TRILL

Touch sensing for makers

Capacitive Touch Sensors

Revision C

Bela, a part of Augmented Instruments Ltd

Datasheet Version 1.3 – Revised 2025-02-03

Contents

1 Overview

Trill is a family of touch sensors designed by the team at Bela. Trill sensors are a convenient way to integrate capacitive touch sensing into interactive projects.

Trill sensors are compatible with any system that supports I^2C communication.

Visit the Trill libraries repositories to download libraries and examples for working with Linux computers and microcontrollers [1].

For complete Trill documentation as well as a Get Started Guide for multiple hardware platforms, please refer to the Trill section of the Bela Knowledge Base: https://bela.io/trill.

1.1 Revision history

This datasheet is version 1.3 of Rev C of Trill. Version changes are as follows:

Version 1.1:

- Added OSHW UIDs in subsection 10.1.
- Added a reference to the Trill Craft symbol and footprint for KiCad to 4.2.

Version 1.2:

- Removed the term 'pseudo multi-touch' in Note 1 of Table 2, and added detail of why 2-axis sensors do not support multi-touch.
- Reworked Table 12 and added Table 13 for clarity.
- Reworked Section 6.2.4 to make content clearer.
- Fixed errors in Table 8 and Figure 4 (offset configuration was incorrect, solder pad pattern was mirrored).
- General edit of entire text to improve clarity, consistency of terms, language use, and formatting.
- Added instructions on how to attach the Trill Flex sensor to the FCC connector in Section 8.6.

Version 1.3:

- Added notes about the changes in the Trill Craft design in sections 4.2, 8.3, and 9
- Included the zero point and direction of increasing position values to the illustrations in section 8.

For previous revs, please refer to this datasheet: https://github.com/BelaPlatform/Trill/blob/main/datasheet/REV_B/trill_datasheet.pdf

---

[1]Trill Linux library https://github.com/BelaPlatform/Trill-Linux, Trill Arduino library https://github.com/BelaPlatform/Trill-Arduino

2 Manufacturer Numbers

Table 1 lists the manufacturer number and the Global Trade Item Number (formerly known as European Article Number) for each sensor.

Table 1: Manufacturer Product Numbers

Name	Manufacturer Number	GTIN
Trill Bar	160701-BAR	5060821690137
Trill Square	160702-SQUARE	5060821690144
Trill Craft	160703-CRAFT	5060821690151
Trill Hex	160704-HEX	5060821690168
Trill Ring	160705-RING	5060821690175
Trill Flex	160706-FLEX	5060821690212
Trill Hub	160707-HUB	5060821690205

3 Trill Sensor Types

There are six Trill sensor types: Bar, Square, Craft, Hex, Ring and Flex. Each Trill type offers a different combination of physical form factor and sensing affordances:

Table 2: Sensor affordances

Trill Type	Sensing Mode	Multi-touch	CapSense channels
Trill Bar	1-axis slider	Yes	26
Trill Square	2-axis pad	No[1]	30 (15 per axis)
Trill Craft[2]	30-channel breakout	Yes	30
Trill Hex	2-axis pad	No[1]	30 (15 per axis)
Trill Ring	1-axis slider	Yes	30 (28 + 2 buttons)
Trill Flex[3]	1-axis slider	Yes	30

[1] 2-axis Trill sensors (Trill Square and Trill Hex) do not support multi-touch, meaning they do not return accurate touch location data when more than one touch is present. These sensors feature a square matrix configuration of pads, with the 30 channels of capacitive sensing distributed between two axes (15 channels per axis). A single touch results in the return of one pair of X/Y coordinates and an accurate touch location. However, when more than one touch is present, Trill will register **all** touches and return them as two lists, one list for touches on the X axis and the other list for touches on the Y axis, but these sets of values cannot be reliably paired to produce any accurate touch locations.

[2] Trill Craft is a 30-channel breakout board for creating custom touch interfaces out of any conductive material.

[3] Trill Flex comes with a single-axis, multi-touch sensor printed on a flexible PCB. This sensor is detachable and can be replaced with a custom flexible PCB sensor. See the tutorial 'Trill: designing a custom flex sensor' for details: https://learn.bela.io/flex-design

4 Pinout

Table 3 lists the pinout and applies to Trill Bar, Square, Hex, Ring and Flex. Each of these sensors includes a cable with QWIIC connector[2] that attaches to the sensor and ends in pins that can be plugged into a breadboard or

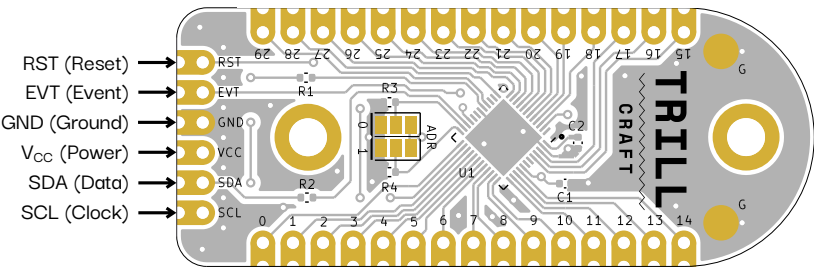[2] 4-pin JST SH SM04B-SRSS-TB (Pitch 1mm) https://www.sparkfun.com/qwiic

Figure 1: Trill Craft Pinout

development board.

Table 3: Cable colour code

Pin	Signal	Colour[1]	Illustration[2]
1	GND	Black	
2	V_{CC}	Red	
3	SDA (I²C Data)	Blue	
4	SCL (I²C Clock)	Yellow	

[1] The colour listed corresponds to the wire on the provided QWIIC cable

[2] From left to right facing the receptacle: Pin 1, 2, 3, 4

4.1 Additional connections

The Event (EVT) and Reset (RST) signals are available as unpopulated, labelled solder pads. When a voltage is applied to the Reset pad the sensor will reset. The Event pad is pulsed by the sensor according to the setting of the EventMode command, as explained in Section 7.5.

The Address (ADR) pads can be used to change the I^2C address of the sensor (see Section 7.1).

Trill Ring has two pads on its reverse side labelled A and B that can be used to connect external capacitive surfaces. These pads behave in the same way as the channels on Trill Craft—solder a wire to the pads and connect them to any conductive material to create a capacitive touch surface. This is to provide a way of adding, for example, a button to the centre of the Ring to create a control interface.

4.2 Trill Craft

Trill Craft has a total of 30 capacitive sensing pads, one for each channel of sensing (15 castellated pads on each side). Trill Craft also has two SMD pads on the front side labelled 'G' which are connected to GND. The power, I^2C, RST and EVT signals are available as through-hole pads along the shorter straight edge (see Figure 1). The Trill Craft footprint for KiCad projects is available in the Trill KiCad library [3]. **Please note that there is a form factor difference for Trill Craft, between Rev A (length = 52.0mm) and the current Rev C (length = 53.8mm). Both footprints are included in the above library.**

4.3 Trill Hub

Trill Hub is a passive breakout board for connecting multiple I^2C devices together. It features 10 QWIIC connectors, 2 Grove connectors and 6 sets of 2.54 mm through-hole pads connected in series, in addition to 2.2 k pull-up resistors on the SDA and SCL data lines. This removes the need for external pull-ups, provides solid connection points, and offers a single point of connection for the host processor.

[3] https://github.com/BelaPlatform/Trill/tree/main/hardware/trill-KiCad-libs/footprints

5 Electrical specifications

The Trill family of touch sensors uses the Infineon Semiconductor PSoC CY8C20XX6A IC.[4]

Table 4: Power and data specifications

	Unit	Value	Condition
Operating voltage (V_{CC})	V	1.71 V to 5.5 V[1]	
Operating current	mA	4	3.3 V, 5 V
I²C bus speed	kHz	50/100/400	

[1] These are the values reported in the CY8C20XX6A/S datasheet. Trill devices have been tested with V_{CC} = 3.3 V and V_{CC} = 5 V.
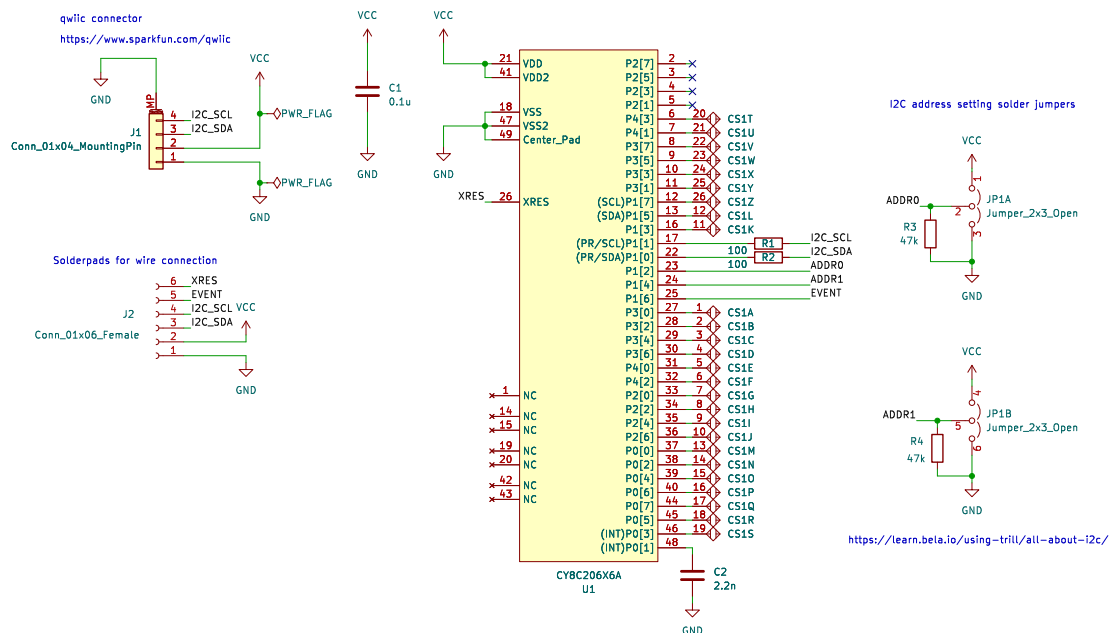
5.1 Schematics

Figure 2: Trill Schematic (Bar), for other sensors see table 6

---

[4]CY8C20XX6A/S datasheet, Infineon Semiconductor company, Document number: 001-54459
https://www.infineon.com/dgdl/Infineon-CY8C20XX6A_S_1.8_V_Programmable_CapSense_Controller_with_SmartSense_Auto-tuning_1-33_Buttons_0-6_Sliders-DataSheet-v26_00-EN.pdf?fileId=8ac78c8c7d0d8da4017d0ecc6dc04671

Table 5: CY8C20XX6A common pinout and usage on all Trill sensors

Pin number	Name	Usage on Trill Sensors
1, 14, 15, 42, 43, 19, 20	NC (Not connected)	
17	P1[1]	I²C SCL
18, 47, Center pad	V_{SS}	GND
21, 41	V_{DD}	V_{CC}
22	P1[0]	I²C SDA
23	P1[2]	ADDR0
24	P1[4]	ADDR1
26	XRES	RESET
48	P0[1]	CapSense internal[1]

[1] CapSense Integrating Capacitor

Table 6: CY8C20XX6A pinout and usage on Trill sensors

Pin #	Name	Bar	Square[1]	Craft[2]	Ring	Hex[1]	Flex
2	P2[7]	NC	R8	EVENT	1	R8	18
3	P2[5]	NC	C12	18	2	C11	19
4	P2[3]	NC	C13	19	3	C12	20
5	P2[1]	NC	C14	20	4	C13	21
6	P4[3]	20	C1	21	5	C14	22
7	P4[1]	21	R7	22	6	R7	23
8	P3[7]	22	R8	23	7	R6	24
9	P3[5]	23	R5	24	8	R5	25
10	P3[3]	24	R4	25	9	R4	26
11	P3[1]	25	R3	26	10	R3	27
12	P1[7]	26	R2	27	11	R2	28
13	P1[5]	12	C8	28	Button B	C8	29
16	P1[3]	11	R1	29	Button A	R1	30
25	P1[6]	EVENT	EVENT	0	EVENT	EVENT	EVENT
27	P3[0]	1	C1	1	12	C1	1
28	P3[2]	2	C2	2	13	C2	2
29	P3[4]	3	C3	3	14	C3	3
30	P3[6]	4	C4	4	15	C4	4
31	P4[0]	5	C5	5	16	C5	5
32	P4[2]	6	C6	6	17	C6	6
33	P2[0]	7	C7	7	18	C7	7
34	P2[2]	8	R9	8	19	R10	8
35	P2[4]	9	R10	9	20	R11	9
36	P2[6]	10	R11	10	21	R12	10
37	P0[0]	13	R12	11	22	R13	11
38	P0[2]	14	R13	12	23	R14	12
39	P0[4]	15	R14	13	24	R15	13
40	P0[6]	16	R15	14	25	R16	14
44	P0[7]	17	C9	15	26	C9	15
45	P0[5]	18	C10	16	27	C10	16
46	P0[3]	19	C11	17	28	R9	17

[1] 2-axis sensors (Trill Square and Trill Hex) have rows and columns.

[2] Trill Craft pins are labelled on the silkscreen and start at index 0.

6 Functional overview

The PSoC runs custom firmware[5]. This firmware uses the EZ I^{2}C Slave library[6] for I^{2}C communication and the CapSense Sigma-Delta library[7] for capacitive sensing.

The PSoC firmware's main procedure processes incoming messages received via I^{2}C from the host processor, which then scans the enabled capacitive channels, and then processes and formats the scan results. A host processor (meaning any device that can control the I^{2}C bus, such as a microcontroller, single board computer, or any other compatible system) sends commands, retrieves responses and accesses the scan data via the I^{2}C bus, accessing 64 bytes of the PSoC's RAM. The layout and contents of this shared RAM buffer are described in Section 7.3.

There are up to 30 capacitive channels on each sensor, as listed in Table 2. The firmware uses the CSD library to scan the capacitive sensing channels. For details on any function or variable name prefixed with CSD_ mentioned in the remainder of this document, please refer to the CSD library documentation.

6.1 Operating modes

A Trill sensor can operate in any of the following modes:

- **Centroid mode** processes the readings of the individual capacitive sensing channels, taking into account the geometry and layout of the sensing pads on the circuit board in order to compute the position and size of discrete touch points via a Trill-specific moving average algorithm. The reported size of a touch is a measure of the total activation measured on the sensing channels that contribute to the touch. This mode uses the differential capacitive readings stored in CSD_waSnsDiff, but it does not use any of the CSD library's centroid or slider features.
  Trill Bar, Square, Ring, Hex and Flex use Centroid mode by default.
- **Raw mode** transmits the raw capacitance values from CSD_waSnsResult
  Raw mode is normally used for testing and debugging.
- **Baseline mode** transmits the baseline capacitance values from CSD_waSnsBaseline. This baseline value is used as a reference value when the sensor is in Differential or Centroid mode.
  Baseline mode is normally used for testing and debugging.
- **Differential mode** transmits the difference between the raw and baseline capacitive readings after applying a noise threshold, as given by CSD_waSnsDiff.
  Trill Craft uses Differential mode by default, to sense activity on individual capacitive pads.

If Trill Craft or Trill Flex is set in Centroid mode, it expects the capacitive channels to be connected in sequential order to sensing pads placed in a linear fashion, similar to the sensing pads on Trill Bar or on the flexible PCB that comes with Trill Flex. If these sensors are using a different geometry for the sensing surface, Differential mode should be used and the host processor should read the scan data for each channel and perform any further processing itself.

6.2 Timing characteristics

When Trill receives a scan trigger, it performs a scan of the sensor channels. This can take several milliseconds. When the scan is complete, the results are atomically placed into the shared memory buffer. The EVT pin may also be set high depending on the current settings.

---

[5] https://github.com/BelaPlatform/Trill
[6] Cypress Semiconductor Corporation, EZ I^{2}C Slave v1.20 https://www.infineon.com/dgdl/Infineon-Component_EZI2C_Slave_V1.20-Software+Module+Datasheets-v02_00-EN.pdf?fileId=8ac78c8c7d0d8da4017d0e7f54ef111e
[7] Cypress Semiconductor Corporation, CapSense® Sigma-Delta Datasheet CSD v2.20 https://www.infineon.com/dgdl/Infineon-CapSense_Sigma-Delta_Datasheet_CSD-Software%20Module%20Datasheets-v02_02-EN.pdf?fileId=8ac78c8c7d0d8da4017d0f9f5d9b0e82

6.2.1 Scan trigger

A full scan of all enabled sensor channels can be triggered on an I^{2}C read transaction, or when a customisable timer expires. The `ScanTrigger` command is used to set whether either, neither or both of these conditions are enabled, while the `AutoScanTimer` command is used to set the period of the timer.

If the scan is set to trigger on I^{2}C, a new scan starts as soon as a new I^{2}C read transaction begins. If the scan is set to trigger on a timer, the timer restarts at the beginning of each scan and a new scan starts when the timer expires. If the timer expires while a scan is in progress, a new scan will start as soon as the processing of the current scan is complete.

6.2.2 Acquisition time

Table 7: Scan times for each capacitive channel for different scan settings (times in μs)

Bits[2]	Scan speed setting[1]			
	CSD_ULTRA_FAST_SPEED	CSD_FAST_SPEED	CSD_NORMAL_SPEED	CSD_SLOW_SPEED
9	57	78	125	205
10	78	125	205	380
11	125	205	380	720
12	205	380	720	1400
13	380	720	1400	2800
14	720	1400	2800	5600
15	1400	2800	5600	11000
16	2800	5600	11000	22000

[1] These are the values available to the `ScanSettings` command and the `CSD_SetScanMode()` function
[2] Resolution

The time required to scan all enabled channels on the sensor depends on the number of enabled channels, the bit resolution, and the scanning speed. Table 7 details the scan duration for all combinations of speed and resolution settings for each of the capacitive sensing channels on your sensor. To calculate the overall scan time, find the scan time that applies to your Trill settings in the table, and then multiply it by the number of sensing channels you have enabled.

Each sensing channel can be enabled or disabled via dedicated commands. We recommend disabling scanning for unused channels, in order to reduce overall scanning and required data transfer time. However, if your sensor is in Centroid mode we do not recommend disabling any channels, as this will disrupt the touch detection routine.

The minimum achievable sampling period for a Trill sensor is largely limited by the scan time, combined with the additional time required by the PSoC firmware to handle the logic and formatting of data. The actual additional time depends on the number of enabled channels and data format settings, but it is normally below 1 ms.

6.2.3 The Event (EVT) pin

The `EventMode` command makes it possible to specify the conditions under which the EVT pin goes high when a scan has completed. When set to `EventModeTouch`, the EVT pin goes high if activity is detected in the current scan.[8] When set to `EventModeChange`, the EVT pin goes high if activity is detected in the current or past scan. When set to `EventModeAlways`, the EVT pin goes high after every scan, regardless of whether activity is detected or not.

Once the data from a new scan has been placed into the memory buffer, the EVT pin will go high if its specified conditions are met. The host processor can monitor the EVT pin and start a new I^{2}C read transaction to access

---

[8]When a sensor is in Centroid mode, activity is present any time a touch is detected. In all other modes, activity is present if any of the channel data is non-zero. If activity was detected in a given set of scan data, the corresponding bit in the Status byte will also be set (see Section 7.6).

the scan data upon detecting a positive edge on the pin. The EVT pin stays high until the next scan is complete. As soon as the scanning routine is complete and just before performing any post-processing or formatting of the data, the EVT pin goes low. If a new scan is triggered to start immediately after the previous one, the EVT pin go low for less than 1 ms before returning to high.

6.2.4 Use cases

In this section we highlight use cases that illustrate how to leverage the Trill sensor's flexibility in applications, such as using multiple sensors, or when prioritising latency, sampling rate, I^2C bus use, host processor pin use, or power usage.

Use Case 1: Using the minimum number of connections for reading one or more Trill sensors

Trill's address pins allow the use of multiple sensors on the same I^2C bus using only the SDA and SCL lines. When using multiple sensors, each sensor's ScanTrigger command should be set to scan on I^2C transaction, and then the host processor should periodically read all connected sensors. When the host processor begins a read over the I^2C bus, this simultaneously triggers a new scan on the sensor. This means that each read transaction returns the payload from the scan that was initiated on the previous read transaction.

Typically, reading the data from the sensor takes less time than the sensor requires to scan its enabled channels. This means if the host processor reads too quickly it may read the same data multiple times, because a new payload may not yet be available since the previous read. There are two strategies to avoid this: First, you can set the host processor's read timer to a rate that's slower than the time that the sensor requires to scan and second, you can check that the received payload is new by monitoring the Status byte (see Section 7.6 for details).

Use Case 2: Achieving the lowest possible latency and highest possible sampling rate from a single sensor

This use case assumes that the host processor is capable of monitoring the EVT pin and immediately beginning an I^2C read transaction as soon as it detects a rising edge. The sensor's ScanTrigger should be set to scan on I^2C transaction, and the EVT pin should be set to EventModeAlways so it goes high after every scan.

The host processor starts by reading the data. This triggers the sensor to scan, and as a result brings its EVT pin high. The host processor, which is monitoring the EVT pin, starts a new read when it detects this rising edge. This transaction will in turn trigger the sensor to scan, bringing its EVT pin low before returning it to high. This rising edge will make the host processor start a new read, and so on. In this way the sampling rate is determined by the sensor's scan time, plus the host processor's response time.

Because of these tight timing constrains this use case is only suitable for a single sensor.

Use Case 3: Achieving high sampling rate with variable latency for multiple sensors

Set each sensors' EventMode to EventModeChange, which brings the EVT pin high if new activity is detected in a scan. Set each sensor's ScanTrigger to timer. If maximum sampling rate is required, set AutoScanTimer to a minimum value. Otherwise, set it to a larger value suitable for your application.

If the host processor can monitor the EVT pin for each sensor, it should read data from a sensor after detecting a rising edge on that sensor's EVT pin. In this case, the host processor can read the data at any point before the next rising edge of the sensor's EVT pin without affecting the sampling rate of the data.

If the host processor cannot monitor the EVT pin for each sensor, it should periodically read data from each sensor to verify whether new scan data is available. This is best achieved by reading the Status byte to determine whether this is a new scan, and if new activity was detected. If both of these are true, the host processor can proceed with a new read of the full Payload. These reads have to take place more often than the acquisition time of the sensor in order to avoid missing frames, but they should not occur too often as they would slow down the sensor during acquisition.

This technique of polling the Status byte or monitoring the EVT pin can extend to applications where many sensors are in use. Reading the full scan data from all sensors at the desired sampling rate at all times is likely not possible because it's limited by the bandwidth available on the I^2C bus, rather than by the acquisition time. However, as long as all sensors are not showing activity at the same time, the host processor should be able to scan only the active sensors while maintaining a high sampling rate.

Note that each sensor will have a slightly different scan time, so the host processor should be prepared to handle timing drift.

Use Case 4: Limiting power consumption

When the host processor expects long waits between bursts of activity on one or more sensors, or wants to limit its own (or the sensors') power consumption and can tolerate a slight delay when activity starts being detected, it's possible to use the EVT pin as a 'wake-up' signal for the host processor.

In this case, set the sensors' EventMode to EventModeChange, ScanTrigger to timer, and a timer interval compatible with latency requirements before entering low-power mode. Once activity is detected by the sensor, it will set its EVT pin high, a signal that can be used to wake up the host processor. Then, the host processor can scan and process incoming data on each rising edge of the EVT pin.

Because EventModeChange means the EVT pin goes high when activity is detected and stays high until a next scan is completed, a persistently high EVT pin indicates no more activity. At this point the host processor can go back into low-power mode, waiting for the next burst of activity.

7   Communication

A host processor can communicate with the Trill sensor using the I^{2}C protocol, over the SDA and SCL lines. These lines are connected directly to the SDA and SCL pins of the Infineon CY8C20XX6A PSoC (Programmable System on Chip) IC located on the Trill sensor, so for the rest of this section we will refer to the PSoC directly. More details about the I^{2}C protocol, including timing information, can be found in the CYBC20XX6A datasheet (see Footnote 4).

There are no pull-up resistors for the SDA or SCL lines on the sensors themselves. When using Trill sensors without Trill Hub (which provides 2.2 k pull-up resistors), it is the user's responsibility to provide pull-up resistors on these lines if required. More information on pull-up resistors can be found in the Texas Instruments Application Report SLVA689[9].

7.1   I^{2}C addresses

Each Trill sensor type has a different I^{2}C address. In order to use more than 1 of any single sensor type on a single I^{2}C bus, you'll have to change the address of each additional sensor so they can be unequivocally addressed on the bus.

Setting the I^{2}C address

On the back of each Trill sensor are six pads labelled ADR, in 2 groups of 3. These pads can be bridged with solder in various configurations to offset the sensor's address.

The 3 pads in each group are unconnected by default. The middle pad of the group (either ADR0 and ADR1) can be connected to one of the pads adjacent to it (either GND or V$_{CC}$) with a solder bridge. The GND pads are indicated by the thicker and separated line on that side (see Figure 3). By bridging ADR0 and ADR1 to GND or V$_{CC}$ or leaving them unconnected, you can offset the sensor address by up to 8. This means up to 9 Trill sensors of a single type can be connected on one I^{2}C bus (1 sensor with the default address, and 8 more with each of the offset addresses). See Table 8 for details on how to configure these connections.

Do not connect all three pads together.

---

[9]Texas Instruments Application Report SLVA689 I2C Bus Pullup Resistor Calculation https://www.ti.com/lit/an/slva689/slva689.pdf
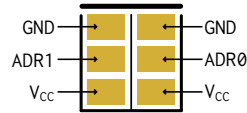
Figure 3: Solder pads for offsetting the sensor address (2 groups of 3 pads). Note the thicker and
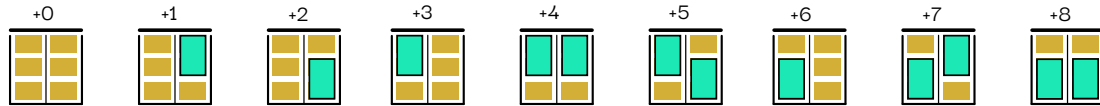separated line indicating the side with the GND pads.

Figure 4: Setting the solder pad addresses

Table 8: Address pad connections and resulting addresses
NC: Not Connected

ADR1	ADR0	Offset	Trill Bar	Trill Square	Trill Craft	Trill Ring	Trill Hex	Trill Flex
NC[1]		+0	0x20	0x28	0x30	0x38	0x40	0x48
NC	GND	+1	0x21	0x29	0x31	0x39	0x41	0x49
NC	V_{CC}	+2	0x22	0x2A	0x32	0x3A	0x42	0X4A
GND	NC	+3	0x23	0x2B	0x33	0x3B	0x43	0X4B
GND	GND	+4	0x24	0x2C	0x34	0x3C	0x44	0X4C
GND	V_{CC}	+5	0x25	0x2D	0x35	0x3D	0x45	0X4D
V_{CC}	NC	+6	0x26	0x2E	0x36	0x3E	0x46	0X4E
V_{CC}	GND	+7	0x27	0x2F	0x37	0x3F	0x47	0X4F
V_{CC}	V_{CC}	+8	0x28	0x30	0x38	0x40	0x48	0X50

[1] No solder bridge (+0 offset) is the default for all Trill sensors

7.2 Writing and reading data

When writing to the PSoC, the first byte in a transaction written by the host processor determines the memory offset at which the bytes in that transaction are written. For example, to write bytes [0xAA 0xBB] at offset 0x00, the host processor should write 3 bytes in a single transaction: [0x00 0xAA 0xBB] (the memory offset, followed by the two data bytes).

Reads of any length start at the memory offset set during the last write transaction. This offset can be changed by performing a dedicated 1-byte write transaction. For example, if the last write was at offset 0x00 and you want to read bytes starting from offset 0x03, the host processor must first perform a 1-byte write transaction in which it writes 0x03 in order to set the offset. Future read transactions will return values starting from memory offset 0x03.

If the host processor tries to write to an offset outside the R/W section of the memory, or if it attempts to read from an offset outside the available memory, the PSoC will trigger a Not Acknowledge (NACK) condition on the I^2C bus and ignore the rest of the transaction until the host processor issues a new start condition.

7.3 Memory map

The shared RAM buffer exposed by the PSoC to the host processor over I^2C is organised as follows:

Table 9: Map of the PSoC's memory that is accessible by the host processor over the I²C bus

Offset	0	1	2	3	4	5	...	63
Access Mode	R/W	R/W	R/W	R/O	R/O	R/O	...	R/O
Host Writes	Cmd code	Cmd arg0	Cmd arg1	x	x	x	x	x
Host Reads	Ack code	Ack arg0	Ack arg1	Status	Payload 0	Payload 1	...	Payload 59

Bytes 0 to 2: The command and acknowledgement region; supports both reads and writes (R/W) from the host processor.

Byte 3: The status byte; is read-only (R/O) for the host processor.

Bytes 4 to 63: The payload region of the memory; these are R/O for the host processor.

7.4 Commands and acknowledgements

To send commands to the PSoC, the host processor writes between one and three bytes to offset 0 in a single I²C transaction. The first byte is the command's code, to which 0, 1 or 2 command-specific bytes follow depending on the command (as detailed in Section 7.5).

When the PSoC has processed the command, it writes an acknowledgement message to offset 0 so that the host processor can read it back. The time it takes for the PSoC to process the command and prepare the acknowledgement message depends on the command and scanning settings.

In order to ensure a command has been received and processed after it's been written, the host processor should repeatedly poll the PSoC memory by reading 3 bytes from offset 0 and waiting for a valid acknowledgement message to appear. This signals that the PSoC is ready to receive the next command.

The acknowledgement message is contains three bytes: [ACK BYTE0 BYTE1]. The ACK byte is 254 (0xFE). When a command requires a response, BYTE0 and BYTE1 are the payload of the response. When a command doesn't require a response, BYTE0 is the command code that was written by the host processor to offset 0 in the command to which this acknowledgement refers. BYTE1 is an 8-bit counter that is incremented by one for each command that the PSoC receives (regardless of whether a response is required or not), and is reset to 0 when the PSoC is reset or when the counter wraps around. By monitoring the value of the counter byte, the host processor can detect unexpected resets of the PSoC.

The acknowledgement message only notifies the host processor that the command was received and processed. It does not provide any information regarding the command's validity or that of its arguments - it is the host processor's responsibility to validate that.

7.5 Command list

The commands supported by the revision C of the Trill sensor family are listed in Table 10 and explained below. The processing of some of these commands results in a function call to the CSD API of the CapSense Sigma-Delta library which is used on the Trill sensor to scan the capacitive sensing channels on board the PSoC.

Mode command: 1 (0x01)

Sets the mode in which the sensor processes and transmits the data read from its capacitive sensing channels. A detailed description of the available modes is available in Section 6.1.

The argument byte can assume one of the following values:

- 0 (0x00): Centroid. Default for all sensors except Trill Craft.
- 1 (0x01): Raw

Table 10: Command list

Name	Command code	Number of argument bytes	Requires response
Mode	1 (0x01)	1	No
ScanSettings	2 (0x02)	2	No
Prescaler	3 (0x03)	1	No
NoiseThreshold	4 (0x04)	1	No
Idac	5 (0x05)	1	No
UpdateBaseline	6 (0x06)	0	No
MinimumSize	7 (0x07)	2	No
AdjacentCentroidNoiseThreshold	8 (0x08)	2	No
EventMode	9 (0x09)	1	No
ChannelMaskLow	10 (0x0A)	2	No
ChannelMaskHigh	11 (0x0B)	2	No
Reset	12 (0x0C)	0	No
Format	13 (0x0D)	2	No
AutoScanTimer	14 (0x0E)	2	No
ScanTrigger	15 (0x0F)	1	No
Identify	255 (0xFF)	0	Yes

- 2 (0x02): Baseline
- 3 (0x03): Differential. Default for Trill Craft.

ScanSettings command: 2 (0x02)

Sets scanning speed and resolution.

The first argument byte is speed and can assume one of the following values:

- 0 (0x00): CSD_ULTRA_FAST_SPEED
- 1 (0x01): CSD_FAST_SPEED
- 2 (0x02): CSD_NORMAL_SPEED
- 3 (0x03): CSD_SLOW_SPEED

The second argument byte is the resolution in bits and can be any integer value between 9 and 16.

The arguments are passed to CSD_SetScanMode(). Defaults are 0 (CSD_ULTRA_FAST_SPEED) for speed and 12 for resolution.

Prescaler command: 3 (0x03)

Sets the prescaler value.

The argument byte is passed to CSD_SetPrescaler() and can assume one of the following values:

- 0 (0x00): CSD_PRESCALER_1 sets prescaler to 1
- 1 (0x01): CSD_PRESCALER_2 sets prescaler to 2
- 2 (0x02): CSD_PRESCALER_4 sets prescaler to 4
- 3 (0x03): CSD_PRESCALER_8 sets prescaler to 8
- 4 (0x04): CSD_PRESCALER_16 sets prescaler to 16
- 5 (0x05): CSD_PRESCALER_32 sets prescaler to 32
- 6 (0x06): CSD_PRESCALER_64 sets prescaler to 64
- 7 (0x07): CSD_PRESCALER_128 sets prescaler to 128

- 8 (0x08): CSD_PRESCALER_256 sets prescaler to 256

Default value is 4 on Trill Bar, and 2 on all other sensors.

NoiseThreshold command: 4 (0x05)

Sets the noise threshold for the sensing channels.

The argument byte is used to set `CSD_bNoiseThreshold`. Default value is 40.

Idac command: 5 (0x05)

Sets the iDAC value.

The argument byte is passed to `CSD_SetIdacValue()`. Default value is 20.

UpdateBaseline command: 6 (0x06)

Updates the baseline values. This results in a complete sensor scan followed by a call to `CSD_InitializeBaselines()`.

MinimumSize command: 7 (0x07)

When the sensor is in Centroid mode, this command sets the minimum size that a centroid has to have in order to be considered a valid touch.

The two argument bytes are interpreted as a 16-bit Big-Endian word representing the centroid size. Default value is 0.

AdjacentCentroidNoiseThreshold command: 8 (0x08)

When the sensor is in Centroid mode, this command sets the noise threshold to detect a trough between two adjacent touches.

The two argument bytes are interpreted as a 16-bit Big-Endian word representing the noise threshold. Default value is 400.

EventMode command: 9 (0x09)

Sets the conditions under which the EVT pin should be pulsed high after the data from a new scan is ready.

The argument byte can assume one of the following values:

- 0 (0x00): EventModeTouch, the EVT pin goes high every time activity is detected in the data of the current scan (default)
- 1 (0x01): EventModeChange, the EVT pin goes high every time activity is detected in the data of the current or previous scan
- 2 (0x02): EventModeAlways, the EVT pin goes high after every scan, regardless of whether activity is detected or not

ChannelMaskLow command: 10 (0x0A)

Each capacitive channel on the sensor can be individually enabled or disabled by setting the respective bits in a 32-bit channel mask. If a channel's bit is 1, it will be scanned and its data made available for the host processor to read. If a channel's bit is 0, it will not be scanned and its data will not be made available for the host processor to read. There are always fewer than 32 channels on each sensor, as shown in Table 2. Any bit indexes higher than the number of available channels will be ignored.

By default the channel mask is set to 0xFFFFFFFF (all channels enabled). When the sensor is in Centroid mode, the channel mask should be set to its default value. See section 7.7 for more details on how disabling a channel affects the data stored in the Payload region.

The `ChannelMaskLow` command sets the channel mask for the lowest 16 channels. See Table 11 for more details.

The first argument byte is the mask for sensing channels 8 through 15.

The second argument byte is the mask for sensing channels 0 through 7.

Table 11: Map of the channel mask

Command Code	ChannelMaskHigh 11 (0x0B)		ChannelMaskLow 10 (0x0A)	
Argument Byte	0	1	0	1
Mask Bits	31:24	23:16	15:8	7:0
Default Value	0xFF	0xFF	0xFF	0xFF

ChannelMaskHigh command: 11 (0x0B)

Sets the sensing channel mask for the highest 16 channels. See Table 11 for more details.

The first argument byte is the mask for sensing channels 24 through 31.

The second argument byte is the mask for sensing channels 16 through 23.

Reset command: 12 (0x0C)

Performs a reset of the PSoC by restoring the CPU to the power-on reset state.

Format command: 13 (0x0D)

Sets the format in which the data for individual sensing channels is made available to the host processor. This does not affect the format of data in Centroid mode.

The first argument byte sets the data width (in bits) and can assume one of the following values: 8, 12, 16 (defaults to 16).

The second argument sets the number of positions by which the value should be right shifted before storing it in the allocated data width (defaults to 0). See Section 7.7 for more details.

AutoScanTimer command: 14 (0x0E)

Sets the period of an internal timer that can be used to automatically scan and process the capacitive sensing channels. The effective minimum scanning period will be limited by the scanning speed, bit depth and any computation happening on the sensor. The interval set with this command is only used if the `ScanTrigger` command is set to 2 or 3.

The values of the two argument bytes, when multiplied together, give the number of periods of an internal 32 kHz clock after which a new scan is triggered. If either is set to 0, the timer is disabled. The nominal scanning period, expressed in seconds, is therefore given by `byte0 * byte1 / 32000`.

Note that the 32 kHz clock (the ILO clock of the PSoC) can deviate more than 10% from its nominal frequency, correspondingly affecting the accuracy of the period set here.

ScanTrigger command: 15 (0x0F)

Sets how the sensor triggers a new scan of its enabled channels.

The argument byte can assume one of the following values:

- 0 (0x00): Scanning is disabled.
- 1 (0x01): Start a new scan after every I²C transaction (default).
- 2 (0x02): Start a new scan when the time interval (specified with the `AutoScanTimer` command) has elapsed since the last scan.
- 3 (0x03): Start a new scan after every I²C transaction, or when the time interval specified with the `AutoScanTimer` command has elapsed since the last scan.

Identify command: 255 (0xFF)

Returns the sensor's type and firmware version.

This command requires a response. Therefore once the command has been processed, the command buffer will contain the bytes [ACK SENSOR_TYPE FW_VERSION], where ACK is 254 (0xFE), FW_VERSION is 3 (0x03) and SENSOR_TYPE is one of the following:

- 1 (0x01): Bar
- 2 (0x02): Square
- 3 (0x03): Craft
- 4 (0x04): Ring
- 5 (0x05): Hex
- 6 (0x06): Flex

7.6 Status byte

The byte at offset 3 is the Status byte, which contains details about the scan data currently present in the Payload region and the state of the sensor.

Bits 5:0 contain the frameId, a counter that is incremented every time the results of a new scan are made available on the I²C bus. By reading the frameId, the host processor can detect duplicated or dropped scans.

Bit 6 is set to 1 if any activity was detected in the current scan, or 0 otherwise. See Section 6.2.3 for more information about detected activity.

Bit 7 is set to 0 upon reset and set to 1 after an Identify message is received. The host processor can monitor this bit to detect an unexpected reset of the PSoC.

7.7 Payload

The memory region starting at offset 4 contains the payload data obtained from scanning and processing the capacitive channels on the PSoC. The format and length of this section depend on the Trill sensor type, the sensor's mode (as set by the Mode command), the number of active sensing channels (as set by the SensorMaskLow and SensorMaskHigh commands), and the transmission width (as set by the Format command).

7.7.1 Centroid mode

The data in the Payload region contains the values for location and size for the touches detected by the sensor. The location and size of a touch are each represented by a 2-byte word. Each word represents a 16-bit unsigned integer in Big Endian format. Touch location and size values are stored in a buffer of contiguous memory, starting from the beginning of the Payload region. For each sensing axis, location data for all touches is presented first, followed by size data. 2-axis sensors (Trill Square, Trill Hex) store vertical (Y-axis) touches first, followed by horizontal (X-axis) touches. The readings of the two exposed pads on Trill Ring are stored after the touch values as a 16-bit unsigned integers in Big Endian format.

The memory offset for the start of the location data for a given touch n on a given sensing axis is $p + 2n$. The memory offset for the start of the size data for a given touch n is $p + 2N + 2n$, where N is the total number of touches on the sensing axis. The offset p is 4 (the beginning of the Payload region) for 1-axis sensors and the Y-axis of 2-axis sensors, and is $4 + 2N$ for the X-axis of 2-axis sensors. Tables 12 and 13 detail the memory map of the Payload region for 1-axis and 2-axis sensors, respectively.

The size of valid data (in bytes) in the Payload memory region for a sensor that provides a total of N touches across its axes and E extra pads is $M = 4N + 2E$. The host processor should read M bytes starting at offset 4 to retrieve the full touch data. The number of available touches and the corresponding data size for each sensor type in Centroid mode is shown in Table 14 .

Table 12: Map of the payload memory when a 1-axis sensor is in Centroid mode

Offset	4	5	...	12	13	14	15	...	22	23
Touch	0		...	4		0		...	4	
Value	Location		...	Location		Size		...	Size	
Bits	15:8	7:0	...	15:8	7:0	15:8	7:0	...	15:8	7:0

Table 13: Map of the payload memory when a 2-axis sensor is in Centroid mode

Offset	4	5	...	10	11	12	13	...	18	19
Touch	0		...	3		0		...	3	
Value	Y-axis location		...	Y-axis location		Y-axis size		...	Y-axis size	
Bits	15:8	7:0	...	15:8	7:0	15:8	7:0	...	15:8	7:0

Offset	20	21	...	26	27	28	29	...	34	35
Touch	0		...	3		0		...	3	
Value	X-axis location		...	X-axis location		X-axis size		...	X-axis size	
Bits	15:8	7:0	...	15:8	7:0	15:8	7:0	...	15:8	7:0

The touch location values range from 0 to $128 * (n - 1)$, where n is the number of capacitive pads on the sensing axis (See Table 2). The touch size value is the sum of the differential readings for all pads assigned to the touch. The special value 65535 (`0xFFFF`) is used to denote the location and size values of inactive touches.

7.7.2 Raw, Baseline or Diff mode

The data in the Payload region represents the amount of capacitive activation on the corresponding sensing channel. Values of all enabled channels are stored contiguously in the memory buffer. This means there are no gaps in the buffer if a channel is disabled. If a channel is disabled in the channel mask, its value is removed from the buffer, and the values of all subsequent channels are moved up so the values remain contiguous.

Each channel reading is stored using a data width (8, 12, or 16 bits), as set by the first argument of the `Format` command (default is 16). The value is right shifted by the number of bits specified via the second argument of the `Format` command (default is 0) before it is stored in the available bits. The value for a given channel prior to the right shift ranges between 0 and $2^n - 1$, where n is the bit resolution set via the `ScanSettings` command (defaults to 12). If the result of the right shift would overflow the available bits, it is saturated to the maximum value that can be stored.

For a data width of 8 bits each channel's data is stored in one byte. For a data width of 12 bits the data from a pair of sensing channels occupies a total of 3 bytes:

- Bits 7:0 of the first byte represent bits 11:4 of the 12-bit value of the first channel in the pair.
- Bits 3:0 of the second byte represent bits 3:0 of the 12-bit value of the first channel in the pair.
- Bits 7:4 of the second byte represent bits 11:8 of the 12-bit value of the second channel in the pair.
- Bits 7:0 of the third byte represent bits 7:0 of the 12-bit value of the second channel in the pair.

For a data width of 16 bits each sensing channel's data occupies 2 bytes and is stored as an unsigned integer in Big Endian format.

Table 14: Content of the Payload buffer in Centroid mode for each Trill sensor

Trill type	Y-axis touches	X-axis touches	Extra pads	Data length (M)
Bar	5	0	0	20
Square	4	4	0	32
Craft	5	0	0	20
Ring	5	0	2	24
Hex	4	4	0	32
Flex	5	0	0	20

Table 15: Map of the payload memory when the sensor is in Raw, Baseline or Diff mode and the data width is 8 bits

Memory Offset	4	5	6	7	...
Memory Bits	7:0	7:0	7:0	7:0	...
Sensing Channel	0	1	2	3	...
Bits in word	7:0	7:0	7:0	7:0	...

Table 16: Map of the payload memory when the sensor is in Raw, Baseline or Diff mode and the data width is 12 bits

Source Offset	4	5		6	...
Source Bits	7:0	7:4	3:0	7:0	...
Sensing Channel	0	1	0	1	...
Bits in word	11:4	11:8	3:0	7:0	...

Table 17: Map of the payload memory when the sensor is in Raw, Baseline or Diff mode and the data width is 16 bits

Memory Offset	4	5	6	7	...
Memory Bits	7:0	7:0	7:0	7:0	...
Sensing Channel	0		1		...
Bits in word	15:8	7:0	15:8	7:0	...

7.8 Typical operation

Examples of host processor libraries for use with Trill sensors are available online (see Footnote 1). These take care of the low-level communication detail and expose a user-friendly C++ API which allows the sending of commands from 7.5, retrieves scan and touch data, and parses them into C++ data types.

This section is meant for those intending to communicate with Trill sensors without using the provided libraries. It includes examples of barebones I^2C communication written in pseudo-code to be used as a reference for implementing communication. Users of the provided libraries should not need to worry about the details of communication or the examples in this section, and can instead refer to the documentation and examples provided with the library.

The following is a summary of the salient characteristics of I^2C communication between a host processor and a Trill sensor:

- In each write transaction, the first byte is the offset at which the following bytes are written, and at which successive reads start.
- Commands are written to offset 0 (0x00) and have length 1, 2 or 3.
- Acknowledgements are read from offset 0 (0x00) and have length 3.
- Status byte is read from offset 3 (0x03) and has length 1.
- Payload data starts from offset 4 (0x04) and has variable length.

A typical interaction of a host processor with a Trill sensor involves writing commands to the command region to set up the sensor's scanning properties, and reading back acknowledgements to verify that commands have been processed and to ascertain the sensor type and its firmware revision number.

After this initial setup, the sensor must be prepared for reading scan data by writing a single byte with the value of 3 or 4 (corresponding to the offsets of the Status byte and Payload region respectively), depending on whether the host processor intends to read the status byte while reading data. From that point on, successive reads of the appropriate length will return the content of the data region.

7.8.1 Code listing 1

Communication between a host processor and a Trill Bar at default address 0x20. Default values are used, therefore the sensor is already in Centroid mode. The host processor reads data for up to 5 touches and ignores the Status byte.

```
// pseudo-code
// write() and read() calls should perform a write or read transaction
// on the I2C bus at address 0x20.

// Write the identify command
write([0, 0xFF])

uint8 arr[3]
// Poll and wait to read the identify acknowledgement
do { read(arr, 1) } while (arr[0] != 0xFE)
sensor_type = arr[1] // should verify that it is 0x01: Trill Bar
fw_revision = arr[2] // should verify that it is 3

// To start reading data, write the offset from which we want to read from.
// This is 4 as we start from the Payload memory region
write([4])

// Repeatedly read the touch data for 5 touches (4 byte each, 20 bytes total)
while(1) {
 uint8 data[20]
 read(data, 20)
 // here, parse and handle touches
 // wait before the next read to allow
 // the sensor to perform a new scan
 sleepMs(10)
}
```

7.8.2 Code listing 2

Communication between a host processor and a Trill Craft at default address 0x30. The sensor is in Differential mode, the prescaler value is adjusted and the bit resolution is increased to 16. Only capacitive channels 7 through 20 are enabled. All other channels are disabled in the channel mask. The sensor is then set to scan automatically every 50 ms. The host processor polls the Status byte every 30 ms to verify whether the current scan is a new, and whether it has detected activity. If so, the host processor performs a full read of the Status byte and the Payload. The Payload contains the data for the 14 enabled channels, and each channel's data occupies 2 bytes. Therefore the total size of the read, including the Status byte, is 29.

```
// pseudo-code
// write() and read() calls should perform a write or read transaction
// on the I2C bus at address 0x30.

// Write the identify command
write([0, 0xFF])

uint8 arr[3]
// Poll and wait to read the identify acknowledgement
do { read(arr, 3) } while (arr[0] != 0xFE)
sensor_type = arr[1] // should verify that it is 0x03: Trill Craft
 fw_revision = arr[2] // should verify that it is 3
```

```
// disable scanning altogether with the ScanTrigger command:
write([0, 0x0F, 0x00])
// wait for acknowledgement
do { read(arr, 1) } while (arr[0] != 0xFE)


// ScanSettings command to set 16 bit and CSD_ULTRA_FAST_SPEED (0x00)
write([0, 0x02, 0x00, 16])
// wait for acknowledgement
do { read(arr, 1) } while (arr[0] != 0xFE)


// Set the prescaler to CSD_PRESCALER_1 (0x01) with the Prescaler command:
write([0, 0x03, 0x01])
// wait for acknowledgement
do { read(arr, 1) } while (arr[0] != 0xFE)


// Manipulate the channel mask so that only channels 7 through 20 are enabled
// The desired content of the channel mask is thus 0x001FFF80
// Using the ChannelMaskLow command we set the lowest bytes
write([0, 0x0A, 0xFF, 0x80])
// wait for acknowledgement
do { read(arr, 1) } while (arr[0] != 0xFE)


// Using the ChannelMaskHigh command we set the highest bytes
write([0, 0x0B, 0x00, 0x1F])
// wait for acknowledgement
do { read(arr, 1) } while (arr[0] != 0xFE)


// Set the internal timer to 50ms with the AutoScanTimer command.
// The timer duration is expressed in periods of a 32kHz clock.
// 50ms is equivalent to 1600 periods.
// The two argument bytes multiplied together should give 1600.
// We use 200 and 8.
write([0, 0x0E, 200, 8])
// wait for acknowledgement
do { read(arr, 1) } while (arr[0] != 0xFE)


// the sensor is all set up. Enable scanning on timer with the ScanTrigger command
write([0, 0x0F, 0x02])
// wait for acknowledgement
do { read(arr, 1) } while (arr[0] != 0xFE)


// To start reading data, write the offset from which we want to read from.
// This is 3 as we start from the Status byte
write([3])

while(1) {
 uint8 data[29]
 // Repeatedly read the Status byte until activity on a new frame is detected.
 oldFrameId = -1
 do {
  // we read faster than the period in order not to miss any frames
  sleepMs(30)
  read(data, 1)
  status = data[0]
  // bit six of the status bits is set if activity was detected
```

```
  activityDetected = status & (1 << 6)
  // lowest five bits are the frameId
  frameId = status & 0x1F
  newData = frameId != oldFrameId
  oldFrameId = frameId
} while (!(newData && activity))


// once we get here, we should read the full payload.
// We re-read the status byte we just read above because
// that's faster than performing a write to change the read offset
read(data, 29)
// here, parse and handle data. Note that the
// first byte is the status byte and can be discarded
}
```

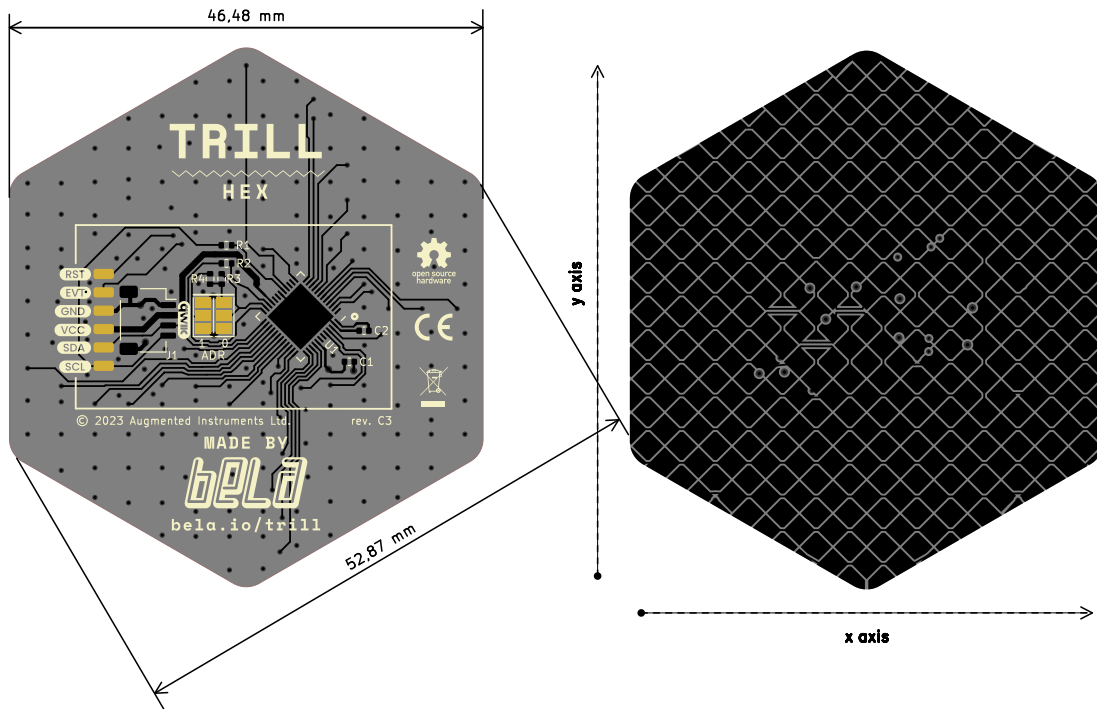# 8 Dimensions

8.1 Trill Bar
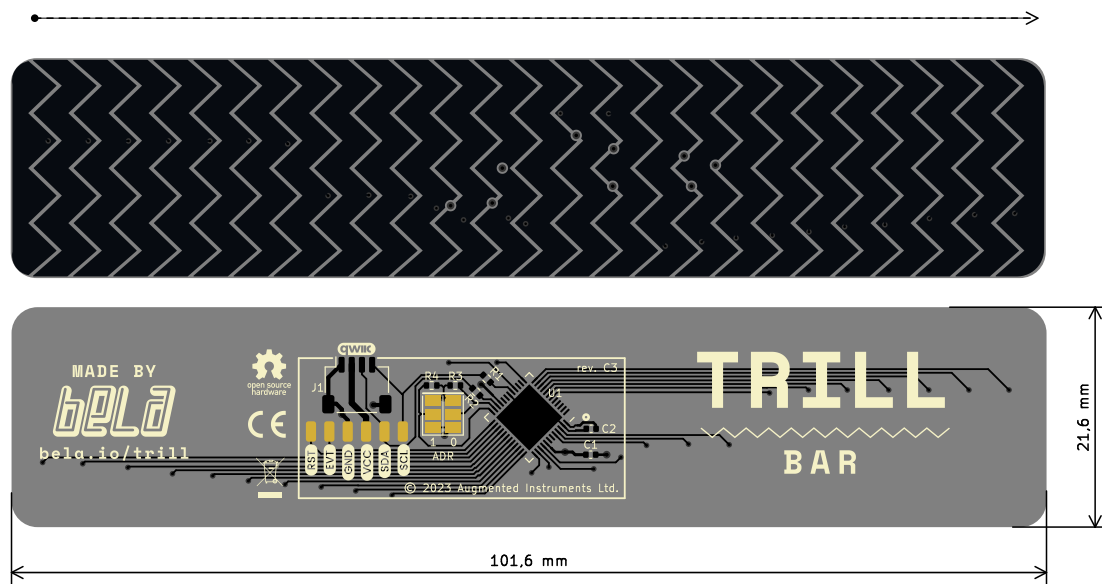
Figure 5: Trill Bar dimensions

4-layer PCB with 0.8 mm thickness. This sensor can be trimmed. The marked rectangular area shows the minimum size (32x14 mm). Don't cut inside this area.

8.2 Trill Square

4-layer PCB with 0.8 mm thickness. This sensor can be trimmed. The marked rectangular area shows the minimum size (32x14 mm). Don't cut inside this area.
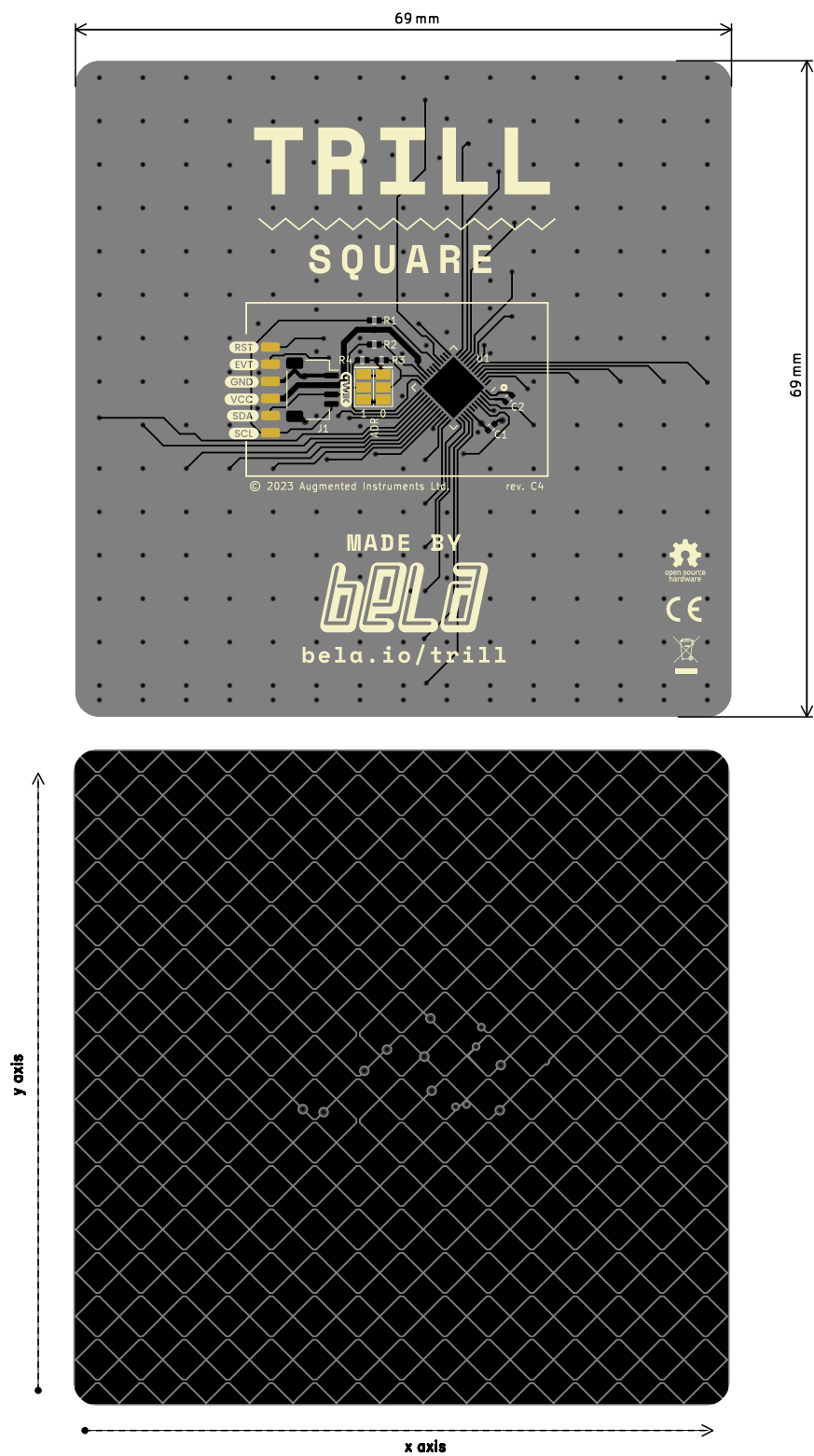
Figure 6: Trill Square dimensions

8.3 Trill Craft
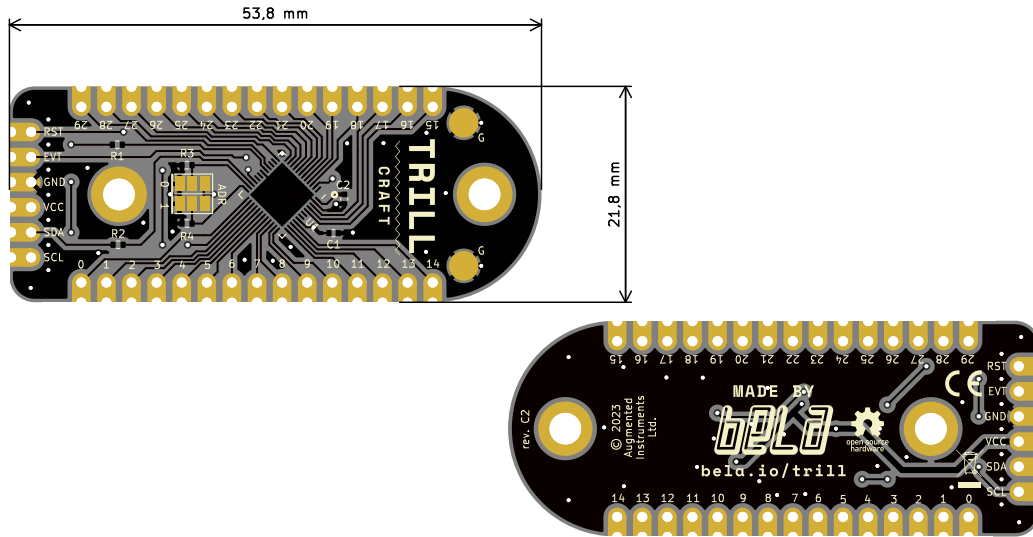
Figure 7: Trill Craft dimensions

2-layer PCB with 1.6 mm thickness. Castellated pads with 2.54 mm pitch. The two rows of holes on each side are 18 mm apart. Trill Craft has two mounting holes fitting for M3 screws 37 mm apart. These mounting holes are not connected to ground.

Important Note: The form factor for Trill Craft has been amended from Rev A to Rev C, with its length increasing from 52mm to 53.8mm. Footprints for both versions are available in the Trill KiCad footprint library[10].

8.4 Trill Ring
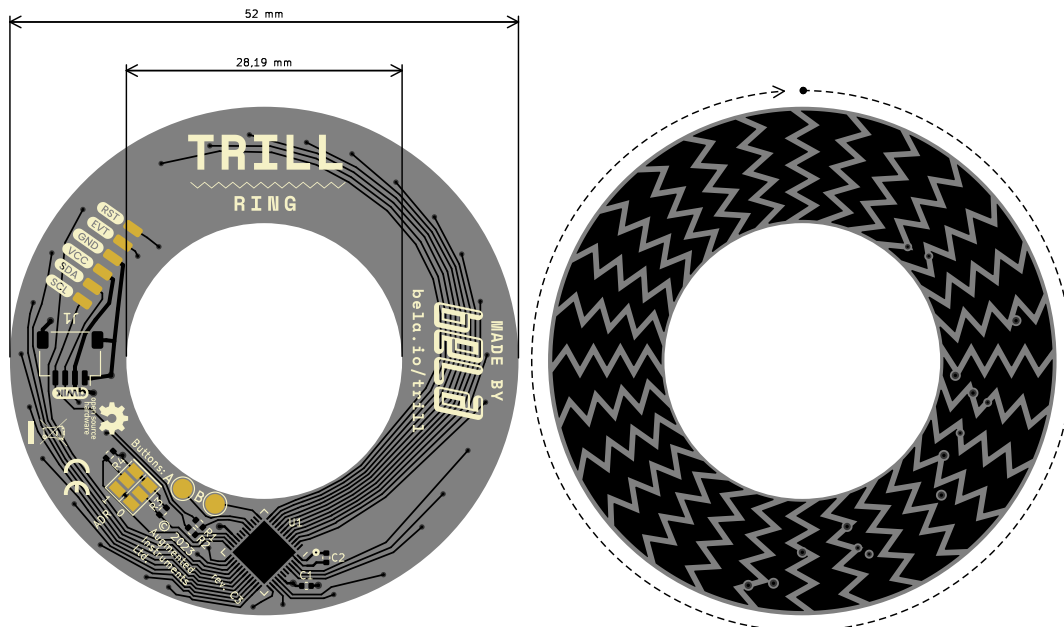
Figure 8: Trill Ring dimensions

4-layer PCB with 0.8 mm thickness. Two buttons can be connected using the pads labelled 'A' and 'B'.

---

[10] https://github.com/BelaPlatform/Trill/tree/main/hardware/trill-KiCad-libs/footprints

8.5 Trill Hex

Figure 9: Trill Hex dimensions

4-layer PCB with 0.8 mm thickness. This sensor can be trimmed. The marked rectangular area shows the minimum size (32x14 mm). Don't cut inside this area.

8.6 Trill Flex
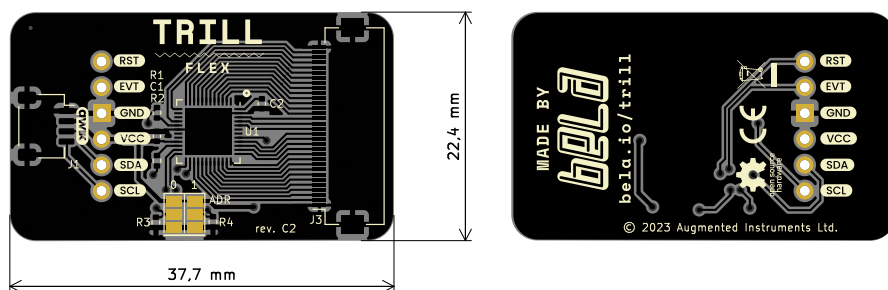
Figure 10: Trill Flex Base dimensions

Base: 2-layer PCB with 1.6 mm thickness. Flat Flexible Connector (FFC): 32 pin, 0.5mm pitch. Slider: 2-layer Flex-PCB.

To attach Trill Flex sensor: Hold the base facing up with the Trill logo at the top. Locate the high density connector on the right side of the base, and open its gate by gently pulling it towards the side of the PCB. Hold the sensor with the chevron-pattern touch surface facing up. Insert the end of the sensor fully into the high density connector. Push the gate on the connector gate closed, and give the sensor a gentle tug to make sure it's properly attached.
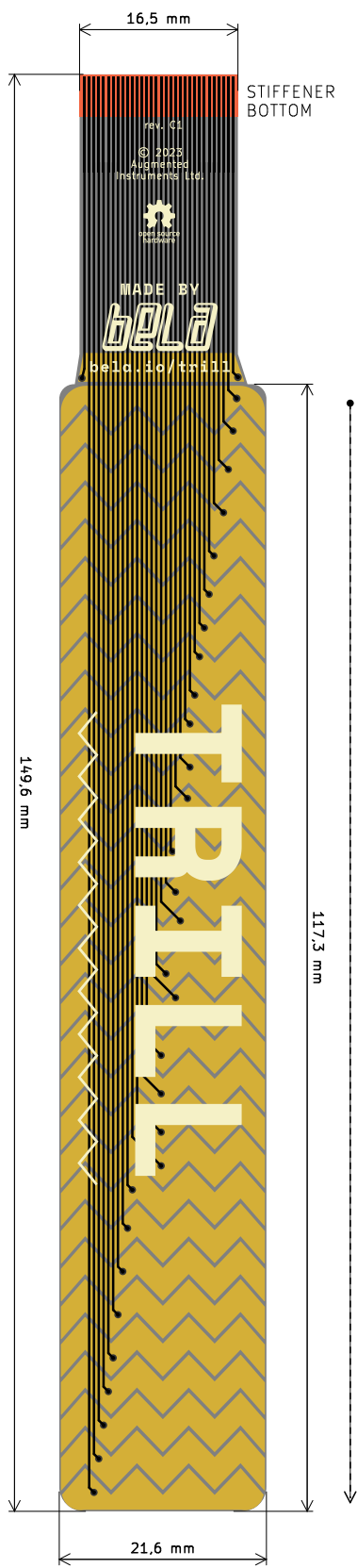
Figure 11: Trill Flex Slider dimensions

8.7 Trill Hub

2-layer PCB with 1.6 mm thickness.
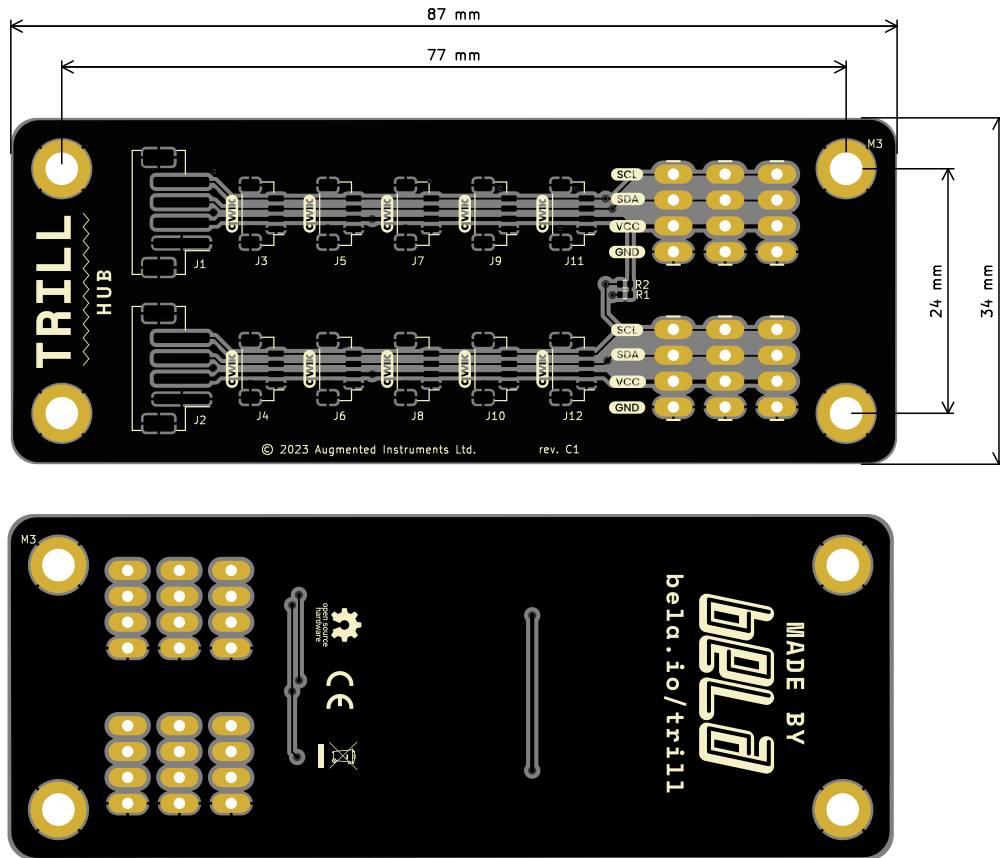
Figure 12: Trill Flex Hub dimensions

9 History

The original Trill sensors (Revisions A and B) were funded on Kickstarter in autumn 2019. They feature GROVE connectors, were designed in Eagle and have their own datasheet[11]. This datasheet is for revision C of the Trill sensors, for which we used the open source KiCad schematic capture and PCB design software suite and switched to more compact horizontal QWIIC connector.

9.1 Updates to Trill Craft in Rev C

There are two changes to Trill Craft in the current Rev C:

1. **Castellated pads.** Trill Craft now has castellated pads to make it easier to integrate into e-textile projects.

2. **Amended footprint.** Rev C of Trill Craft has an amended form factor, with its length increased from 52.0mm (Rev A) to 53.8mm (Rev C). Footprints for both versions are available in the Trill KiCad footprint library[12].

---

[11] https://github.com/BelaPlatform/Trill/blob/main/datasheet/REV_B/trill_datasheet.pdf
[12] https://github.com/BelaPlatform/Trill/tree/main/hardware/trill-KiCad-libs/footprints

---

10 Source Files and Licenses

Trill hardware designs and this datasheet are available under a Creative Commons Attribution-ShareAlike 4.0 International (CC BY-SA 4.0) License, meaning they can be freely reused and remixed with attribution, provided modifications remain open source. Trill firmware is licensed under GNU General Public License 3, meaning that you can freely use, remix, change and extend the code, but you are obligated to make the modified source code available alongside any binaries (flashed, or as files) that you release.

Source files can be found here: https://github.com/BelaPlatform/Trill

10.1 Certified Open Source Hardware

We strongly believe in the power of Open Source Hardware. All Trill sensors, and Trill Hub are certified as open source hardware by OSHWA.[13]

Table 18: OSHWA UIDs

Name	OSHWA UID	Certification page
Trill Bar	OSHW UK000052	https://certification.oshwa.org/uk000052.html
Trill Square	OSHW UK000051	https://certification.oshwa.org/uk000051.html
Trill Craft	OSHW UK000053	https://certification.oshwa.org/uk000053.html
Trill Hex	OSHW UK000055	https://certification.oshwa.org/uk000055.html
Trill Ring	OSHW UK000054	https://certification.oshwa.org/uk000054.html
Trill Flex	OSHW UK000056	https://certification.oshwa.org/uk000056.html
Trill Hub	OSHW UK000057	https://certification.oshwa.org/uk000057.html

10.2 Commercial Licensing

It's possible to use Trill design files and firmware for commercial projects free of charge under CC-BY-SA 4.0 (hardware) and GPL 3 (firmware). Both licenses allow you to use these assets yourself under the license requirements, which include publicly releasing any changes you make to the designs under the same license.

If you want to use Trill design files but not provide attribution and/or not release your source code – for example, because you want to create something to sell using these files but don't want to make your source files public – this is still possible, but requires a commercial license. Augmented Instruments Ltd. can provide you with a commercial license that fits your project and its scope. Get in touch at info@bela.io to discuss your product and the license that's right for you.[14]

Please note: The above commercial licensing applies only to modifications to our provided firmware code and PCB designs. You can buy Trill sensors and use them in any commercial or personal project without any additional licensing costs.

10.3 Trill/Bela Name and Logo

Trill and Bela names and logos are copyright Augmented Instruments Ltd. Any designs that you release or produce should **not** use the Trill name or logo and/or the Bela name and logo, whether modified or exact copies.

---

[13]Open Source Hardware Association https://oshwa.org

[14]More information may be found at https://learn.bela.io/products/products-overview/open-source-licenses/#using-trill-in-commercial-products

10.4 Disclaimer

All trademarks, logos and brand names including, but not limited to Infineon, Kickstarter, Eagle, KiCad, Texas Instruments, CapSense or Arduino are the property of their respective owners. All company, product and service names used in this datasheet are for identification purposes only. Use of these names, trademarks and brands does not imply affiliation or endorsement. All information in this datasheet is subject to change without notice.