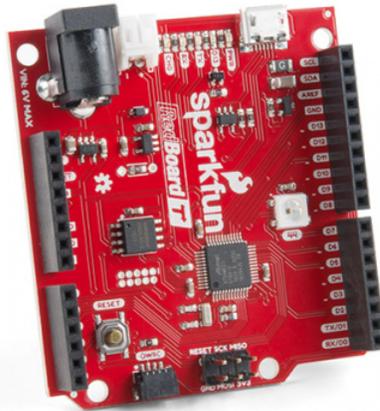


# RedBoard Turbo Hookup Guide

## Introduction

If you're ready to step up your Arduino game from older 8-bit/16MHz microcontrollers, the RedBoard Turbo is an awesome alternative. The RedBoard Turbo uses the ATSAM21G18, which is an ARM Cortex M0+, 32-bit microcontroller that can run at up to 48MHz. The RedBoard Turbo is similar to the SAMD21 Dev Breakout, with a few improvements. The RedBoard Turbo steps up the flash memory from the 256kB of internal memory to 4MB of external memory. Along with the UF2 Bootloader, the RedBoard Turbo is even easier to program than before!



## SparkFun RedBoard Turbo - SAMD21 Development Board

● DEV-14812

Product Showcase: Going Turbo





The RedBoard Turbo equips the ATSAM21G18 with a USB interface for programming and power, a Qwiic connector, an RTC crystal, WS2812-based addressable RGB LED, 600mA 3.3V regulator, LiPo charger, and a variety of other components.

## Required Materials

In addition to the RedBoard Turbo, you'll also need a Micro-B Cable (as if you don't already have dozens in your USB cable drawer!). That's all you'll need to get started. You can also take advantage of its LiPo charger with a single-cell Lithium Polymer battery. You may not need everything though depending on what you have. Add it to your cart, read through the guide, and adjust the cart as necessary.



SparkFun RedBoard Turbo - SAMD21 Development Board

● DEV-14812



USB micro-B Cable - 6 Foot

● CAB-10215

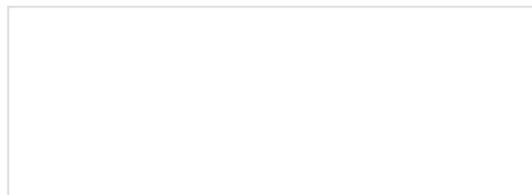
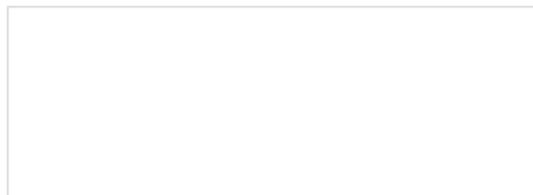
★★★★☆ 11

## Suggested Reading

Before continuing on with this tutorial, you may want to familiarize yourself with some of these topics if they're unfamiliar to you:



*Qwiic Connect System*





### Analog to Digital Conversion

The world is analog. Use analog to digital conversion to help digital devices interpret the world.



### What is an Arduino?

What is this 'Arduino' thing anyway?



### Installing Arduino IDE

A step-by-step guide to installing and testing the Arduino software on Windows, Mac, and Linux.



### SAMD21 Mini/Dev Breakout Hookup Guide

An introduction to the Atmel ATSAMD21G18 microprocessor and our Mini and Pro R3 breakout boards. Level up your Arduino-skills with the powerful ARM Cortex M0+ processor.

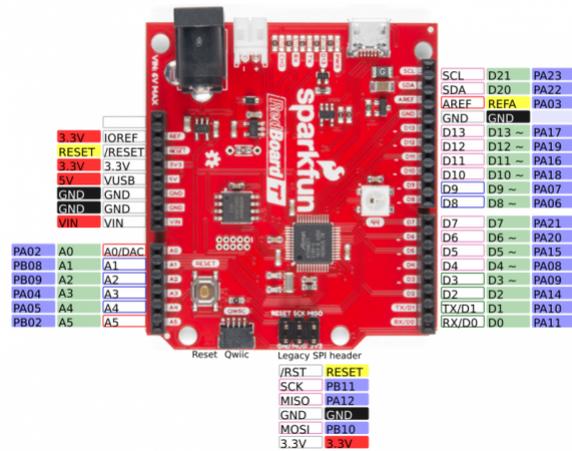
## SAMD21 RedBoard Turbo Overview

**Note:** For those interested in the nitty, gritty details of the SAMD21, check out the section from the Dev Breakout's SAMD21 overview or the datasheet.

Before we get into programming the SAMD21, let's first cover some of the features built into the RedBoard Turbo. The RedBoard Turbo is similar to our SAMD21 Dev Breakout, except turbocharged. In this section we'll cover powering the board, outlining the I/O pins, and what the various LEDs on the board are for.

### I/O Pins

If you've used any Arduino before, this pinout shouldn't surprise you – the layout meets the Arduino 1.0 footprint standard, including a separate SPI header and additional I<sup>2</sup>C header. For a quick reference, consult our graphical datasheet, which exhaustively shows the capability of each I/O pin and some of the other features on the board.



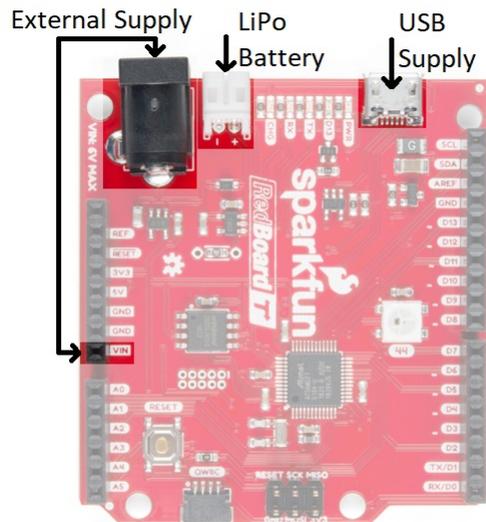
All **PWM-capable** pins are indicated with a tilde (~) adjacent to the pin-label. Speaking of “analog output”, true analog output is available on the A0 pin.

⚠ **3.3V Logic Levels!** When you start interfacing the SAMD21's I/O pins with external sensors and other components, keep in mind that each I/O will produce, at most, 3.3V for a high-level output.

When configured as an input, the maximum input voltage for each I/O is 3.6V (VDD+0.3V). If you're interfacing the SAMD21 with 5V devices, you may need some level shifters in between.

## Supplying Power

Power can be supplied to the RedBoard Turbo through either USB, a single-cell (3.7-4.2V) lithium-polymer battery, or an external 5V source via barrel jack. Each of the power supply inputs are available on the top edge of the board (the VIN pin on the power header can also be used).



⚠ **Warning** The barrel jack connection on the RedBoard Turbo has a lower input voltage than most Arduino development boards. Make sure that you are using a power supply below 6V!

## USB Power

The USB jack comes in the form of a **micro-B** connector. It should work with one of the many USB phone-charging cables you have lying around, or one of our Micro-B cables. You can plug the other end into a computer USB port, or use a USB Wall Adapter. The USB supply input includes a 500mA PTC resettable fuse – if something on or connected to the breakout fails, it should help protect your supply from damage.



Wall Adapter Power Supply - 5V DC 2A (USB Micro-B)

● TOL-12890



USB micro-B Cable - 6 Foot

● CAB-10215



Wall Adapter Power Supply - 5V DC 2A (Barrel Jack)

● TOL-12889



USB Wall Charger - 5V, 1A (Black)

● TOL-11456

### Single-Cell Lithium-Polymer (LiPo) Battery Charger

The SAMD21 touts many low-power features, so using it in battery-powered projects should be a common occurrence. We've integrated our standard 2-pin JST connector, and a single-cell USB battery charger into the board. Any of our single-cell lithium polymer batteries can be used to power the board.



Lithium Ion Battery - 1Ah

● PRT-13813



Lithium Ion Battery - 400mAh

● PRT-13851



Lithium Ion Battery - 2Ah

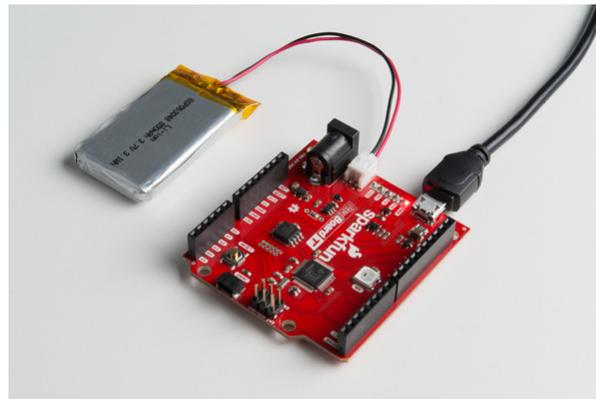
● PRT-13855



Lithium Ion Battery - 110mAh

● PRT-13853

To charge the battery, simply **connect USB or a 5V wall adapter** while the battery is also connected.

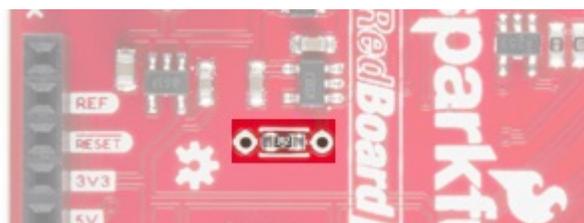


The “Charge” LED should illuminate while the battery is charging, and it should eventually turn off once fully juiced up.

### Configuring Battery Charge Current

The MCP73831's charge current is configured by a resistor value between 66k $\Omega$  and 2k $\Omega$ , to charge the battery at a rate between 15mA and 500mA, respectively. By default, the board is configured to charge the battery at around **250mA**.

Most batteries shouldn't be charged at a rate over 1C (for example, a 110mAh battery's 1C charge current would be 110mA). If you need to adjust the charge current, we've added pads for a through-hole resistor. This resistor can be added in parallel with the 3.9k $\Omega$  resistor already on board, or the CHG SET resistor can be removed with a soldering iron.



If you need a smaller charge current, the charge set resistor *must* be removed, before adding your own. Increasing the charge current can be achieved by adding a resistor in parallel. Here are a few resistor value/charge current examples:

Charge Current ( $I_{\text{Charge}}$ )	Total Resistance ( $R_{\text{Prog}}$ )	Parallel Resistor
40mA	25k $\Omega$	No, must remove CHG SET resistor
100mA	10k $\Omega$	No, must remove CHG SET resistor
400mA	2.5k $\Omega$	6.9k $\Omega$
500mA	2k $\Omega$	4.1k $\Omega$

The charge current is calculated as:

$$I_{\text{Charge}} = 1000/R_{\text{Prog}}$$

$R_{\text{Prog}}$  is the total programming resistor resistance, which may include the 3.9k $\Omega$  resistor in parallel.

## Current Capabilities

Depending on the task it's given, the SAMD21's core will usually consume between 3-17mA. There should be plenty of juice left from the 600mA 3.3V regulator to power other sensors or components off the Turbo's 3.3V supply rail.

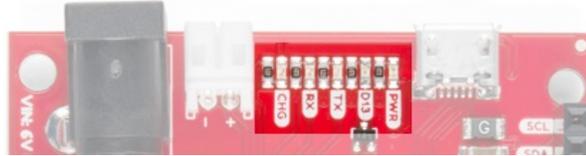
Each I/O pin can sink up to 10mA and source up to 7mA, with one caveat: **each cluster of I/O is limited to sourcing 14mA or sinking 19.5mA**. The GPIO clusters are:

Cluster	GPIO	Cluster Supply (Pin)	Cluster Ground (Pin)
1	SWCLK, SWDIO	VDDIN (44)	GND (42)
2	30, 31 (USB_HOST_EN, TX_LED)	VDDIN (44) VDDIO (36)	GND (42) GND (35)
3	D2, D5, D6, D7, D10, D11, D12, D13, D38 SCL, SDA, MISO, SCK, MOSI (USB_D-, USB_D+)	VDDIO (36) VDDIO (17)	GND (35) GND (18)
4	D0, D1, D3, D4	VDDIO (17)	GND (18)
5	A1, A2, A3, A4 D8, D9	VDDANA (6)	GNDANA (5)
6	A0, A5, AREF (RX_LED, RTC1, RTC2)	VDDANA (6)	GNDANA (5)

So, for example, if you're sourcing current to four LEDs tied to pins 0, 1, 3, and 4 (cluster 4), the sum of that current must be less than 14mA (~3.5mA per LED).

## LEDs

Speaking of LEDs, the RedBoard Turbo has a lot of them: a power indicator, pin 13 "status" LED, USB transmit and receive LED indicators, a battery charge status indicator, and addressable WS2812 LED.



### Status LED

The blue LED driven by the Arduino's pin 13 is actually sourced through an N-channel MOSFET, so less of our precious cluster-current is eaten up. The LED still turns on when you write the pin HIGH and off when pin 13 is LOW.

### Serial UART LEDs

The RX and TX LEDs indicate activity on the USB serial port. They are also addressable within an Arduino sketch, using the macros `PIN_LED_RXL` and `PIN_LED_TXL`. These LEDs are **active-low**, so writing the pin HIGH will turn the LED off.

### Charge LED

The charge LED is controlled by the board's integrated MCP73831 battery charger. If a battery is connected and 5V supplied (via USB or the external jack), it will illuminate when a battery is being charged and should turn off once fully-charged.

### Addressable WS2812 LED

The RGB LED uses the WS2812, which is connected to pin 44 which can be used for any purpose.



## UF2 Bootloader

The RedBoard Turbo is now easier than ever to program, thanks to the UF2 bootloader. With this bootloader, the RedBoard Turbo shows up on your computer as a USB storage device **without having to install drivers!**

From the Arduino IDE, you'll still need to select the correct port on your machine, but you can just as easily use another programming language such as CircuitPython or MakeCode, which will be available in the near future.

### What is UF2?

UF2 stands for USB Flashing Format, which was developed by Microsoft for PXT (now known as MakeCode) for flashing microcontrollers over the Mass Storage Class (MSC), just like a removable flash drive. The file format is unique, so unfortunately, you cannot simply drag and drop a compiled binary or hex file onto the Turbo. Instead, the format of the file has extra information to tell the processor where the data goes, in addition to the data itself.

For Arduino users, the UF2 bootloader is **BOSSA compatible**, which the Arduino IDE expects on ATSAMd boards. For more information about UF2, you can read more from the MakeCode blog, as well as the UF2 file format specification.

## Setting Up Arduino

While the SAMD21 alone is powerful enough, what truly makes it special is its growing support in the Arduino IDE. With just a couple click's, copies, and pastes, you can add ARM Cortex-M0+-support to your Arduino IDE. This page will list every step required for getting RedBoard Turbo support into your Arduino IDE.

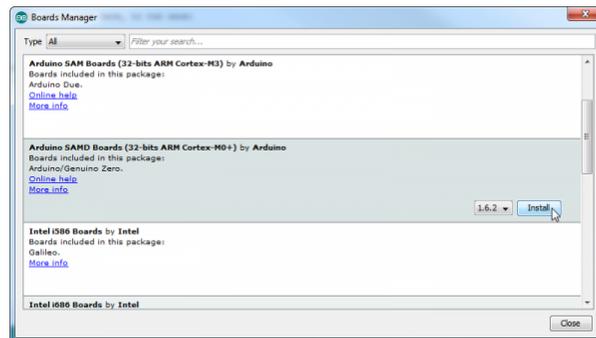
**Update Arduino!** This setup requires *at least* Arduino version 1.6.4 or later. We've tested it on 1.6.5 and the latest version – 1.8.8.

If you're running an older version of Arduino, consider visiting [arduino.cc](http://arduino.cc) to get the latest, greatest release.

### Install Arduino SAMD Board Add-Ons

First, you'll need to install a variety of tools, including low-level ARM Cortex libraries full of generic code, arm-gcc to compile your code, and bossa to upload over the bootloader. These tools come packaged along with Arduino's SAMD board definitions for the Arduino Zero.

To install the Arduino SAMD board definitions, navigate to your board manager (**Tools > Board > Boards Manager...**), then find an entry for **Arduino SAMD Boards (32-bits ARM Cortex-M0+)**. Select it, and install the latest version (recently updated to v1.6.19).



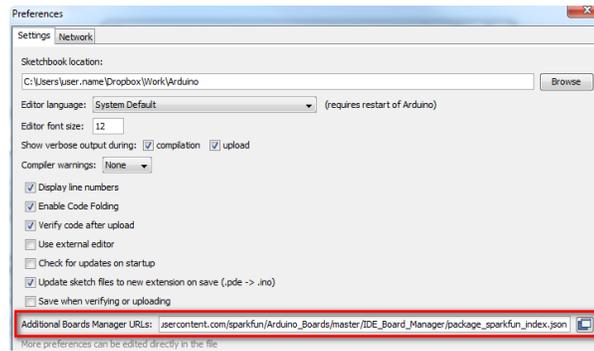
Downloading and installing the tools may take a couple minutes – arm-gcc in particular will take the longest, it's about 250MB unpacked.

Once installed, Arduino-blue "Installed" text should appear next to the SAMD boards list entry.

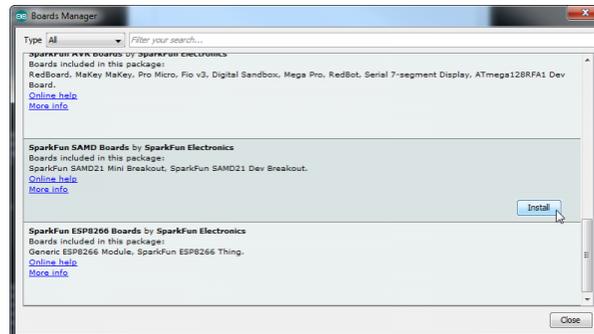
### Install SparkFun Board Add-On

Now that your ARM tools are installed, one last bit of setup is required to add support for the SparkFun SAMD boards. First, open your Arduino preferences (**File > Preferences**). Then find the **Additional Board Manager URLs** text box, and paste the below link in:

```
https://raw.githubusercontent.com/sparkfun/Arduino_Boards/master/IDE_Board_Manager/package_sparkfun_index.json
```



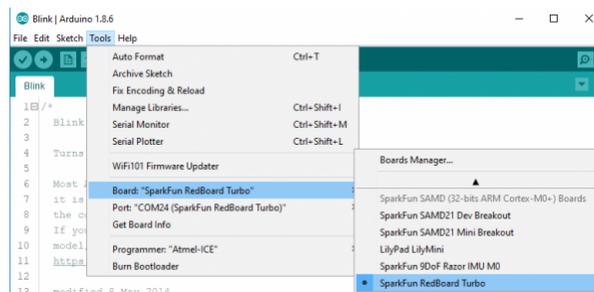
Then hit “OK”, and travel back to the **Board Manager** menu. You should (but probably won’t) be able to find a new entry for **SparkFun SAMD Boards**. If you don’t see it, close the board manager and open it again. ☹️



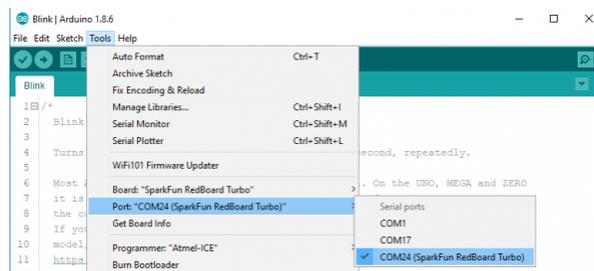
This installation should be much faster; you’ve already done the heavy lifting in the previous section.

### Select the Board and Serial Port

Once the board is installed, you should see a new entry in your **Tools > Board** list. **Select your SparkFun RedBoard Turbo.**



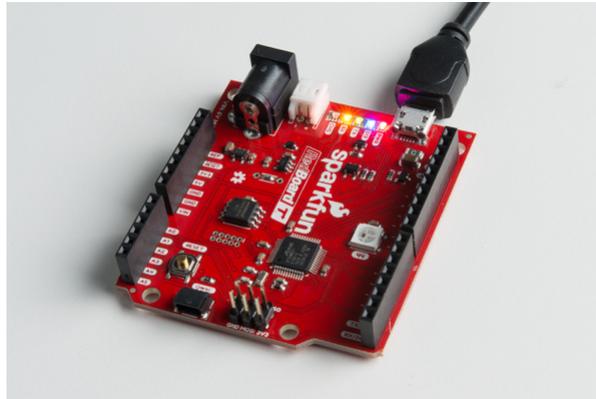
Finally, select your Turbo’s port. Navigate back up to the **Tool > Port** menu. The port menu may magically know which of your ports (if you have more than one) is the RedBoard Turbo board. On a Windows machine, the serial port should come in the form of “**COM#**”. On a Mac or Linux machine, the port will look like “**/dev/cu.usbmodem####**”.



Once you find it, select it!

### Example: Blink

As with any development board, if you can blink an LED, you're well on your way to controlling the rest of the world. Since the RedBoard Turbo has 3 user-controllable LEDs, let's blink them all!



The RX and TX LEDs are on pins 25 and 26, respectively, a couple pre-defined macros ( `PIN_LED_RXL` and `PIN_LED_TXL` ) can be used to access those pins, just in case you forget the numbers.

Here's a quick example sketch to blink the LEDs and make sure your environment is properly set up. Copy and paste from below, and upload!

```
const int BLUE_LED = 13; // Blue "stat" LED on pin 13
const int RX_LED = PIN_LED_RXL; // RX LED on pin 25, we use the predefined PIN_LED_RXL to make sure
const int TX_LED = PIN_LED_TXL; // TX LED on pin 26, we use the predefined PIN_LED_TXL to make sure

bool ledState = LOW;

void setup()
{
  pinMode(BLUE_LED, OUTPUT);
  pinMode(RX_LED, OUTPUT);

  pinMode(TX_LED, OUTPUT);
  digitalWrite(RX_LED, HIGH);
  digitalWrite(TX_LED, HIGH);
  digitalWrite(BLUE_LED, LOW);
}

void loop()
{
  digitalWrite(RX_LED, LOW); // RX LED on
  delay(333);
  digitalWrite(RX_LED, HIGH); // RX LED off
  digitalWrite(TX_LED, LOW); // TX LED on
  delay(333);
  digitalWrite(TX_LED, HIGH); // TX LED off
  digitalWrite(BLUE_LED, HIGH); // Blue LED on
  delay(333);
  digitalWrite(BLUE_LED, LOW); // Blue LED off
}
```

After hitting the “Upload” button, wait a handful of seconds while the code compiles and sends. While the code uploads, you should see the blue LED flicker. Once you’ve verified that the IDE is all set up, you can start exploring the world of the ATSAM21!

## Example: Serial Ports

One of the SAMD21’s most exciting features is SERCOM – its multiple, configurable serial ports. The Arduino IDE equips the SAMD21 with two hardware serial ports, by default, plus a third “USB serial port” for communicating between the serial monitor.

Each of these serial ports has a unique `Serial` object which you’ll refer to in code:

Serial Object	Serial Port	RX Pin	TX Pin
<code>SerialUSB</code>	USB Serial (Serial Monitor)		
<code>Serial1</code>	Hardware Serial Port 1	0	1

There are a couple critical things to notice here. First of all, if you’re trying to use the Serial Monitor to debug, you’ll need to use `SerialUSB.begin(<baud>)` and `SerialUSB.print()`. (Thankfully find/replace exists for adjusting example code.)

Here’s a quick example demonstrating the differences between Serial Monitor and `Serial1`. It is designed to route data from `Serial1` to the Serial Monitor, and vice-versa.

```

void setup()
{
  SerialUSB.begin(9600); // Initialize Serial Monitor USB
  Serial1.begin(9600); // Initialize hardware serial port, pins 0/1

  while (!SerialUSB) ; // Wait for Serial monitor to open

  // Send a welcome message to the serial monitor:
  SerialUSB.println("Send character(s) to relay it over Serial1");
}

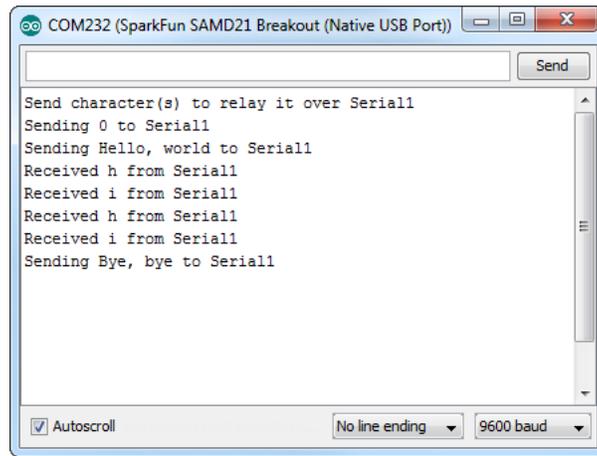
void loop()
{
  if (SerialUSB.available()) // If data is sent to the monitor
  {
    String toSend = ""; // Create a new string
    while (SerialUSB.available()) // While data is available
    {
      // Read from SerialUSB and add to the string:
      toSend += (char)SerialUSB.read();
    }
    // Print a message stating what we're sending:
    SerialUSB.println("Sending " + toSend + " to Serial1");

    // Send the assembled string out over the hardware
    // Serial1 port (TX pin 1).
    Serial1.print(toSend);
  }

  if (Serial1.available()) // If data is sent from device
  {
    String toSend = ""; // Create a new string
    while (Serial1.available()) // While data is available
    {
      // Read from hardware port and add to the string:
      toSend += (char)Serial1.read();
    }
    // Print a message stating what we've received:
    SerialUSB.println("Received " + toSend + " from Serial1");
  }
}

```

Then try typing something into the serial monitor. Even with nothing connected to the hardware serial port, you should see what you typed echoed back at you.



You can further test this sketch out by connecting an FTDI Basic or any other serial device to the SAMD21's pins 0 (RX) and 1 (TX). Data sent from the FTDI should end up in your Serial Monitor, and data sent to your Serial Monitor will route over to the FTDI.

## Example: Analog Input and Output

While it still has PWM-based “analog outputs”, the SAMD21 also features true analog output in the form of a digital-to-analog converter (DAC). This module can produce an analog voltages between 0 and 3.3V. It can be used to produce audio with more natural sound, or as a kind of “digital potentiometer” to control analog devices.

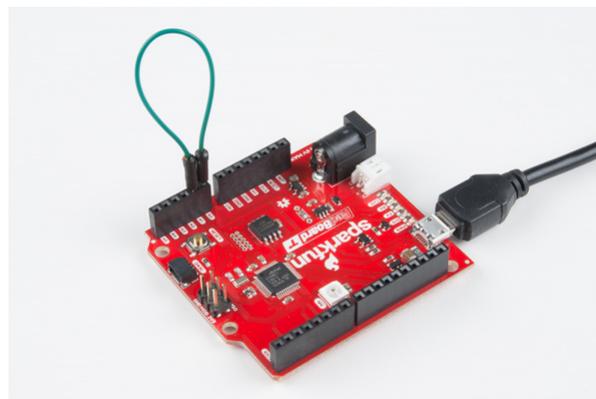
The DAC is only available on the **Arduino pin A0**, and is controlled using `analogWrite(A0, <value>)`. The DAC can be set up to 10-bit resolution (making sure to call `analogWriteResolution(10)` in your setup), which means values between 0 and 1023 will set the voltage to somewhere between 0 and 3.3V.

In addition to the DAC, the SAMD21's ADC channels also stand apart from the ATmega328: they're equipped with up to **12-bit resolution**. That means the analog input values can range from 0-4095, representing a voltage between 0 and 3.3V. To use the ADC's in 12-bit mode, make sure you call `analogReadResolution(12)` in your setup.

### Serial Plotting the DAC

The **Serial Plotter** in this example requires Arduino 1.6.6 or later. Visit [arduino.cc](http://arduino.cc) to get the latest, greatest version.

Here's an example that demonstrates both the 10-bit DAC and the 12-bit ADC. To set the experiment up, **connect A0 to A1** – we'll drive A0 with an analog voltage, then read it with A1. It's the simplest circuit we've ever put in a tutorial:



## Jumping a temporary connection between A0 (our DAC) and A1.

Then copy and paste the code below into your Arduino IDE, and upload!

```
// Connect A0 to A1, then open the Serial Plotter.

#define DAC_PIN A0 // Make code a bit more legible

float x = 0; // Value to take the sin of
float increment = 0.02; // Value to increment x by each time
int frequency = 440; // Frequency of sine wave

void setup()
{
  analogWriteResolution(10); // Set analog out resolution to max, 10-bits
  analogReadResolution(12); // Set analog input resolution to max, 12-bits

  SerialUSB.begin(9600);
}

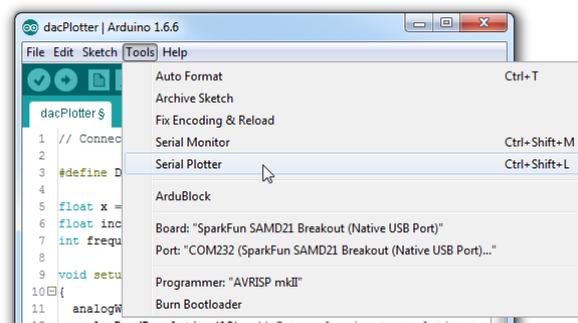
void loop()
{
  // Generate a voltage value between 0 and 1023.
  // Let's scale a sin wave between those values:
  // Offset by 511.5, then multiply sin by 511.5.
  int dacVoltage = (int)(511.5 + 511.5 * sin(x));
  x += increment; // Increase value of x

  // Generate a voltage between 0 and 3.3V.
  // 0= 0V, 1023=3.3V, 512=1.65V, etc.
  analogWrite(DAC_PIN, dacVoltage);

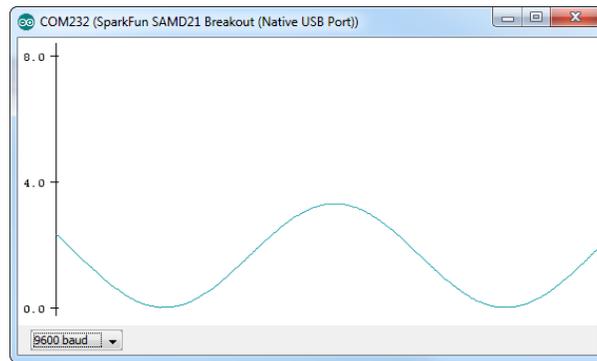
  // Now read A1 (connected to A0), and convert that
  // 12-bit ADC value to a voltage between 0 and 3.3.
  float voltage = analogRead(A1) * 3.3 / 4096.0;
  SerialUSB.println(voltage); // Print the voltage.
  delay(1); // Delay 1ms
}
```

This sketch produces a sine wave output on A0, with values ranging from 0 to 3.3V. Then it uses A1 to read that output into its 12-bit ADC, and convert it into a voltage between 0 and 3.3V.

You can, of course, open the serial monitor to view the voltage values stream by. But if the the sine wave is hard to visualize through text, check out Arduino's new **Serial Plotter**, by going to **Tools > Serial Plotter**.



And take in the majesty of that sine wave.



## Example: Addressable RGB LED

**Heads up!** Since the addressable WS2812 LED is attached to pin 44, we will be using the NeoPixel library. The FastLED will not be able to work at that high of an I/O number for the SAMD21

In this last example, we'll take a look at how to use the RGB LED on the RedBoard Turbo. The RGB LED comes in the form of a WS2812, which could be great as a status LED or for debugging if you don't want or need to use serial terminal. In the example below, we'll test the functionality of the LED by using the rainbow fade code below. To use this code, you will need to install the NeoPixel library. You can obtain these libraries through the Arduino Library Manager. Search for NeoPixel and you should be able to install the latest version. If you prefer downloading the libraries manually you can grab them from the GitHub repository:

[DOWNLOAD NEOPIXEL LIBRARY \(ZIP\)](#)

Once the library has been installed, copy and paste the following code into your Arduino IDE.

```

#include <Adafruit_NeoPixel.h>
#define LEDPIN RGB_LED // connect the Data from the strip to this pin on the Arduino
#define NUMBER_PIXELS 1 // the number of pixels in your LED strip
Adafruit_NeoPixel strip = Adafruit_NeoPixel(NUMBER_PIXELS, LEDPIN, NEO_GRB + NEO_KHZ800);

int wait = 10; // how long we wait on each color (milliseconds)

void setup() {
  strip.begin();
}

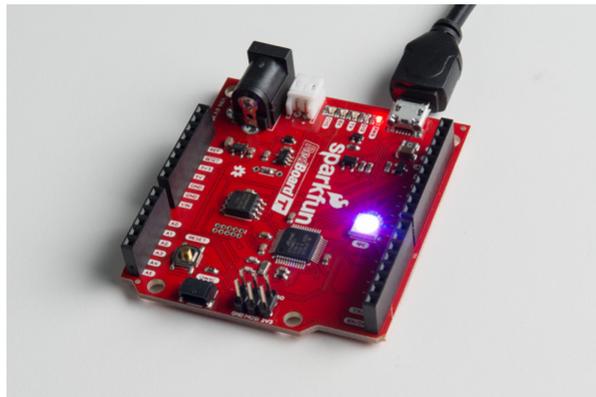
void loop() {

  for (int color=0; color<255; color++) {
    for (int i=0; i<strip.numPixels(); i++) {
      strip.setPixelColor(i, Wheel(color));
    }
    strip.show();
    delay(wait);
  }
}

// Input a value 0 to 255 to get a color value.
// The colours are a transition r - g - b - back to r.
uint32_t Wheel(byte WheelPos) {
  WheelPos = 255 - WheelPos;
  if(WheelPos < 85) {
    return strip.Color(255 - WheelPos * 3, 0, WheelPos * 3);
  } else if(WheelPos < 170) {
    WheelPos -= 85;
    return strip.Color(0, WheelPos * 3, 255 - WheelPos * 3);
  } else {
    WheelPos -= 170;
    return strip.Color(WheelPos * 3, 255 - WheelPos * 3, 0);
  }
}

```

Once uploaded, you should see the LED changing colors. Notice in the code, the RGB LED's pin is defined using **RGB\_LED**. You could also call it using **LED4** or it's pin number, **44** .



## Troubleshooting

For troubleshooting tips, checkout the SAMD21 Troubleshooting guide here for common issues that you might run into when using the SAMD21 with Arduino. The only exception is that the RedBoard Turbo does not require drivers so tips for re-installing drivers will not apply.

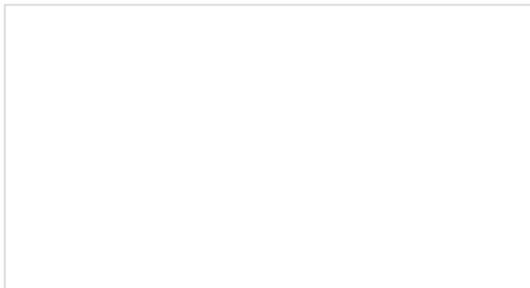
## SAMD21 MINI/DEV BREAKOUT HOOKUP GUIDE: TROUBLESHOOTING

### Resources and Going Further

There is a wealth of information out there, whether you're looking for datasheets, schematics, or design files. Additional resources, here are a few links you might find handy:

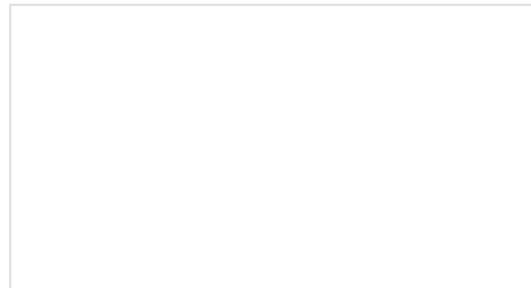
- **SparkFun RedBoard Turbo Design Resources**
  - GitHub Product Repository
  - Schematic (PDF)
  - Eagle (ZIP)
  - Graphical Datasheet
  - GitHub: Arduino Board Definitions
- **ATmel ATSAM21 Resources**
  - Atmel ATSAM21G18A Product Page
  - ATSAM21 Summary Datasheet
  - ATSAM21 Full Datasheet
- **Arduino ATSAM21 Resources**
  - SAMD Arduino Core GitHub Repository
- SFE Product Showcase

It's a brave new world out there – Arduinos and ARMs working together! What are you going to create with your powerful, new RedBoard Turbo? Looking for some inspiration, check out these tutorials!



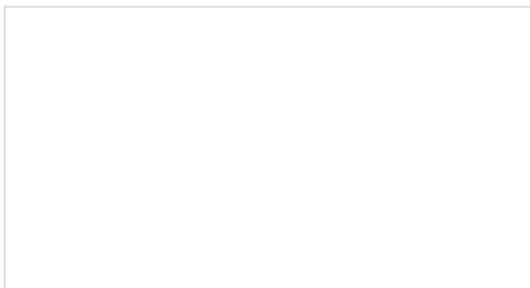
#### Arduino Shields

All things Arduino Shields. What they are and how to assemble them.

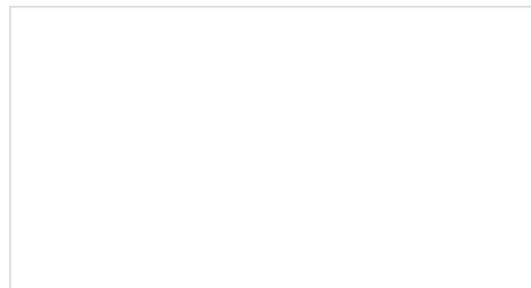


#### Using GitHub to Share with SparkFun

A simple step-by-step tutorial to help you download files from SparkFun's GitHub site, make changes, and share the changes with SparkFun.



#### Connecting Arduino to Processing



#### Data Types in Arduino

Send serial data from Arduino to Processing and back - even at the same time!

Learn about the common data types and what they signify in the Arduino programming environment.