# 3-Axis Accelerometer - ADXL345 - Trēo™ Module

## Module Features

- Analog Devices ADXL345
- RoHS Compliant
- Software Library
- NightShade Trēo™ Compatible
- Breakout Headers


## ADXL345 Features

(from Analog Devices)

- 10-bit Resolution
- ±16g Range
- Single/Double Tap Detection
- Free-Fall Detection
- 10,000g Shock Survival

## Applications

- Handsets
- Medical Instrumentation
- Gaming & Pointing Devices
- Industrial Instrumentation
- Personal Navigation Devices

## Description

The ADXL345 Trēo™ Module is a 3-Axis Accelerometer module that that features Analog Devices' ADXL345 3-Axis Accelerometer. It offers 13-bit resolution and a ±16g range. Built-in modules also provide tap and free-fall detection. This module is a part of the NightShade Treo system, patent pending.

## Trēo™ Compatibility

### Electrical

| | |
|---|---|
| **Communication** | I2C |
| **Max Current, 3.3V** | 1mA |
| **Max Current, 5V** | 0mA |

### Mechanical

- 25mm x 25mm Outline
- 20mm x 20mm Hole Pattern
- M2.5 Mounting Holes

## Table of Contents

# 1 Summary

The ADXL345 module measures acceleration in 3 axes with a range of ±2g, ±4g, ±8g, or ±16g and up to 13-bit resolution. It can be configured to only provide the most recent measurement or to collect measurements into a FIFO buffer. The begin() function configures the module to bypass the FIFO register, providing only the last measurement, with a ±4g range and 13-bit precision. Use the retrieveData() method to collect measurement data to a local buffer, which can be read with the readX(), readY(), and readZ() functions.

# 2 What is Trēo™?

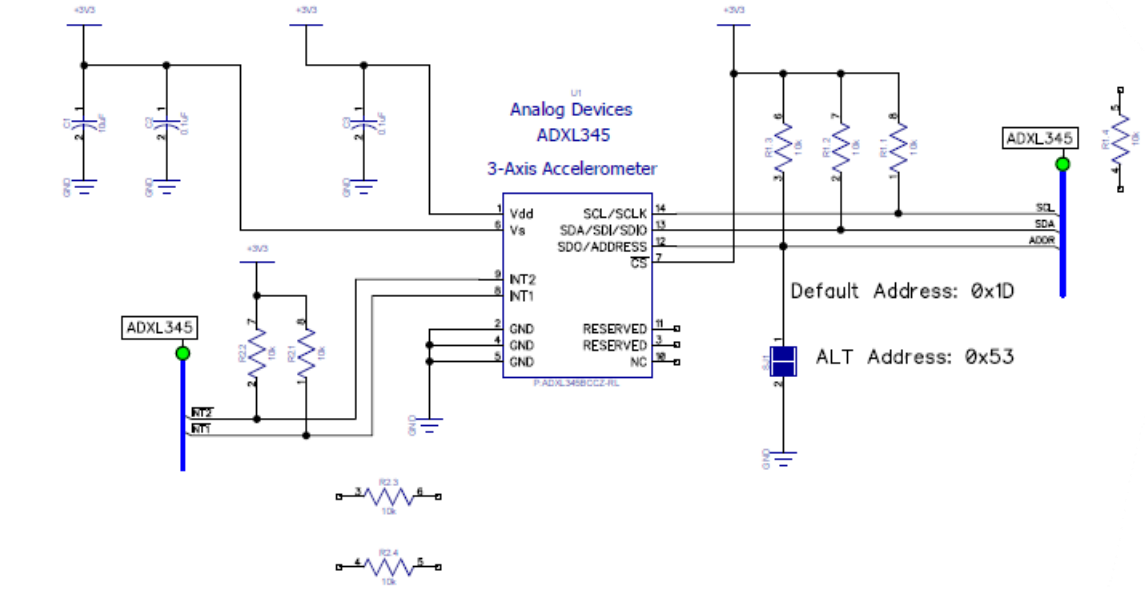
NightShade Trēo is a system of electronic modules that have standardized mechanical, electrical, and software interfaces. It provides you with a way to quickly develop electronic systems around microprocessor development boards. The grid attachment system, common connector/cabling, and extensive cross-platform software library allow you more time to focus on your application. Trēo is supported with detailed documentation and CAD models for each device.

Learn more about Trēo here.

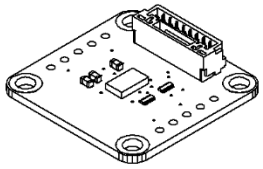
# 3 Electrical Characteristics

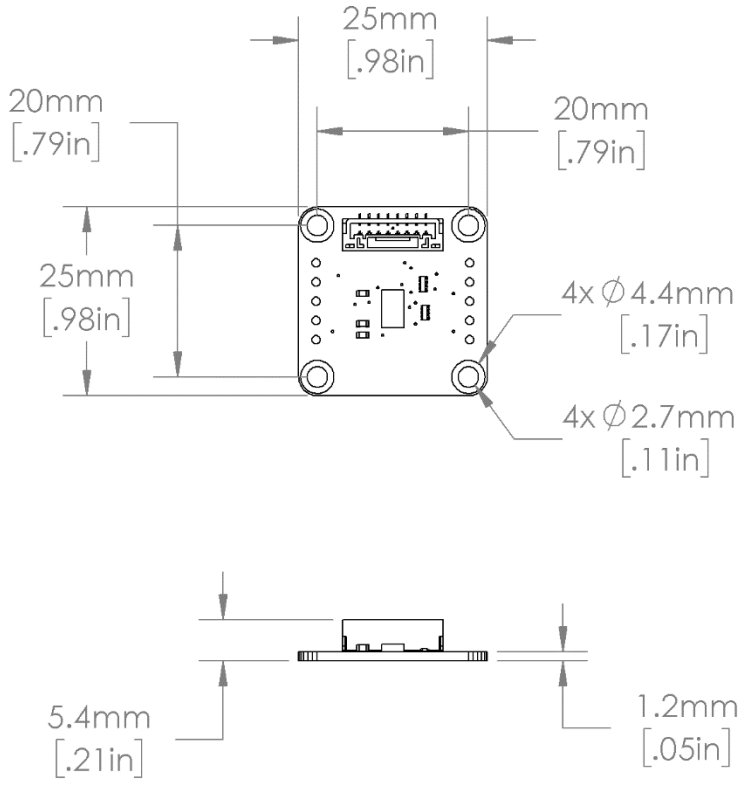
| | Minimum | Nominal | Maximum |
|---|---|---|---|
| **Voltages** | | | |
| $V_{i/o}$ (SDA, SCL, INT) | -0.3V | - | 3.6V |
| $V_{3.3V}$ | 3.1V | 3.3V | 3.5V |
| | | | |
| **Measurement** | | | |
| Bandwidth | 0.05Hz | - | 1600Hz |
| Range | -16g | - | +16g |
| Precision | 0.004g | - | 0.031g |
| Error | -1.7% | - | +1.7% |
| | | | |
| **I2C Slave Address** | | | |
| SJ1 Open (Default) | | 0x1D | |
| SJ1 Closed (Soldered) | | 0x53 | |
| | | | |
| **Operating Temperature** | -25°C | - | +85°C |

# 4 Electrical Schematic



# 5 Mechanical Outline





# 6 Example Arduino Program

```
/*********************************************************
  ADXL345_Accelerometer - NightShade_Treo by NightShade Electronics

  This sketch demonstrates the functionality of the
  NightShade Trēo ADXL345 accelerometer module. (NSE-1123-1) It reads the
  accelerations measured by the sensor and prints them over
  serial at 115200 baudrate.

  Created by Aaron D. Liebold
  on February 15, 2021

  Links:
  NightShade Trēo System: https://nightshade.net/treo
  Product Page: https://nightshade.net/product/treo-3-axis-accelerometer-adxl345/

  Distributed under the MIT license
  Copyright (C) 2021 NightShade Electronics
  https://opensource.org/licenses/MIT
*********************************************************/

// Include NightShade Treo Library
#include <NightShade_Treo.h>

// Declare Objects
NightShade_Treo_ADXL345 accel(1);

void setup() {
  Serial.begin(115200);
  accel.begin();
  accel.setMeasurementRange(0);
}

void loop() {
  accel.retrieveData();
  Serial.print("X: ");
  Serial.print(accel.readX());
  Serial.print("\tY: ");
  Serial.print(accel.readY());
  Serial.print("\tZ: ");
  Serial.print(accel.readZ());
  Serial.println();
  delay(250);
}
```

# 7 Library Overview (C++ & Python)

**C++ Class**

NightShade_Treo_ADXL345 <classObject>();

**Python Module**

<classObject> = NightShade_Treo.ADXL345()

## 7.1 Constructors

**NightShade_Treo_ADXL345(int port, uint8_t slaveAddress, uint32_t clockSpeed)**

Creates a class object.

Arguments:

| | |
|---|---|
| port | Integer of the I2C port used. (e.g. 0 = "/dev/i2c_0") |
| slaveAddress | The 7-bit slave address of the controller. |
| clockSpeed | The desired clock speed for the I2C bus. |

Returns:

Nothing

**NightShade_Treo_ADXL345(int port)**

Creates a class object assuming the default slave address and clock speed.

Arguments:

| | |
|---|---|
| port | Integer of the I2C port used. (e.g. 0 = "/dev/i2c_0") |

Returns:

Nothing

## 7.2 Methods

**begin()**

Initializes the ADXL345 into measurement mode with 4g range and 13-bit resolution.

Arguments:

None

Returns:

| | |
|---|---|
| Error | 0 = Success (int) |

**retrieveData()**

Reads X, Y, Z data from ADXL345 and stores it in a local buffer.

Arguments:
None

Returns

| | |
|---|---|
| Error | 0 = Success (int) |

**readX()**

Reads X value from local buffer.

Arguments
None

Returns
X-axis accelerometer value (int)

**readY()**

Reads Y value from local buffer.

Arguments
None

Returns
Y-axis accelerometer value (int)

**readZ()**

Reads Z value from local buffer.

Arguments
None

Returns
Z-axis accelerometer value (int)

**enableMeasureMode(uint8_t enable)**

Starts a single or continuous measurement cycle.

Arguments

| | |
|---|---|
| enable | True – Measurement is enabled<br>False – Measurement is disabled |

Returns

| | |
|---|---|
| Error | 0 = successful (int) |

**setBandwidth(uint8_t setting)**

Sets the data collection rate.

Arguments

| | |
|---|---|
| setting | 0: 0.1Hz Sample Rate, 0.05Hz Bandwidth<br>1: 0.2Hz Sample Rate, 0.1Hz Bandwidth<br>2: 0.39Hz Sample Rate, 0.2Hz Bandwidth<br>3: 0.78Hz Sample Rate, 0.39Hz Bandwidth<br>4: 1.56Hz Sample Rate, 0.78Hz Bandwidth<br>5: 3.13Hz Sample Rate, 1.56Hz Bandwidth<br>6: 6.25Hz Sample Rate, 3.13Hz Bandwidth<br>7: 12.5Hz Sample Rate, 6.25Hz Bandwidth<br>8: 25Hz Sample Rate, 12.5Hz Bandwidth<br>9: 50Hz Sample Rate, 25Hz Bandwidth<br>10: 100Hz Sample Rate, 50Hz Bandwidth<br>11: 200Hz Sample Rate, 100Hz Bandwidth<br>12: 400Hz Sample Rate, 200Hz Bandwidth<br>13: 800Hz Sample Rate, 400Hz Bandwidth<br>14: 1600Hz Sample Rate, 800Hz Bandwidth<br>15: 3200Hz Sample Rate, 1600Hz Bandwidth |

Returns

| | |
|---|---|
| Error | 0 = Success (int) |

**setMeasurementRange(uint8_t mode)**

Sets the full scale range of the sensor.

Arguments

| | |
|---|---|
| mode | 0: ±2g Range<br>1: ±4g Range<br>2: ±8g Range<br>3: ±16g Range |

Returns

| | |
|---|---|
| Error | 0 = Success (int) |

**enableSleep(uint8_t enable)**

Enables sleep mode.

Arguments

| | |
|---|---|
| enable | True – Measurement is enabled<br>False – Measurement is disabled |

Returns

| | |
|---|---|
| Error | 0 = Success (int) |

**int setWakeUp(uint8_t mode)**

Sets the frequency of reading during sleep mode.

Arguments

| | |
|---|---|
| enable | True – Measurement is enabled<br>False – Measurement is disabled |

Returns

| | |
|---|---|
| Error | 0 = Success (int) |

**enableLowPower(uint8_t enable)**

Enables low-power mode.

Arguments

| | |
|---|---|
| enable | True – Measurement is enabled<br>False – Measurement is disabled |

Returns

| | |
|---|---|
| Error | 0 = Success (int) |

**enableSelfTest(uint8_t enable)**

Enables the self-test mode.

Arguments

| | |
|---|---|
| enable | True – Measurement is enabled<br>False – Measurement is disabled |

Returns

| | |
|---|---|
| Error | 0 = Success (int) |

**setOffsetValues(uint8_t xOffest, uint8_t yOffest, uint8_t zOffest)**

Set the calibration offset values for the X, Y, and Z accelerometer axes.

Arguments

| | |
|---|---|
| xOffset | X-axis offset at zero acceleration. (uint8_t) |
| yOffset | Y-axis offset at zero acceleration. (uint8_t) |
| zOffset | Z-axis offset at zero acceleration. (uint8_t) |

Returns

| | |
|---|---|
| Error | 0 = Success (int) |

**invertInterruptOutput(uint8_t enable)**

Sets the interrupt pin to active-low behavior. (DO NOT CHANGE this setting if you are working with the Trēo system, this must always be set as active-low (enabled).)

Arguments

| | |
|---|---|
| enable | True – Measurement is enabled<br>False – Measurement is disabled |

Returns

| | |
|---|---|
| Error | 0 = Success (int) |

**enableFullResolution(uint8_t enable)**

Enables the full 13-bit resolution. The device operates at 10-bit resolution if this is disabled.

Arguments

| | |
|---|---|
| enable | True – Measurement is enabled<br>False – Measurement is disabled |

Returns

| | |
|---|---|
| Error | 0 = Success (int) |

**enableLink(uint8_t enable)**

Links the Activity and Inactivity functions.

Arguments

| | |
|---|---|
| enable | True – Link is enable<br>False – Link is disabled |

Returns

| | |
|---|---|
| Error | 0 = Success (int) |

**enableAutoSleep(uint8_t enable)**

Enables the auto-sleep function.

Arguments

| | |
|---|---|
| enable | True – Link is enable<br>False – Link is disabled |

Returns

| | |
|---|---|
| Error | 0 = Success (int) |

**setFifoMode(uint8_t mode)**

Sets the FIFO buffer mode.

Arguments

| | |
|---|---|
| mode | 0: Bypass<br>1: FIFO, 32 Values, Stops if full<br>2: Stream, Holds the last 32 values, Overwrites oldest<br>3: Trigger, Starts on trigger bit, FIFO behavior |

Returns

| | |
|---|---|
| Error | 0 = Success (int) |

**enableFifoTrigger(uint8_t mode)**

The trigger bit is set by the external interrupt pin. This setting switches the trigger between the INT1 and INT2 pins.

Arguments

| | |
|---|---|
| mode | 0: INT1 triggers FIFO<br>1: INT2 triggers FIFO |

Returns

| | |
|---|---|
| Error | 0 = Success (int) |

**setFifoSamples(uint8_t numberOfSamples)**

Sets the number of samples to be collected to the FIFO buffer.

Arguments

numberOfSamples

Returns

| | |
|---|---|
| Error | 0 = Success (int) |

**numberFifoEntries()**

Returns the number of values in the FIFO butter.

Arguments

None.

Returns

Number of FIFO entries. (int)

**checkFifoTrigger()**

Returns the status of the trigger bit.

Arguments

None

Returns

Trigger bit (Boolean)

**enableInterrupts(uint8_t regValue)**

Writes a value to the INT_ENABLE register. (see ADXL345 datasheet)

Arguments
regValue

Returns
Error 0 = Success (int)

**writeInterruptMap(uint8_t regValue);**

Writes a value to the INT_MAP register. (see ADXL345 datasheet)

Arguments
regValue

Returns
Error 0 = Success (int)

**readInterruptSources()**

Reads the value of the INT_SOURCE register. (see ADXL345 datasheet)

Arguments
None

Returns
Register value (uint8_t)

**setTapThreshold(uint8_t threshold)**

Sets the *tap* threshold.

Arguments
threshold

Returns
Error 0 = Success (int)

**setTapDuration(uint8_t duration)**

Sets the maximum duration of a *tap* to prevent the recognition of movement as a *tap*.

Arguments
duration 625µs/LSB

Returns
Error 0 = Success (int)

**setTapLatency(uint8_t latencyTime)**

Sets the minimum length of time between taps in a *double-tap*.

Arguments

| | |
|---|---|
| latencyTime | 1.25ms/LSB, 0 = double-tap disabled |

Returns

| | |
|---|---|
| Error | 0 = Success (int) |

**setDoubleTapWindow(uint8_t windowTime)**

Sets the maximum length of time between taps in a *double-tap*.

Arguments

| | |
|---|---|
| windowTime | 1.25ms/LSB, 0 = double-tap disabled |

Returns

| | |
|---|---|
| Error | 0 = Success (int) |

**setActivityThreshold(uint8_t threshold)**

Set the threshold level for detecting activity.

Arguments

| | |
|---|---|
| threshold | 62.5mg/LSB |

Returns

| | |
|---|---|
| Error | 0 = Success (int) |

**setInactivitiyThreshold(uint8_t threshold)**

Set the threshold level for detecting inactivity.

Arguments

| | |
|---|---|
| threshold | 62.5mg/LSB |

Returns

| | |
|---|---|
| Error | 0 = Success (int) |

**setInactivityTime(uint8_t time)**

Sets the amount of time that the accelerations must be less than the inactivity threshold before going to sleep.

Arguments

| | |
|---|---|
| time | 1sec/LSB |

Returns

| | |
|---|---|
| Error | 0 = Success (int) |

**setActInactControl(uint8_t regValue)**

Writes a value to the INT_ENABLE register. (see ADXL345 datasheet)

Arguments

| | |
|---|---|
| regValue | |

Returns

| | |
|---|---|
| Error | 0 = Success (int) |

**setTapAxes(uint8_t regValue)**

Writes a value to the TAP_AXES register. (see ADXL345 datasheet)

Arguments

| | |
|---|---|
| regValue | |

Returns

| | |
|---|---|
| Error | 0 = Success (int) |

**readActivityTapStatus()**

Reads the current value of the ACT_TAP_STATUS register. (see ADXL345 datasheet)

Arguments

None

Returns

Register value (uint8_t)

**setFFThreshold(uint8_t threshold)**

Sets the threshold for freefall detection.

Arguments

| | |
|---|---|
| threshold | 62.5mg/LSB |

Returns

| | |
|---|---|
| Error | 0 = Success (int) |

**setFFTime(uint8_t time)**

Sets the minimum time all accelerations must be lower than the freefall threshold for freefall detection.

Arguments

| | |
|---|---|
| time | 5ms/LSB |

Returns

| | |
|---|---|
| Error | 0 = Success (int) |