cnt = cnt + 1;
        angleArray(cnt, 1) = testArray(idx);
    end
end

%Find starting point
start = 1;
while start < numel(testArray) && isnan(testArray(start))
    start = start + 1;
end

%% Check the quadrants of the first 1 / 4 of the elements
angleLabel = get_quadrant(testArray);

checkArray = cell(numel(angleLabel, 1));
checkArray(1, 1) = angleLabel(start);

[maxRow, ~] = size(angleArray);
maxCheck = start + ceil(maxRow / 8);

cnt = 1;
disp([start maxCheck]);
for idx = start:maxCheck
    if ~isequal(angleLabel(idx), checkArray(cnt)) %check if qudrant is not the same
        cnt = cnt + 1;  %increment
        checkArray(cnt, 1) = angleLabel(idx);  %save new quadrant label to another cell
    end
end

%% Correct angles (in radians) opposite that of reference quadrants
for idx = 1:numel(testArray)
    notequal = 0;

    for idx2 = 1:numel(checkArray)
        if isequal(angleLabel{idx}, checkArray{idx2})
            notequal = notequal + 1;
        end
    end

    if ~notequal
        testArray(idx) = testArray(idx) + pi;
        angleLabel{idx} = get_quadrant(testArray(idx));
    end
end

%% Return output
outputArray = testArray;
end
```

get matrix.m

```
function outputMatrix = get_matrix(orderedArray, dataWindow)
  %% This function maps lux data into a matrix
  orderedArrayMsum = movsum(orderedArray, dataWindow, 1, Endpoints='discard'); % TODO: Figure out why a moving sum is being applied
  matSize = sqrt(width(orderedArrayMsum)); %TODO: Replace with more general definition
  outputMatrix = reshape(orderedArrayMsum', matSize, matSize, []);
  outputMatrix = permute(outputMatrix,  [2, 1, 3]);
  outputMatrix = outputMatrix / dataWindow;
end
```

get norm.m

```
function dataSampleNorm = get_norm(dataSample)
%% This function normalizes data with respect to each column
[nRows, nCols] = size(dataSample);
dataNorm = zeros(nRows, nCols);
for iCol = 1:nCols
    dataNorm(:, iCol) = dataSample(:, iCol) / max(abs(dataSample(:, iCol)));
end
dataSampleNorm = dataNorm;
```

get quadrant.m

```
function angleLabel = get_quadrant(array)
  angleLabel = cell(1);
  for idx = 1:numel(array)
    if array(idx)>=0 && array(idx) < pi/2
      angleLabel(idx, 1) = {'Quadrant1'};
    elseif array(idx)>=pi/2 && array(idx) < pi
      angleLabel(idx, 1) = {'Quadrant2'};
    elseif array(idx)>=pi && array(idx) < 3*pi/2
      angleLabel(idx, 1) = {'Quadrant3'};
    elseif array(idx)>=3*pi/2 && array(idx) < 2*pi
      angleLabel(idx, 1) = {'Quadrant4'};
    else
      angleLabel(idx, 1) = {NaN};
    end
  end
end
```

get resultant vec.m

```
function [M, phase] = get_resultant_vec(magnitude, theta)
% This function converts the magnitudes and angles into a phasor. The
% resultant vector's is then decomposed as magnitude, M, and angle, phase.

zTotal = 0;
threshold = 0.02;
[nRows, nCols, nPages] = size(magnitude);
M = zeros(nPages, 1);
phase = zeros(nPages, 1);

for iPage = 1:(nPages - 1)
    for iRow = 1:nRows
        for iCol = 1:nCols
            R = magnitude(iRow, iCol, iPage);
            rtheta = deg2rad(theta(iRow, iCol, iPage));

            if R > threshold
                z = R * (cos(rtheta) + j * sin(rtheta));                         %convert into complex form
                zTotal = zTotal + z;                                      %add complex numbers
            end
        end
    end

    M(iPage) = abs(zTotal);
    phase(iPage) = angle(zTotal);
end
end
```

get sample range.m

```
function dataSampleRange = get_sample_range(data, dataStart, dataEnd)
%% This function gets a specified frame from x_start to x_end of the dataset.
nSamples = dataEnd - dataStart + 1;
nSensors = width(data);
dataSampleRange = zeros(nSamples, nSensors);
for i = 1:nSensors
%     actual_temp = data(x_start:x_end, i);
    dataSampleRange(:, i) = data(dataStart:dataEnd, i);
end
% data_sample = reshape(data_sample, num_samples, num_sensors);
```

get vid.m

```
function get_vid(inputStruct, filename)
%% Turn images in struct into an avi file
v = VideoWriter(filename);
open(v);

for k = 1:length(inputStruct)
   frame = inputStruct(:, k);
   writeVideo(v, frame);
end

close(v);
```

main.m

```
clear frames paddedData;

DATA_FILE = '../Data/11-9-sensors-only.csv';
OUTPUT_DIR = '11-9-output';
% OUTPUT_DIR = 'output';
MATRIX_TYPE = 'normalized';
THRESHOLD = 0.03;
PEAK_DISTANCE = 50;
PEAK_PROMINENCE = 0.15;%0.016;
PEAK_WIDTH = 15;
SENSOR_ORDER  = [5 4 6 8 7 9 2 1 3]; % Northwest to Southeast
DATA_ORDER  =   [9 4 8 3 5 1 7 2 6]; % Northwest to Southeast
FILL_ORDER  =   [4 1 8 9 6 2 5 3 7]; % N S W E NE SW NW SE O
CREATE_VIDEO = false;
CREATE_PLOTS = true;
% SENSOR_ORDER = FILL_ORDER;
if ispc % Check to see if operating system is Windows
  DELIMITER = '\';
else % otherwise, use unix-style path delimiters.
  DELIMITER = '/';
end
figure_setup;
load figure_setup SENSOR_STRINGS FIGURE_STRINGS FMT BIN_EDGES SCALE TILE;

if ~isfile(DATA_FILE)
  [DATA_FILE, DATA_FILE_PATH] = uigetfile('*.csv;*.txt;*.dat',...
    'Select Input CSV Data', 'data.csv');
  DATA_FILE = [DATA_FILE_PATH, DATA_FILE];
end
if ~isfolder(OUTPUT_DIR)
  OUTPUT_DIR = uigetdir('.', 'Select Output Directory');
end

% Read and transfer raw lux data to new array
data = readmatrix(DATA_FILE, OutputType='string');
if ~isempty(regexp(data, '[a-fx]', 'once')) % Check if data is hex
  data = hex2dec(data); % If hex, convert to decimal
end
data = uint32(data);

```

```matlab

%%%%%% THINGS PEAK
% i = 0;
% lastMat = 0;
% T = thingSpeakRead(1552033, ... % Get table from TSpeak
%     Fields=1, ...
%     NumPoints=1661, ...
%     ReadKey='AD8ZB04MFD6HIYI8', ...
%     OutputFormat='table');
% dataCol = T(:, {'cmvsData'});       % time & data cols -> data col
% dataCol = rowfun(@string, dataCol); % char table -> str table
% data = dataCol{:,:};                %  str table -> str array
% data = arrayfun(@(x) uint32(str2num(x)), data, ...
%                 uniform=false);     % str array -> uint32 cell array
% data = cell2mat(data);              % uint32 cell array -> uint32 array
%%%%%

nNotNan  = sum(~isnan(data),2); % count number of valid values in each row
nSensors = round(mean(nNotNan)); % Use mean to get number of sensors
data = data(nNotNan == nSensors, :); % Get rows with a reading for each sensor
data = rmmissing(data, 2); % Exclude any remaining columns that contain a Nan

iLoop = 1;
% iDataStart = 660;
% iDataEnd = 770;
% iDataStart = 1;
% iDataEnd = 101;
iDataStart = 840; % 10 ft/ 30 s -- whole array
iDataEnd = 940;

% iDataStart = 1030 + 24; % 10 ft / 10 s -- whole array
% iDataEnd = 1130 - 24;
iDelta = iDataEnd - iDataStart;
dataWindow = 10; % specifies sliding window length for moving sum
filterWindow = 11; %  specifies smoothing window length


%% Validate, Normalize, and Smooth Data
if dataWindow > length(data)
  dataWindow = length(data);
  dataWindowWarn = sprintf("dataWindow exceeds length of data and has been trimmed");
else
  dataWindowWarn = sprintf('');
end
if filterWindow > length(data)
  filterWindow = length(data);
  filterWindowWarn = sprintf("filterWindow exceeds length of data and has been trimmed");
else
  filterWindowWarn = sprintf('');
end
if iDataEnd > length(data)
  iDataEnd = length(data);
  iDataStart = max(iDataEnd - iDelta, 1);
  dataEndWarn = sprintf("iDataEnd exceeds length of data. Range parameters have been changed");
else
  dataEndWarn = sprintf('');
```

```
end
if ~ismissing([dataWindowWarn, filterWindowWarn, dataEndWarn])
  warning('\n\t%s\n\t%s\n\t%s', dataWindowWarn, filterWindowWarn, dataEndWarn);
end

if nSensors < 9
  %   paddedData = padarray(data', 9 - width(data), nan, 'post')';
  paddedData = NaN(length(data), 9);
  if mod(nSensors, 2)
    fillOrder = [FILL_ORDER(1:nSensors) FILL_ORDER(end)];
  else
    fillOrder = FILL_ORDER;
  end
  for iSensor = 1:nSensors
    paddedData(:, fillOrder(iSensor)) = data(:, iSensor);
  end
  % paddedData(:, floor(linspace(1,9,nSensors))) = data;
  data = fillmissing(paddedData, 'movmean', max(9 - nSensors,2), 2, ...
    EndValues='nearest');
  warning("%d sensors detected. Missing data is being interpolated, and may be inaccurate.", ...
    nSensors);
  nSensors = 9;
end

% while iDataStart + iDelta < height(dataCol)
% Tsl = Sensor;
% data             = calculate_lux(Tsl, data);

iDataEnd         = iDataStart +  iDelta;

dataSample       = get_sample_range(data, iDataStart, iDataEnd);
dataSample       = dataSample(:, DATA_ORDER);
dataSampleNorm   = get_norm(dataSample);
smoothSample     = smoothdata(dataSample, 'sgolay', filterWindow);
smoothSampleNorm = smoothdata(dataSampleNorm, 'sgolay', filterWindow);

%% Plot Sensor Data
plotSets = {
  data
  dataSample
  smoothSample
  smoothSampleNorm
  };
if CREATE_PLOTS
  figure(FMT.FIG);
  dataPlotFmt.LineWidth = 2;
  for iPlotSet = 1:length(plotSets)
    dataPlot = plot(plotSets{iPlotSet});
    for iSensor = 1:nSensors
      dataPlotFmt.DisplayName = SENSOR_STRINGS(iSensor, :);
      set(dataPlot(iSensor), dataPlotFmt);
    end % iSensor = 1:nSensors
    legend('show');
    dataAx = gca;
    xlabel('Time Elapsed (milliseconds)');
    ylabel('Irradiance (W/m^2)');
    dataAx.XTickLabel = arrayfun(@(x) sprintf('%d', SCALE * x), dataAx.XTick,...
      'un', 0);
    set(dataAx, FMT.AX);
    saveas(gca, fullfile(OUTPUT_DIR, FIGURE_STRINGS(iPlotSet, :)), 'fig');
    saveas(gca, fullfile(OUTPUT_DIR, FIGURE_STRINGS(iPlotSet, :)), 'png');
    close
  end % iPlotSet = 1:length(plotSets)
end % CREATE_PLOTS
%% Find peaks and dips
t = (iDataStart:iDataEnd); %/ Fs
peakArr = zeros(nSensors, 1);
peakLocArr = zeros(nSensors, 1);

dipArr = zeros(nSensors, 1);
dipLocArr = zeros(nSensors, 1);

% Peak and dip parameters
dipFmt.MinPeakDistance = PEAK_DISTANCE;
dipFmt.MinPeakProminence = PEAK_PROMINENCE;
dipFmt.NPeaks = PEAK_WIDTH;

% Plot local maxima and minima
if CREATE_PLOTS
  sensorPlot = repelem(0, nSensors);
  figure(FMT.FIG);

  hold on
  for iSensor = 1:nSensors
    sensorInv = 1 ./ smoothSampleNorm(:, iSensor);
    [dip, dipLoc] = findpeaks(sensorInv, dipFmt);

    if isempty(dipLoc)
      dipLocArr(iSensor) = 0;
    else
      dipLocArr(iSensor) = dipLoc(1);
    end

    sensorPlot(iSensor) = plot(t, smoothSampleNorm(:, iSensor), ...
      DisplayName='Origin Sensor', LineWidth=2);
    set(sensorPlot(iSensor), dataPlotFmt);
    plot(t(dipLoc), 1 / dip, 'rs', 'MarkerSize', 10);
  end
  hold off
  sensorAx = gca;
  set(sensorAx, FMT.AX);
  xlabel('Time Elapsed (milliseconds)');
  ylabel('Normalized Irradiance');
  sensorAx.XTickLabel = arrayfun(@(x) sprintf('%d', SCALE * x), sensorAx.XTick, 'un', 0);
  saveas(gca, fullfile(OUTPUT_DIR, 'CMV_Sample_Norm'), 'fig');
  saveas(gca, fullfile(OUTPUT_DIR, 'CMV_Sample_Norm'), 'png');
end % CREATE_PLOTS

if strcmp(MATRIX_TYPE, 'normalized')
  smoothSampleNorm2 = get_norm(smoothSample); % FIXME: Why is the normalization of smooth sample being defined differently here?
  luxMatrix =  get_matrix(smoothSampleNorm2(:, SENSOR_ORDER), dataWindow);
else
```

```
  luxMatrix =  get_matrix(smoothSample(:, SENSOR_ORDER), dataWindow);
end

pages = length(luxMatrix);                                              %find maxnumber of frames
imData =  luxMatrix(:, :, 1:pages);                                           %set dataset to be analyzed
[imageRow, imageCol, ~] = size(imData);
theta = zeros(imageRow, imageCol, pages);
magnitude = zeros(imageRow, imageCol, pages);

%% Prepare Frames
clear XLim yLim
frames(pages) = struct('cdata',[],'colormap',[]);
figure(FMT.FIG);
% set(gcf, Visible = false);
progressBar = waitbar(0, '1', Name='Populating Frames');

qFigs = nan(1, pages);
for iFrame = 1:(pages)
  waitbar(iFrame/pages, progressBar, sprintf("Frame %4d / %4d\n%3d%% complete", iFrame, pages, ceil(iFrame/pages * 100)));
  [gx, gy] = imgradientxy( imData(:, :, iFrame), 'sobel'); % Find cmv direction using Gradient Matrix Method
  [gmag, gdir] = imgradient(gx, gy);
  theta(:, :, iFrame) = gdir;
  magnitude(:, :, iFrame) = gmag;
  if CREATE_VIDEO
    figure(FMT.FIG);
    q = quiver(gx, -gy); %invert to correct visual vector orientation
    xAbsPos = [floor(q.XData + q.UData); ceil(q.XData + q.UData)];
    [xLim(1), xLim(2)] = bounds(xAbsPos, 'all');
    yAbsPos = [floor(q.YData - q.VData); ceil(q.YData - q.VData)];
    [yLim(1), yLim(2)] = bounds(yAbsPos, 'all');
    qFigs(iFrame) = gcf;
  end % if CREATE_VIDEO
end
delete(progressBar);
%% Create Video
if CREATE_VIDEO
  vidDir = 'Gradient Matrix Animations';
  [~, ~] = mkdir([OUTPUT_DIR, DELIMITER, vidDir]);
  videoFmt = 'MPEG-4';
  videoTitle = string([OUTPUT_DIR, DELIMITER, vidDir, DELIMITER, ' Mat', char(datetime('now', Format='yy-MM-dd_HH-mm-ss'))]);
  v = VideoWriter(videoTitle, videoFmt);
  v.FrameRate = 30;
  open(v);
  txt = sprintf('dataWindow = %d filterWindow = %d\n', dataWindow, filterWindow);
  progressBar = waitbar(0, '1', Name='Creating Video');
  for iFrame = 1:(pages)
    waitbar(iFrame/pages, progressBar, sprintf("Frame %4d / %4d\n%3d%% complete", iFrame, pages, ceil(iFrame/pages * 100)));
    ax = gca(qFigs(iFrame));
  %   xlim(ax, [0, xLim(2)]);
    xlim(ax, [0, 5]);
  %   ylim(ax, [0, yLim(2)]);
```

```
    ylim(ax, [0, 5]);
    textWrapper(txt, ax);
    frames(cast(iFrame, 'uint16')) = getframe(qFigs(iFrame));
    writeVideo(v, frames(iFrame));
  end % iFrame = 1:(pages)
  close(v);
  delete(progressBar);
end % if CREATE_VIDEO
%% Create Polar Histograms
mtd1.shadow = struct;
mtd2.shadow = struct;
mtd1.shadow.ang = get_csd(magnitude, theta, THRESHOLD); %correct raw angles
[mtd2.shadow.mag, mtd2.shadow.ang] =  get_resultant_vec(magnitude, theta);

figure(99);
set(gcf, FMT.FIG);
tlo = tiledlayout(TILE.ROWS, TILE.COLS);
title(tlo, 'Shadow Direction Probability');
set(tlo,FMT.TLO);
nexttile(TILE.POS(1), TILE.LARGE_SPAN); % Large Left Tile BEGIN
  mtd1.phistBig = polarhistogram(mtd1.shadow.ang, 10, Normalization="probability");
  hold on;
    mtd2.phistBig = polarhistogram(mtd2.shadow.ang, BIN_EDGES, Normalization="probability");
    legendLabels(1) = "Method One";
    legendLabels(2) = "Method Two";

    % Plot dotted projection lines
    mtd1.phistBigProj = polarhistogram(mtd1.shadow.ang, 10, ...
      Normalization="count", EdgeColor=FMT.COLORORDER(1, :), ...
      FaceColor='none', LineStyle=':');
    mtd2.phistBigProj = polarhistogram(mtd2.shadow.ang, BIN_EDGES, ...
      Normalization="count", EdgeColor=FMT.COLORORDER(2, :), ...
      FaceColor='none', LineStyle=':');
    % Find bins w/ probability >= 5% and extend to edges
    mtd1.phistBigProj.BinCounts(mtd1.phistBig.Values >= 0.05) = 1;
    mtd2.phistBigProj.BinCounts(mtd2.phistBig.Values >= 0.05) = 1;
    % Set bins w/ probablility < 5% to zero
    mtd1.phistBigProj.BinCounts(mtd1.phistBig.Values  < 0.05) = 0;
    mtd2.phistBigProj.BinCounts(mtd2.phistBig.Values  < 0.05) = 0;

    bothMtds.polarAx = gca;
    set(bothMtds.polarAx, FMT.POLAX);

    FMT.RTICKSET();
    the = 0:45:315;
    rho = repmat(gca().RLim, 1, length(the));
    the = repelem(deg2rad(the), 2);

    for iTheta = 1:2:(length(the)-1)
      polarplot(the(iTheta:iTheta+1), rho(iTheta:iTheta+1), ...
        LineWidth=1, LineStyle='-', Color=[0 0 0 0.25]);
    end
  hold off;
  legendLabels(3:length(gca().Children)) = repelem("", length(gca().Children) - 2);

  legend(bothMtds.polarAx, legendLabels, ...
    Location='northoutside', Orientation='horizontal');
  set(gca, Children=flipud(gca().Children));
% Large Left Tile END
nexttile(TILE.POS(2)); % Upper Right Tile BEGIN
  mtd1.phist = polarhistogram(mtd1.shadow.ang, 10, Normalization="probability");
  mtd1.polarAx = gca;
  mtd1.phist.FaceColor = FMT.COLORORDER(1,:);
  set(mtd1.polarAx, FMT.POLAX);
  FMT.RTICKSET();
% Upper Right Tile END
nexttile(TILE.POS(3)); % Lower Right Tile BEGIN
  mtd2.phist = polarhistogram(mtd2.shadow.ang, BIN_EDGES, Normalization="probability");
  mtd2.polarAx = gca;
  mtd2.phist.FaceColor = FMT.COLORORDER(2,:);
  set(mtd2.polarAx, FMT.POLAX);
  FMT.RTICKSET();
% Upper Left Tile END

cmvDirection1 = get_cmv_direction(mtd1.shadow.ang, mtd1.phist, 1);
cmvDirection2 = get_cmv_direction(mtd2.shadow.ang, mtd2.phist, 2);
cmvSpeed1     = get_cmv_speed(cmvDirection1, dipLocArr);
cmvSpeed2     = get_cmv_speed(cmvDirection2, dipLocArr);
cmvSpeed1     = fillmissing(cmvSpeed1, "nearest", EndValues='nearest');
cmvSpeed2     = fillmissing(cmvSpeed2, "nearest", EndValues='nearest');
cmv           = [cmvDirection1 cmvSpeed1 cmvDirection2 cmvSpeed2];

% tlo.OuterPosition = tlo.OuterPosition .* [1 1 1 1 + 0.125];
% tlo.InnerPosition = tlo.InnerPosition .* [1 1 1 1 + 0.125];
txt = sprintf('Dir(1)=% 6.5g Speed(1)=% 6.5g <> Dir(2)=% 6.5g Speed(2)=% 6.5g', cmv);
textWrapper(txt, gca, [1 -0.18]);
figure(gcf);
% saveas(gcf, fullfile(OUTPUT_DIR, 'cmv_histogram'), 'fig');                          %save figure
saveas(gcf, fullfile(OUTPUT_DIR, 'cmv_histogram'), 'png');                          %save image
iLoop = iLoop + 1;
iDataStart = iDataEnd;
% end % while iDataStart + iLoop * iDelta < height(dataCol)
%% Find the optical flow
% OpF =  get_optical_flow(imData);
%  get_vid(OpF, strcat(OUTPUT_DIR, 'OpticalFlow'));                         %save OpF run as .AVI file

fileID = fopen(strcat(OUTPUT_DIR, 'cmv.txt'), 'w');
fprintf(fileID, '%6s %6s %6s %6s\n', 'CMV_Direction1', 'CMV_Direction2', 'CMV_Speed1', 'CMV_Speed2');
fprintf(fileID, '%0.2f %0.2f %0.2f %0.2f\n', cmv);
fclose(fileID);
```

read cmv data.m

```
function dataRaw = read_cmv_data(filepath)
%% This function grabs the filepath of the CMV sensor data and ouputs a table version of
```

```
  the data
%   data = output table version of original data
%   filepath = directory of the input file

fd = fopen(filepath, 'rt');

% formatSpec = '%s % * f %f %f'; => doesn't work!!!
% So use the following code to create the same format
%formatSpec = '%s %d %d %d %d %d %d %d %d %d';
%formatSpec = '%s %f %f %f %f %f %f %f %f %f';
formatSpec = '%s % * f %f % * f %f % * f %f % * f %f % * f %f % * f %f % * f %f % * f %f % * f %f';

% Transfer csv file data into dataRaw array
dataRaw = textscan(fd, formatSpec, 'Delimiter', {', ', '\n'}, 'CollectOutput', 1, 'EndOfLine', '\n');
fclose(fd); %close csv file

for i = 1:length(dataRaw{1})
    dateTemp = dataRaw{1, 1}{i, 1};
    dateTemp = dateTemp(1:end);
    dateTemp = strrep(dateTemp, 'T', ' '); %remove T from timestamp
    dataRaw{1, 1}{i, 1} = dateTemp;
end

%data = [dataRaw{:, 1} dataRaw{:, 2}];
```

Sensor.m

```
classdef Sensor
  properties (Constant = false)
    aTime = Sensor.INTTIME_100MS;
    aGain = Sensor.GAIN_LOW;
  end
  properties (Constant = true)
    INTTIME_100MS = 0x00 % 100 millis
    INTTIME_200MS = 0x01 % 200 millis
    INTTIME_300MS = 0x02 % 300 millis
    INTTIME_400MS = 0x03 % 400 millis
    INTTIME_500MS = 0x04 % 500 millis
    INTTIME_600MS = 0x05 % 600 millis
    GAIN_LOW      = 0x00
    GAIN_MED      = 0x10
    GAIN_HIGH     = 0x20
    GAIN_MAX      = 0x30
    GA            = 1.0   % Glass Attenuation (equals 1 if uncovered)
    % DF            = 408.0 % Device Factor (specific to TSL2591)
    % COEFA         = 1.00  % IR channel (ch0) coefficients
    % COEFB         = 1.64  % IR channel (ch0) coefficients
    % COEFC         = 0.59  % FULL channel (ch1) coefficients
    % COEFD         = 0.86  % FULL channel (ch1) coefficients
    DF            = 60    % Device Factor (specific to TSL2591)
    COEFA         = 1.00  % IR channel (ch0) coefficients
    COEFB         = 1.87  % IR channel (ch0) coefficients
    COEFC          = 0.63  % FULL channel (ch1) coefficients
    COEFD          = 1.00  % FULL channel (ch1) coefficients
  end
  methods
    function s = Sensor(setTime, setGain)
      arguments
        setTime = 0;
        setGain = 0;
      end
      switch setTime % How long the sensor collects light
        case s.INTTIME_100MS, s.aTime = 100.0;
        case s.INTTIME_200MS, s.aTime = 200.0;
        case s.INTTIME_300MS, s.aTime = 300.0;
        case s.INTTIME_400MS, s.aTime = 400.0;
        case s.INTTIME_500MS, s.aTime = 500.0;
        case s.INTTIME_600MS, s.aTime = 600.0;
        otherwise           , s.aTime = 100.0; % Default
      end
      switch setGain
        case s.GAIN_LOW , s.aGain = 1.0;
        case s.GAIN_MED , s.aGain = 25.0;
        case s.GAIN_HIGH, s.aGain = 428.0;
        case s.GAIN_MAX , s.aGain = 9876.0;
        otherwise       , s.aGain = 1.0; % Default
      end
    end
  end
end
```

text wrapper.m

```
function text_wrapper(txt, ax, pos)
arguments
txt
ax = gca;
pos = [0.95 -0.1];
end
text(ax, pos(1), pos(2), txt, HorizontalAlignment='right', Units='normalized',
  FontName='FixedWidth', FontSize=10);
end
```

thing speak test.m

```
function [Mat, times] = thing_speak_test(nPoints, nLoops, forever)
arguments
  nPoints=1;
  nLoops = 5;
  forever = false;
end

times = repelem(datetime,nLoops * nPoints, 9);
Mat = zeros(nLoops * nPoints, 9);
```

```matlab

iLoop = 1;
lastTime = datetime;
tRead = tic;
if forever, loopCondition = @(iLoop) 1; else, loopCondition = @(iLoop) iLoop <= nLoops; end
% loopCondition = @(iLoop) 1 if forever, else loopCondition = @(iLoop) iLoop <= nLoops, end
%   while iLoop <= nLoops
  while loopCondition(iLoop)
    Tab = thingSpeakRead(1552033, ...
          Fields=1, ...
          NumPoints=nPoints, ...
          ReadKey='AD8ZB04MFD6HIYI8', ...
          OutputFormat='timetable');

  times = Tab.Timestamps;
%     times(iLoop*nPoints:nPoints*(iLoop+1)-1, nPoints) = Tab.Timestamps;
%     timeTab = rowfun(@datenum, timeTab);
    if  times(end) == lastTime
      continue
    end
    toc(tRead);
    tRead = tic;
    fprintf("Loop %d: ", iLoop);

    Tstr = string(Tab.cmvsData);
    Tstr = replace(Tstr{:,:}, ' ', '');
%     if Tstr(end) == ',', Tstr = Tstr(1:end-1); end
    Tarr = str2double(split(Tstr,','));
%     Cell = arrayfun(@(x) uint32(str2num(x)),Tarr,'uniform',0); %#ok<ST2NM>
    Mat(iLoop, :) = Tarr;

    disp(Mat(iLoop, :));
    fprintf("\n");
    try
    plot(times, Mat);
    catch
      disp("breakpoint")
    end
%     plot(time,Mat);
%     if ~isequal(Mat, lastMat)
%       disp(Mat(end, :)), else, disp("same"), end
    % if ~isequal(Mat, lastMat)
    %     disp(Mat(end, :));
    % else
    %     disp("same");
    % end
%     [p.XData] = deal(time);
%     for k_ = 1:nLoops
%     end
  lastTime = times(end);
  iLoop = iLoop+1;
  if forever && (iLoop > nLoops)
    iLoop = 1;
  end

  end % while

end %function
```

Appendix B: Arduino Code Listings

The code in this section is written for use with an Arduino Nano 33 IoTand contains a programmatic implementation of the I2C protocol, allowing the SDA and SCL pins to be arbitrarily assigned to any digital input/output pin. As a result, a system using this design does not need to be connected to a computer, BeagleBone Black, or any other peripheral beyond the TSL2591sensors themselves.

cmvs sw wifi.ino

```
#include <Wire.h>
#include <avr/dtostrf.h>
#include <stdio.h>


#include "constants.h"
// #include "led_codes.h"
#include "tsl_softwire.h"
#include "wifi_setup.h"

#define SETTINGS TSL2591_INTEGRATIONTIME_100MS | TSL2591_GAIN_LOW
#define SERIAL_PRINT true

tsl2591_t tsl[N_SENSORS];


char data_str[N_SENSORS * RD_WIDTH] = "";
const uint8_t mode = LUM_MODE;

void setup(void) {
  delay(1000);
  Serial.begin(9600);
  pinMode(LED_PIN, OUTPUT);
  initializeArray(SETTINGS);
  // WiFi & ThingSpeak setup
  printArduinoMac();
  wifiSetup();
}

void loop(void) {
  // ledSignal(FAIL);
  // ledSignal(ZERO);
  // ledSignal(PASS);
  // ledSignal(UPLOAD);
  // ledSignal(STUCK);
  // ledSignal(SEARCHING);
  // ledSignal();
  // digitalWrite(LED_PIN, HIGH);
  // delay(500);
  // digitalWrite(LED_PIN, LOW);
  // delay(500);
  refreshArray();
  if(SERIAL_PRINT) printArrayData();
  uploadData(formatDataStr());
}
```

constants.h

```
#ifndef CONSTANTS_H
#define CONSTANTS_H

#define RD_WIDTH 16
#define DLY 500
#define DATA_CH 1
#define STATUS_CH 2

#define N_SENSORS 9
#define BUFLEN 2
#define CMVS_INTEGRATION TSL2591_INTEGRATIONTIME_100MS
#define CMVS_GAIN TSL2591_GAIN_LOW
#define LUX_MODE 0b1000
#define LUM_MODE 0b0100
#define IR_MODE 0b0010
#define VIS_MODE 0b0001
#define FULL_MODE 0b0000

#define T_SCANNING_ON 50
#define T_SCANNING_OFF 100
#define T_STUCK_ON 4000
#define T_STUCK_OFF 4000
#define T_PROCESSING_ON 10
#define T_PROCESSING_OFF 1000
// #define LED_PIN 19
#define LED_PIN LED_BUILTIN

#define SDA1 17
#define SCL1 16

#define SDA2 14
#define SCL2 15

#define SDA3 12
#define SCL3 13

#define SDA4 11
#define SCL4 10

#define SDAX  8
#define SCLX  9

#define SDA5  7
#define SCL5  6

#define SDA6  4
#define SCL6  5

#define SDA7  2
#define SCL7  3

#define SDA8 21
#define SCL8 20
#endif // CONSTANTS_H
```

secrets.h

```c
// Use this file to store all of the private credentials
// and connection details
#ifndef SECRETS_H
#define SECRETS_H
#define HOT_SSID "iPhone"
#define HOT_PASS "wet$paghetti1979"
#define SECRET_SSID ""
#define SECRET_PASS ""
#define WPI_SSID "WPI-Open"
#define WPI_PASS "\0"

#define SECRET_CH_ID 1552033
#define SECRET_WRITE_APIKEY "AD8ZB04MFD6HIYI8"

const char *ssid_list[] = {SECRET_SSID, WPI_SSID};
const char *pass_list[] = {SECRET_PASS, WPI_PASS};
#endif // SECRETS_H
```

led codes.h

```cpp
#ifndef LED_CODES_H
#define LED_CODES_H

#include "constants.h"
#include <Arduino.h>
template <typename F>
void repeat(unsigned n, F f) {
    while (n--) f();
}
void ledToggle(uint16_t t_on_ms, uint16_t t_off_ms);
void ledSignal(uint8_t status = 0xFF);
void ledOn(uint16_t delay_ms = 0);
void ledOff(uint16_t delay_ms = 0);

enum{FAIL , ZERO, PASS, SEARCHING, STUCK, UPLOAD};
#endif // LED_CODES_H
```

led codes.cpp

```
#include "led_codes.h"

uint8_t led_state = LOW;

void ledOn(uint16_t delay_ms) {
  led_state = HIGH;
  digitalWrite(LED_PIN, led_state);
  delay(delay_ms);
}
void ledOff(uint16_t delay_ms) {
  led_state = LOW;
  digitalWrite(LED_PIN, led_state);
  delay(delay_ms);
}

void ledToggle(uint16_t t_on_ms, uint16_t t_off_ms) {
  if (!led_state)
    ledOn(t_on_ms);
  else
    ledOff(t_off_ms);
}

void ledSignal(uint8_t status) {
  static uint8_t i = 0;
  static uint8_t j = 0;
  static uint8_t prev_status = 0xFF;
  if (status != prev_status) {
    i = 0;
    j = 0;
  }
  switch (status) {
  case FAIL: {
    // repeat(4, [] {ledToggle(100,100);});
    repeat(4, [] {ledToggle(50,50);});
    repeat(8, [] {ledToggle(25,25);});
    repeat(16, [] {ledToggle(10,10);});
    break;
  }
  case ZERO: {
    repeat(10, [] { ledToggle(1, 24); });
    break;
  }
  case PASS: {
    repeat(4, [] { ledToggle(1, 9); });
    break;
  }
  case UPLOAD: {
    repeat(4, [] { ledToggle(5, 190); });
    break;
  }
  case STUCK: {
    ledToggle(400, 100);
    ledToggle(400, 100);
    break;
  }
  case SEARCHING: {
    for(uint8_t i = 0; i < 20; ++i) {
      ledToggle(i, 25-i);
    }

    break;
  }
  default:
    ledToggle(1000, 1000);
    ledToggle(1000, 1000);
  }
  prev_status = status;
  // digitalWrite(LED_PIN, LOW);
}
```

tsl softwire.h

```
#ifndef TSL_SOFTWIRE
#define TSL_SOFTWIRE

#include "constants.h"
// #include "led_codes.h"
#include <Adafruit_Sensor.h>
#include <Adafruit_TSL2591.h>
#include <SoftWire.h>
#include "led_codes.h"

#define DEFAULT_SETTINGS TSL2591_INTEGRATIONTIME_100MS | TSL2591_GAIN_LOW
extern char tsl_data_str[N_SENSORS * RD_WIDTH];

typedef enum tsl_ch {
  FULL = 1,
  IR = 2,
  BOTH = 3,
} tsl_ch_t;

typedef enum loc {
  NORTHWEST,
  NORTH,
  NORTHEAST,
  WEST,
  ORIGIN,
  EAST,
  SOUTHWEST,
  SOUTH,
  SOUTHEAST,
} loc_t;

typedef struct tsl2591 {
  uint8_t settings;
  uint8_t sda;
  uint8_t scl;
  uint16_t ch0;
  uint16_t ch1;
} tsl2591_t;

extern tsl2591_t tsl[N_SENSORS];

uint8_t read8(uint8_t reg);
void write8(uint8_t reg, uint8_t value);
void write8(uint8_t reg);
void enable(void);
void disable(void);
void setTslPins(uint8_t location);
void sensorWrite(uint8_t *buffer, size_t len, uint8_t stop = true);
void sensorRead(uint8_t *buffer, size_t len, uint8_t stop = true);
void initializeArray(uint8_t settings = DEFAULT_SETTINGS);
void configureSensor(uint8_t settings = DEFAULT_SETTINGS);
void configureArray(uint8_t settings = DEFAULT_SETTINGS);
void refreshTsl(uint8_t location);
void refreshArray(void);
void printArrayData(void);
void printSensorData(uint8_t location);
String formatDataStr(void);
void setSwPins(void);
void getTslData(uint8_t location);
float calculateLux(uint32_t lum);
#endif /*TSL_SOFTWIRE*/
```

tsl softwire.cpp

```
#include "tsl_softwire.h"

const uint8_t sda_pins[] = {SDA8, SDA1, SDA2, SDA7, SDAX,
                            SDA3, SDA6, SDA5, SDA4}; // 17 14 12 11 8 7 4 2 21
const uint8_t scl_pins[] = {SCL8, SCL1, SCL2, SCL7, SCLX,
                            SCL3, SCL6, SCL5, SCL4}; // 16 15 13 10 9 6 5 3 20
uint8_t sw_txbuff[BUFLEN];
uint8_t sw_rxbuff[BUFLEN];

SoftWire swire(sda_pins[0], scl_pins[0]);

void sensorWrite(uint8_t *buffer, size_t len, uint8_t stop) {
  swire.beginTransmission(TSL2591_ADDR);
  swire.write(buffer, len);
  swire.endTransmission(stop);
}
//
void sensorRead(uint8_t *buffer, size_t len, uint8_t stop) {
  for (uint8_t i = 0; i < len; i++) {
    swire.requestFrom((uint8_t)TSL2591_ADDR, (uint8_t)len, (uint8_t)stop);
    buffer[i] = swire.read();
  }
}

uint8_t read8(uint8_t reg) {
  uint8_t buffer[1];
  buffer[0] = reg;
  sensorWrite(buffer, 1, true);
  sensorRead(buffer, 1);
  return buffer[0];
}
void write8(uint8_t reg, uint8_t value) {
  uint8_t buffer[2];
  buffer[0] = reg;
  buffer[1] = value;
  sensorWrite(buffer, 2);
}

void write8(uint8_t reg) {
  uint8_t buffer[1];
  buffer[0] = reg;
  sensorWrite(buffer, 1);
}

void enable(void) {
  // Enable the device by setting the control bit to 0x01
  write8(TSL2591_COMMAND_BIT | TSL2591_REGISTER_ENABLE,
         TSL2591_ENABLE_POWERON | TSL2591_ENABLE_AEN | TSL2591_ENABLE_AIEN |
             TSL2591_ENABLE_NPIEN);
}
void disable(void) {
  // Disable the device by setting the control bit to 0x00
  write8(TSL2591_COMMAND_BIT | TSL2591_REGISTER_ENABLE,
         TSL2591_ENABLE_POWEROFF);
```

```
}

void configureSensor(uint8_t settings) {
  // set timing and gain
  enable();
  write8(TSL2591_COMMAND_BIT | TSL2591_REGISTER_CONTROL, settings);
  write8(TSL2591_COMMAND_BIT | TSL2591_REGISTER_CONTROL, settings);
  disable();
}

void setTslPins(uint8_t location) {
  swire.setSda(sda_pins[location]);
  swire.setScl(scl_pins[location]);
  swire.begin();
}

void initializeArray(uint8_t settings) {
  swire.setTxBuffer(sw_txbuff, sizeof(sw_txbuff));
  swire.setRxBuffer(sw_rxbuff, sizeof(sw_rxbuff));
  swire.setTimeout_ms(10);
  delay(500);
  Serial.print("Initializing sensors...\n");
  for (uint8_t loc = NORTHWEST; loc < N_SENSORS; ++loc) {
    tsl2591_t *p_tsl = &tsl[loc];
    // setTslPins(loc);
    // p_tsl->sda = swire.getSda();
    // p_tsl->scl = swire.getScl();
    // configureSensor(settings);
    // refreshTsl(loc);
    printSensorData(loc);
    Serial.println();
  }
}

void refreshTsl(uint8_t location) {
  tsl2591_t *p_tsl = &tsl[location];
  enable();
  delay(((p_tsl->settings & 0x0F) + 1) * 120); // Wait x ms for ADC to complete
  p_tsl->ch0 = read8(TSL2591_COMMAND_BIT | TSL2591_REGISTER_CHAN0_LOW) |
               read8(TSL2591_COMMAND_BIT | TSL2591_REGISTER_CHAN0_HIGH) << 8;
  p_tsl->ch1 = read8(TSL2591_COMMAND_BIT | TSL2591_REGISTER_CHAN1_LOW) |
               read8(TSL2591_COMMAND_BIT | TSL2591_REGISTER_CHAN1_HIGH) << 8;
  disable();
}
void refreshArray(void) {
  for (uint8_t loc = NORTHWEST; loc < N_SENSORS; ++loc) {
    setTslPins(loc);
    refreshTsl(loc);
  }
}

void printSensorData(uint8_t location) {
  char data_str[N_SENSORS * RD_WIDTH];
  tsl2591_t *p_tsl = &tsl[location];
  if ((p_tsl->ch0 & p_tsl->ch1) == 0xFFFF) {
    sprintf(data_str, "%u:.._..ERR ....", location);
    ledSignal(FAIL);
  } else if (!(p_tsl->ch0 || p_tsl->ch1)) {
    sprintf(data_str, "%u:.._..NIL ....", location);
    ledSignal(ZERO);
  } else {
    sprintf(data_str, "%u:0x_%04x_%04 x", location, p_tsl->ch0, p_tsl->ch1);
    ledSignal(PASS);
  }
  Serial.print(data_str);
}

void printArrayData(void) {
  char data_str[N_SENSORS * RD_WIDTH] = "";
  for (uint8_t loc = NORTHWEST; loc < N_SENSORS; ++loc) {
    printSensorData(loc);
    char delimiter = (loc < N_SENSORS - 1) ? ',' : '\0';
    Serial.print(delimiter);
  }
  Serial.println();
}

// Formats the incoming data according to the mode.
String formatDataStr(void) {
  char data_str[N_SENSORS * RD_WIDTH];
  String out_Str = "";
  for (uint8_t loc = NORTHWEST; loc < N_SENSORS; ++loc) {
    tsl2591_t *p_tsl = &tsl[loc];
    if ((p_tsl->ch0 & p_tsl->ch1) == 0xFFFF) {

    } else if (!(p_tsl->ch0 || p_tsl->ch1)) {
    }
    if (loc < N_SENSORS - 1)
      sprintf(data_str, "0x%04x%04x, ", (p_tsl->ch0), (p_tsl->ch1));
    else
      sprintf(data_str, "0x%04x%04x", (p_tsl->ch0), (p_tsl->ch1));
    out_Str.concat(data_str);
  }
  return out_Str;
}

float calculateLux(uint32_t lum) {
  float a_time, a_gain;
  float cpl, lux1, lux2, lux;
  // uint32_t sw_reading = ;
  uint16_t ir = (lum & 0xFFFF);
  uint16_t full = (lum >> 16);

  // Check for overflow conditions first
  if ((ir == 0xFFFF) | (full == 0xFFFF)) {
    // Signal an overflow
    return -1;
  }

  // Note: This algorithm is based on preliminary coefficients
  // provided by AMS and may need to be updated in the future

  switch (CMVS_INTEGRATION) {
  case TSL2591_INTEGRATIONTIME_100MS:
    a_time = 100.0F;
    break;
  case TSL2591_INTEGRATIONTIME_200MS:
    a_time = 200.0F;
    break;
  case TSL2591_INTEGRATIONTIME_300MS:
    a_time = 300.0F;
    break;
  case TSL2591_INTEGRATIONTIME_400MS:
    a_time = 400.0F;
    break;
  case TSL2591_INTEGRATIONTIME_500MS:
    a_time = 500.0F;
    break;
  case TSL2591_INTEGRATIONTIME_600MS:
    a_time = 600.0F;
    break;
  default: // 100ms
    a_time = 100.0F;
    break;
  }

  switch (CMVS_GAIN) {
  case TSL2591_GAIN_LOW:
    a_gain = 1.0F;
    break;
  case TSL2591_GAIN_MED:
    a_gain = 25.0F;
    break;
  case TSL2591_GAIN_HIGH:
    a_gain = 428.0F;
    break;
  case TSL2591_GAIN_MAX:
    a_gain = 9876.0F;
    break;
  default:
    a_gain = 1.0F;
    break;
  }

  // cpl = (ATIME * AGAIN) / DF
  cpl = (a_time * a_gain) / TSL2591_LUX_DF;

  // Original lux calculation (for reference sake)
  lux1 = ((float)ir - (TSL2591_LUX_COEFB * (float)full)) / cpl;
  lux2 = ((TSL2591_LUX_COEFC * (float)ir) - (TSL2591_LUX_COEFD * (float)full)) /
         cpl;
  lux = (0 > lux1) ? 0 : (lux1 > lux2) ? lux1 : lux2;

  // Alternate lux calculation 1
  // See: https://github.com/adafruit/Adafruit_TSL2591_Library/issues/14
  // lux = (((float)ir - (float)full)) * (1.0F - ((float)full / (float)ir)) /
  // cpl;

  // Alternate lux calculation 2
  // lux = ((float)ir - (1.7F * (float)full)) / cpl;

  // Signal I2C had no errors
  return lux;
}
```

wifi setup.h

```
#ifndef WIFI_SETUP
#define WIFI_SETUP
#include "ThingSpeak.h"

#define DISCONNECTED "Disconnected!"
#define CONNECTED "Connected."

#if __has_include("local_secrets.h")
  #include "local_secrets.h"
#else
  #include "secrets.h"
#endif

#include "led_codes.h"
#include <WiFiNINA.h>





uint8_t getNetworkIndex(void);
void wifiSetup(void);
String getConnectionStatus(void);
void printArduinoMac(void);
void uploadData(String fmt_Str);
#endif /*WIFI_SETUP*/
```

wifi setup.cpp

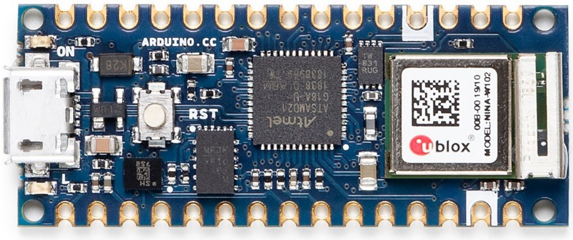
```cpp
#include "wifi_setup.h"

unsigned long tspeak_id = SECRET_CH_ID;
const char *tspeak_key = SECRET_WRITE_APIKEY;
const char *ssid_list[] = {SECRET_SSID ,HOT_SSID, WPI_SSID};
const char *pass_list[] = {SECRET_PASS ,HOT_PASS, WPI_PASS};

WiFiClient client;


uint8_t getNetworkIndex(void) {
  // scan for nearby networks:
  Serial.print("Scanning Known Networks...");

  int n_ssid = WiFi.scanNetworks();
  if (n_ssid == -1) {
    Serial.println("Couldn't get a WiFi connection");
    while (true)
      ledSignal(STUCK);
  }
  for (uint8_t this_net = 0; this_net < n_ssid; this_net++) {
    Serial.println(WiFi.SSID(this_net));
    for (const String &ssid : ssid_list) {
      ledSignal(SEARCHING);
      if ((String)WiFi.SSID(this_net) == ssid) {
        Serial.print("found ");
        Serial.println(ssid_list[this_net]);
        return this_net;
      }
    }
  }
  return 0xFF;
}

void wifiSetup(void) {
  if (WiFi.status() == WL_NO_MODULE) {
    Serial.println("Communication with WiFi module failed!");
    while (true)
      ledSignal(STUCK);
  }

  uint8_t i_network = getNetworkIndex();
  if (WiFi.status() != WL_CONNECTED) {
    Serial.print("attempting to connect...");
    Serial.println(ssid_list[i_network]);
    while (WiFi.status() != WL_CONNECTED) {
      ledSignal(SEARCHING);
      WiFi.begin(ssid_list[i_network], pass_list[i_network]);
      Serial.print("·");
    }
  }
  Serial.println("success!");
  ThingSpeak.begin(client); // Initialize ThingSpeak
}
```

```

String getConnectionStatus(void) {
  String status = (WiFi.status() != WL_CONNECTED) ? DISCONNECTED : CONNECTED;
  return status;
}

void uploadData(String fmt_Str) {
  ThingSpeak.writeField(tspeak_id, DATA_CH, fmt_Str, tspeak_key);
  ledSignal(UPLOAD);
}
void printArduinoMac(void) {
  byte mac[6];
  WiFi.macAddress(mac);
  Serial.print("Arduino MAC: ");
  Serial.print(mac[5],HEX);
  Serial.print(":");
  Serial.print(mac[4],HEX);
  Serial.print(":");
  Serial.print(mac[3],HEX);
  Serial.print(":");
  Serial.print(mac[2],HEX);
  Serial.print(":");
  Serial.print(mac[1],HEX);
  Serial.print(":");
  Serial.println(mac[0],HEX);
}
```

Appendix C: Datasheets

The following pages contain datasheets pertinent to the various components discussed throughout the report.

Product Reference Manual
SKU: ABX00027



Description

Nano 33 IoT is a miniature sized module containing a Cortex M0+ SAMD21 processor, a WiFi+BT module based on ESP32, a crypto chip which can securely store certificates and pre-shared keys and a 6 axis IMU. The module can either be mounted as a DIP component (when mounting pin headers), or as a SMT component, directly soldering it via the castellated pads.

Target areas:

Maker, enhancements, basic IoT application scenarios

Features

- **SAMD21G18A**
  - **Processor**
    - 256KB Flash
    - 32KB Flash
    - Power On Reset (POR) and Brown Out Detection (BOD)
  - **Peripherals**
    - 12 channel DMA
    - 12 channel event system
    - 5x 16 bit Timer/Counter
    - 3x 24 bit timer/counter with extended functions
    - 32 bit RTC
    - Watchdog Time
    - CRC-32 generator
    - Full speed Host/Device USB with 8 end points
    - 6x SERCOM (USART, I^{2}C, SPI, LIN)
    - Two channel I^{2}S
    - 12 bit 350ksps ADC (up to 16 bit with oversampling)
    - 10 bit 350ksps DAC
    - External Interrupt Controller (up to 16 lines)

- **Nina W102**
  - **Module**
    - Dual Core Tensilica LX6 CPU at up to 240MHz
    - 448 KB ROM, 520KB SRAM, 2MB Flash
  - **WiFi**
    - IEEE 802.11b up to 11Mbit
    - IEEE 802.11g up to 54MBit
    - IEEE 802.11n up to 72MBit
    - 2.4 GHz, 13 channels
    - 16dBm output power
    - 19 dBm EIRP
    - -96 dBm sensitivity
  - **Bluetooth BR/EDR**
    - Max 7 peripherals
    - 2.4 GHz, 79 channels
    - Up to 3 Mbit/s
    - 8 dBm output power at 2/3 Mbit/s
    - 11 dBm EIRP at 2/3 Mbit/s
    - -88 dBm sensitivity
  - **Bluetooth Low Energy**
    - Bluetooth 4.2 dual mode
    - 2.4GHz 40 channels
    - 6 dBm output power
    - 9 dBm EIRP
    - -88 dBm sensitivity
    - Up to 1 Mbit/
  - **MPM3610** (DC-DC)
    - Regulates input voltage from up to 21V with a minimum of 65% efficiency @minimum load
    - More than 85% efficiency @12V
  - **ATECC608A** (Crypto Chip)
    - Cryptographic co-processor with secure hardware based key storage
    - Protected storage for up to 16 keys, certificates or data
    - ECDH: FIPS SP800-56A Elliptic Curve Diffie-Hellman
    - NIST standard P256 elliptic curve support
    - SHA-256 & HMAC hash including off-chip context save/restore
    - AES-128 encrypt/decrypt, galois field multiply for GCM

- **LSM6DSL** (6 axis IMU)
  - Always-on 3D accelerometer and 3D gyroscope
  - Smart FIFO up to 4 KByte based
  - ±2/±4/±8/±16 g full scale
  - ±125/±250/±500/±1000/±2000 dps full scale

Contents

1 The Board **6**

1.1 Application Examples 6

2 Ratings **6**

2.1 Recommended Operating Conditions 6

2.2 Power Consumption 6

3 Functional Overview **7**

3.1 Board topology 7

3.2 Processor 8

3.3 WiFi/BT Communication Module 8

3.4 Crypto 9

3.5 IMU 9

3.6 Power Tree 9

4 Board Operation **10**

4.1 Getting started - IDE 10

4.2 Getting started - Arduino Web Editor 10

4.3 Getting started - Arduino IoT Cloud 10

4.4 Sample Sketches 10

4.5 Online Resources 10

4.6 Board Recovery 11

5 Connector Pinouts **11**

5.1 USB 12

5.2 Headers 12

5.3 Debug 13

6 Mechanical Information **13**

6.1 Board Outline and Mounting Holes 13

6.2 Connector Positions 14

7 Certifications **15**

7.1 Declaration of Conformity CE DoC (EU) 15

7.2 Declaration of Conformity to EU RoHS & REACH 211 01/19/2021 15

7.3 Conflict Minerals Declaration 16

8 FCC Caution **16**

9 Company Information **17**

10 Reference Documentation **17**

11 Revision History **18**

1 The Board

As all Nano form factor boards, Nano 33 IoT does not have a battery charger but can be powered through USB or headers.

NOTE: Arduino Nano 33 IoT only supports 3.3V I/Os and is **NOT** 5V tolerant so please make sure you are not directly connecting 5V signals to this board or it will be damaged. Also, as opposed to Arduino Nano boards that support 5V operation, the 5V pin does NOT supply voltage but is rather connected, through a jumper, to the USB power input.

1.1 Application Examples

Weather station: Using the Arduino Nano 33 IoT together with a sensor and a OLED display, we can create a small weather station communicating temperature, humidity etc. directly to your phone.

Air quality monitor: Bad air quality may have serious effects on your health. By assembling the Nano 33 IoT, with a sensor and monitor you can make sure that the air quality is kept in indoor-environments. By connecting the hardware assembly to an IoT application/API, you will receive real time values.

Air drum: A quick and fun project is to create a small air drum. Connect your Nano 33 IoT and upload your sketch from the Create Web Editor and start creating beats with your audio workstation of your choice.

2 Ratings

2.1 Recommended Operating Conditions

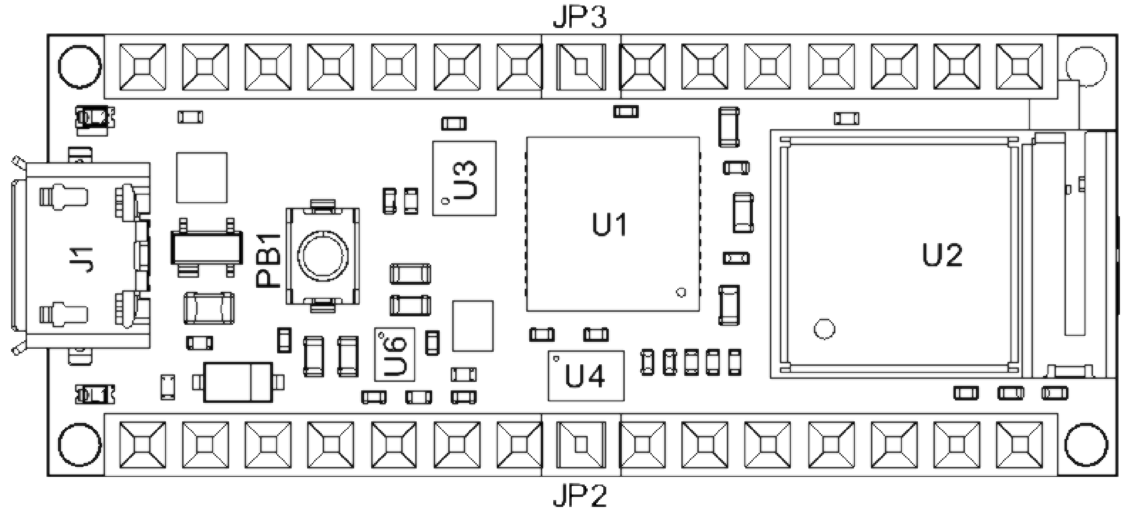
Symbol	Description	Min	Max
	Conservative thermal limits for the whole board:	-40 °C (40 °F)	85°C (185 °F)

2.2 Power Consumption

Symbol	Description	Min	Typ	Max	Unit
VINMax	Maximum input voltage from VIN pad	-0.3	-	21	V
VUSBMax	Maximum input voltage from USB connector	-0.3	-	21	V
PMax	Maximum Power Consumption	-	-	TBC	mW

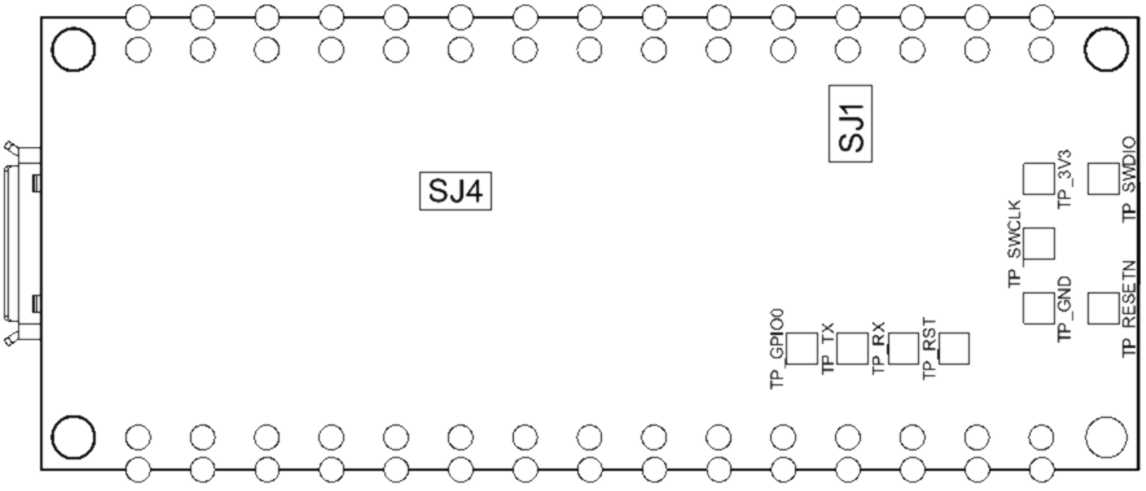
3 Functional Overview

3.1 Board topology



Board topology top

Ref.	Description	Ref.	Description
U1	ATSAMD21G18A Controller	U3	LSM6DSOXTR IMU Sensor
U2	NINA-W102-00B WiFi/BLE Module	U4	ATECC608A-MAHDA-T Crypto Chip
J1	Micro USB Connector	PB1	IT-1185-160G-GTR Push button



Board topology bottom

Ref.	Description	Ref.	Description
SJ1	Open solder bridge (VUSB)	SJ4	Closed solder bridge (+3V3)
TP	Test points	xx	Lorem Ipsum

3.2 Processor

The Main Processor is a Cortex M0+ running at up to 48MHz. Most of its pins are connected to the external headers, however some are reserved for internal communication with the wireless module and the on-board internal I^{2}C peripherals (IMU and Crypto).

NOTE: As opposed to other Arduino Nano boards, pins A4 and A5 have an internal pull up and default to be used as an I^{2}C Bus so usage as analog inputs is not recommended.

Communication with NINA W102 happens through a serial port and a SPI bus through the following pins.

SAMD21 Pin	SAMD21 Acronym	NINA Pin	NINA Acronym	Description
13	PA08	19	RESET_N	Reset
39	PA27	27	GPIO0	Attention Request
41	PA28	7	GPIO33	Acknowledge
23	PA14	28	GPIO5	SPI CS
21	GPIO19	UART RTS		
24	PA15	29	GPIO18	SPI CLK
20	GPIO22	UART CTS		
22	PA13	1	GPIO21	SPI MISO
21	PA12	36	GPIO12	SPI MOSI
31	PA22	23	GPIO3	Processor TX Nina RX
32	PA23	22	GPIO1	Processor RX Nina TX

3.3 WiFi/BT Communication Module

Nina W102 is based on ESP32 and is delivered with a pre-certified software stack from Arduino. Source code for the firmware is available **[9]**.

NOTE: Reprogramming the wireless module's firmware with a custom one will invalidate compliance with radio standards as certified by Arduino, hence this is not recommended unless the application is used in private laboratories far from other electronic equipment and people. Usage of custom firmware on radio modules is the sole responsibility of the user.

Some of the module's pins are connected to the external headers and can be directly driven by ESP32 provided SAMD21's corresponding pins are aptly tri-stated. Below is a list of such signals:

SAMD21 Pin	SAMD21 Acronym	NINA Pin	NINA Acronym	Description
48	PB03	8	GPIO21	A7
14	PA09	5	GPIO32	A6
8	PB09	31	GPIO14	A5/SCL
7	PB08	35	GPIO13	A4/SDA

3.4 Crypto

The crypto chip in Arduino IoT boards is what makes the difference with other less secure boards as it provides a secure way to store secrets (such as certificates) and accelerates secure protocols while never exposing secrets in plain text.

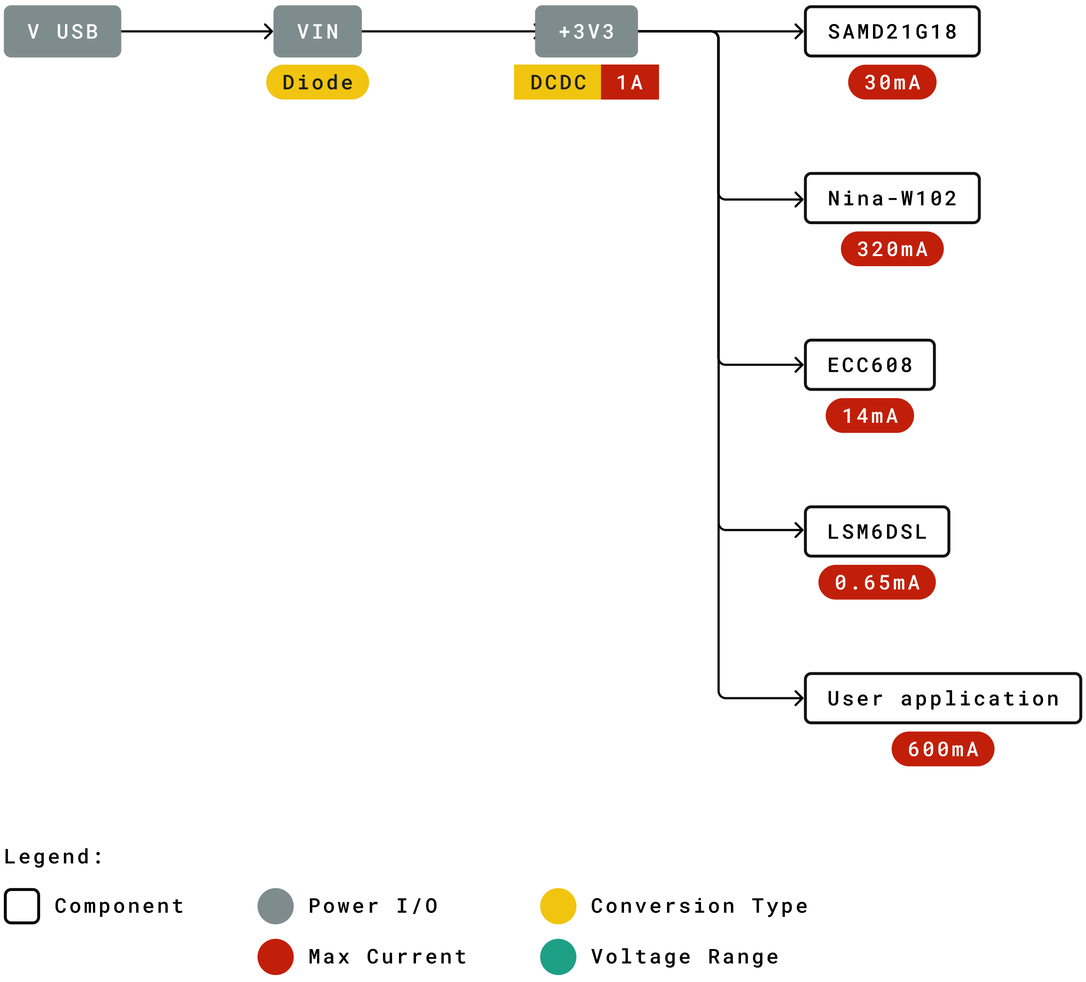
Source code for the Arduino Library that supports the Crypto is available **[10]**

3.5 IMU

Arduino Nano 33 IoT has an embedded 6 axis IMU which can be used to measure board orientation (by checking the gravity acceleration vector orientation) or to measure shocks, vibration, acceleration and rotation speed.

Source code for the Arduino Library that supports the IMU is available **[11]**

3.6 Power Tree



Power tree

4 Board Operation

4.1 Getting started - IDE

If you want to program your Arduino 33 IoT while offline you need to install the Arduino Desktop IDE [1] To connect the Arduino 33 IoT to your computer, you'll need a Micro-B USB cable. This also provides power to the board, as indicated by the LED.

4.2 Getting started - Arduino Web Editor

All Arduino boards, including this one, work out-of-the-box on the Arduino Web Editor [2], by just installing a simple plugin.

The Arduino Web Editor is hosted online, therefore it will always be up-to-date with the latest features and support for all boards. Follow **[3]** to start coding on the browser and upload your sketches onto your board.

4.3 Getting started - Arduino IoT Cloud

All Arduino IoT enabled products are supported on Arduino IoT Cloud which allows you to Log, graph and analyze sensor data, trigger events, and automate your home or business.

4.4 Sample Sketches

Sample sketches for the Arduino 33 IoT can be found either in the "Examples" menu in the Arduino IDE or in the "Documentation" section of the Arduino Pro website [4]

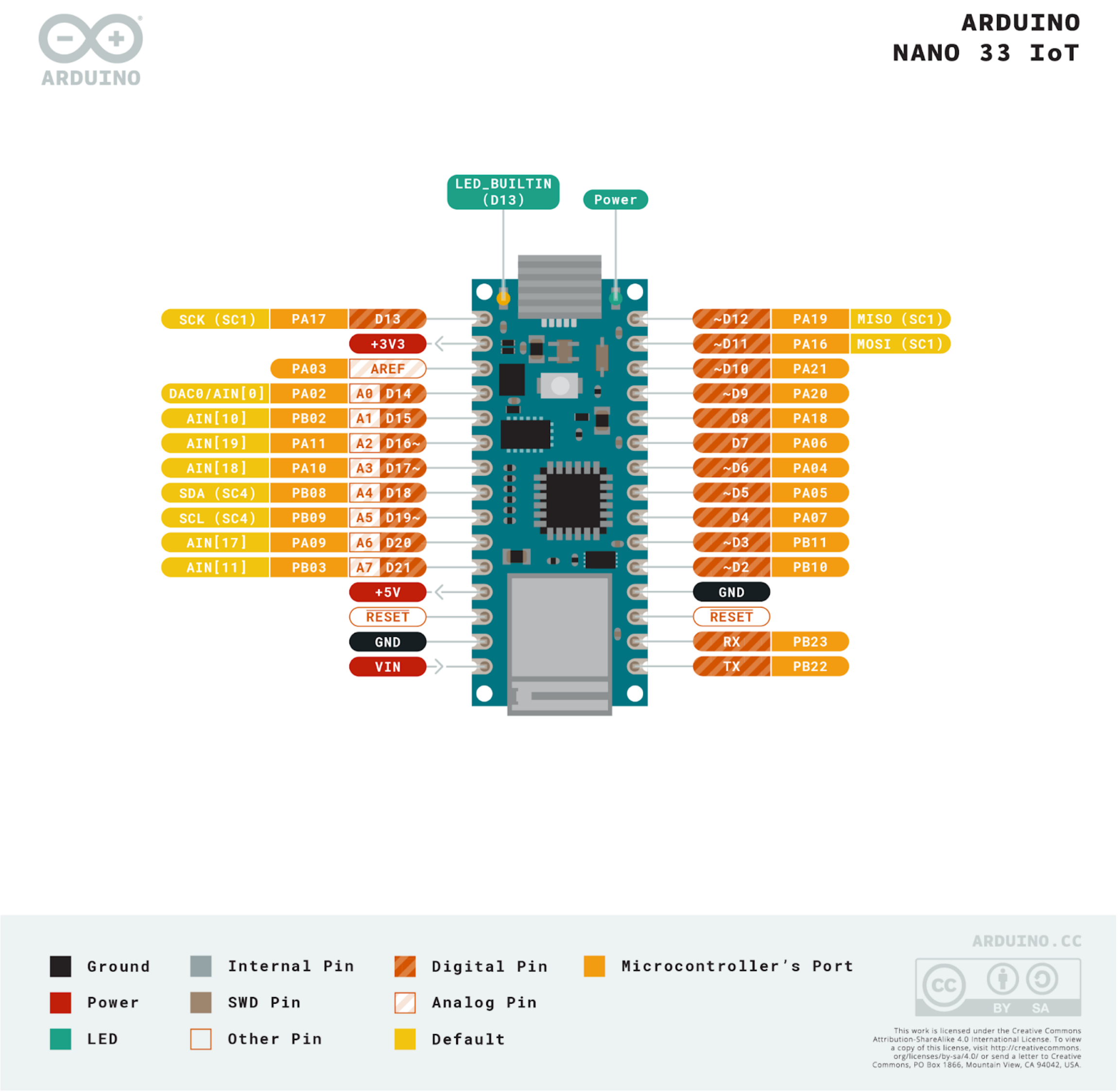
4.5 Online Resources

Now that you have gone through the basics of what you can do with the board you can explore the endless possibilities it provides by checking exciting projects on ProjectHub **[5]**, the Arduino Library Reference **[6]** and the online store **[7]** where you will be able to complement your board with sensors, actuators and more

4.6 Board Recovery

All Arduino boards have a built-in bootloader which allows flashing the board via USB. In case a sketch locks up the processor and the board is not reachable anymore via USB it is possible to enter bootloader mode by double-tapping the reset button right after power up.

5 Connector Pinouts



Pinout

5.1 USB

Pin	Function	Type	Description
1	VUSB	Power	Power Supply Input. If board is powered via VUSB from header this is an Output **(1)**
2	D-	Differential	USB differential data -
3	D+	Differential	USB differential data +
4	ID	Analog	Selects Host/Device functionality
5	GND	Power	Power Ground

1. The board can support USB host mode only if powered via the V_{USB} pin and if the jumper close to the VUSB pin is shorted.

5.2 Headers

The board exposes two 15 pin connectors which can either be assembled with pin headers or soldered through castellated vias.

Pin	Function	Type	Description
1	D13	Digital	GPIO
2	+3V3	Power Out	Internally generated power output to external devices
3	AREF	Analog	Analog Reference; can be used as GPIO
4	A0/DAC0	Analog	ADC in/DAC out; can be used as GPIO
5	A1	Analog	ADC in; can be used as GPIO
6	A2	Analog	ADC in; can be used as GPIO
7	A3	Analog	ADC in; can be used as GPIO
8	A4/SDA	Analog	ADC in; I2C SDA; Can be used as GPIO **(1)**
9	A5/SCL	Analog	ADC in; I2C SCL; Can be used as GPIO **(1)**
10	A6	Analog	ADC in; can be used as GPIO
11	A7	Analog	ADC in; can be used as GPIO
12	VUSB	Power In/Out	Normally NC; can be connected to VUSB pin of the USB connector by shorting a jumper
13	RST	Digital In	Active low reset input (duplicate of pin 18)
14	GND	Power	Power Ground
15	VIN	Power In	Vin Power input
16	TX	Digital	USART TX; can be used as GPIO
17	RX	Digital	USART RX; can be used as GPIO
18	RST	Digital	Active low reset input (duplicate of pin 13)
19	GND	Power	Power Ground
20	D2	Digital	GPIO
21	D3/PWM	Digital	GPIO; can be used as PWM
22	D4	Digital	GPIO
23	D5/PWM	Digital	GPIO; can be used as PWM
24	D6/PWM	Digital	GPIO, can be used as PWM
25	D7	Digital	GPIO
26	D8	Digital	GPIO

Pin	Function	Type	Description
27	D9/PWM	Digital	GPIO; can be used as PWM
28	D10/PWM	Digital	GPIO; can be used as PWM
29	D11/MOSI	Digital	SPI MOSI; can be used as GPIO
30	D12/MISO	Digital	SPI MISO; can be used as GPIO

5.3 Debug

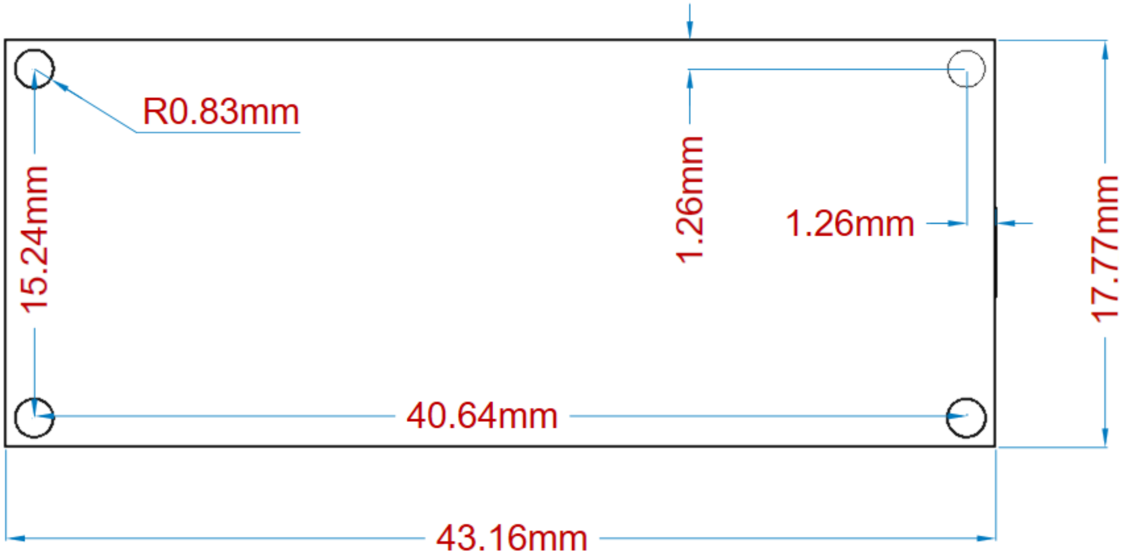
On the bottom side of the board, under the communication module, debug signals are arranged as 3x2 test pads with 100 mil pitch. Pin 1 is depicted in Figure 3 – Connector Positions

Pin	Function	Type	Description
1	+3V3	Power Out	Internally generated power output to be used as voltage reference
2	SWD	Digital	SAMD11 Single Wire Debug Data
3	SWCLK	Digital In	SAMD11 Single Wire Debug Clock
4	UPDI	Digital	ATMega4809 update interface
5	GND	Power	Power Ground
6	RST	Digital In	Active low reset input

6 Mechanical Information

6.1 Board Outline and Mounting Holes

The board measures are mixed between metric and imperial. Imperial measures are used to maintain a 100 mil pitch grid between pin rows to allow them to fit a breadboard whereas board length is Metric.

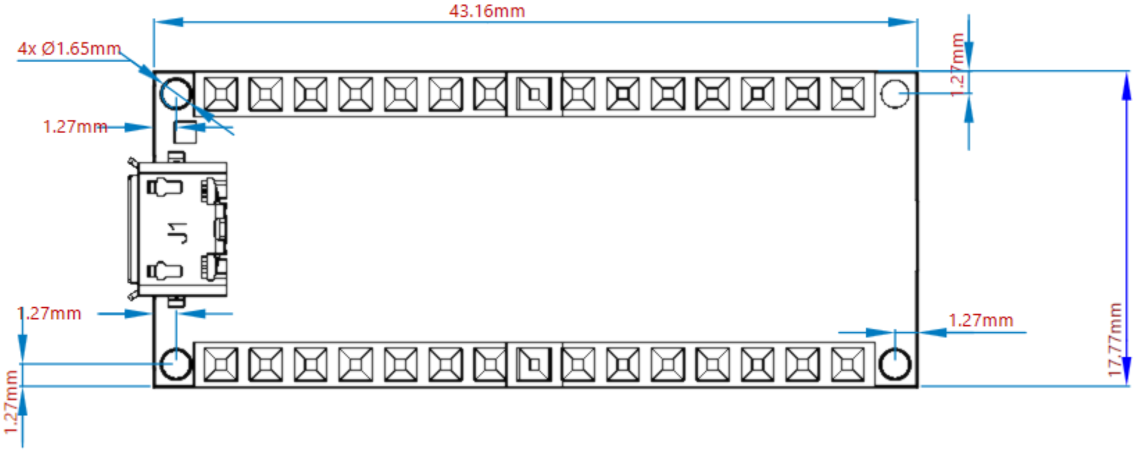


Layout

6.2 Connector Positions

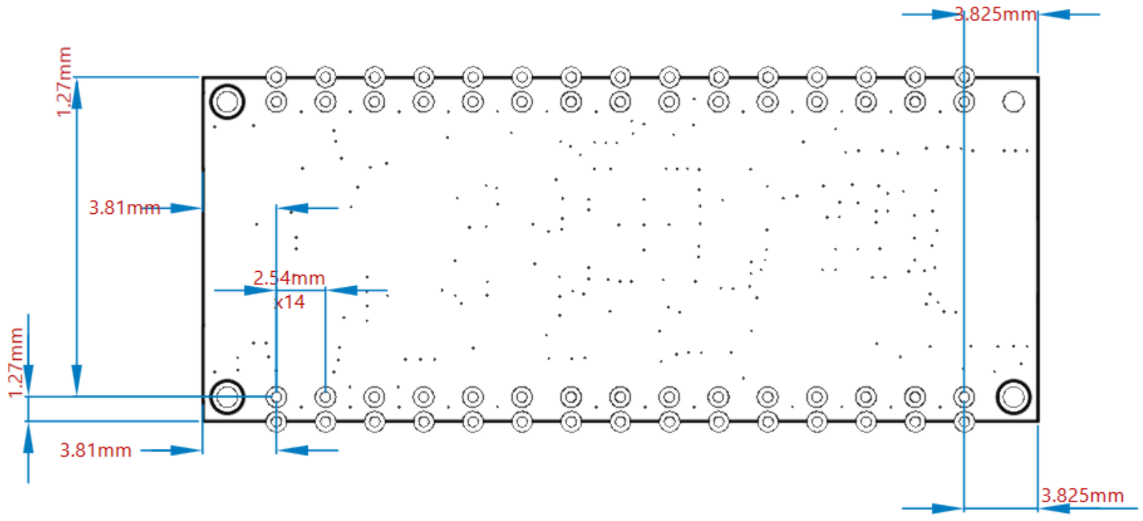
The view below is from top however it shows Debug connector pads which are on the bottom side. Highlighted pins are pin 1 for each connector'

Top view:



Top side connectors

Bottom view:



Bottom side connectors

7 Certifications

7.1 Declaration of Conformity CE DoC (EU)

We declare under our sole responsibility that the products above are in conformity with the essential requirements of the following EU Directives and therefore qualify for free movement within markets comprising the European Union (EU) and European Economic Area (EEA).

7.2 Declaration of Conformity to EU RoHS & REACH 211 01/19/2021

Arduino boards are in compliance with RoHS 2 Directive 2011/65/EU of the European Parliament and RoHS 3 Directive 2015/863/EU of the Council of 4 June 2015 on the restriction of the use of certain hazardous substances in electrical and electronic equipment.

Substance	**Maximum limit (ppm)**
Lead (Pb)	1000
Cadmium (Cd)	100
Mercury (Hg)	1000
Hexavalent Chromium (Cr6+)	1000
Poly Brominated Biphenyls (PBB)	1000
Poly Brominated Diphenyl ethers (PBDE)	1000
Bis(2-Ethylhexyl} phthalate (DEHP)	1000
Benzyl butyl phthalate (BBP)	1000
Dibutyl phthalate (DBP)	1000
Diisobutyl phthalate (DIBP)	1000

Exemptions : No exemptions are claimed.

Arduino Boards are fully compliant with the related requirements of European Union Regulation (EC) 1907 /2006 concerning the Registration, Evaluation, Authorization and Restriction of Chemicals (REACH). We declare none of the SVHCs (https://echa.europa.eu/web/guest/candidate-list-table), the Candidate List of Substances of Very High Concern for authorization currently released by ECHA, is present in all products (and also package) in quantities totaling in a concentration equal or above 0.1%. To the best of our knowledge, we also declare that our products do not contain any of the substances listed on the "Authorization List" (Annex XIV of the REACH regulations) and Substances of Very High Concern (SVHC) in any significant amounts as specified by the Annex XVII of Candidate list published by ECHA (European Chemical Agency) 1907 /2006/EC.

7.3 Conflict Minerals Declaration

As a global supplier of electronic and electrical components, Arduino is aware of our obligations with regards to laws and regulations regarding Conflict Minerals, specifically the Dodd-Frank Wall Street Reform and Consumer Protection Act, Section 1502. Arduino does not directly source or process conflict minerals such as Tin, Tantalum, Tungsten, or Gold. Conflict minerals are contained in our products in the form of solder, or as a component in metal alloys. As part of our reasonable due diligence Arduino has contacted component suppliers within our supply chain to verify their continued compliance with the regulations. Based on the information received thus far we declare that our products contain Conflict Minerals sourced from conflict-free areas.

8 FCC Caution

Any Changes or modifications not expressly approved by the party responsible for compliance could void the user's authority to operate the equipment.

This device complies with part 15 of the FCC Rules. Operation is subject to the following two conditions:

(1) This device may not cause harmful interference

(2) this device must accept any interference received, including interference that may cause undesired operation.

FCC RF Radiation Exposure Statement:

1. This Transmitter must not be co-located or operating in conjunction with any other antenna or transmitter.
2. This equipment complies with RF radiation exposure limits set forth for an uncontrolled environment.
3. This equipment should be installed and operated with minimum distance 20cm between the radiator & your body.

English: User manuals for license-exempt radio apparatus shall contain the following or equivalent notice in a conspicuous location in the user manual or alternatively on the device or both. This device complies with Industry Canada license-exempt RSS standard(s). Operation is subject to the following two conditions:

(1) this device may not cause interference

(2) this device must accept any interference, including interference that may cause undesired operation of the device.

French: Le présent appareil est conforme aux CNR d'Industrie Canada applicables aux appareils radio exempts de licence. L'exploitation est autorisée aux deux conditions suivantes :

(1) l' appareil nedoit pas produire de brouillage

(2) l'utilisateur de l'appareil doit accepter tout brouillage radioélectrique subi, même si le brouillage est susceptible d'en compromettre le fonctionnement.

IC SAR Waring:

English This equipment should be installed and operated with minimum distance 20 cm between the radiator and your body.

French: Lors de l' installation et de l' exploitation de ce dispositif, la distance entre le radiateur et le corps est d 'au moins 20 cm.

Important: The operating temperature of the EUT can't exceed 85℃ and shouldn't be lower than -40℃.

Hereby, Arduino S.r.l. declares that this product is in compliance with essential requirements and other relevant provisions of Directive 2014/53/EU. This product is allowed to be used in all EU member states.

Frequency bands	Maximum output power (ERP)
863-870Mhz	-3.22dBm

9 Company Information

Company name	Arduino SA.
Company Address	Via Ferruccio Pelli 14 6900 Lugano Switzerland

10 Reference Documentation

Reference	Link
Arduino IDE (Desktop)	https://www.arduino.cc/en/Main/Software
Arduino IDE (Cloud)	https://create.arduino.cc/editor
Cloud IDE Getting Started	https://create.arduino.cc/projecthub/Arduino_Genuino/getting-started-with-arduino-web-editor-4b3e4a
Forum	http://forum.arduino.cc/
SAMD21G18	http://ww1.microchip.com/downloads/en/devicedoc/40001884a.pdf
NINA W102	https://www.u-blox.com/sites/default/files/NINA-W10_DataSheet_%28UBX-17065507%29.pdf
ECC608	http://ww1.microchip.com/downloads/en/DeviceDoc/40001977A.pdf
MPM3610	https://www.monolithicpower.com/pub/media/document/MPM3610_r1.01.pdf
NINA Firmware	https://github.com/arduino/nina-fw
ECC608 Library	https://github.com/arduino-libraries/ArduinoECCX08
LSM6DSL Library	https://github.com/stm32duino/LSM6DSL
ProjectHub	https://create.arduino.cc/projecthub?by=part&part_id=11332&sort=trending
Library Reference	https://www.arduino.cc/reference/en/
Arduino Store	https://store.arduino.cc/

11 Revision History

Date	Revision	Changes
04/15/2021	1	General datasheet updates

TSL2591

Light-to-Digital Converter

General Description

The TSL2591 is a very-high sensitivity light-to-digital converter that transforms light intensity into a digital signal output capable of direct I^2C interface. The device combines one broadband photodiode (visible plus infrared) and one infrared-responding photodiode on a single CMOS integrated circuit. Two integrating ADCs convert the photodiode currents into a digital output that represents the irradiance measured on each channel. This digital output can be input to a microprocessor where illuminance (ambient light level) in lux is derived using an empirical formula to approximate the human eye response. The TSL2591 supports a traditional level style interrupt that remains asserted until the firmware clears it.

Ordering Information and Content Guide appear at end of datasheet.

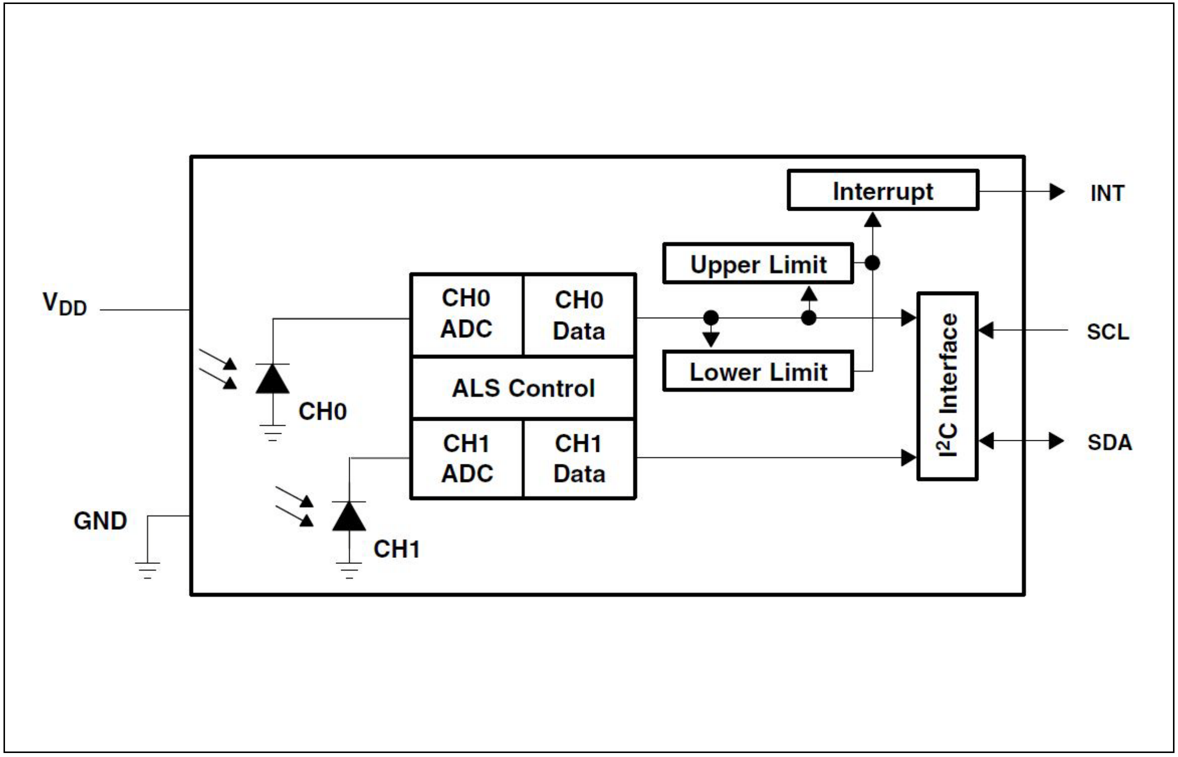
Figure 1:
Added Value of Using TSL2591

Benefits	Features
• Approximates Human Eye Response	• Dual Diode
• Flexible Operation	• Programmable Analog Gain and Integration Time
• Suited for Operation Behind Dark Glass	• 600M:1 Dynamic Range
• Low Operating Overhead	• Two Internal Interrupt Sources • Programmable Upper and Lower Thresholds • One Interrupt Includes Programmable Persistence Filter
• Low Power 3.0 µA Sleep State	• User Selectable Sleep Mode
• I^2C Fast Mode Compatible Interface	• Data Rates up to 400 kbit/s • Input Voltage Levels Compatible with 3.0V Bus

Block Diagram

The functional blocks of this device are shown below:

Figure 2:
Block Diagram



Detailed Description

The TSL2591 contains two integrating analog-to-digital converters (ADC) that integrate currents from two photodiodes. Integration of both channels occurs simultaneously. Upon completion of the conversion cycle, the conversion result is transferred to the Channel 0 and Channel 1 data registers, respectively. The transfers are double-buffered to ensure that the integrity of the data is maintained. After the transfer, the device automatically begins the next integration cycle.

Communication with the device is accomplished through a standard, two-wire I²C serial bus. Consequently, the TSL2591 can be easily connected to a microcontroller or embedded controller. No external circuitry is required for signal conditioning. Because the output of the device is digital, the output is effectively immune to noise when compared to an analog signal.

The TSL2591 also supports an interrupt feature that simplifies and improves system efficiency by eliminating the need to poll a sensor for a light intensity value. The primary purpose of the interrupt function is to detect a meaningful change in light intensity. The concept of a meaningful change can be defined by the user both in terms of light intensity and time, or persistence, of that change in intensity. The device has the ability to define two sets of thresholds, both above and below the current light level. An interrupt is generated when the value of a conversion exceeds either of these limits. One set of thresholds can be configured to trigger an interrupt only when the ambient light exceeds them for a configurable amount of time (persistence) while the other set can be configured to trigger an immediate interrupt.

Pin Assignment

The TSL2591 pin assignments are described below.

Figure 3:
Pin Diagram

Package FN Dual Flat No-Lead (Top View): Package drawing is not to scale.

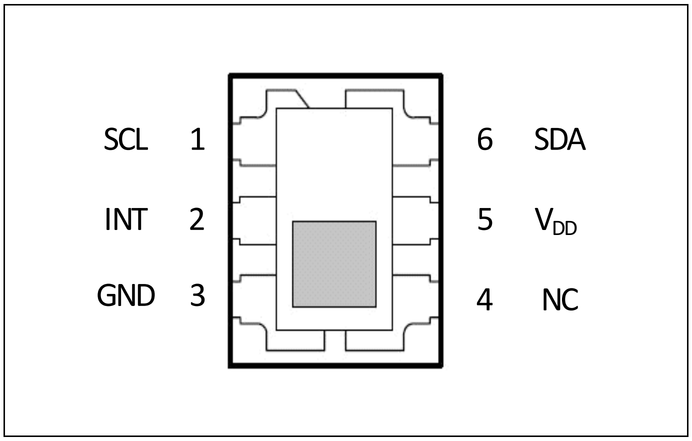


Figure 4:
Pin Description

Pin Number	Pin Name	Description
1	SCL	I^{2}C serial clock input terminal
2	INT	Interrupt — open drain output (active low).
3	GND	Power supply ground. All voltages are referenced to GND.
4	NC	No connect — do not connect.
5	V_{DD}	Supply voltage
6	SDA	I^{2}C serial data I/O terminal

Absolute Maximum Ratings

Stresses beyond those listed under Absolute Maximum Ratings may cause permanent damage to the device. These are stress ratings only. Functional operation of the device at these or any other conditions beyond those indicated under Recommended Operating Conditions is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

Figure 5:
Absolute Maximum Ratings

Parameter	Min	Max	Units	Comments
Supply voltage, V_{DD}		3.8	V	All voltages are with respect to GND
Input terminal voltage	-0.5	3.8	V	
Output terminal voltage	-0.5	3.8	V	
Output terminal current	-1	20	mA	
Storage temperature range, T_{stg}	-40	85	°C	
ESD tolerance, human body model	±2000		V	JESD22-A114-B
ESD tolerance, charge device model (CDM)	±500		V	JESD22-C101

Electrical Characteristics

All limits are guaranteed. The parameters with min and max values are guaranteed with production tests or SQC (Statistical Quality Control) methods. Device parameters are guaranteed at T_A = 25°C unless otherwise noted.

Figure 6:
Recommended Operating Conditions

Symbol	Parameter	Min	Typ	Max	Units
V_{DD}	Supply voltage	2.7	3	3.6	V
T_A	Operating free-air temperature	-30		70	°C

Figure 7:
Operating Characteristics, V_{DD}=3V, T_A=25°C (unless otherwise noted)

Symbol	Parameter	Conditions	Min	Typ	Max	Units
I_{DD}	Supply current	Active Sleep state - no I^2C activity		275 2.3	325 4	µA
V_{OL}	INT, SDA output low voltage	3mA sink current 6mA sink current	0 0		0.4 0.6	V
I_{LEAK}	Leakage current, SDA, SCL, INT pins		-5		5	µA
V_{IH}	SCL, SDA input high voltage	TSL25911 ($V_{bus} = V_{DD}$)	0.7 V_{DD}			V
		TSL25913 (V_{bus} = 1.8)	1.26			
V_{IL}	SCL, SDA input low voltage	TSL25911 ($V_{bus} = V_{DD}$)			0.3 V_{DD}	V
		TSL25913 (V_{bus} = 1.8)			0.54	

Figure 8:
ALS Characteristics, V_{DD}=3V, T_A=25°C, AGAIN = High, AEN=1, (unless otherwise noted)[1] [2] [3]

Parameter	Conditions	Channel	Min	Typ	Max	Units
Dark ADC count value	E_e = 0, AGAIN = Max, ATIME=000b (100ms)	CH0 CH1	0 0		20 20	counts
ADC integration time step size	ATIME = 000b (100ms)		95	100	105	ms
ADC number of integration steps [4]			1		6	steps
Max ADC count	ATIME = 000b (100ms)		0		36863	counts
Max ADC count	ATIME = 001b (200ms), 010b (300ms), 011b (400ms), 100b (500ms), 101b (600ms)		0		65535	counts
ADC count value	White light [2] E_e = 4.98 µW/cm^2 ATIME = 000b (100 ms)	CH0 CH1	1120	1315 174	1510	counts
	λ_p = 850 nm [3] E_e = 5.62 µW/cm^2, ATIME = 000b (100 ms)	CH0 CH1	1230	1447 866	1665	counts
ADC count value ratio: CH1/CH0	White light [2]		0.092	0.132	0.172	
	λ_p = 850 nm [3]		0.558	0.598	0.638	
R_e Irradiance responsivity	White light [2] ATIME = 000b (100 ms)	CH0 CH1		264.1 34.9		counts/ (µW/cm^2)
	λ_p = 850 nm [3] ATIME = 000b (100 ms)	CH0 CH1		257.5 154.1		
Noise [4]	White light [2] E_e = 4.98 µW/cm^2 ATIME = 000b (100 ms)	CH0		1	2	1 standard deviation

Parameter	Conditions	Channel	Min	Typ	Max	Units
Gain scaling, relative to 1× gain setting (AGAIN = Low)	AGAIN = Med	CH0 CH1	22 22	24.5 24.5	27 27	×
	AGAIN = High	CH0 CH1	360 360	400 400	440 440	
	AGAIN = Max	CH0 CH1	8500 9100	9200 9900	9900 10700	

Note(s):

1. Optical measurements are made using small-angle incident radiation from light-emitting diode optical sources. Visible white LEDs and infrared 850 nm LEDs are used for final product testing for compatibility with high-volume production
2. The white LED irradiance is supplied by a white light-emitting diode with a nominal color temperature of 4000 K.
3. The 850 nm irradiance is supplied by a GaAs light-emitting diode with the following typical characteristics: peak wavelength λ_p = 850 nm and spectral halfwidth Δλ½ = 42 nm.
4. Parameter ensured by design and is not 100% tested.

Timing Characteristics

The timing characteristics of TSL2591 are given below.

Figure 9:
AC Electrical Characteristics, V_{DD} = 3 V, T_A = 25ºC (unless otherwise noted)

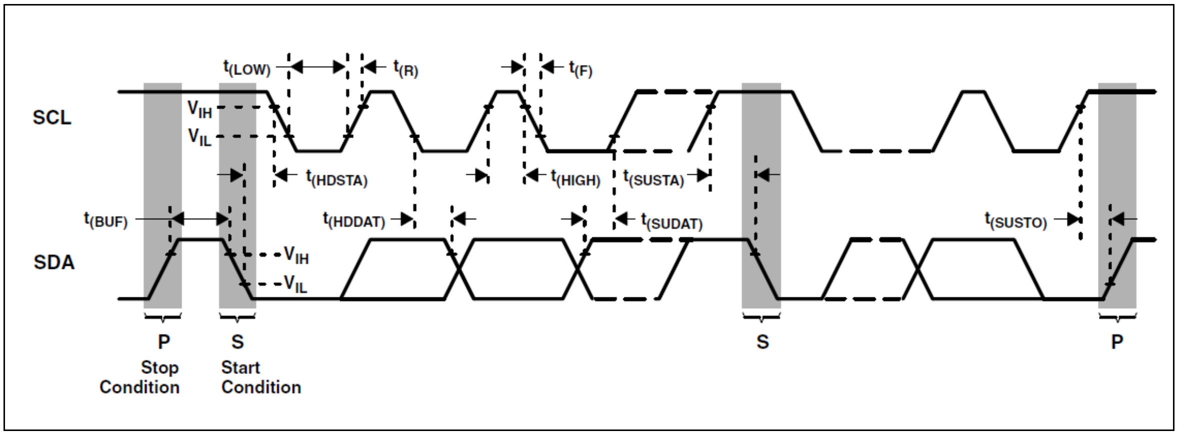
Parameter [1]	Description	Min	Typ	Max	Units
$f_{(SCL)}$	Clock frequency (I²C only)	0		400	kHz
$t_{(BUF)}$	Bus free time between start and stop condition	1.3			µs
$t_{(HDSTA)}$	Hold time after (repeated) start condition. After this period, the first clock is generated.	0.6			µs
$t_{(SUSTA)}$	Repeated start condition setup time	0.6			µs
$t_{(SUSTO)}$	Stop condition setup time	0.6			µs
$t_{(HDDAT)}$	Data hold time	0			µs
$t_{(SUDAT)}$	Data setup time	100			ns
$t_{(LOW)}$	SCL clock low period	1.3			µs
$t_{(HIGH)}$	SCL clock high period	0.6			µs
t_F	Clock/data fall time			300	ns
t_R	Clock/data rise time			300	ns
C_i	Input pin capacitance			10	pF

Note(s):

1. Specified by design and characterization; not production tested.

Timing Diagrams

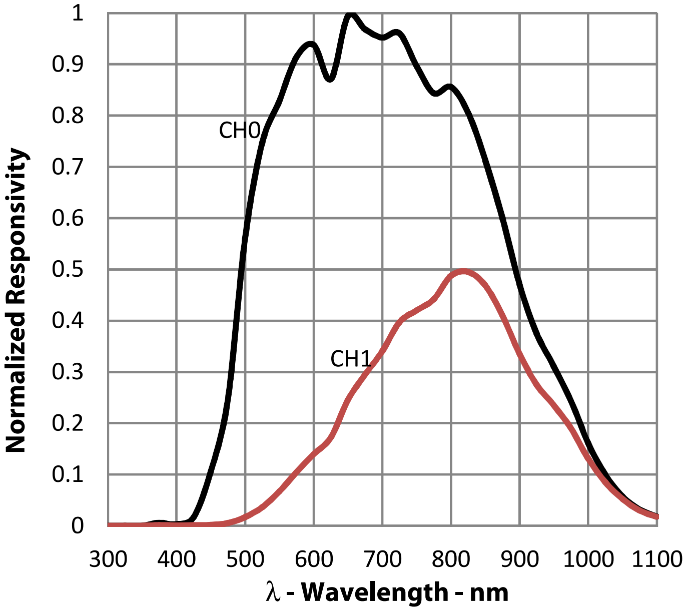
Figure 10:
Parameter Measurement Information



Typical Operating Characteristics

Spectral Responsivity: Two channel response allows for tunable illuminance (lux) calculation regardless of transmissivity of glass.

Figure 11:
Spectral Responsivity



White LED Angular Response: Near cosine angular response for broadband white light sources.

Figure 12:
White Normalized Responsivity vs. Angular Displacement

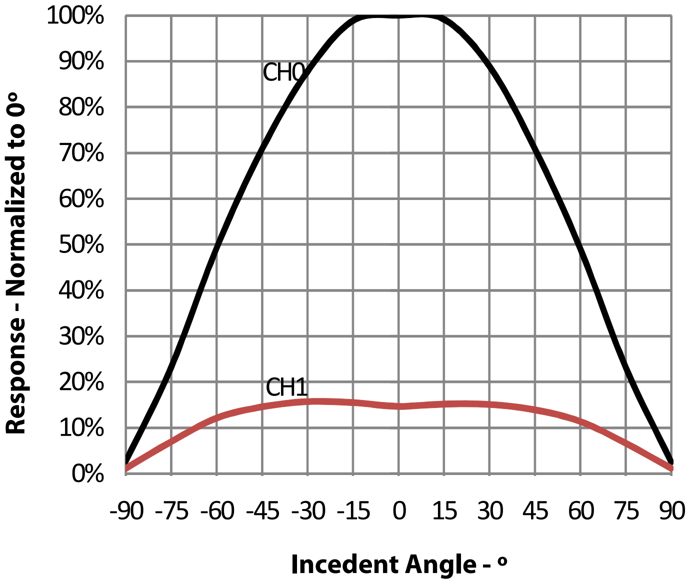


Figure 13:
Normalized I_{DD} vs. V_{DD} and Temperature

I_{DD} vs. V_{DD} and Temp: Effect of supply voltage and temperature on active current.

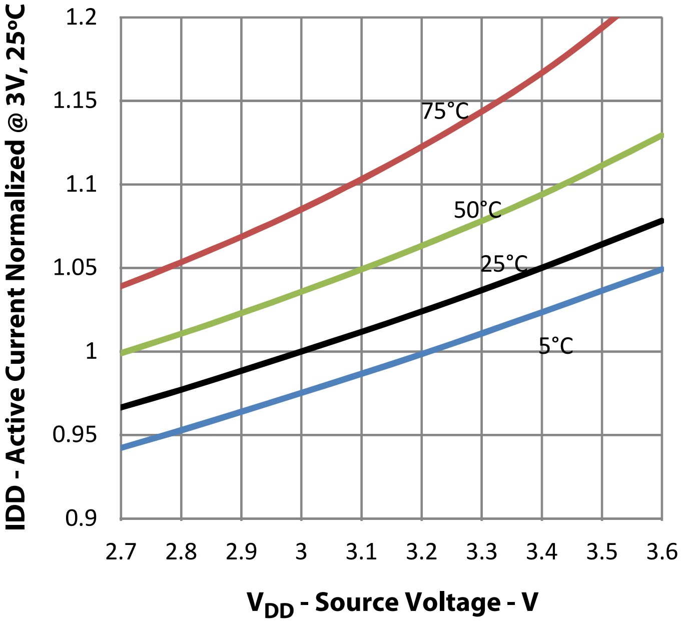
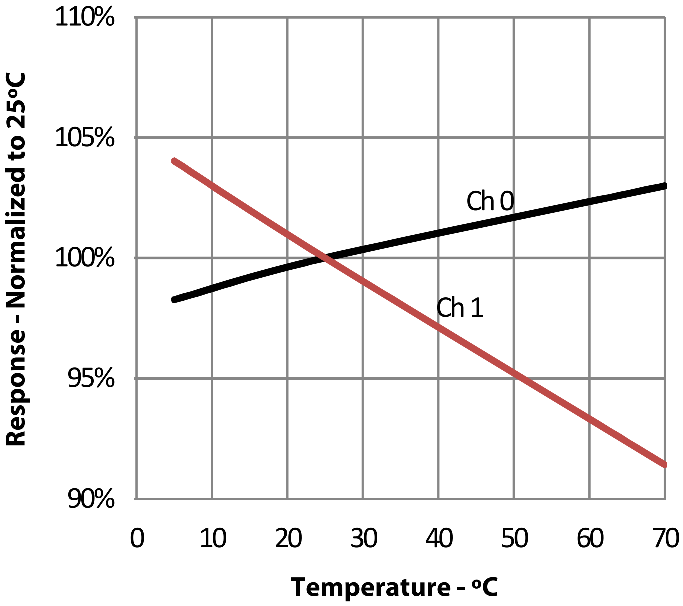


Figure 14:
Response to White LED vs. Temperature

White LED Response vs. Temp: Effect of temperature on the device response for a broadband white light source.



Register Description

The device is controlled and monitored by registers accessed through the I²C serial interface. These registers provide for a variety of control functions and can be read to determine results of the ADC conversions. The register set is summarized in Figure 15.

Figure 15:
Register Description

Address	Register Name	R/W	Register Function	Reset Value
--	COMMAND	W	Specifies Register Address	0x00
0x00	ENABLE	R/W	Enables states and interrupts	0x00
0x01	CONFIG	R/W	ALS gain and integration time configuration	0x00
0x04	AILTL	R/W	ALS interrupt low threshold low byte	0x00
0x05	AILTH	R/W	ALS interrupt low threshold high byte	0x00
0x06	AIHTL	R/W	ALS interrupt high threshold low byte	0x00
0x07	AIHTH	R/W	ALS interrupt high threshold high byte	0x00
0x08	NPAILTL	R/W	No Persist ALS interrupt low threshold low byte	0x00
0x09	NPAILTH	R/W	No Persist ALS interrupt low threshold high byte	0x00
0x0A	NPAIHTL	R/W	No Persist ALS interrupt high threshold low byte	0x00
0x0B	NPAIHTH	R/W	No Persist ALS interrupt high threshold high byte	0x00
0x0C	PERSIST	R/W	Interrupt persistence filter	0x00
0x11	PID	R	Package ID	--
0x12	ID	R	Device ID	ID
0x13	STATUS	R	Device status	0x00
0x14	C0DATAL	R	CH0 ADC low data byte	0x00
0x15	C0DATAH	R	CH0 ADC high data byte	0x00
0x16	C1DATAL	R	CH1 ADC low data byte	0x00
0x17	C1DATAH	R	CH1 ADC high data byte	0x00

Note(s):

1. Devices with a primary I²C address of 0x29 also have a secondary I²C address of 0x28 that can be used for read only registers to quickly read in a single block I²C transaction.

Command Register

The COMMAND register specifies the address of the target register for future read and write operations, as well as issues special function commands.

7	6	5	4	3	2	1	0
CMD	TRANSACTION		ADDR/SF				

<table>
<tr><th>Fields</th><th>Bits</th><th colspan="2">Description</th></tr>
<tr><td>CMD</td><td>7</td><td colspan="2">Select Command Register. Must write as 1 when addressing COMMAND register.</td></tr>
<tr><td rowspan="6">TRANSACTION</td><td rowspan="6">6:5</td><td colspan="2">Select type of transaction to follow in subsequent data transfers</td></tr>
<tr><td><b>FIELD VALUE</b></td><td><b>DESCRIPTION</b></td></tr>
<tr><td>00</td><td>Reserved - Do not use</td></tr>
<tr><td>01</td><td>Normal Operation</td></tr>
<tr><td>10</td><td>Reserved – Do not use</td></tr>
<tr><td>11</td><td>Special Function – See description below</td></tr>
<tr><td rowspan="8">ADDR/SF</td><td rowspan="8">4:0</td><td colspan="2">Address field/special function field. Depending on the transaction type, see above, this field either specifies a special function command or selects the specific control-status-data register for subsequent read and write transactions. The field values listed below apply only to special function commands.</td></tr>
<tr><td><b>FIELD VALUE</b></td><td><b>DESCRIPTION</b></td></tr>
<tr><td>00100</td><td>Interrupt set – forces an interrupt</td></tr>
<tr><td>00110</td><td>Clears ALS interrupt</td></tr>
<tr><td>00111</td><td>Clears ALS and no persist ALS interrupt</td></tr>
<tr><td>01010</td><td>Clears no persist ALS interrupt</td></tr>
<tr><td>other</td><td>Reserved – Do not write</td></tr>
<tr><td colspan="2">The interrupt set special function command sets the interrupt bits in the status register (0x13). For the interrupt to be visible on the INT pin, one of the interrupt enable bits in the enable register (0x00) must be asserted.
The interrupt set special function must be cleared with an interrupt clear special function. The ALS interrupt clear special functions clear any pending interrupt(s) and are self-clearing.</td></tr>
</table>

Enable Register (0x00)

The ENABLE register is used to power the device on/off, enable functions and interrupts.

7	6	5	4	3	2	1	0
NPIEN	SAI	Reserved	AIEN	Reserved		AEN	PON

Fields	Bits	Description
NPIEN	7	No Persist Interrupt Enable. When asserted NP Threshold conditions will generate an interrupt, bypassing the persist filter.
SAI	6	Sleep after interrupt. When asserted, the device will power down at the end of an ALS cycle if an interrupt has been generated.
Reserved	5	Reserved. Write as 0.
AIEN	4	ALS Interrupt Enable. When asserted permits ALS interrupts to be generated, subject to the persist filter.
Reserved	3:2	Reserved. Write as 0.
AEN	1	ALS Enable. This field activates ALS function. Writing a one activates the ALS. Writing a zero disables the ALS.
PON	0	Power ON. This field activates the internal oscillator to permit the timers and ADC channels to operate. Writing a one activates the oscillator. Writing a zero disables the oscillator.

Control Register (0x01)

The CONTROL register is used to configure the ALS gain and integration time. In addition, a system reset is provided. Upon power up, the CONTROL register resets to 0x00.

<table>
<tr><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td>SRESET</td><td>Reserved</td><td colspan="2">AGAIN</td><td>Reserved</td><td colspan="3">ATIME</td></tr>
</table>

<table>
<tr><th>Fields</th><th>Bits</th><th colspan="3">Description</th></tr>
<tr><td>SRESET</td><td>7</td><td colspan="3">System reset. When asserted, the device will reset equivalent to a power-on reset. SRESET is self-clearing.</td></tr>
<tr><td>Reserved</td><td>6</td><td colspan="3">Reserved. Write as 0.</td></tr>
<tr><td rowspan="6">AGAIN</td><td rowspan="6">5:4</td><td colspan="3">ALS gain sets the gain of the internal integration amplifiers for both photodiode channels.</td></tr>
<tr><td><b>FIELD VALUE</b></td><td colspan="2"><b>DESCRIPTION</b></td></tr>
<tr><td>00</td><td colspan="2">Low gain mode</td></tr>
<tr><td>01</td><td colspan="2">Medium gain mode</td></tr>
<tr><td>10</td><td colspan="2">High gain mode</td></tr>
<tr><td>11</td><td colspan="2">Maximum gain mode</td></tr>
<tr><td>Reserved</td><td>3</td><td colspan="3">Reserved. Write as 0.</td></tr>
<tr><td rowspan="8">ATIME</td><td rowspan="8">2:0</td><td colspan="3">ALS time sets the internal ADC integration time for both photodiode channels.</td></tr>
<tr><td><b>FIELD VALUE</b></td><td><b>INTEGRATION TIME</b></td><td><b>MAX COUNT</b></td></tr>
<tr><td>000</td><td>100 ms</td><td>36863</td></tr>
<tr><td>001</td><td>200 ms</td><td>65535</td></tr>
<tr><td>010</td><td>300 ms</td><td>65535</td></tr>
<tr><td>011</td><td>400 ms</td><td>65535</td></tr>
<tr><td>100</td><td>500 ms</td><td>65535</td></tr>
<tr><td>101</td><td>600 ms</td><td>65535</td></tr>
</table>

ALS Interrupt Threshold Register (0x04 – 0x0B)

The ALS interrupt threshold registers provide the values to be used as the high and low trigger points for the comparison function for interrupt generation. If C0DATA crosses below the low threshold specified, or above the higher threshold, an interrupt is asserted on the interrupt pin.

If the C0DATA exceeds the persist thresholds (registers: 0x04 – 0x07) for the number of persist cycles configured in the PERSIST register an interrupt will be triggered. If the C0DATA exceeds the no-persist thresholds (registers: 0x08 – 0x0B) an interrupt will be triggered immediately following the end of the current integration.

Note that while the interrupt is observable in the STATUS register (0x13), it is visible only on the INT pin when AIEN or NPIEN are enabled in the ENABLE register (0x00).

Upon power up, the interrupt threshold registers default to 0x00.

Register	Address	Bits	Description
AILTL	0x04	7:0	ALS low threshold lower byte
AILTH	0x05	7:0	ALS low threshold upper byte
AIHTL	0x06	7:0	ALS high threshold lower byte
AIHTH	0x07	7:0	ALS high threshold upper byte
NPAILTL	0x08	7:0	No Persist ALS low threshold lower byte
NPAILTH	0x09	7:0	No Persist ALS low threshold upper byte
NPAIHTL	0x0A	7:0	No Persist ALS high threshold lower byte
NPAIHTH	0x0B	7:0	No Persist ALS high threshold upper byte

PERSIST Register (0x0C)

The Interrupt persistence filter sets the number of consecutive out-of-range ALS cycles necessary to generate an interrupt. Out-of-range is determined by comparing C0DATA (0x14 and 0x15) to the interrupt threshold registers (0x04 - 0x07). Note that the no-persist ALS interrupt is not affected by the interrupt persistence filter. Upon power up, the interrupt persistence filter register resets to 0x00.

7	6	5	4	3	2	1	0
Reserved				APERS			

Field	Bits	Description	
Reserved	7:4	Reserved. Write as 0.	
APERS	3:0	ALS interrupt persistence filter	
		FIELD VALUE	**PERSISTENCE**
		0000	Every ALS cycle generates an interrupt
		0001	Any value outside of threshold range
		0010	2 consecutive values out of range
		0011	3 consecutive values out of range
		0100	5 consecutive values out of range
		0101	10 consecutive values out of range
		0110	15 consecutive values out of range
		0111	20 consecutive values out of range
		1000	25 consecutive values out of range
		1001	30 consecutive values out of range
		1010	35 consecutive values out of range
		1011	40 consecutive values out of range
		1100	45 consecutive values out of range
		1101	50 consecutive values out of range
		1110	55 consecutive values out of range
		1111	60 consecutive values out of range

PID Register (0x11)

The PID register provides an identification of the devices package. This register is a read-only register whose value never changes.

7	6	5	4	3	2	1	0
Reserved		PACKAGEID		Reserved			

Field	Bits	Description
Reserved	7:6	Reserved.
PID	5:4	Package Identification = 00
Reserved	3:0	Reserved.

ID Register (0x12)

The ID register provides the device identification. This register is a read-only register whose value never changes.

7	6	5	4	3	2	1	0
ID							

Field	Bits	Description
ID	7:0	Device Identification = 0x50

Status Register (0x13)

The Status Register provides the internal status of the device. This register is read only.

7	6	5	4	3	2	1	0
Reserved		NPINTR	AINT	Reserved			AVALID

Field	Bits	Description
Reserved	7:6	Reserved. Write at zero.
NPINTR	5	No-persist Interrupt. Indicates that the device has encountered a no-persist interrupt condition.
AINT	4	ALS Interrupt. Indicates that the device is asserting an ALS interrupt.
Reserved	3:1	Reserved.
AVALID	0	ALS Valid. Indicates that the ADC channels have completed an integration cycle since the AEN bit was asserted.

ALS Data Register (0x14 - 0x17)

ALS data is stored as two 16-bit values; one for each channel. When the lower byte of either channel is read, the upper byte of the same channel is latched into a shadow register. The shadow register ensures that both bytes are the result of the same ALS integration cycle, even if additional integration cycles occur between the lower byte and upper byte register readings.

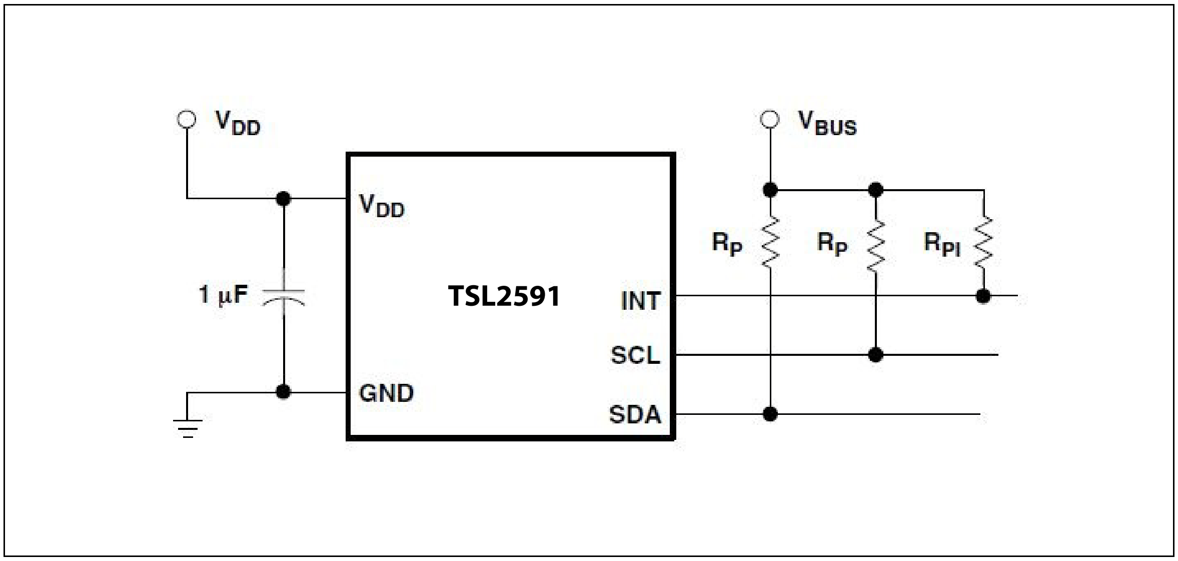
Each channel independently operates the upper byte shadow register. So to minimize the potential for skew between CH0 and CH1 data, it is recommended to read all four ADC bytes in sequence.

Register	Address	Bits	Description
C0DATAL	0x14	7:0	ALS CH0 data low byte
C0DATAH	0x15	7:0	ALS CH0 data high byte
C1DATAL	0x16	7:0	ALS CH1 data low byte
C1DATAH	0x17	7:0	ALS CH1 data high byte

Application Information

Figure 16 shows a typical hardware application circuit. A 1-µF low-ESR decoupling capacitor should be placed as close as possible to the V_{DD} pin. V_{BUS} in this figure refers to the I²C bus voltage, which is equal to V_{DD}.

Figure 16:
Typical Application Hardware Circuit

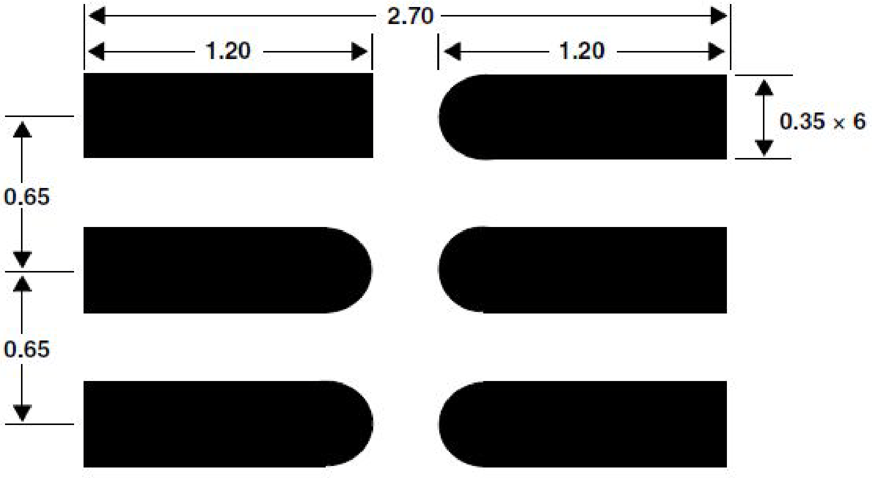


The I²C signals and the Interrupt are open-drain outputs and require pull-up resistors. The pull-up resistor (RP) value is a function of the I²C bus speed, the I²C bus voltage, and the capacitive load. The **ams** EVM running at 400 kbps, uses 1.5-kΩ resistors. A 10-kΩ pull-up resistor (RPI) can be used for the interrupt line.

PCB Pad Layout

Suggested land pattern based on the IPC–7351B Generic Requirements for Surface Mount Design and Land Pattern Standard (2010) for the small outline no-lead (SON) package is shown in Figure 17.

Figure 17:
Suggested FN Package PCB Layout (Top View)

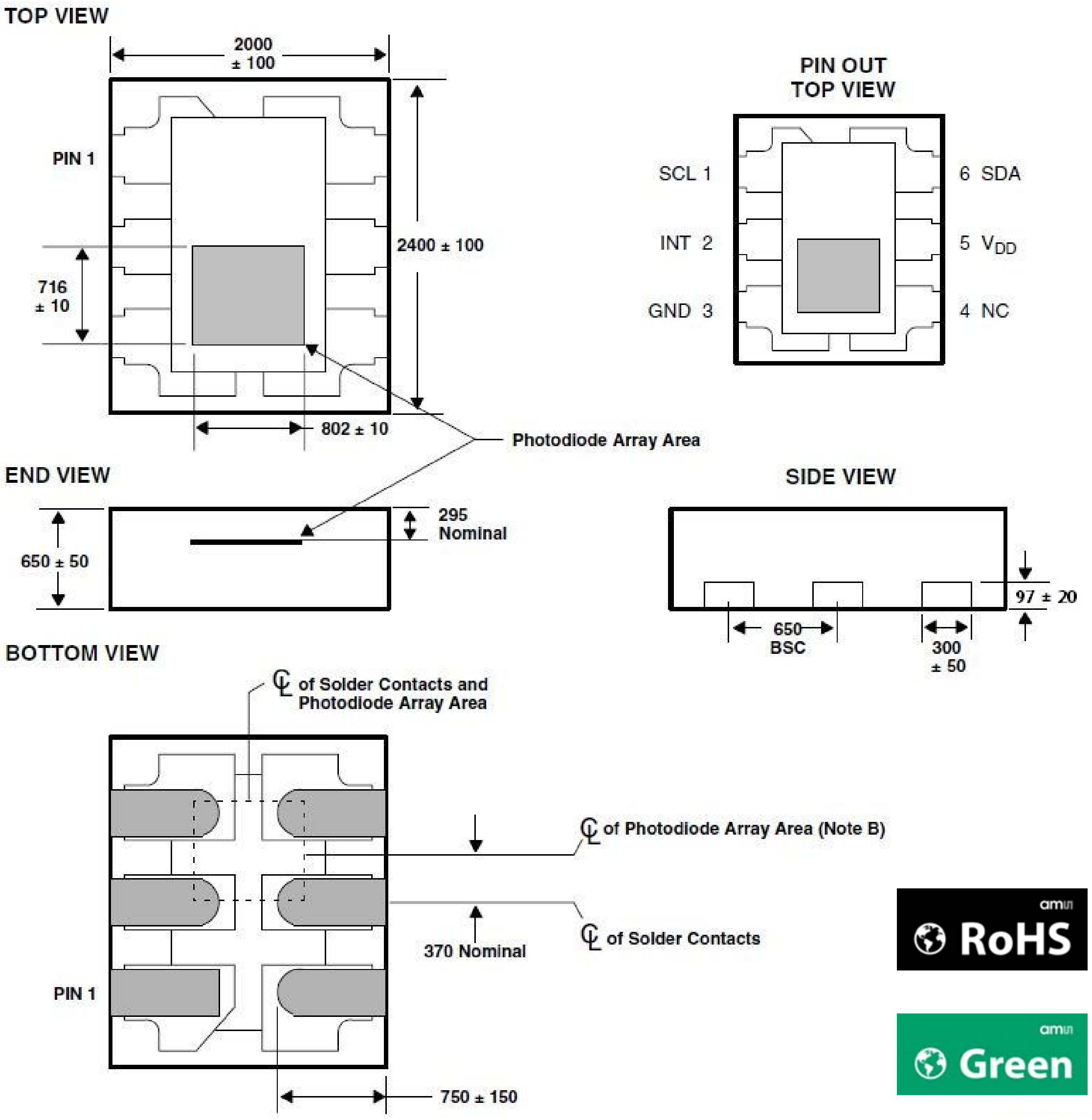


Note(s):

1. All linear dimensions are in millimeters.
2. This drawing is subject to change without notice.

Package Drawings & Markings

Figure 18:
FN Package – Dual Flat No-Lead Packaging Configuration

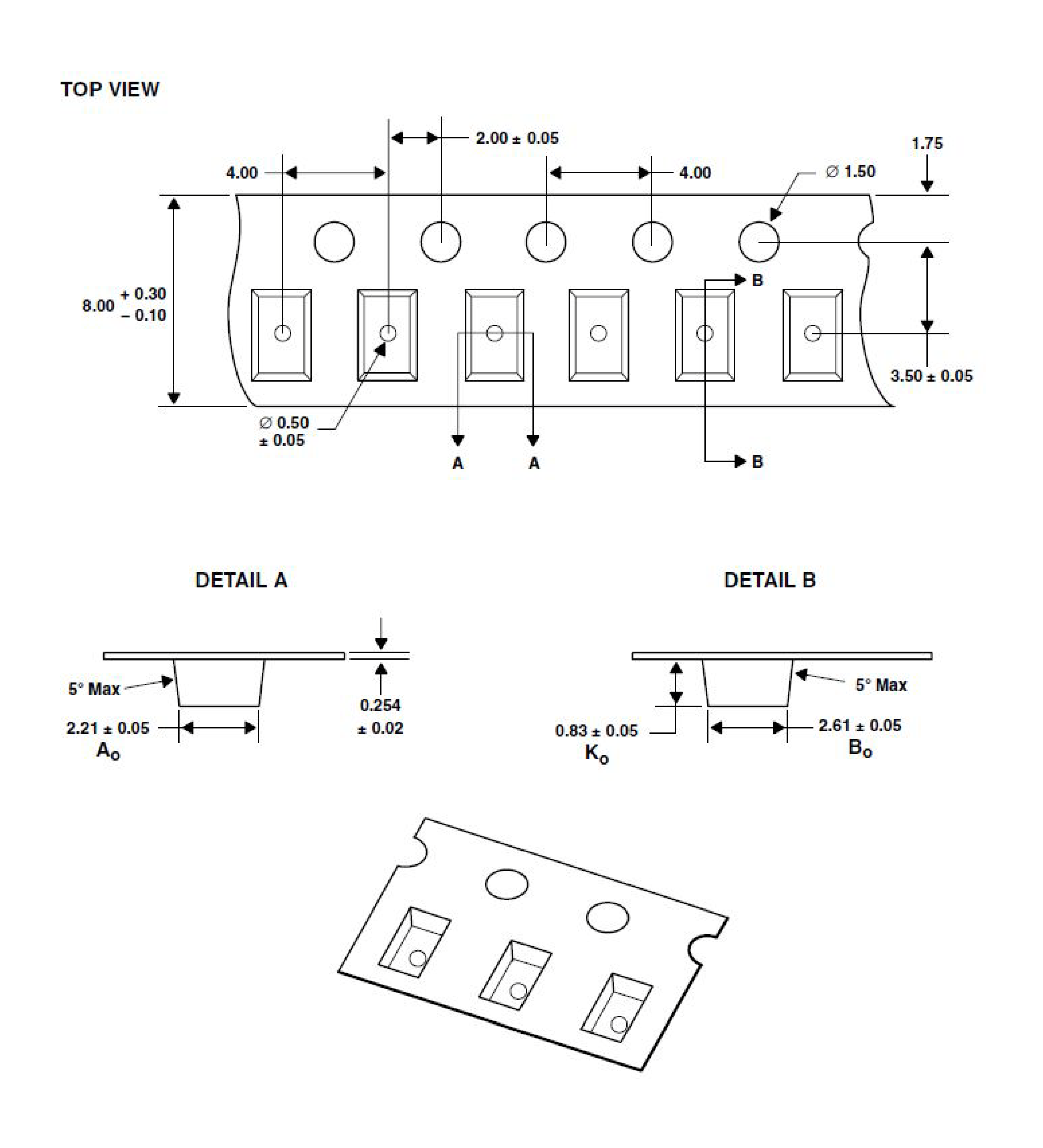


Note(s):

1. All linear dimensions are in micrometers.
2. The die is centered within the package within a tolerence of ±75 µm.
3. Package top surface is molded with an electrically non-conductive clear plastic compound having an index of refraction of 1.55.
4. Contact finish is copper alloy A194 with pre-plated NiPdAu lead finish.
5. This package contains no lead (Pb).
6. This drawing is subject to change without notice.

Mechanical Data

Figure 19:
FN Package Carrier Tape and Reel Information



Note(s):

1. All linear dimensions are in millimeters. Dimension tolerance is ± 0.10 mm unless otherwise noted.
2. The dimensions on this drawing are for illustrative purposes only. Dimensions of an actual carrier may vary slightly.
3. Symbols on drawing A_O, B_O and K_O are defined in ANSI EIA Standard 481-B 2001.
4. Each reel is 178 millimeters in diameter and contains 3500 parts.
5. **ams** packaging tape and reel conform to the requirements of EIA Standard 481 - B.
6. In accordance with EIA Standard, device pin 1 is located next to the sprocket holes in the tape.
7. This drawing is subject to change without notice.

Soldering Information

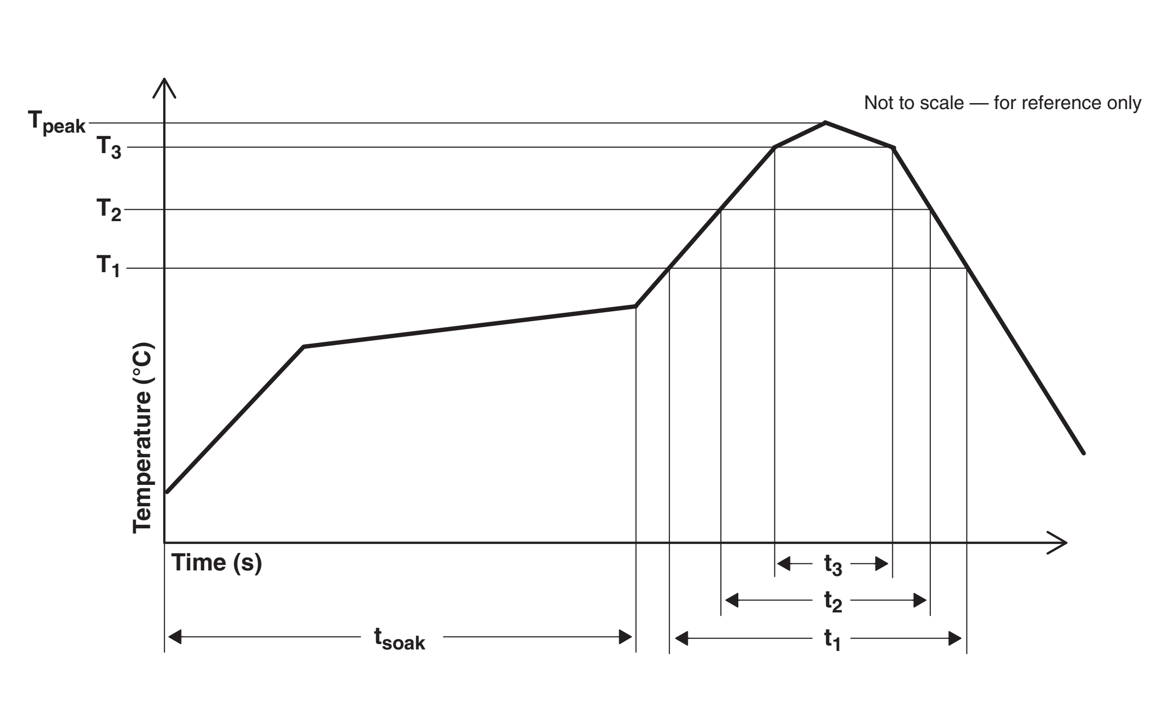
The package has been tested and has demonstrated an ability to be reflow soldered to a PCB substrate.

The solder reflow profile describes the expected maximum heat exposure of components during the solder reflow process of product on a PCB. Temperature is measured on top of component. The components should be limited to a maximum of three passes through this solder reflow profile.

Figure 20:
Solder Reflow Profile

Parameter	Reference	Device
Average temperature gradient in preheating		2.5 ºC/s
Soak time	t_{soak}	2 to 3 minutes
Time above 217 ºC (T_1)	t_1	Max 60 s
Time above 230 ºC (T_2)	t_2	Max 50 s
Time above T_{peak} - 10 ºC (T_3)	t_3	Max 10 s
Peak temperature in reflow	T_{peak}	260 ºC
Temperature gradient in cooling		Max -5 ºC/s

Figure 21:
Solder Reflow Profile Graph



Storage Information

Moisture Sensitivity

Optical characteristics of the device can be adversely affected during the soldering process by the release and vaporization of moisture that has been previously absorbed into the package. To ensure the package contains the smallest amount of absorbed moisture possible, each device is baked prior to being dry packed for shipping.

Devices are dry packed in a sealed aluminized envelope called a moisture-barrier bag with silica gel to protect them from ambient moisture during shipping, handling, and storage before use.

Shelf Life

The calculated shelf life of the device in an unopened moisture barrier bag is 12 months from the date code on the bag when stored under the following conditions:

- Shelf Life: 12 months
- Ambient Temperature: < 40°C
- Relative Humidity: < 90%

Rebaking of the devices will be required if the devices exceed the 12 month shelf life or the Humidity Indicator Card shows that the devices were exposed to conditions beyond the allowable moisture region.

Floor Life

The FN package has been assigned a moisture sensitivity level of MSL 3. As a result, the floor life of devices removed from the moisture barrier bag is 168 hours from the time the bag was opened, provided that the devices are stored under the following conditions:

- Floor Life: 168 hours
- Ambient Temperature: < 30°C
- Relative Humidity: < 60%

If the floor life or the temperature/humidity conditions have been exceeded, the devices must be rebaked prior to solder reflow or dry packing.

Rebaking Instructions

When the shelf life or floor life limits have been exceeded, rebake at 50°C for 12 hours.

Ordering & Contact Information

Figure 22:
Ordering Information

Ordering Code	Address	Interface	Delivery Form
TSL25911FN	0x29	I^2C $V_{bus} = V_{DD}$ Interface	ODFN-6
TSL25913FN	0x29	I^2C V_{bus} = 1.8V	ODFN-6

Buy our products or get free samples online at:
www.ams.com/Product

Technical Support is available at:
www.ams.com/Technical-Support

Provide feedback about this document at:
www.ams.com/Document-Feedback

For further information and requests, e-mail us at:
ams_sales@ams.com

For sales offices, distributors and representatives, please visit:
www.ams.com/Contact

Headquarters
ams AG
Tobelbader Strasse 30
8141 Premstaetten
Austria, Europe

Tel: +43 (0) 3136 500 0

Website: www.ams.com

RoHS Compliant & ams Green Statement

RoHS: The term RoHS compliant means that ams AG products fully comply with current RoHS directives. Our semiconductor products do not contain any chemicals for all 6 substance categories, including the requirement that lead not exceed 0.1% by weight in homogeneous materials. Where designed to be soldered at high temperatures, RoHS compliant products are suitable for use in specified lead-free processes.

ams Green (RoHS compliant and no Sb/Br): ams Green defines that in addition to RoHS compliance, our products are free of Bromine (Br) and Antimony (Sb) based flame retardants (Br or Sb do not exceed 0.1% by weight in homogeneous material).

Important Information: The information provided in this statement represents ams AG knowledge and belief as of the date that it is provided. ams AG bases its knowledge and belief on information provided by third parties, and makes no representation or warranty as to the accuracy of such information. Efforts are underway to better integrate information from third parties. ams AG has taken and continues to take reasonable steps to provide representative and accurate information but may not have conducted destructive testing or chemical analysis on incoming materials and chemicals. ams AG and ams AG suppliers consider certain information to be proprietary, and thus CAS numbers and other limited information may not be available for release.

Copyrights & Disclaimer

Devices sold by ams AG are covered by the warranty and patent indemnification provisions appearing in its General Terms of Trade. ams AG makes no warranty, express, statutory, implied, or by description regarding the information set forth herein. ams AG reserves the right to change specifications and prices at any time and without notice. Therefore, prior to designing this product into a system, it is necessary to check with ams AG for current information. This product is intended for use in commercial applications. Applications requiring extended temperature range, unusual environmental requirements, or high reliability applications, such as military, medical life-support or life-sustaining equipment are specifically not recommended without additional processing by ams AG for each application. This product is provided by ams AG "AS IS" and any express or implied warranties, including, but not limited to the implied warranties of merchantability and fitness for a particular purpose are disclaimed.

ams AG shall not be liable to recipient or any third party for any damages, including but not limited to personal injury, property damage, loss of profits, loss of use, interruption of business or indirect, special, incidental or consequential damages, of any kind, in connection with or arising out of the furnishing, performance or use of the technical data herein. No obligation or liability to recipient or any third party shall arise or flow out of ams AG rendering of technical or other services.

Document Status

Document Status	Product Status	Definition
Product Preview	Pre-Development	Information in this datasheet is based on product ideas in the planning phase of development. All specifications are design goals without any warranty and are subject to change without notice
Preliminary Datasheet	Pre-Production	Information in this datasheet is based on products in the design, validation or qualification phase of development. The performance and parameters shown in this document are preliminary without any warranty and are subject to change without notice
Datasheet	Production	Information in this datasheet is based on products in ramp-up to full production or full production which conform to specifications in accordance with the terms of ams AG standard warranty as given in the General Terms of Trade
Datasheet (discontinued)	Discontinued	Information in this datasheet is based on products which conform to specifications in accordance with the terms of ams AG standard warranty as given in the General Terms of Trade, but these products have been superseded and should not be used for new designs

Revision Information

Changes from 2-03 (2018-Apr-30) to current revision 2-04 (2018-Jun-05)	Page
Updated Figure 5	5
Updated text under Electrical Characteristics	6

Note(s):

1. Page and figure numbers for the previous version may differ from page and figure numbers in the current revision.
2. Correction of typographical errors is not explicitly mentioned.

Content Guide

1 General Description
2 Block Diagram

3 Detailed Description
4 Pin Assignment
5 Absolute Maximum Ratings
6 Electrical Characteristics
9 Timing Characteristics
10 Timing Diagrams
11 Typical Operating Characteristics

13 Register Description
14 Command Register
15 Enable Register (0x00)
16 Control Register (0x01)
17 ALS Interrupt Threshold Register (0x04 – 0x0B)
18 PERSIST Register (0x0C)
19 PID Register (0x11)
19 ID Register (0x12)
20 Status Register (0x13)
21 ALS Data Register (0x14 - 0x17)

22 Application Information
23 PCB Pad Layout
24 Package Drawings & Markings
25 Mechanical Data
26 Soldering Information

27 Storage Information
27 Moisture Sensitivity
27 Shelf Life
27 Floor Life
27 Rebaking Instructions

28 Ordering & Contact Information
29 RoHS Compliant & ams Green Statement
30 Copyrights & Disclaimer
31 Document Status
32 Revision Information

TCA9548A Low-Voltage 8-Channel I^{2}C Switch with Reset

1 Features

- 1-to-8 Bidirectional translating switches
- I^{2}C Bus and SMBus compatible
- Active-low reset input
- Three address pins, allowing up to eight TCA9548A devices on the I^{2}C bus
- Channel selection through an I^{2}C Bus, in any combination
- Power up with all switch channels deselected
- Low R_{ON} switches
- Allows voltage-level translation between 1.8-V, 2.5-V, 3.3-V, and 5-V buses
- No glitch on power up
- Supports hot insertion
- Low standby current
- Operating power-supply voltage range of 1.65 V to 5.5 V
- 5-V Tolerant inputs
- 0- to 400-kHz Clock frequency
- Latch-up performance exceeds 100 mA Per JESD 78, class II
- ESD Protection exceeds JESD 22
  - ±2000-V Human-body model (A114-A)
  - 200-V Machine model (A115-A)
  - ±1000-V Charged-device model (C101)

2 Applications

- Servers
- Routers (telecom switching equipment)
- Factory Automation
- Products with I^{2}C slave address conflicts (such as multiple, identical temperature sensors)

3 Description

The TCA9548A device has eight bidirectional translating switches that can be controlled through the I^{2}C bus. The SCL/SDA upstream pair fans out to eight downstream pairs, or channels. Any individual SCn/SDn channel or combination of channels can be selected, determined by the contents of the programmable control register. These downstream channels can be used to resolve I^{2}C slave address conflicts. For example, if eight identical digital temperature sensors are needed in the application, one sensor can be connected at each channel: 0-7.

The system master can reset the TCA9548A in the event of a time-out or other improper operation by asserting a low in the $\overline{\text{RESET}}$ input. Similarly, the power-on reset deselects all channels and initializes the I^{2}C/SMBus state machine. Asserting $\overline{\text{RESET}}$ causes the same reset and initialization to occur without powering down the part. This allows recovery should one of the downstream I^{2}C buses get stuck in a low state.

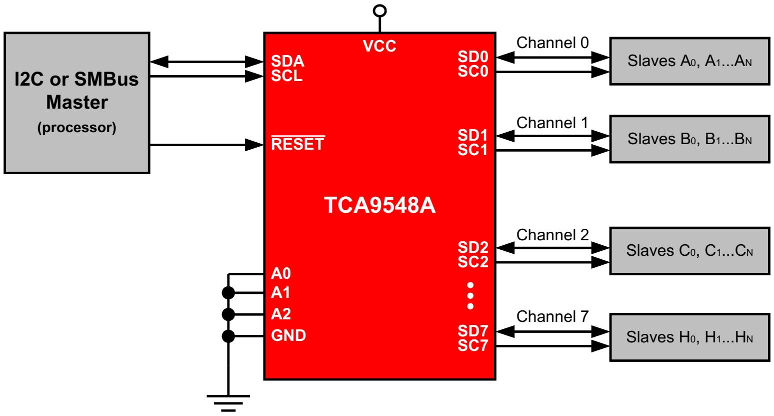
The pass gates of the switches are constructed so that the VCC pin can be used to limit the maximum high voltage, which is passed by the TCA9548A. Limiting the maximum high voltage allows the use of different bus voltages on each pair, so that 1.8-V, 2.5-V or 3.3-V parts can communicate with 5-V parts, without any additional protection. External pullup resistors pull the bus up to the desired voltage level for each channel. All I/O pins are 5-V tolerant.

Device Information[1]

PART NUMBER	PACKAGE	BODY SIZE (NOM)
TCA9548A	TSSOP (24)	7.80 mm × 4.40 mm
	VQFN (24)	4.00 mm × 4.00 mm

[1] For all available packages, see the orderable addendum at the end of the data sheet.

Simplified Application Diagram



An IMPORTANT NOTICE at the end of this data sheet addresses availability, warranty, changes, use in safety-critical applications, intellectual property matters and other important disclaimers. PRODUCTION DATA.

Table of Contents

1 **Features** 1
2 **Applications** 1
3 **Description** 1
4 **Revision History** 2
5 **Pin Configuration and Functions** 4
6 **Specifications** 5
6.1 Absolute Maximum Ratings 5
6.2 ESD Ratings 5
6.3 Recommended Operating Conditions 5
6.4 Thermal Information 5
6.5 Electrical Characteristics 6
6.6 I²C Interface Timing Requirements 7
6.7 Reset Timing Requirements 8
6.8 Switching Characteristics 8
6.9 Typical Characteristics 9
7 **Parameter Measurement Information** 10
8 **Detailed Description** 12
8.1 Overview 12
8.2 Functional Block Diagram 13
8.3 Feature Description 14
8.4 Device Functional Modes 14
8.5 Programming 14
9 **Application and Implementation** 20
9.1 Application Information 20
9.2 Typical Application 20
10 **Power Supply Recommendations** 24
10.1 Power-On Reset Requirements 24
11 **Layout** 26
11.1 Layout Guidelines 26
11.2 Layout Example 26
12 **Device and Documentation Support** 27
12.1 Documentation Support 27
12.2 Receiving Notification of Documentation Updates 27
12.3 Support Resources 27
12.4 Trademarks 27
12.5 Electrostatic Discharge Caution 27
12.6 Glossary 27
13 **Mechanical, Packaging, and Orderable Information** 27

4 Revision History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Changes from Revision F (November 2016) to Revision G	Page
• Changed the appearance of the PW package and the RGE package images	4
• Changed T_J from 90 C to 130 C in lower voltage V_{CC} conditions	5
• Changed T_A from 85 C to 125C for lower voltage V_{CC} conditions	5
• Changed From: V_{CC} = 2.3 V to 3.6 V To: V_{CC} = 1.65 V to 5.5 V in the *Electrical Characteristics* conditions	6
• Changed V_O min from 0.9V to 0.6 V	6
• Added standby mode specifications for > 85 C T_A	6
• Changed R_L = 1 kW To: R_L = 1 KΩ in Figure 6	11

Changes from Revision E (October 2015) to Revision F	Page
• Updated the *Description* section	1
• Added new orderable part number, TCA9548AMRGER	1

Changes from Revision D (January 2015) to Revision E	Page
• Updated Pin Functions table.	4
• Added new I²C Sections and read/write description	16

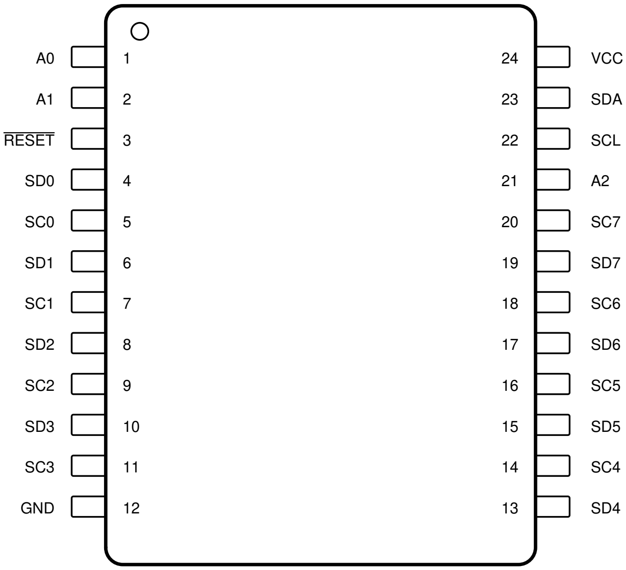
Changes from Revision C (November 2013) to Revision D	Page
• Added *Pin Configuration and Functions* section, *ESD Ratings* table, *Feature Description* section, *Device Functional Modes*, *Application and Implementation* section, *Power Supply Recommendations* section, *Layout* section, *Device and Documentation Support* section, and *Mechanical, Packaging, and Orderable Information* section	1
• Updated Typical Application schematic.	21

Changes from Revision B (November 2013) to Revision C	Page
• Updated V_{POR} and I_{CC} standby specification.	6

Changes from Revision A (July 2012) to Revision B	Page
• Updated document formatting.	1
• Removed ordering information.	1

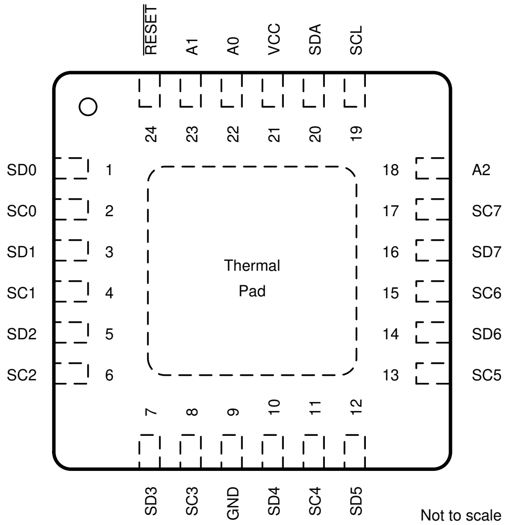
5 Pin Configuration and Functions

PW Package
24-Pin TSSOP
Top View



Not to scale

RGE Package
24-Pin VQFN
Top View



Not to scale

Pin Functions

PIN			TYPE	DESCRIPTION
NAME	TSSOP (PW)	QFN (RGE)		
A0	1	22	I	Address input 0. Connect directly to V_{CC} or ground
A1	2	23	I	Address input 1. Connect directly to V_{CC} or ground
A2	21	18	I	Address input 2. Connect directly to V_{CC} or ground
GND	12	9	—	Ground
$\overline{\text{RESET}}$	3	24	I	Active-low reset input. Connect to V_{CC} or V_{DPUM} [1] through a pull-up resistor, if not used
SD0	4	1	I/O	Serial data 0. Connect to V_{DPU0} [1] through a pull-up resistor
SC0	5	2	I/O	Serial clock 0. Connect to V_{DPU0} [1] through a pull-up resistor
SD1	6	3	I/O	Serial data 1. Connect to V_{DPU1} [1] through a pull-up resistor
SC1	7	4	I/O	Serial clock 1. Connect to V_{DPU1} [1] through a pull-up resistor
SD2	8	5	I/O	Serial data 2. Connect to V_{DPU2} [1] through a pull-up resistor
SC2	9	6	I/O	Serial clock 2. Connect to V_{DPU2} [1] through a pull-up resistor
SD3	10	7	I/O	Serial data 3. Connect to V_{DPU3} [1] through a pull-up resistor
SC3	11	8	I/O	Serial clock 3. Connect to V_{DPU3} [1] through a pull-up resistor
SD4	13	10	I/O	Serial data 4. Connect to V_{DPU4} [1] through a pull-up resistor
SC4	14	11	I/O	Serial clock 4. Connect to V_{DPU4} [1] through a pull-up resistor
SD5	15	12	I/O	Serial data 5. Connect to V_{DPU5} [1] through a pull-up resistor
SC5	16	13	I/O	Serial clock 5. Connect to V_{DPU5} [1] through a pull-up resistor
SD6	17	14	I/O	Serial data 6. Connect to V_{DPU6} [1] through a pull-up resistor
SC6	18	15	I/O	Serial clock 6. Connect to V_{DPU6} [1] through a pull-up resistor
SD7	19	16	I/O	Serial data 7. Connect to V_{DPU7} [1] through a pull-up resistor
SC7	20	17	I/O	Serial clock 7. Connect to V_{DPU7} [1] through a pull-up resistor
SCL	22	19	I/O	Serial clock bus. Connect to V_{DPUM} [1] through a pull-up resistor
SDA	23	20	I/O	Serial data bus. Connect to V_{DPUM} [1] through a pull-up resistor
VCC	24	21	Power	Supply voltage

[1] V_{DPUX} is the pull-up reference voltage for the associated data line. V_{DPUM} is the master I^2C reference voltage and V_{DPU0}-V_{DPU7} are the slave channel reference voltages.

6 Specifications

6.1 Absolute Maximum Ratings[1]

over operating free-air temperature range (unless otherwise noted)

			MIN	MAX	UNIT
V_{CC}	Supply voltage		–0.5	7	V
V_I	Input voltage[2]		–0.5	7	V
I_I	Input current		–20	20	mA
I_O	Output current		–25		mA
I_{CC}	Supply current		–100	100	mA
T_{stg}	Storage temperature		–65	150	°C
T_J	Max Junction Temperature	$V_{CC} \leq 3.6$ V		130	℃
		$V_{CC} \leq 5.5$ V		90	

[1] Stresses beyond those listed under *Absolute Maximum Ratings* may cause permanent damage to the device. These are stress ratings only, which do not imply functional operation of the device at these or any other conditions beyond those indicated under *Recommended Operating Conditions*. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.

[2] The input negative-voltage and output voltage ratings may be exceeded if the input and output current ratings are observed.

6.2 ESD Ratings

			VALUE	UNIT
$V_{(ESD)}$	Electrostatic discharge	Human body model (HBM), per ANSI/ESDA/JEDEC JS-001[1]	±2000	V
		Charged-device model (CDM), per JEDEC specification JESD22-C101[2]	±1000	

[1] JEDEC document JEP155 states that 500-V HBM allows safe manufacturing with a standard ESD control process.

[2] JEDEC document JEP157 states that 250-V CDM allows safe manufacturing with a standard ESD control process.

6.3 Recommended Operating Conditions

			MIN	MAX	UNIT
V_{CC}	Supply voltage	-40 ℃ ≤ T_A ≤ 85 ℃	1.65	5.5	V
		85 ℃ < T_A ≤ 125 ℃	1.65	3.6	
V_{IH}	High-level input voltage	SCL, SDA	0.7 × V_{CC}	6	V
		A2–A0, $\overline{\text{RESET}}$	0.7 × V_{CC}	V_{CC} + 0.5	
V_{IL}	Low-level input voltage	SCL, SDA	–0.5	0.3 × V_{CC}	V
		A2–A0, $\overline{\text{RESET}}$	–0.5	0.3 × V_{CC}	
T_A	Operating free-air temperature	3.6 V < V_{CC} ≤ 5.5 V	–40	85	°C
		1.65 V ≤ V_{CC} ≤ 3.6 V	–40	125	

6.4 Thermal Information

THERMAL METRIC[1]		TCA9548A		UNIT
		PW (TSSOP)	RGE (VQFN)	
		24 PINS	24 PINS	
$R_{\theta JA}$	Junction-to-ambient thermal resistance	108.8	57.2	°C/W
$R_{\theta JC(top)}$	Junction-to-case (top) thermal resistance	54.1	62.5	°C/W
$R_{\theta JB}$	Junction-to-board thermal resistance	62.7	34.4	°C/W
ψ_{JT}	Junction-to-top characterization parameter	10.9	3.8	°C/W
ψ_{JB}	Junction-to-board characterization parameter	62.3	34.4	°C/W
$R_{\theta JC(bot)}$	Junction-to-case (bottom) thermal resistance	N/A	15.5	°C/W

[1] For more information about traditional and new thermal metrics, see the *Semiconductor and IC Package Thermal Metrics* application report.

6.5 Electrical Characteristics[1]

V_{CC} = 1.65 V to 5.5 V, over recommended operating free-air temperature ranges supported by Recommended Operating Conditions (unless otherwise noted)

PARAMETER			TEST CONDITIONS	V_{CC}	MIN	TYP[2]	MAX	UNIT
V_{PORR}	Power-on reset voltage, V_{CC} rising		No load, V_I = V_{CC} or GND[3]			1.2	1.5	V
V_{PORF}	Power-on reset voltage, V_{CC} falling[4]		No load, V_I = V_{CC} or GND[3]		0.8	1		V
$V_{o(sw)}$	Switch output voltage		$V_{i(sw)}$ = V_{CC}, I_{SWout} = –100 μA	5 V		3.6		V
				4.5 V to 5.5 V	2.6		4.5	
				3.3 V		1.9		
				3 V to 3.6 V	1.6		2.8	
				2.5 V		1.5		
				2.3 V to 2.7 V	1.1		2	
				1.8 V		1.1		
				1.65 V to 1.95 V	0.6		1.25	
I_{OL}	SDA		V_{OL} = 0.4 V	1.65 V to 5.5 V	3	6		mA
			V_{OL} = 0.6 V		6	9		
I_I	SCL, SDA		V_I = V_{CC} or GND[3]	1.65 V to 5.5 V	–1		1	μA
	SC7–SC0, SD7–SD0				–1		1	
	A2–A0				–1		1	
	$\overline{\text{RESET}}$				–1		1	
I_{CC}	Operating mode	f_{SCL} = 400 kHz	V_I = V_{CC} or GND[3], I_O = 0	5.5 V		50	80	μA
				3.6 V		20	35	
				2.7 V		11	20	
				1.65 V		6	10	
		f_{SCL} = 100 kHz	V_I = V_{CC} or GND[3], I_O = 0	5.5 V		9	30	
				3.6 V		6	15	
				2.7 V		4	8	
				1.65 V		2	4	
	Standby mode	Low inputs	V_I = GND[3], I_O = 0, -40 ℃ ≤ T_A ≤ 85 ℃	5.5 V		0.2	2	
				3.6 V		0.1	2	
				2.7 V		0.1	1	
				1.65 V		0.1	1	
		High inputs	V_I = V_{CC}, I_O = 0, -40 ℃ ≤ T_A ≤ 85 ℃	5.5 V		0.2	2	
				3.6 V		0.1	2	
				2.7 V		0.1	1	
				1.65 V		0.1	1	
		Low and High Inputs	V_I = V_{CC} or GND, I_O = 0, 85 ℃ < T_A ≤ 125 ℃	3.6 V		1	2	
				2.7 V		0.7	1.5	
				1.65 V		0.4	1	
ΔI_{CC}	Supply-current change	SCL, SDA	SCL or SDA input at 0.6 V, Other inputs at V_{CC} or GND[3]	1.65 V to 5.5 V		3	20	μA
			SCL or SDA input at V_{CC} – 0.6 V, Other inputs at V_{CC} or GND[3]			3	20	
C_i	A2–A0		V_I = V_{CC} or GND[3]	1.65 V to 5.5 V		4	5	pF
	$\overline{\text{RESET}}$					4	5	
	SCL		V_I = V_{CC} or GND[3], Switch OFF			20	28	

[1] For operation between specified voltage ranges, refer to the worst-case parameter in both applicable ranges.
[2] All typical values are at nominal supply voltage (1.8-, 2.5-, 3.3-, or 5-V V_{CC}), T_A = 25°C.
[3] $\overline{\text{RESET}}$ = V_{CC} (held high) when all other input voltages, V_I = GND.
[4] The power-on reset circuit resets the I^2C bus logic with V_{CC} < V_{PORF}.

Electrical Characteristics[1] (continued)

V_{CC} = 1.65 V to 5.5 V, over recommended operating free-air temperature ranges supported by Recommended Operating Conditions (unless otherwise noted)

PARAMETER		TEST CONDITIONS	V_{CC}	MIN	TYP[2]	MAX	UNIT
$C_{io(off)}$ [5]	SDA	$V_I = V_{CC}$ or GND[3], Switch OFF	1.65 V to 5.5 V		20	28	pF
	SC7–SC0, SD7–SD0				5.5	7.5	
R_{ON}	Switch-on resistance	V_O = 0.4 V, I_O = 15 mA	4.5 V to 5.5 V	4	10	20	Ω
			3 V to 3.6 V	5	12	30	
		V_O = 0.4 V, I_O = 10 mA	2.3 V to 2.7 V	7	15	45	
			1.65 V to 1.95 V	10	25	70	

[5] $C_{io(ON)}$ depends on internal capacitance and external capacitance added to the SCn lines when channels(s) are ON.

6.6 I²C Interface Timing Requirements

over recommended operating free-air temperature range (unless otherwise noted) (see Figure 5)

			MIN	MAX	UNIT
STANDARD MODE					
f_{scl}	I²C clock frequency		0	100	kHz
t_{sch}	I²C clock high time		4		µs
t_{scl}	I²C clock low time		4.7		µs
t_{sp}	I²C spike time			50	ns
t_{sds}	I²C serial-data setup time		250		ns
t_{sdh}	I²C serial-data hold time		0[1]		µs
t_{icr}	I²C input rise time			1000	ns
t_{icf}	I²C input fall time			300	ns
t_{ocf}	I²C output (SDn) fall time (10-pF to 400-pF bus)			300	ns
t_{buf}	I²C bus free time between stop and start		4.7		µs
t_{sts}	I²C start or repeated start condition setup		4.7		µs
t_{sth}	I²C start or repeated start condition hold		4		µs
t_{sps}	I²C stop condition setup		4		µs
$t_{vdL(Data)}$	Valid-data time (high to low)[2]	SCL low to SDA output low valid		1	µs
$t_{vdH(Data)}$	Valid-data time (low to high)[2]	SCL low to SDA output high valid		0.6	µs
$t_{vd(ack)}$	Valid-data time of ACK condition	ACK signal from SCL low to SDA output low		1	µs
C_b	I²C bus capacitive load			400	pF
FAST MODE					
f_{scl}	I²C clock frequency		0	400	kHz
t_{sch}	I²C clock high time		0.6		µs
t_{scl}	I²C clock low time		1.3		µs
t_{sp}	I²C spike time			50	ns
t_{sds}	I²C serial-data setup time		100		ns
t_{sdh}	I²C serial-data hold time		0[1]		µs
t_{icr}	I²C input rise time		20 + 0.1C_b[3]	300	ns
t_{icf}	I²C input fall time		20 + 0.1C_b[3]	300	ns
t_{ocf}	I²C output (SDn) fall time (10-pF to 400-pF bus)		20 + 0.1C_b[3]	300	ns

[1] A device internally must provide a hold time of at least 300 ns for the SDA signal (referred to the V_{IH} min of the SCL signal), to bridge the undefined region of the falling edge of SCL.

[2] Data taken using a 1-kΩ pull-up resistor and 50-pF load (see Figure 6)

[3] C_b = total bus capacitance of one bus line in pF

I²C Interface Timing Requirements (continued)

over recommended operating free-air temperature range (unless otherwise noted) (see Figure 5)

			MIN	MAX	UNIT
t_{buf}	I²C bus free time between stop and start		1.3		µs
t_{sts}	I²C start or repeated start condition setup		0.6		µs
t_{sth}	I²C start or repeated start condition hold		0.6		µs
t_{sps}	I²C stop condition setup		0.6		µs
$t_{vdL(Data)}$	Valid-data time (high to low)[2]	SCL low to SDA output low valid		1	µs
$t_{vdH(Data)}$	Valid-data time (low to high)[2]	SCL low to SDA output high valid		0.6	µs
$t_{vd(ack)}$	Valid-data time of ACK condition	ACK signal from SCL low to SDA output low		1	µs
C_b	I²C bus capacitive load			400	pF

6.7 Reset Timing Requirements

over recommended operating free-air temperature range (unless otherwise noted)

PARAMETER		MIN	MAX	UNIT
$t_{W(L)}$	Pulse duration, $\overline{\text{RESET}}$ low	6		ns
$t_{REC(STA)}$	Recovery time from $\overline{\text{RESET}}$ to start	0		ns

6.8 Switching Characteristics

over recommended operating free-air temperature range, $C_L \le 100$ pF (unless otherwise noted) (see Figure 5)

PARAMETER			FROM (INPUT)	TO (OUTPUT)	MIN	MAX	UNIT
t_{pd} [1]	Propagation delay time	$R_{ON} = 20\ \Omega$, $C_L = 15$ pF	SDA or SCL	SDn or SCn		0.3	ns
		$R_{ON} = 20\ \Omega$, $C_L = 50$ pF				1	
t_{rst} [2]	$\overline{\text{RESET}}$ time (SDA clear)		$\overline{\text{RESET}}$	SDA		500	ns

[1] The propagation delay is the calculated RC time constant of the typical ON-state resistance of the switch and the specified load capacitance, when driven by an ideal voltage source (zero output impedance).

[2] t_{rst} is the propagation delay measured from the time the $\overline{\text{RESET}}$ pin is first asserted low to the time the SDA pin is asserted high, signaling a stop condition. It must be a minimum of t_{WL}.

6.9 Typical Characteristics

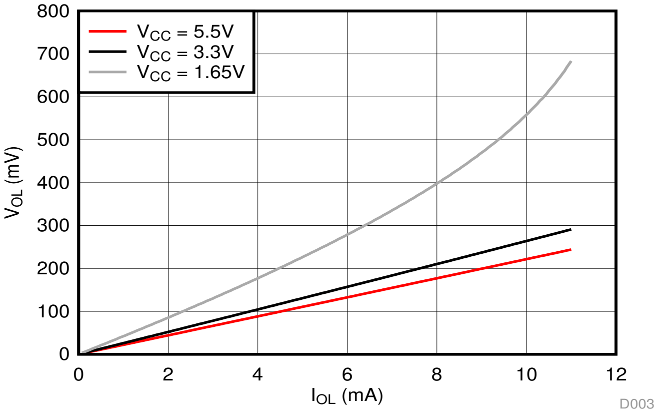


Figure 1. SDA Output Low Voltage (V_{OL}) vs Load Current (I_{OL}) at Three V_{CC} Levels

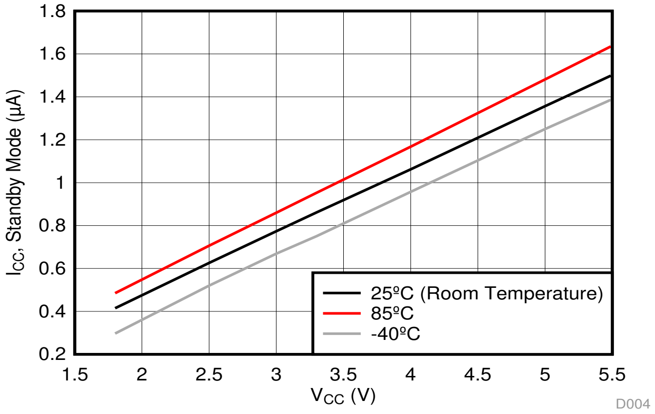


Figure 2. Standby Current (I_{CC}) vs Supply Voltage (V_{CC}) at Three Temperature Points

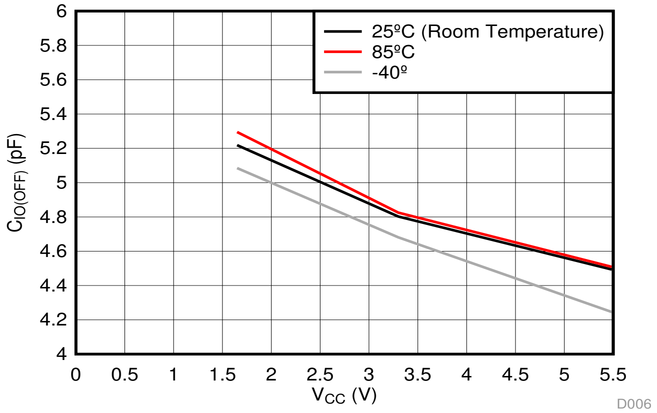


Figure 3. Slave Channel (SCn/SDn) Capacitance ($C_{io(OFF)}$) vs Supply Voltage (V_{CC}) at Three Temperature Points

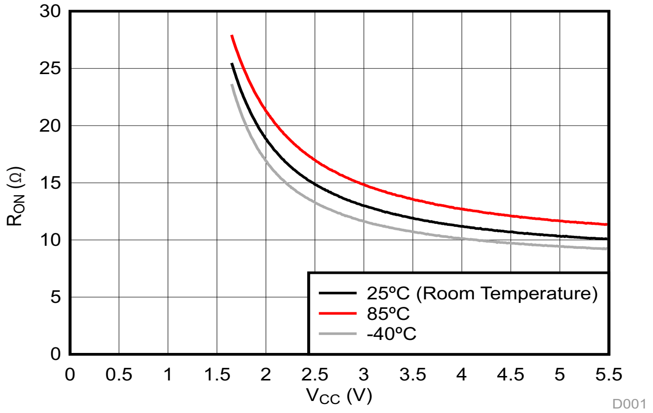
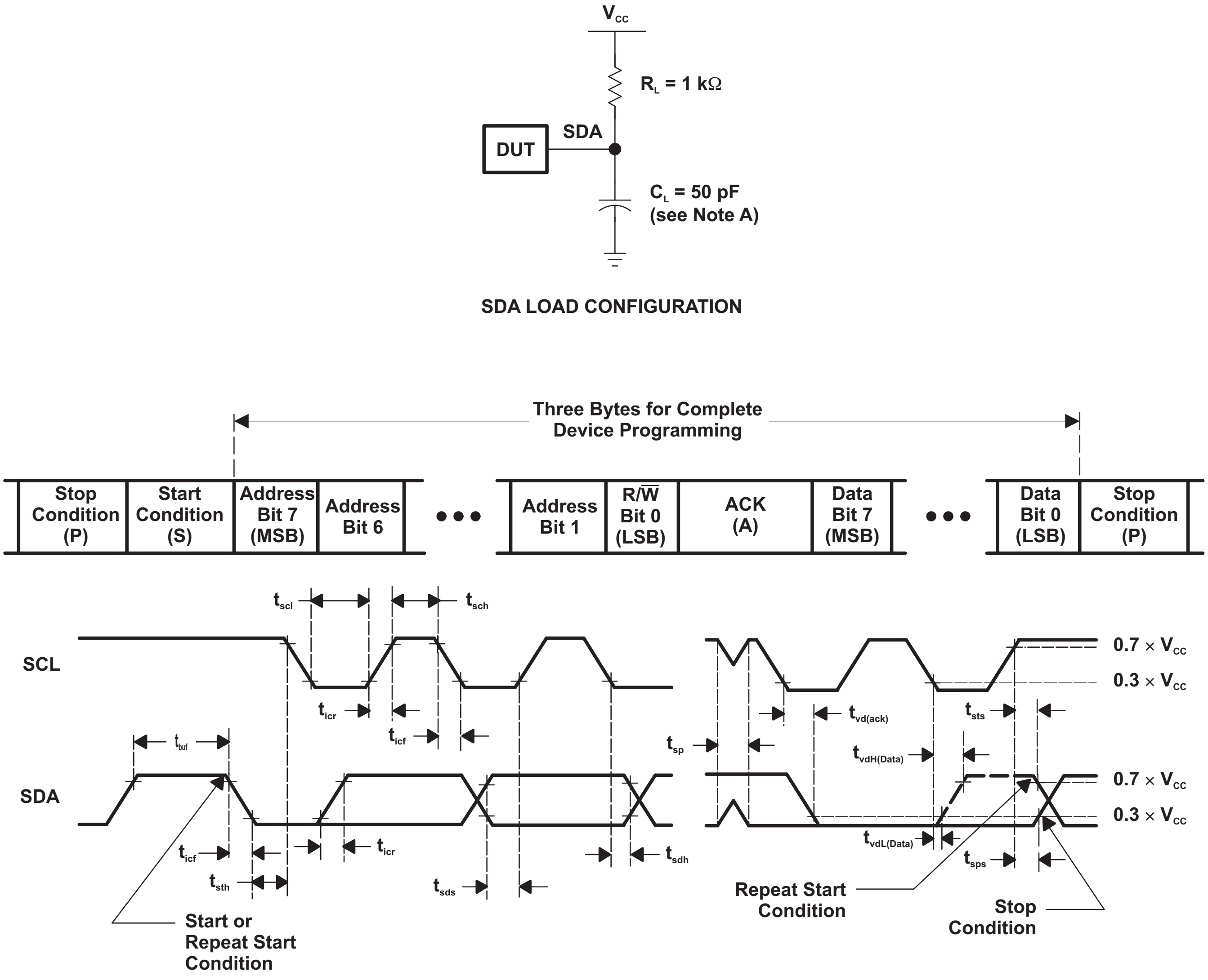


Figure 4. On-Resistance (R_{ON}) vs Supply Voltage (V_{CC}) at Three Temperatures

\

7 Parameter Measurement Information



SDA LOAD CONFIGURATION

VOLTAGE WAVEFORMS

BYTE	DESCRIPTION
1	I^2C address
2, 3	P-port data

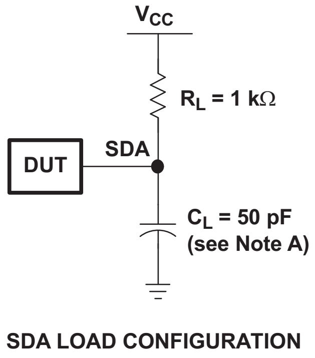
A. C_L includes probe and jig capacitance.

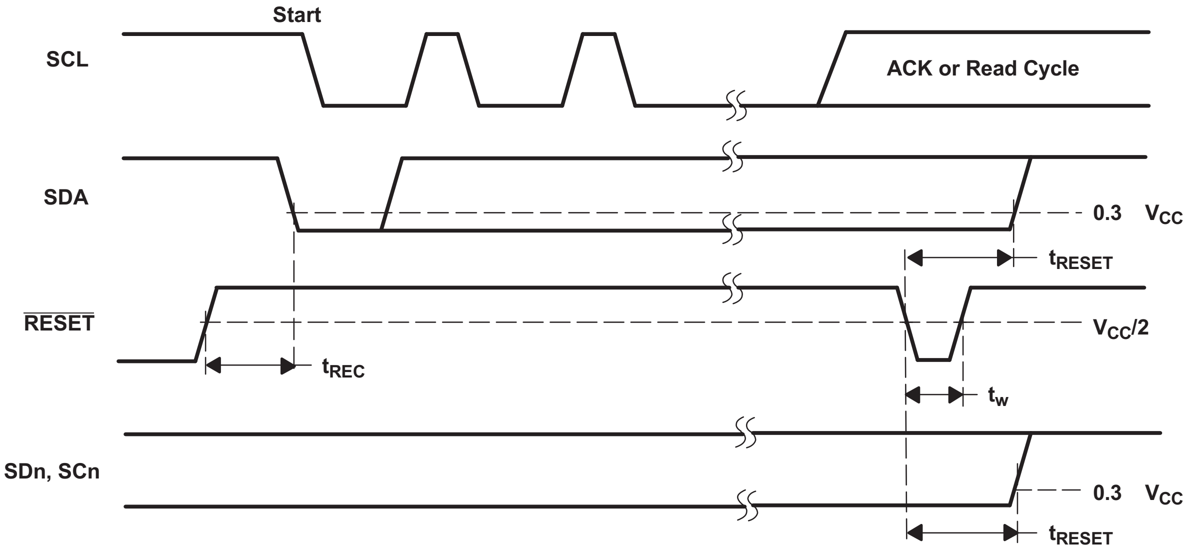
B. All inputs are supplied by generators having the following characteristics: PRR ≤ 10 MHz, Z_O = 50 Ω, t_r/t_f ≤ 30 ns.

C. Not all parameters and waveforms are applicable to all devices.

Figure 5. I^2C Load Circuit and Voltage Waveforms

Parameter Measurement Information (continued)





A. C_L includes probe and jig capacitance.

B. All inputs are supplied by generators having the following characteristics: PRR ≤ 10 MHz, Z_O = 50 Ω, t_r/t_f ≤ 30 ns.

C. I/Os are configured as inputs.

D. Not all parameters and waveforms are applicable to all devices.

Figure 6. Reset Load Circuit and Voltage Waveforms

8 Detailed Description

8.1 Overview

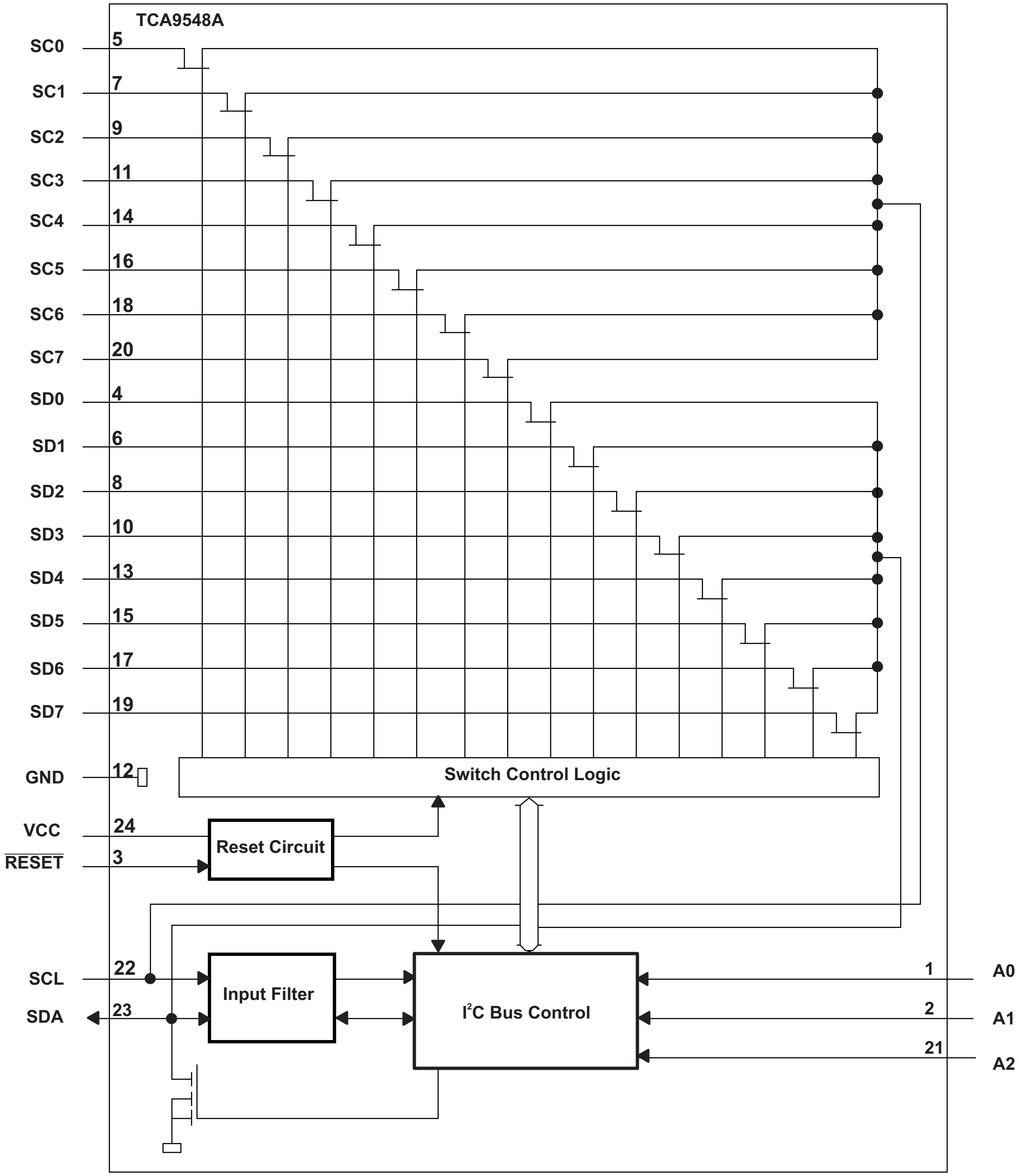
The TCA9548A is an 8-channel, bidirectional translating I^{2}C switch. The master SCL/SDA signal pair is directed to eight channels of slave devices, SC0/SD0-SC7/SD7. Any individual downstream channel can be selected as well as any combination of the eight channels.

The device offers an active-low $\overline{\text{RESET}}$ input which resets the state machine and allows the TCA9548A to recover must one of the downstream I^{2}C buses get stuck in a low state. The state machine of the device can also be reset by cycling the power supply, V_{CC}, also known as a power-on reset (POR). Both the $\overline{\text{RESET}}$ function and a POR cause all channels to be deselected.

The connections of the I^{2}C data path are controlled by the same I^{2}C master device that is switched to communicate with multiple I^{2}C slaves. After the successful acknowledgment of the slave address (hardware selectable by A0, A1, and A2 pins), a single 8-bit control register is written to or read from to determine the selected channels.

The TCA9548A may also be used for voltage translation, allowing the use of different bus voltages on each SCn/SDn pair such that 1.8-V, 2.5-V, or 3.3-V parts can communicate with 5-V parts. This is achieved by using external pull-up resistors to pull the bus up to the desired voltage for the master and each slave channel.

8.2 Functional Block Diagram



8.3 Feature Description

The TCA9548A is an 8-channel, bidirectional translating switch for I^{2}C buses that supports Standard-Mode (100 kHz) and Fast-Mode (400 kHz) operation. The TCA9548A features I^{2}C control using a single 8-bit control register in which each bit controls the enabling and disabling of one of the corresponding 8 switch channels for I^{2}C data flow. Depending on the application, voltage translation of the I^{2}C bus can also be achieved using the TCA9548A to allow 1.8-V, 2.5-V, or 3.3-V parts to communicate with 5-V parts. Additionally, in the event that communication on the I^{2}C bus enters a fault state, the TCA9548A can be reset to resume normal operation using the $\overline{\text{RESET}}$ pin feature or by a power-on reset which results from cycling power to the device.

8.4 Device Functional Modes

8.4.1 $\overline{\text{RESET}}$ Input

The $\overline{\text{RESET}}$ input is an active-low signal that may be used to recover from a bus-fault condition. When this signal is asserted low for a minimum of t_{WL}, the TCA9548A resets its registers and I^{2}C state machine and deselects all channels. The $\overline{\text{RESET}}$ input must be connected to V_{CC} through a pull-up resistor.

8.4.2 Power-On Reset

When power is applied to the VCC pin, an internal power-on reset holds the TCA9548A in a reset condition until V_{CC} has reached V_{PORR}. At this point, the reset condition is released, and the TCA9548A registers and I^{2}C state machine are initialized to their default states, all zeroes, causing all the channels to be deselected. Thereafter, V_{CC} must be lowered below V_{PORF} to reset the device.

8.5 Programming

8.5.1 I^{2}C Interface

The TCA9548A has a standard bidirectional I^{2}C interface that is controlled by a master device in order to be configured or read the status of this device. Each slave on the I^{2}C bus has a specific device address to differentiate between other slave devices that are on the same I^{2}C bus. Many slave devices require configuration upon startup to set the behavior of the device. This is typically done when the master accesses internal register maps of the slave, which have unique register addresses. A device can have one or multiple registers where data is stored, written, or read.

The physical I^{2}C interface consists of the serial clock (SCL) and serial data (SDA) lines. Both SDA and SCL lines must be connected to V_{CC} through a pull-up resistor. The size of the pull-up resistor is determined by the amount of capacitance on the I^{2}C lines. (For further details, see the *I^{2}C Pull-up Resistor Calculation* application report. Data transfer may be initiated only when the bus is idle. A bus is considered idle if both SDA and SCL lines are high after a STOP condition (See Figure 7 and Figure 8).

The following is the general procedure for a master to access a slave device:

1. If a master wants to send data to a slave:
   - Master-transmitter sends a START condition and addresses the slave-receiver.
   - Master-transmitter sends data to slave-receiver.
   - Master-transmitter terminates the transfer with a STOP condition.
2. If a master wants to receive or read data from a slave:
   - Master-receiver sends a START condition and addresses the slave-transmitter.
   - Master-receiver sends the requested register to read to slave-transmitter.
   - Master-receiver receives data from the slave-transmitter.

Programming (continued)

- Master-receiver terminates the transfer with a STOP condition.

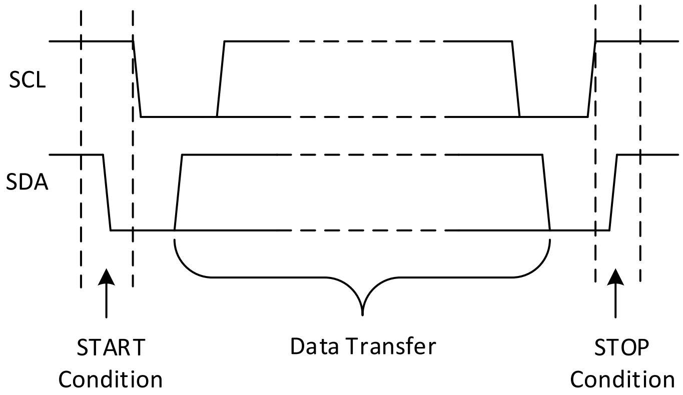


Figure 7. Definition of Start and Stop Conditions

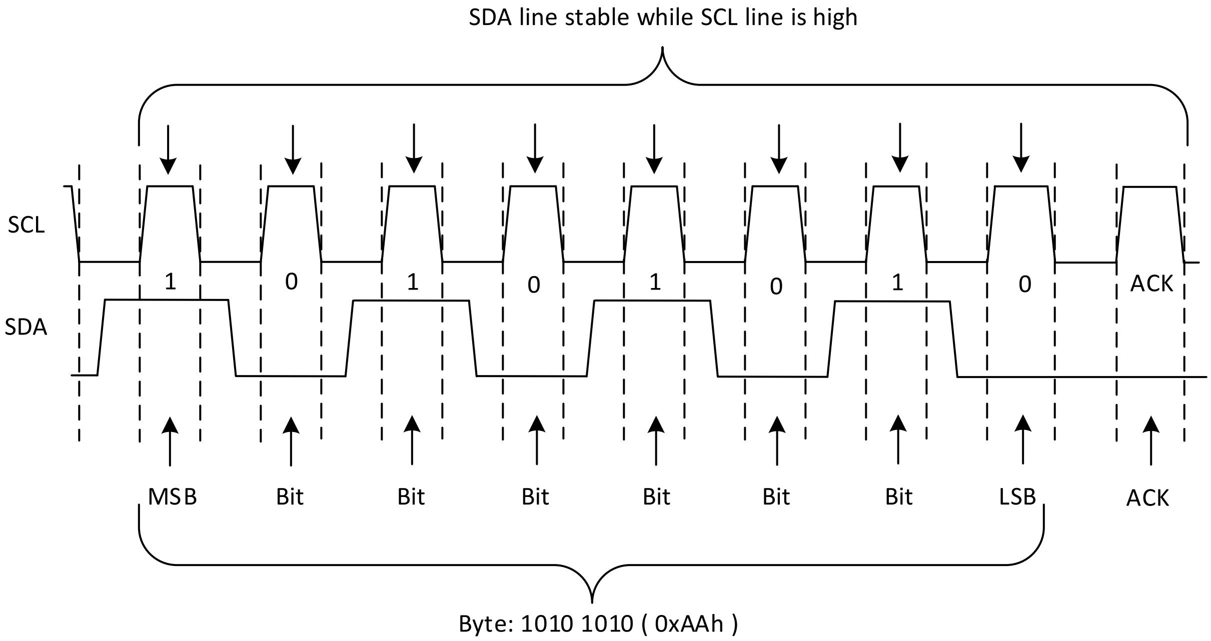

SDA line stable while SCL line is high
SCL
SDA
1
0
1
0
1
0
1
0
ACK
MSB
Bit
Bit
Bit
Bit
Bit
Bit
LSB
ACK
Byte: 1010 1010 (0xAAh)

Figure 8. Bit Transfer

8.5.2 Device Address

Figure 9 shows the address byte of the TCA9548A.

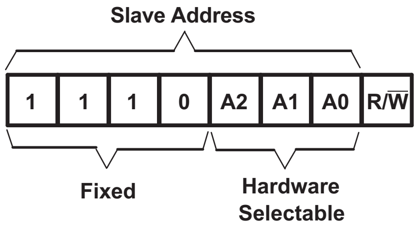


Figure 9. TCA9548A Address

Programming (continued)

The last bit of the slave address defines the operation (read or write) to be performed. When it is high (1), a read is selected, while a low (0) selects a write operation.

Table 1 shows the TCA9548A address reference.

Table 1. Address Reference

INPUTS			I^2C BUS SLAVE ADDRESS
A2	A1	A0	
L	L	L	112 (decimal), 70 (hexadecimal)
L	L	H	113 (decimal), 71 (hexadecimal)
L	H	L	114 (decimal), 72 (hexadecimal)
L	H	H	115 (decimal), 73 (hexadecimal)
H	L	L	116 (decimal), 74 (hexadecimal)
H	L	H	117 (decimal), 75 (hexadecimal)
H	H	L	118 (decimal), 76 (hexadecimal)
H	H	H	119 (decimal), 77 (hexadecimal)

8.5.3 Bus Transactions

Data must be sent to and received from the slave devices, and this is accomplished by reading from or writing to registers in the slave device.

Registers are locations in the memory of the slave which contain information, whether it be the configuration information or some sampled data to send back to the master. The master must write information to these registers in order to instruct the slave device to perform a task.

While it is common to have registers in I^2C slaves, note that not all slave devices have registers. Some devices are simple and contain only 1 register, which may be written to directly by sending the register data immediately after the slave address, instead of addressing a register. The TCA9548A is example of a single-register device, which is controlled via I^2C commands. Since it has 1 bit to enable or disable a channel, there is only 1 register needed, and the master merely writes the register data after the slave address, skipping the register number.

8.5.3.1 Writes

To write on the I^2C bus, the master sends a START condition on the bus with the address of the slave, as well as the last bit (the R/$\overline{W}$ bit) set to 0, which signifies a write. The slave acknowledges, letting the master know it is ready. After this, the master starts sending the control register data to the slave until the master has sent all the data necessary (which is sometimes only a single byte), and the master terminates the transmission with a STOP condition.

There is no limit to the number of bytes sent, but the last byte sent is what is in the register.

Figure 10 shows an example of writing a single byte to a slave register.

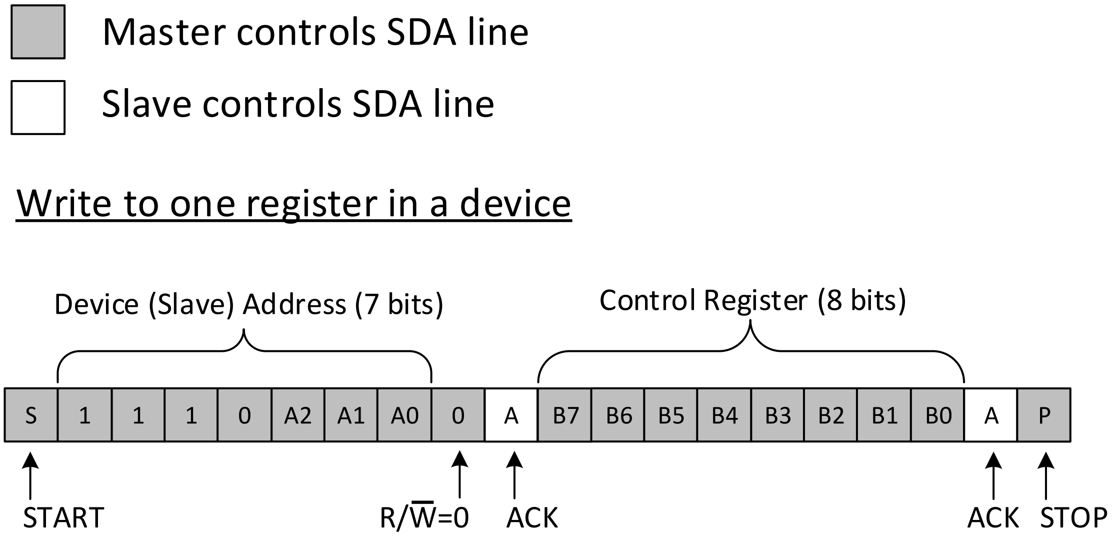


Figure 10. Write to Register

8.5.3.2 Reads

Reading from a slave is very similar to writing, but the master sends a START condition, followed by the slave address with the R/$\overline{W}$ bit set to 1 (signifying a read). The slave acknowledges the read request, and the master releases the SDA bus but continues supplying the clock to the slave. During this part of the transaction, the master becomes the master-receiver, and the slave becomes the slave-transmitter.

The master continues to send out the clock pulses, but releases the SDA line so that the slave can transmit data. At the end of every byte of data, the master sends an ACK to the slave, letting the slave know that it is ready for more data. Once the master has received the number of bytes it is expecting, it sends a NACK, signaling to the slave to halt communications and release the bus. The master follows this up with a STOP condition.

Figure 11 shows an example of reading a single byte from a slave register.

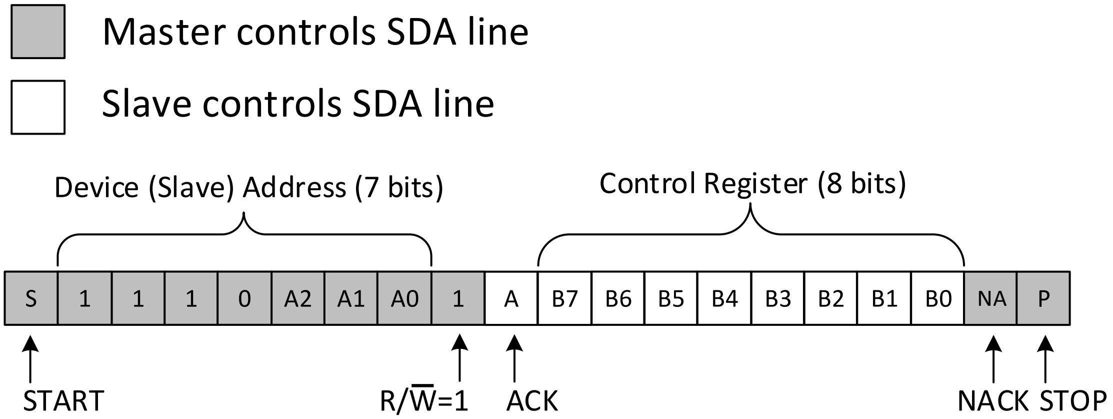


Figure 11. Read from Control Register

8.5.4 Control Register

Following the successful acknowledgment of the address byte, the bus master sends a command byte that is stored in the control register in the TCA9548A (see Figure 12). This register can be written and read via the I^{2}C bus. Each bit in the command byte corresponds to a SCn/SDn channel and a high (or 1) selects this channel. Multiple SCn/SDn channels may be selected at the same time. When a channel is selected, the channel becomes active after a stop condition has been placed on the I^{2}C bus. This ensures that all SCn/SDn lines are in a high state when the channel is made active, so that no false conditions are generated at the time of connection. A stop condition always must occur immediately after the acknowledge cycle. If multiple bytes are received by the TCA9548A, it saves the last byte received.

Channel Selection Bits (Read/Write)

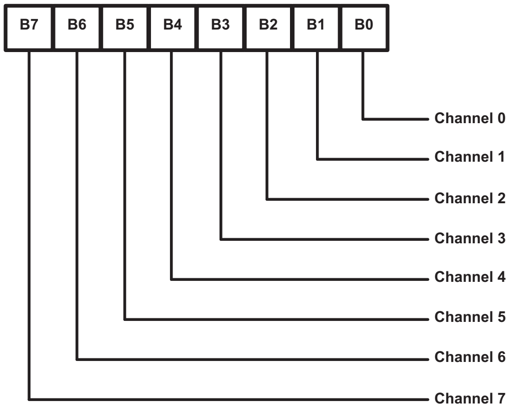


Figure 12. Control Register

Table 2 shows the TCA9548A Command Byte Definition.

Table 2. Command Byte Definition

CONTROL REGISTER BITS								COMMAND
B7	B6	B5	B4	B3	B2	B1	B0	
X	X	X	X	X	X	X	0	Channel 0 disabled
							1	Channel 0 enabled
X	X	X	X	X	X	0	X	Channel 1 disabled
						1		Channel 1 enabled
X	X	X	X	X	0	X	X	Channel 2 disabled
					1			Channel 2 enabled
X	X	X	X	0	X	X	X	Channel 3 disabled
				1				Channel 3 enabled
X	X	X	0	X	X	X	X	Channel 4 disabled
			1					Channel 4 enabled
X	X	0	X	X	X	X	X	Channel 5 disabled
		1						Channel 5 enabled
X	0	X	X	X	X	X	X	Channel 6 disabled
	1							Channel 6 enabled
0	X	X	X	X	X	X	X	Channel 7 disabled
1								Channel 7 enabled
0	0	0	0	0	0	0	0	No channel selected, power-up/reset default state

8.5.5 $\overline{\text{RESET}}$ Input

The $\overline{\text{RESET}}$ input is an active-low signal that may be used to recover from a bus-fault condition. When this signal is asserted low for a minimum of t_{WL}, the TCA9548A resets its registers and I^2C state machine and deselects all channels. The $\overline{\text{RESET}}$ input must be connected to V_{CC} through a pull-up resistor.

8.5.6 Power-On Reset

When power (from 0 V) is applied to V_{CC}, an internal power-on reset holds the TCA9548A in a reset condition until V_{CC} has reached V_{POR}. At that point, the reset condition is released and the TCA9548A registers and I^2C state machine initialize to their default states. After that, V_{CC} must be lowered to below V_{POR} and then back up to the operating voltage for a power-reset cycle.

9 Application and Implementation

NOTE

Information in the following applications sections is not part of the TI component specification, and TI does not warrant its accuracy or completeness. TI's customers are responsible for determining suitability of components for their purposes. Customers should validate and test their design implementation to confirm system functionality.

9.1 Application Information

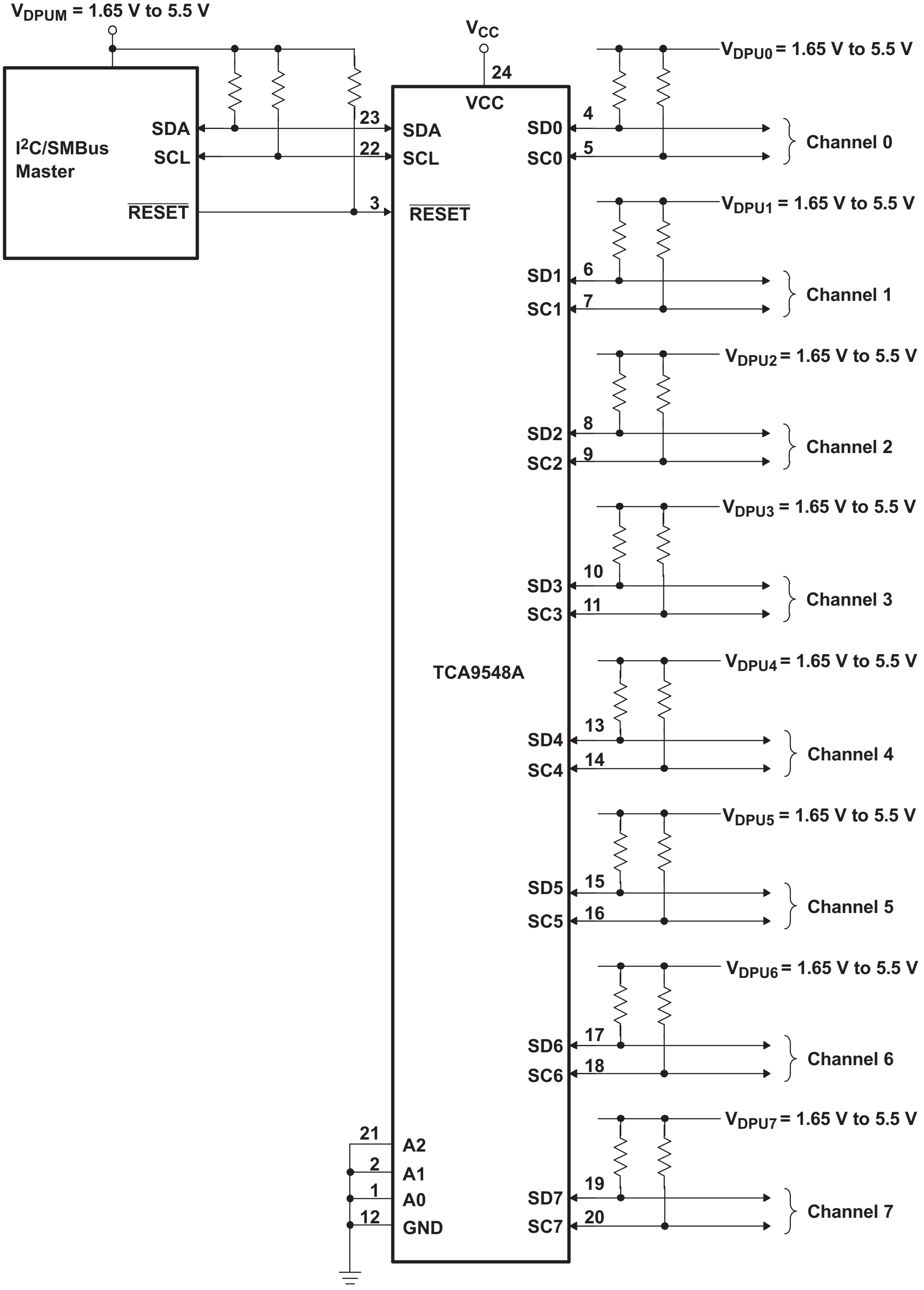
Applications of the TCA9548A contain an I^{2}C (or SMBus) master device and up to eight I^{2}C slave devices. The downstream channels are ideally used to resolve I^{2}C slave address conflicts. For example, if eight identical digital temperature sensors are needed in the application, one sensor can be connected at each channel: 0-7. When the temperature at a specific location needs to be read, the appropriate channel can be enabled and all other channels switched off, the data can be retrieved, and the I^{2}C master can move on and read the next channel.

In an application where the I^{2}C bus contains many additional slave devices that do not result in I^{2}C slave address conflicts, these slave devices can be connected to any desired channel to distribute the total bus capacitance across multiple channels. If multiple switches are enabled simultaneously, additional design requirements must be considered (see the *Design Requirements* section and *Detailed Design Procedure* section).

9.2 Typical Application

Figure 13 shows an application in which the TCA9548A can be used.

Typical Application (continued)


V_{DPUM} = 1.65 V to 5.5 V
V_{CC}
24
VCC
I²C/SMBus Master
SDA
SCL
RESET
23
22
3
SDA
SCL
RESET
SD0
SC0
4
5
V_{DPU0} = 1.65 V to 5.5 V
Channel 0
V_{DPU1} = 1.65 V to 5.5 V
SD1
SC1
6
7
Channel 1
V_{DPU2} = 1.65 V to 5.5 V
SD2
SC2
8
9
Channel 2
V_{DPU3} = 1.65 V to 5.5 V
SD3
SC3
10
11
Channel 3
TCA9548A
V_{DPU4} = 1.65 V to 5.5 V
SD4
SC4
13
14
Channel 4
V_{DPU5} = 1.65 V to 5.5 V
SD5
SC5
15
16
Channel 5
V_{DPU6} = 1.65 V to 5.5 V
SD6
SC6
17
18
Channel 6
V_{DPU7} = 1.65 V to 5.5 V
21
A2
2
A1
1
A0
12
GND
SD7
SC7
19
20
Channel 7

Pin numbers shown are for the PW package.

Figure 13. Typical Application Schematic

Typical Application (continued)

9.2.1 Design Requirements

A typical application of the TCA9548A contains one or more data pull-up voltages, V_{DPUX}, one for the master device (V_{DPUM}) and one for each of the selectable slave channels (V_{DPU0} – V_{DPU7}). In the event where the master device and all slave devices operate at the same voltage, then $V_{DPUM} = V_{DPUX} = VCC$. In an application where voltage translation is necessary, additional design requirements must be considered to determine an appropriate V_{CC} voltage.

The A0, A1, and A2 pins are hardware selectable to control the slave address of the TCA9548A. These pins may be tied directly to GND or V_{CC} in the application.

If multiple slave channels are activated simultaneously in the application, then the total I_{OL} from SCL/SDA to GND on the master side is the sum of the currents through all pull-up resistors, R_p.

The pass-gate transistors of the TCA9548A are constructed such that the V_{CC} voltage can be used to limit the maximum voltage that is passed from one I^2C bus to another.

Figure 14 shows the voltage characteristics of the pass-gate transistors (note that the graph was generated using data specified in the *Electrical Characteristics* table). In order for the TCA9548A to act as a voltage translator, the V_{pass} voltage must be equal to or lower than the lowest bus voltage. For example, if the main bus is running at 5 V and the downstream buses are 3.3 V and 2.7 V, V_{pass} must be equal to or below 2.7 V to effectively clamp the downstream bus voltages. As shown in Figure 14, $V_{pass(max)}$ is 2.7 V when the TCA9548A supply voltage is 4 V or lower, so the TCA9548A supply voltage could be set to 3.3 V. Pull-up resistors then can be used to bring the bus voltages to their appropriate levels (see Figure 13).

9.2.2 Detailed Design Procedure

Once all the slaves are assigned to the appropriate slave channels and bus voltages are identified, the pull-up resistors, R_p, for each of the buses need to be selected appropriately. The minimum pull-up resistance is a function of V_{DPUX}, $V_{OL,(max)}$, and I_{OL} as shown in Equation 1:

$$R_{p(min)} = \frac{V_{DPUX} - V_{OL(max)}}{I_{OL}} \tag{1}$$

The maximum pull-up resistance is a function of the maximum rise time, t_r (300 ns for fast-mode operation, f_{SCL} = 400 kHz) and bus capacitance, C_b as shown in Equation 2:

$$R_{p(max)} = \frac{t_r}{0.8473 \times C_b} \tag{2}$$

The maximum bus capacitance for an I^2C bus must not exceed 400 pF for fast-mode operation. The bus capacitance can be approximated by adding the capacitance of the TCA9548A, $C_{io(OFF)}$, the capacitance of wires, connections and traces, and the capacitance of each individual slave on a given channel. If multiple channels are activated simultaneously, each of the slaves on all channels contribute to total bus capacitance.

Typical Application (continued)

9.2.3 Application Curves

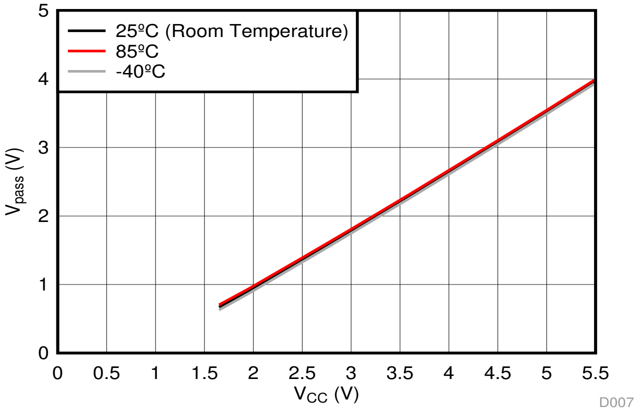
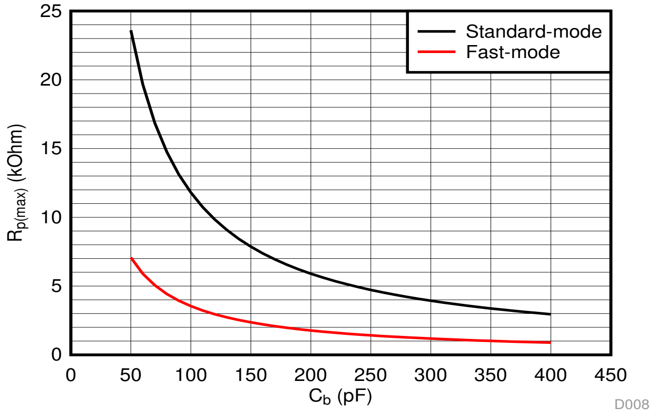


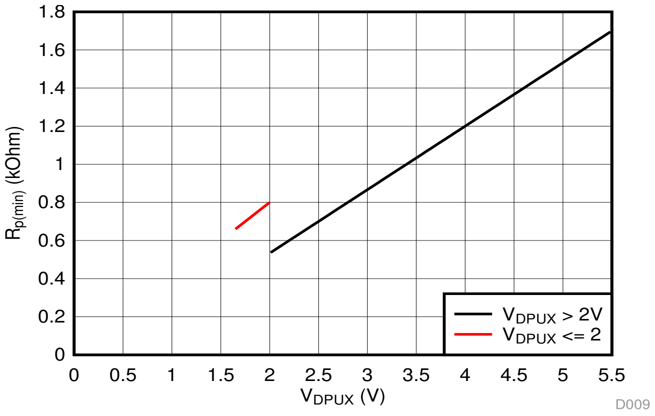
Figure 14. Pass-Gate Voltage (V_{pass}) vs Supply Voltage (V_{CC}) at Three Temperature Points



Standard-mode (f_{SCL} = 100 kHz, t_r = 1 µs)

Fast-mode (f_{SCL} = 400 kHz, t_r = 300 ns)

Figure 15. Maximum Pull-Up Resistance ($R_{p(max)}$) vs Bus Capacitance (C_b)



V_{OL} = 0.2*V_{DPUX}, I_{OL} = 2 mA when V_{DPUX} ≤ 2 V
V_{OL} = 0.4 V, I_{OL} = 3 mA when V_{DPUX} > 2 V

Figure 16. Minimum Pullup Resistance ($R_{p(min)}$) vs Pullup Reference Voltage (V_{DPUX})

10 Power Supply Recommendations

The operating power-supply voltage range of the TCA9548A is 1.65 V to 5.5 V applied at the VCC pin. When the TCA9548A is powered on for the first time or anytime the device must be reset by cycling the power supply, the power-on reset requirements must be followed to ensure the I^{2}C bus logic is initialized properly.

10.1 Power-On Reset Requirements

In the event of a glitch or data corruption, TCA9548A can be reset to its default conditions by using the power-on reset feature. Power-on reset requires that the device go through a power cycle to be completely reset. This reset also happens when the device is powered on for the first time in an application.

A power-on reset is shown in Figure 17.

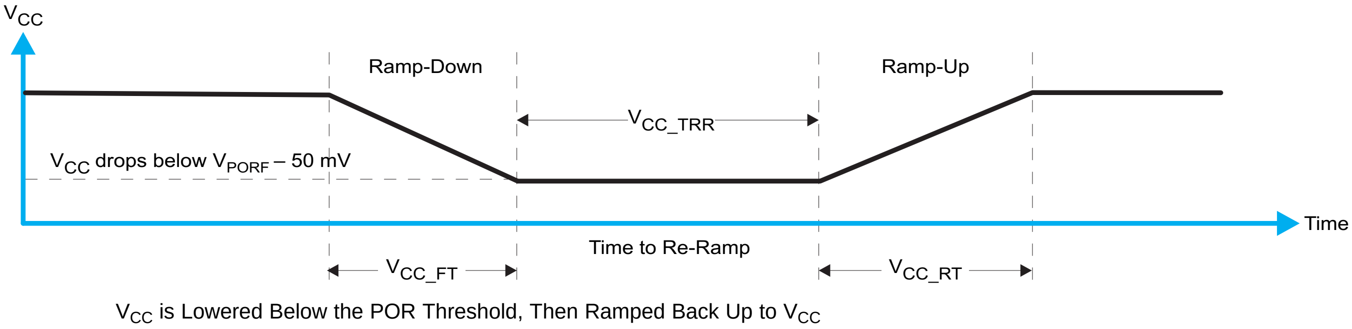

V_{CC}
Ramp-Down
Ramp-Up
V_{CC_TRR}
V_{CC} drops below V_{PORF} – 50 mV
Time
Time to Re-Ramp
V_{CC_FT}
V_{CC_RT}
V_{CC} is Lowered Below the POR Threshold, Then Ramped Back Up to V_{CC}

Figure 17. Power-On Reset Waveform

Table 3 specifies the performance of the power-on reset feature for TCA9548A for both types of power-on reset.

Table 3. Recommended Supply Sequencing and Ramp Rates[1]

PARAMETER			MIN	MAX	UNIT
V_{CC_FT}	Fall time	See Figure 17	1	100	ms
V_{CC_RT}	Rise time	See Figure 17	0.1	100	ms
V_{CC_TRR}	Time to re-ramp (when V_{CC} drops below $V_{PORF(min)}$ – 50 mV or when V_{CC} drops to GND)	See Figure 17	40		µs
V_{CC_GH}	Level that V_{CC} can glitch down to, but not cause a functional disruption when V_{CC_GW} = 1 µs	See Figure 18		1.2	V
V_{CC_GW}	Glitch width that does not cause a functional disruption when V_{CC_GH} = 0.5 × V_{CC}	See Figure 18		10	µs

[1] All supply sequencing and ramp rate values are measured at T_A = 25°C

Glitches in the power supply can also affect the power-on reset performance of this device. The glitch width (V_{CC_GW}) and height (V_{CC_GH}) are dependent on each other. The bypass capacitance, source impedance, and device impedance are factors that affect power-on reset performance. Figure 18 and Table 3 provide more information on how to measure these specifications.

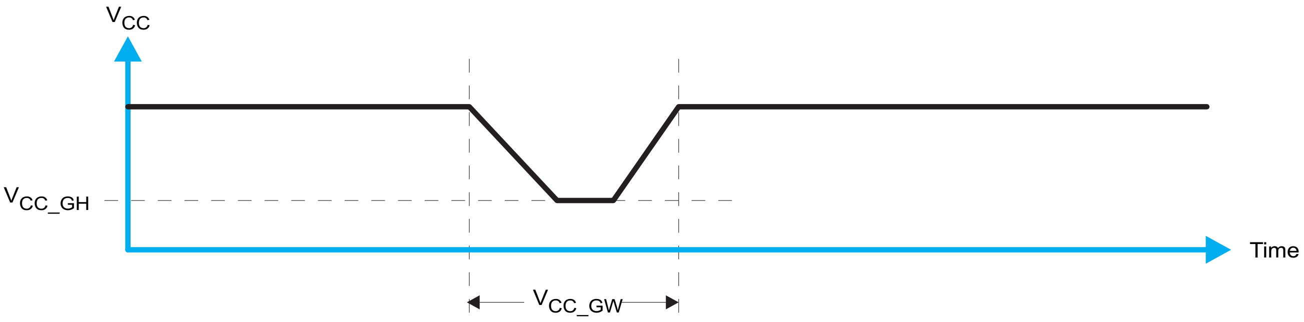


Figure 18. Glitch Width and Glitch Height

V_{POR} is critical to the power-on reset. V_{POR} is the voltage level at which the reset condition is released and all the registers and the I^2C/SMBus state machine are initialized to their default states. The value of V_{POR} differs based on the V_{CC} being lowered to or from 0. Figure 19 and Table 3 provide more details on this specification.

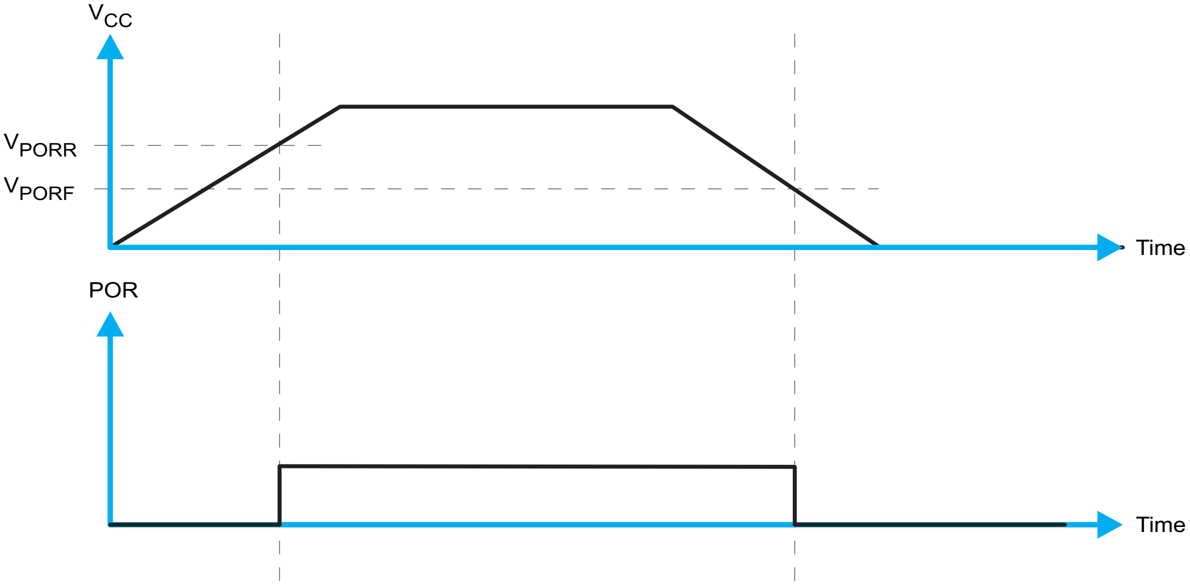


Figure 19. V_{POR}

11 Layout

11.1 Layout Guidelines

For PCB layout of the TCA9548A, common PCB layout practices must be followed but additional concerns related to high-speed data transfer such as matched impedances and differential pairs are not a concern for I^2C signal speeds. It is common to have a dedicated ground plane on an inner layer of the board and pins that are connected to ground must have a low-impedance path to the ground plane in the form of wide polygon pours and multiple vias. By-pass and de-coupling capacitors are commonly used to control the voltage on the VCC pin, using a larger capacitor to provide additional power in the event of a short power supply glitch and a smaller capacitor to filter out high-frequency ripple.

In an application where voltage translation is not required, all V_{DPUX} voltages and V_{CC} could be at the same potential and a single copper plane could connect all of pull-up resistors to the appropriate reference voltage. In an application where voltage translation is required, V_{DPUM} and V_{DPU0} – V_{DPU7}, may all be on the same layer of the board with split planes to isolate different voltage potentials.

To reduce the total I^2C bus capacitance added by PCB parasitics, data lines (SCn and SDn) must be a short as possible and the widths of the traces must also be minimized (for example, 5-10 mils depending on copper weight).

11.2 Layout Example

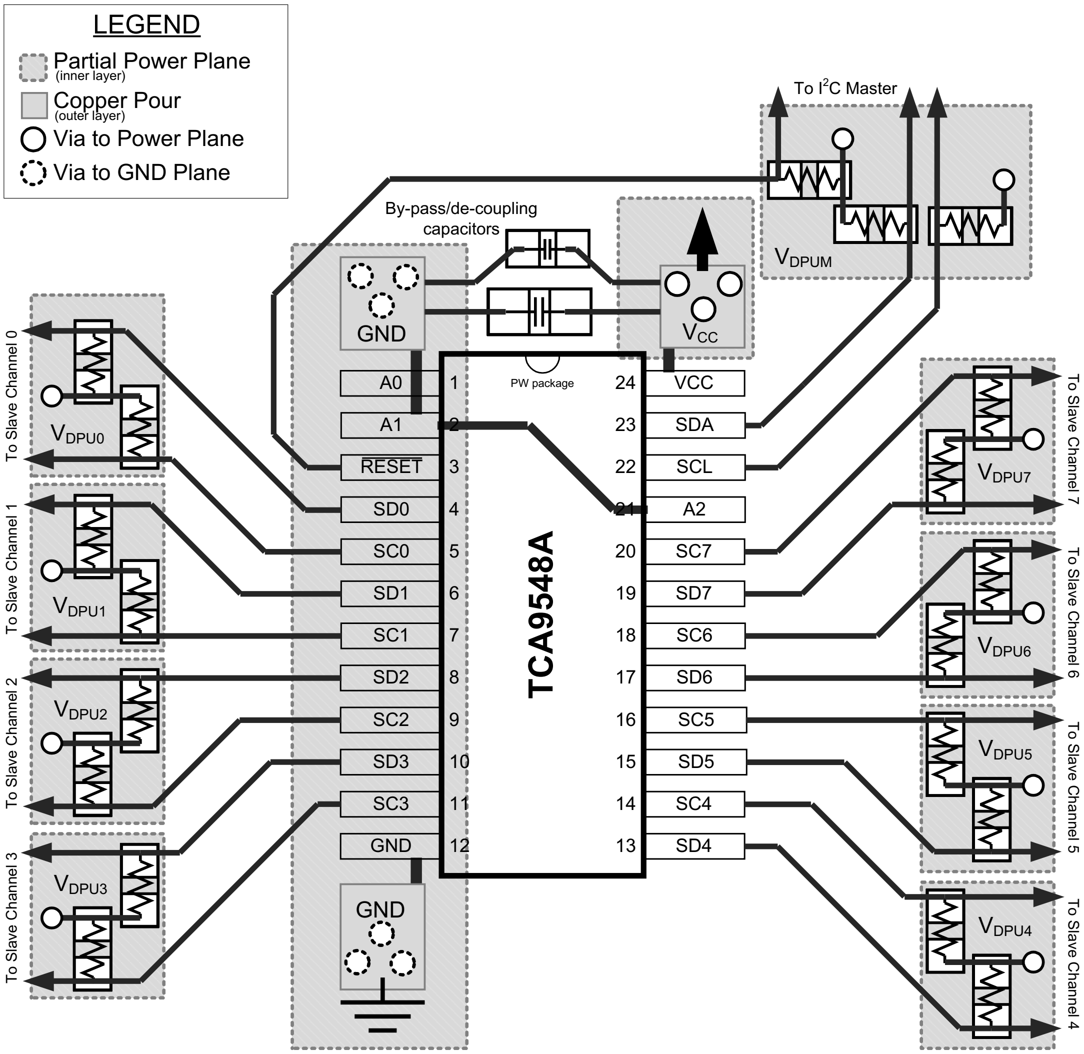


Figure 20. Layout Schematic

12 Device and Documentation Support

12.1 Documentation Support

12.1.1 Related Documentation

For related documentation see the following:

- *I2C Bus Pull-Up Resistor Calculation*
- *Maximum Clock Frequency of I2C Bus Using Repeaters*
- *Introduction to Logic*
- *Understanding the I2C Bus*
- *Choosing the Correct I2C Device for New Designs*
- *TCA9548AEVM User's Guide*

12.2 Receiving Notification of Documentation Updates

To receive notification of documentation updates, navigate to the device product folder on ti.com. In the upper right corner, click on *Alert me* to register and receive a weekly digest of any product information that has changed. For change details, review the revision history included in any revised document.

12.3 Support Resources

TI E2E™ support forums are an engineer's go-to source for fast, verified answers and design help — straight from the experts. Search existing answers or ask your own question to get the quick design help you need.

Linked content is provided "AS IS" by the respective contributors. They do not constitute TI specifications and do not necessarily reflect TI's views; see TI's Terms of Use.

12.4 Trademarks

E2E is a trademark of Texas Instruments.
All other trademarks are the property of their respective owners.

12.5 Electrostatic Discharge Caution

This integrated circuit can be damaged by ESD. Texas Instruments recommends that all integrated circuits be handled with appropriate precautions. Failure to observe proper handling and installation procedures can cause damage.

ESD damage can range from subtle performance degradation to complete device failure. Precision integrated circuits may be more susceptible to damage because very small parametric changes could cause the device not to meet its published specifications.

12.6 Glossary

SLYZ022 — *TI Glossary*.

This glossary lists and explains terms, acronyms, and definitions.

13 Mechanical, Packaging, and Orderable Information

The following pages include mechanical, packaging, and orderable information. This information is the most current data available for the designated devices. This data is subject to change without notice and revision of this document. For browser-based versions of this data sheet, refer to the left-hand navigation.

PACKAGE OPTION ADDENDUM

PACKAGING INFORMATION

Orderable Device	Status (1)	Package Type	Package Drawing	Pins	Package Qty	Eco Plan (2)	Lead finish/ Ball material (6)	MSL Peak Temp (3)	Op Temp (°C)	Device Marking (4/5)	Samples
TCA9548AMRGER	ACTIVE	VQFN	RGE	24	3000	RoHS & Green	NIPDAU	Level-2-260C-1 YEAR	-40 to 85	PW548A	Samples
TCA9548APWR	ACTIVE	TSSOP	PW	24	2000	RoHS & Green	NIPDAU	Level-1-260C-UNLIM	-40 to 85	PW548A	Samples
TCA9548ARGER	ACTIVE	VQFN	RGE	24	3000	RoHS & Green	NIPDAU	Level-2-260C-1 YEAR	-40 to 85	PW548A	Samples

[1] The marketing status values are defined as follows:
ACTIVE: Product device recommended for new designs.
LIFEBUY: TI has announced that the device will be discontinued, and a lifetime-buy period is in effect.
NRND: Not recommended for new designs. Device is in production to support existing customers, but TI does not recommend using this part in a new design.
PREVIEW: Device has been announced but is not in production. Samples may or may not be available.
OBSOLETE: TI has discontinued the production of the device.

[2] **RoHS:** TI defines "RoHS" to mean semiconductor products that are compliant with the current EU RoHS requirements for all 10 RoHS substances, including the requirement that RoHS substance do not exceed 0.1% by weight in homogeneous materials. Where designed to be soldered at high temperatures, "RoHS" products are suitable for use in specified lead-free processes. TI may reference these types of products as "Pb-Free".
RoHS Exempt: TI defines "RoHS Exempt" to mean products that contain lead but are compliant with EU RoHS pursuant to a specific EU RoHS exemption.
Green: TI defines "Green" to mean the content of Chlorine (Cl) and Bromine (Br) based flame retardants meet JS709B low halogen requirements of <=1000ppm threshold. Antimony trioxide based flame retardants must also meet the <=1000ppm threshold requirement.

[3] MSL, Peak Temp. - The Moisture Sensitivity Level rating according to the JEDEC industry standard classifications, and peak solder temperature.

[4] There may be additional marking, which relates to the logo, the lot trace code information, or the environmental category on the device.

[5] Multiple Device Markings will be inside parentheses. Only one Device Marking contained in parentheses and separated by a "~" will appear on a device. If a line is indented then it is a continuation of the previous line and the two combined represent the entire Device Marking for that device.

[6] Lead finish/Ball material - Orderable Devices may have multiple material finish options. Finish options are separated by a vertical ruled line. Lead finish/Ball material values may wrap to two lines if the finish value exceeds the maximum column width.

Important Information and Disclaimer: The information provided on this page represents TI's knowledge and belief as of the date that it is provided. TI bases its knowledge and belief on information provided by third parties, and makes no representation or warranty as to the accuracy of such information. Efforts are underway to better integrate information from third parties. TI has taken and continues to take reasonable steps to provide representative and accurate information but may not have conducted destructive testing or chemical analysis on incoming materials and chemicals. TI and TI suppliers consider certain information to be proprietary, and thus CAS numbers and other limited information may not be available for release.

In no event shall TI's liability arising out of such information exceed the total purchase price of the TI part(s) at issue in this document sold by TI to Customer on an annual basis.

OTHER QUALIFIED VERSIONS OF TCA9548A :

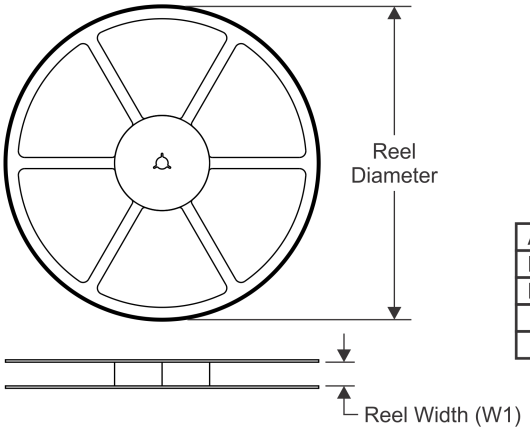
- Automotive: TCA9548A-Q1

NOTE: Qualified Version Definitions:

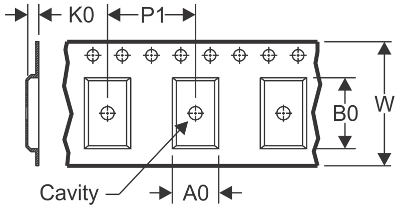
- Automotive - Q100 devices qualified for high-reliability automotive applications targeting zero defects

TAPE AND REEL INFORMATION

REEL DIMENSIONS

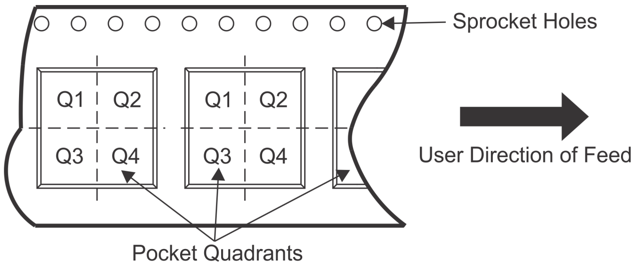


TAPE DIMENSIONS



A0	Dimension designed to accommodate the component width
B0	Dimension designed to accommodate the component length
K0	Dimension designed to accommodate the component thickness
W	Overall width of the carrier tape
P1	Pitch between successive cavity centers

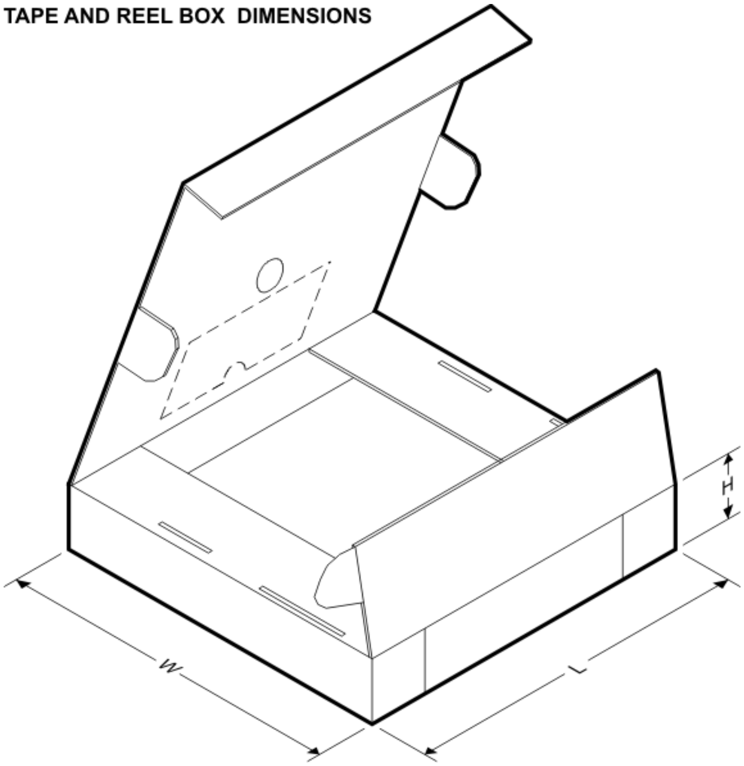
QUADRANT ASSIGNMENTS FOR PIN 1 ORIENTATION IN TAPE



*All dimensions are nominal

Device	Package Type	Package Drawing	Pins	SPQ	Reel Diameter (mm)	Reel Width W1 (mm)	A0 (mm)	B0 (mm)	K0 (mm)	P1 (mm)	W (mm)	Pin1 Quadrant
TCA9548AMRGER	VQFN	RGE	24	3000	330.0	12.4	4.25	4.25	1.15	8.0	12.0	Q1
TCA9548APWR	TSSOP	PW	24	2000	330.0	16.4	6.95	8.3	1.6	8.0	16.0	Q1
TCA9548ARGER	VQFN	RGE	24	3000	330.0	12.4	4.25	4.25	1.15	8.0	12.0	Q2

TAPE AND REEL BOX DIMENSIONS



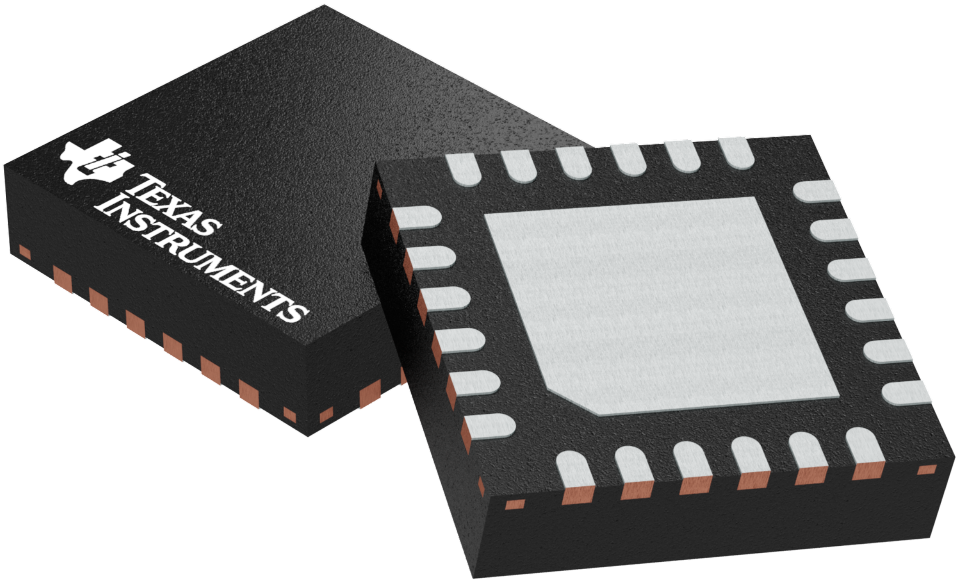
*All dimensions are nominal

Device	Package Type	Package Drawing	Pins	SPQ	Length (mm)	Width (mm)	Height (mm)
TCA9548AMRGER	VQFN	RGE	24	3000	853.0	449.0	35.0
TCA9548APWR	TSSOP	PW	24	2000	853.0	449.0	35.0
TCA9548ARGER	VQFN	RGE	24	3000	853.0	449.0	35.0

GENERIC PACKAGE VIEW

RGE 24 VQFN - 1 mm max height

PLASTIC QUAD FLATPACK - NO LEAD



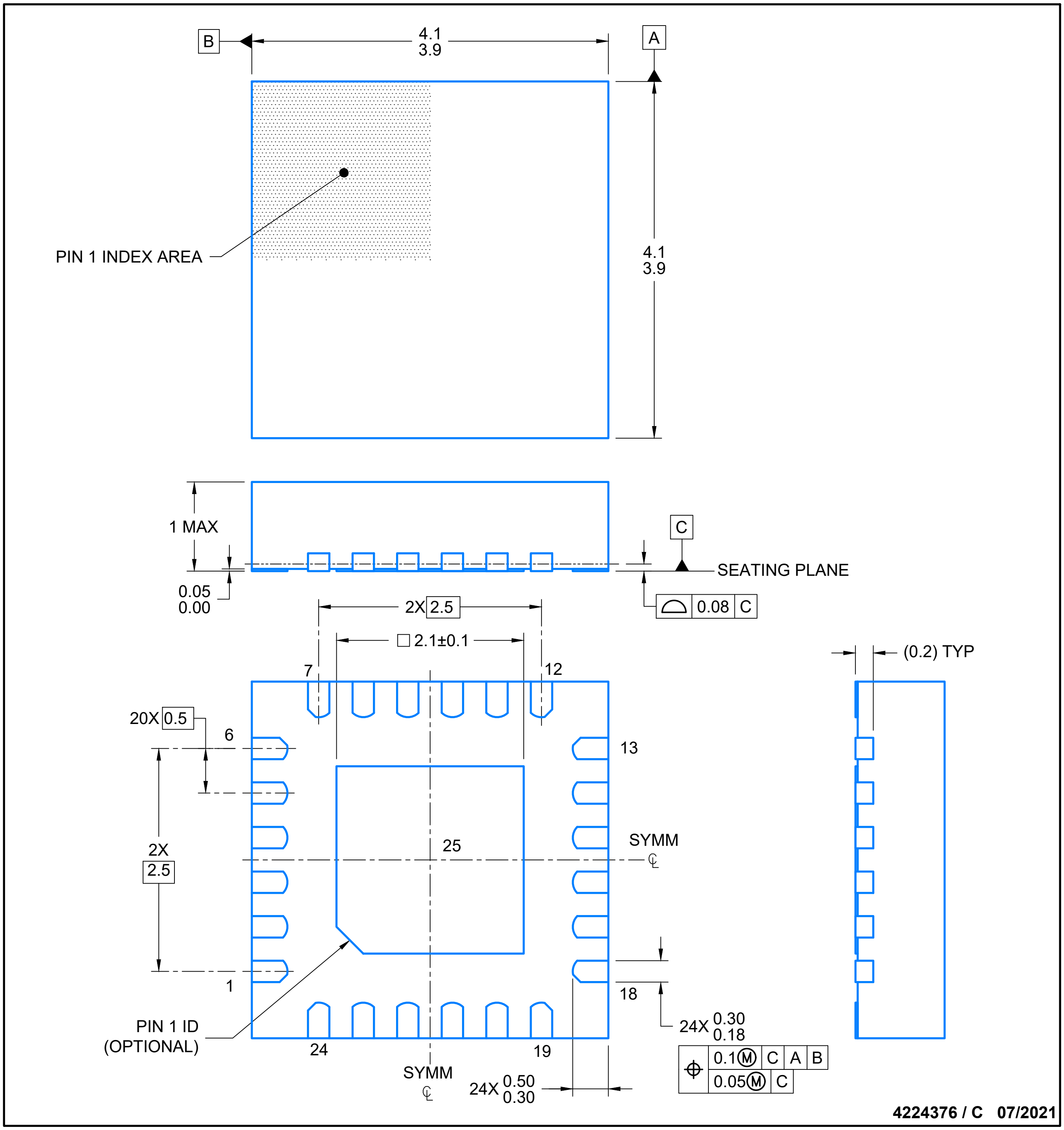
Images above are just a representation of the package family, actual package may vary.
Refer to the product data sheet for package details.

4204104/H

PACKAGE OUTLINE

VQFN - 1 mm max height

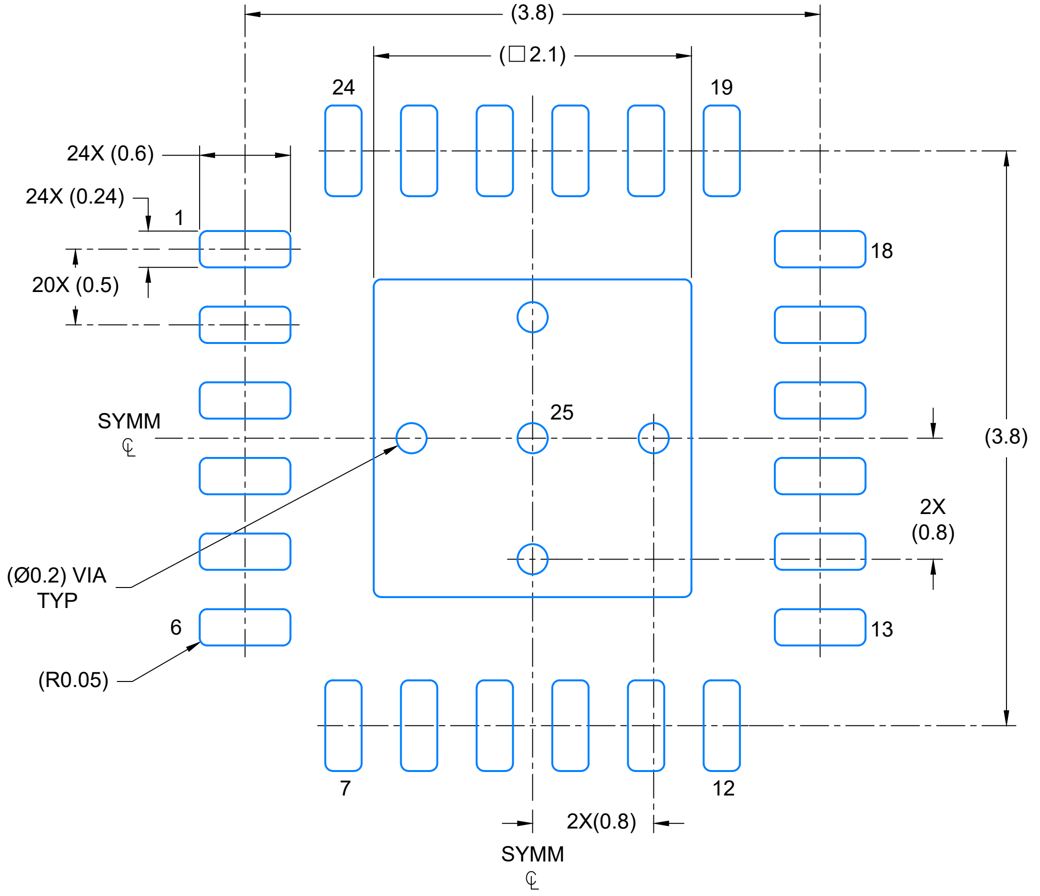
PLASTIC QUAD FLATPACK- NO LEAD



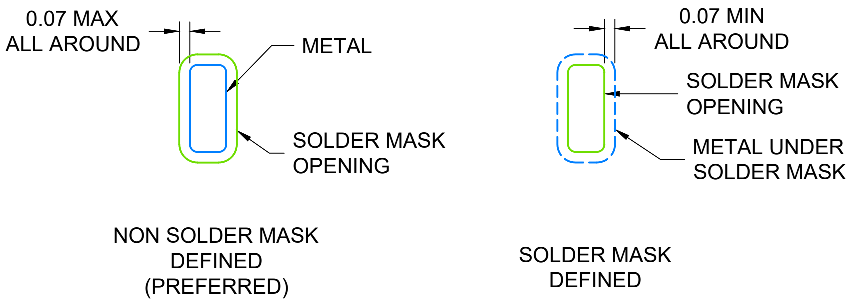
4224376 / C 07/2021

NOTES:

1. All linear dimensions are in millimeters. Any dimensions in parenthesis are for reference only. Dimensioning and tolerancing per ASME Y14.5M.
2. This drawing is subject to change without notice.
3. The package thermal pad must be soldered to the printed circuit board for thermal and mechanical performance.



LAND PATTERN EXAMPLE
SCALE: 20X

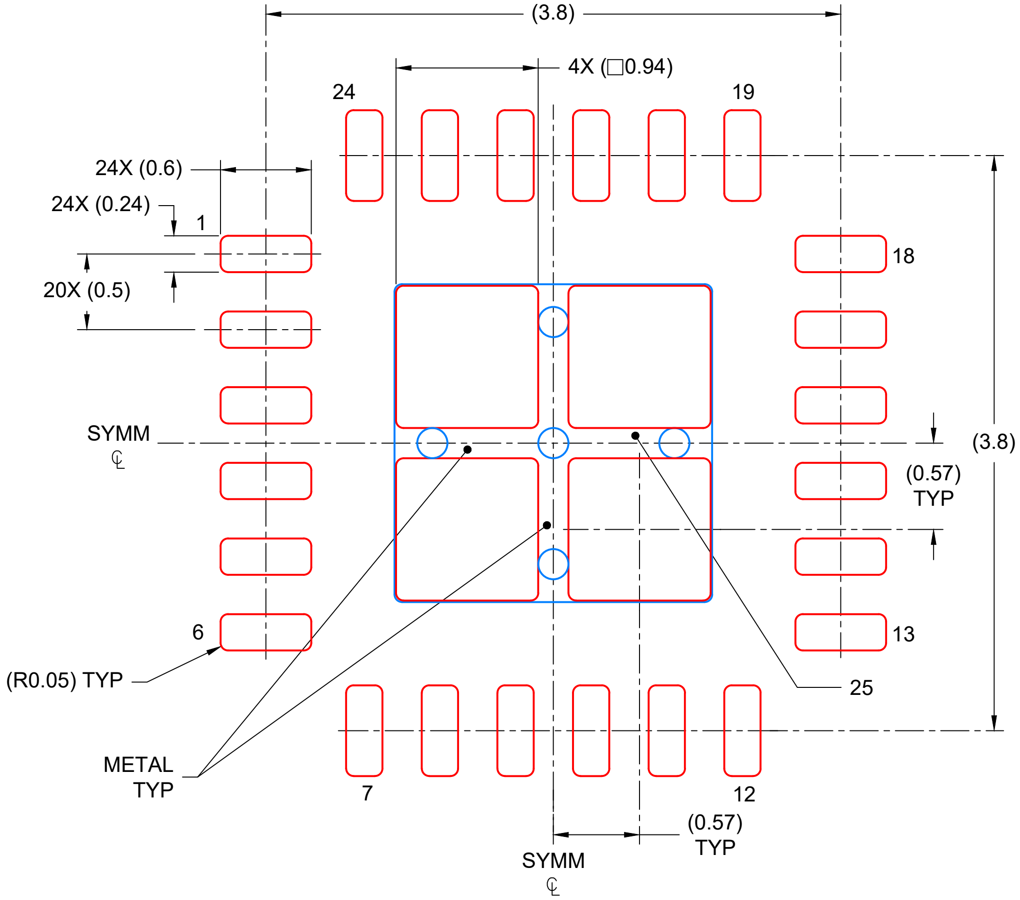


SOLDER MASK DETAILS

4224376 / C 06/2021

NOTES: (continued)

4. This package is designed to be soldered to a thermal pad on the board. For more information, see Texas Instruments literature number SLUA271 (www.ti.com/lit/slua271).
5. Solder mask tolerances between and around signal pads can vary based on board fabrication site.



SOLDER PASTE EXAMPLE
BASED ON 0.125 mm THICK STENCIL

EXPOSED PAD
80% PRINTED COVERAGE BY AREA
SCALE: 20X

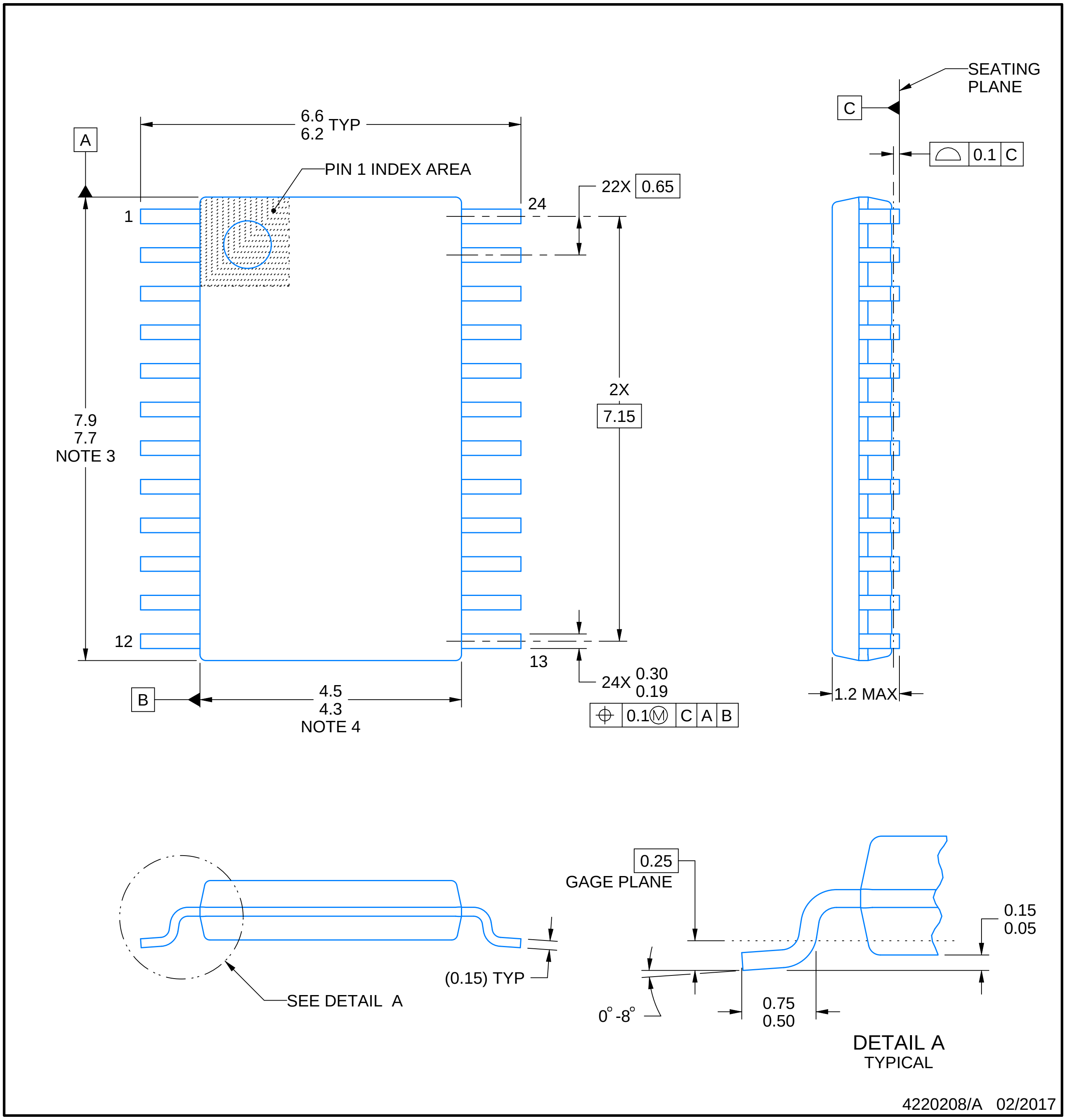
NOTES: (continued)

6. Laser cutting apertures with trapezoidal walls and rounded corners may offer better paste release. IPC-7525 may have alternate design recommendations..

PACKAGE OUTLINE

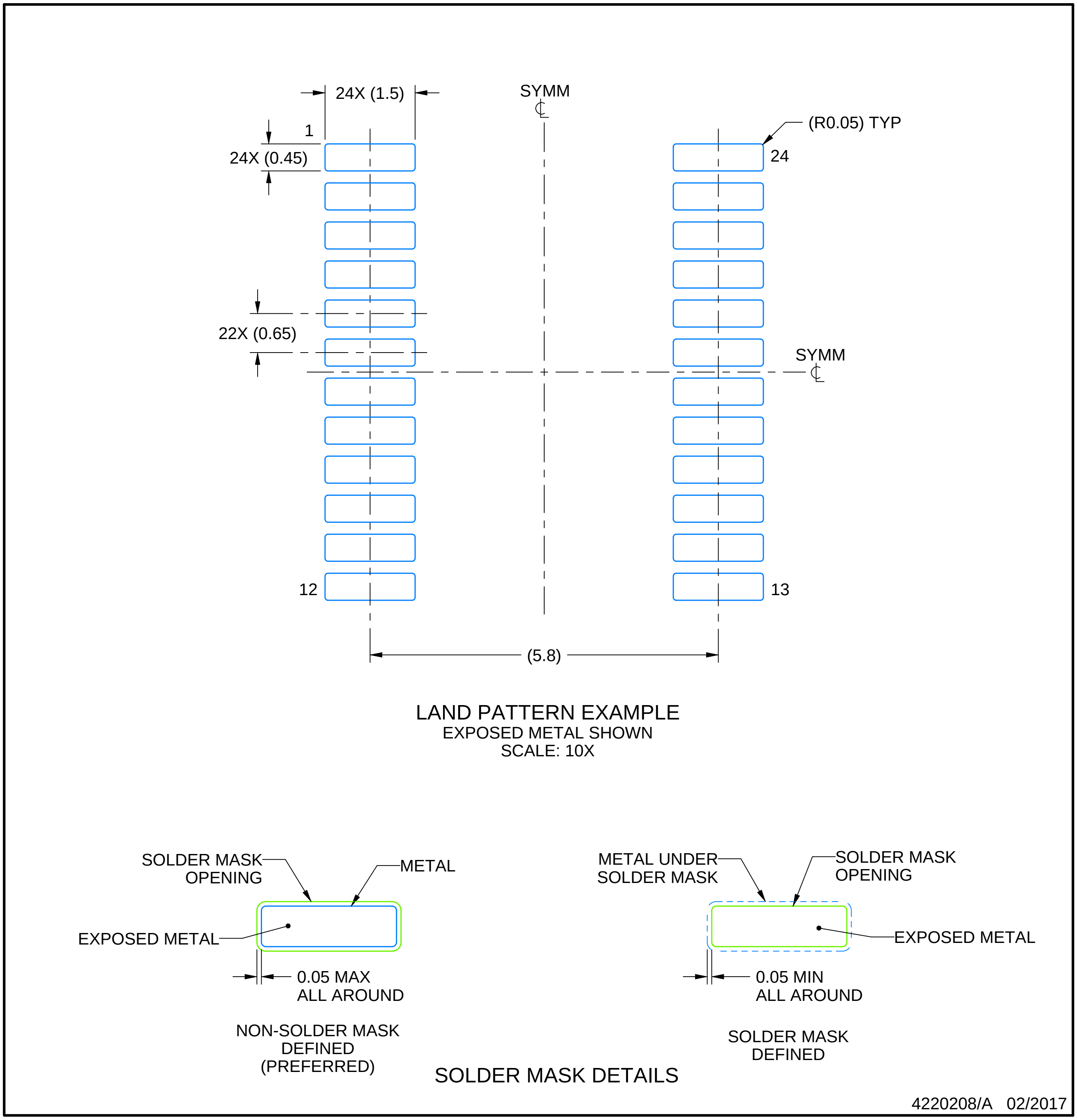
TSSOP - 1.2 mm max height

SMALL OUTLINE PACKAGE



NOTES:

1. All linear dimensions are in millimeters. Any dimensions in parenthesis are for reference only. Dimensioning and tolerancing per ASME Y14.5M.
2. This drawing is subject to change without notice.
3. This dimension does not include mold flash, protrusions, or gate burrs. Mold flash, protrusions, or gate burrs shall not exceed 0.15 mm per side.
4. This dimension does not include interlead flash. Interlead flash shall not exceed 0.25 mm per side.
5. Reference JEDEC registration MO-153.



LAND PATTERN EXAMPLE
EXPOSED METAL SHOWN
SCALE: 10X

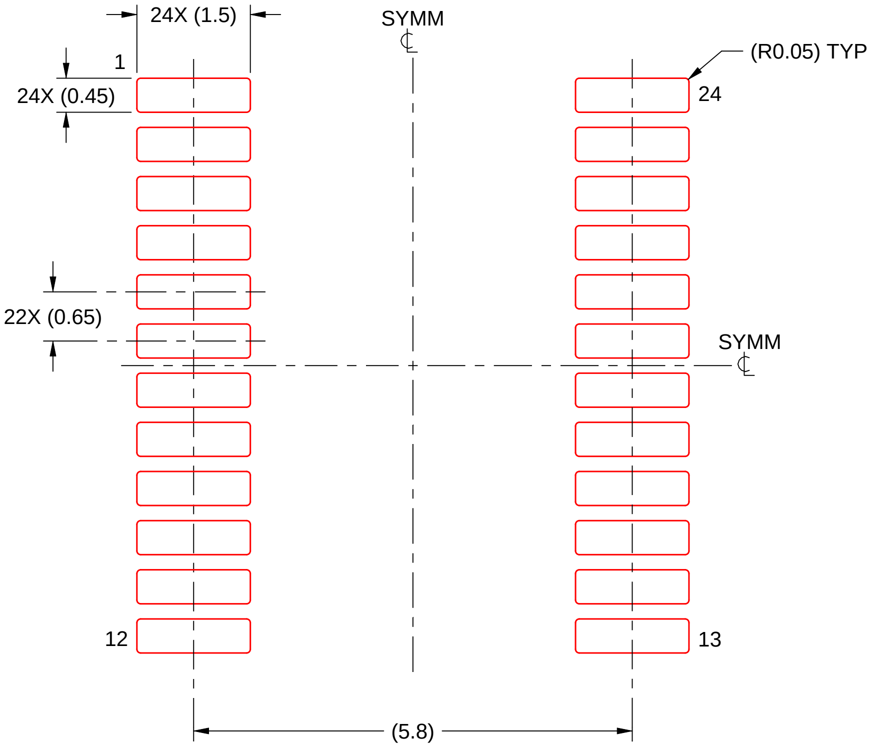
NON-SOLDER MASK
DEFINED
(PREFERRED)

SOLDER MASK
DEFINED

SOLDER MASK DETAILS

NOTES: (continued)

6. Publication IPC-7351 may have alternate designs.
7. Solder mask tolerances between and around signal pads can vary based on board fabrication site.



SOLDER PASTE EXAMPLE
BASED ON 0.125 mm THICK STENCIL
SCALE: 10X

4220208/A 02/2017

NOTES: (continued)

8. Laser cutting apertures with trapezoidal walls and rounded corners may offer better paste release. IPC-7525 may have alternate design recommendations.
9. Board assembly site may have different recommendations for stencil design.

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES "AS IS" AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI's products are provided subject to TI's Terms of Sale (https:www.ti.com/legal/termsofsale.html) or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI's provision of these resources does not expand or otherwise alter TI's applicable warranties or warranty disclaimers for TI products.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265