
Table of Contents

Introduction	1.1
API	1.2
CLI	1.2.1
Hardware API	1.2.2
Modules	1.2.3
Network API	1.2.4
Supported Languages	1.2.5
Debugging	1.3
LAN Discovery	1.3.1
Node.js	1.3.2
Root Access	1.3.3
Technical Overview	1.3.4
USB	1.3.5
Hardware	1.4
Modules	1.4.1
Tessel 2 Overview	1.4.2
Tutorials	1.5
Communication Protocols	1.5.1
Interrupts	1.5.2
Making Your Own Module	1.5.3
Pinpull	1.5.4
Pulse Width Modulation	1.5.5

Tessel 2 Docs

Looking for Tessel 1 docs? Look [here](#).

Welcome to the Tessel 2 documentation.

All of the information you need is separated into three sections at left: [API](#), Debugging, and Tutorials.

If you are looking for information on how to use Tessel or one of the Tessel modules, look in the [API](#) section.

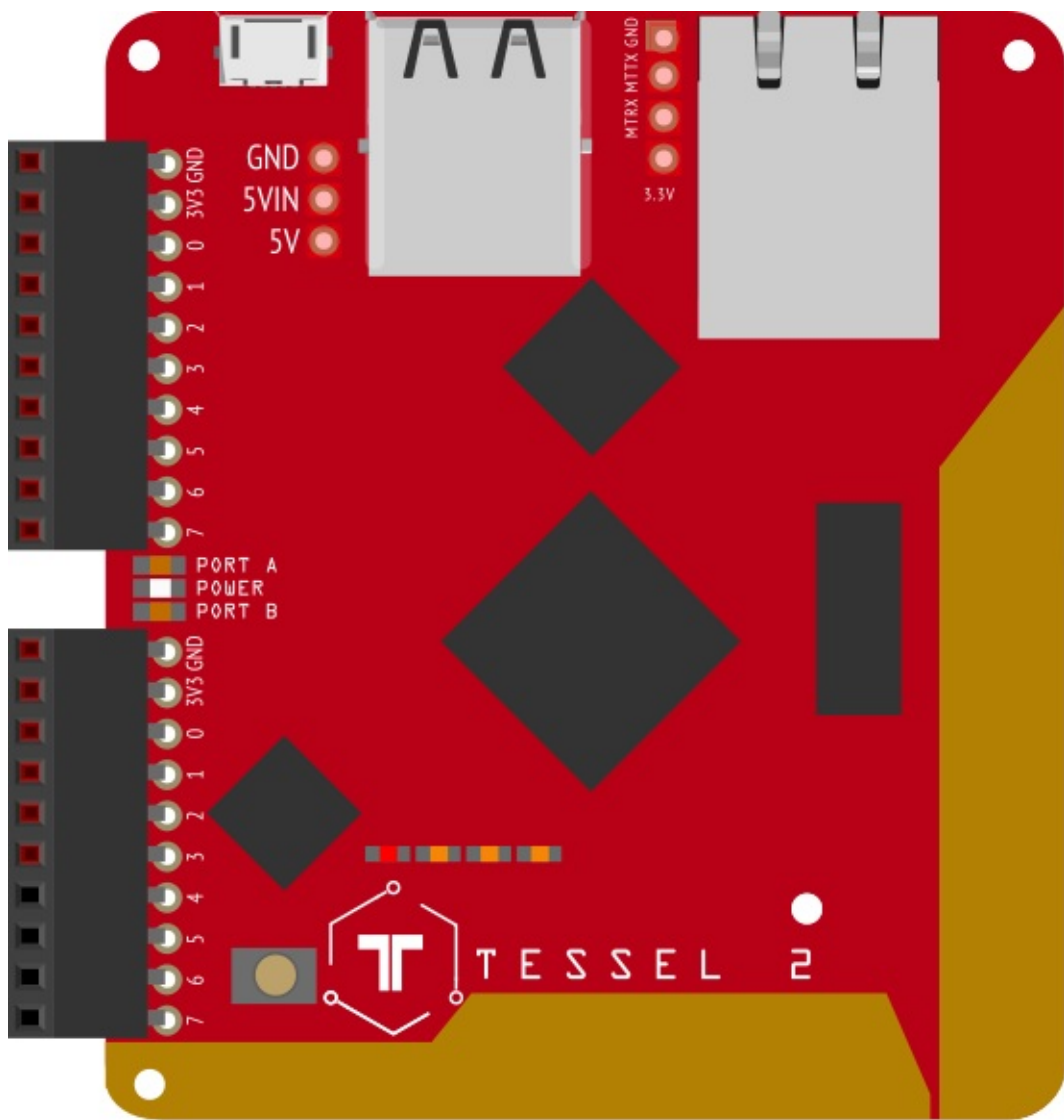
Looking to learn more about hardware and how to create it? Check out the information in the Tutorials section.

If you are here to debug an issue or get up to speed on the open source project, the Debugging section should help you, particularly the [Technical Overview](#).

Looking for information you can't find? Please report incorrect or missing information as an issue on the [Tessel 2 docs repo](#).

About Tessel

Tessel is an open source project under open governance. Read more [here](#).



fritzing

[Fritzing part for Tessel](#)

Get involved: The Tessel project welcomes contributors! Get started [here](#) and reach out if you have any questions.

Stay in the loop: Sign up for our weekly-ish newsletter [This Week in Tessel](#)

License: This documentation repo is CC-BY-SA 4.0 licensed.

Tessel 2 Command Line Interface

- [Installation](#)
- [Updating Tessel](#)
- [Setup](#)
- [Usage/CLI commands](#)

Installation

Prerequisites for installation: [Node.js](#) 4.2.x or greater

```
npm install -g t2-cli
```

The best place to go next is [the Tessel 2 start experience](#), which will walk you through a tutorial.

Updating Tessel 2 On-board OS/Firmware

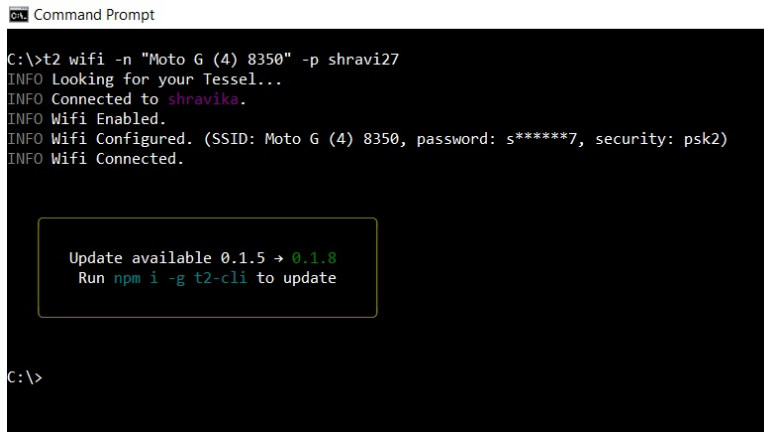
How do I know if I need to update my T2?

[t2-cli](#) and [t2-firmware](#) are separately versioned.

- For updating [t2-cli](#):

When a [CLI](#) command is run on an internet-connected computer, the [CLI](#) automatically checks for updates to the `t2-cli` module. If an update is available, the [CLI](#) will provide the update instructions using a notifier to the user (as shown below). The [CLI](#) can be updated with a global npm re-install of the [CLI](#) using the command `npm install -g t2-cli`.

The [CLI](#) only checks once per day to ensure that the [CLI](#) functioning does not slow down.



```
Command Prompt
C:\>t2 wifi -n "Moto G (4) 8350" -p shravi27
INFO Looking for your Tessel...
INFO Connected to shravika.
INFO Wifi Enabled.
INFO Wifi Configured. (SSID: Moto G (4) 8350, password: s*****7, security: psk2)
INFO Wifi Connected.

Update available 0.1.5 → 0.1.8
Run npm i -g t2-cli to update

C:\>
```

- For updating **t2-firmware**:

The user does not get update information for **t2-firmware** from the **CLI**. The most convenient and fastest way to find out if a firmware update is needed is by running the `t2 update` command and the `t2-firmware` will update automatically if there is a newer firmware version available. Otherwise, you can run `t2 version` to get the version running on your Tessel, and then `t2 update -l` to see the 10 newest versions available.

Updating

Simply run `t2 update`. If you want to update to a specific version, run `t2 update -v VERSION_NUM` where `VERSION_NUM` is one of the versions returned by `t2 update -l` (like `t2 update -v 0.0.6`).

Setup

USB

Connecting to a Tessel 2 over USB requires no special setup.

LAN

In order to authorize the device with your computer to work over a **LAN** connection, call `t2 provision` after connecting it via USB. This will place an SSH key on the device. Use the `t2 wifi` command as described below to connect Tessel 2 to a local network. You should now be able to access your Tessel 2 remotely.

SSH

To connect to your Tessel 2 over SSH use the `t2 root` command. Before you connect, call `t2 provision` to authorize your computer to access your Tessel via SSH.

Usage

- `t2 root`
 - `--timeout` Set timeout in seconds for scanning networked Tessels. The default is `5` seconds.
 - `--key` SSH key for authorization with your Tessel. Optional, only required if you have changed the keypath after your Tessel was provisioned.
 - `--name` The name of the Tessel where the command will be executed.
 - `--output` Enable or disable writing command output to stdout/stderr. Useful for [CLI API](#) consumers. Set to `true` by default
 - `--loglevel` Set the loglevel. It is set to `basic` by default. Available options are `'trace'`, `'debug'`, `'basic'`, `'info'`, `'http'`, `'warn'`, `'error'`

Virtual Machine

Check out the [Virtual Machine repo](#) for instructions on how to set up the VM. All [CLI](#) commands except `provision` and `wifi` should be functional with the VM.

Usage

Specify which Tessel to use with the `--name <name>` option appended to any command. If `--name` is not specified, [CLI](#) will look for an environment variable, e.g. `export TESSEL=Bulbasaur`. If none of the above are specified and there is one Tessel connected over USB, this Tessel will be preferred. Finally, if there is only one Tessel available and none of the above are specified, [CLI](#) will choose that Tessel.

Starting Projects

- `t2 init` in the current directory, create a `package.json` and `index.js` with [Hello World code](#).

Project Files

Along with the `package.json` and `index.js` included in the `t2 init` process, there are some other files that may be useful for your project:

- `.tesselignore` similar to `.gitignore` or `.npmignore`, this file should list any files or directories you want ignored by the [T2](#) bundling and deployment process. This is handy

when using the `--full` flag, which tells [T2](#) to bundle everything in the project directory.

- `.tesselinclude` the overriding and opposite behavior of `.tesselignore`, this file should list any files or directories you want included with the [T2](#) bundling and deployment process. This is handy for including non-JavaScript assets, like HTML, CSS, and images, for use within your project.
 - In your JavaScript program, assets listed in the `.tesselinclude` file should always be accessed using `__dirname` (read more about `__dirname` in the [Node.js docs](#)). For example, if there is a file at `public/index.html` and it's listed in your `.tesselinclude`, from your JavaScript program, you'd read it like this:

```
fs.readFile(path.join(__dirname, "public/index.html"), "utf8", (error, contents)
=> ...);
```

Tessel Management

- `t2 key` will generate a local SSH keypair for authenticating to a Tessel. Use with the `generate` argument to generate a new local SSH keypair for authenticating to a Tessel.
- `t2 list` will show what Tessels are available over WiFi and USB.
- `t2 provision` will authorize your computer to access a Tessel over SSH. (USB-connected Tessel only)
- `t2 rename` will change the name of a Tessel.
- `t2 reboot` will reboot your Tessel.
- `t2 restore` Restore your Tessel to factory defaults by installing the factory version of [OpenWrt-Tessel](#), and [T2-Firmware](#).

Code Deployment

During code deployment, [CLI](#) looks for `.tesselignore` and `.tesselinclude` files to let it know which files it should bundle up and push over to Tessel. In the default bundling process, [CLI](#) takes the file passed into the `run` or `push` command and uses it as an entry point to build a dependency graph (similar to [Browserify](#)). Once the dependencies are known, binary modules are replaced by pre-compiled (for mips) binaries, assets are copied, and everything is tarred before sent to the awaiting Tessel.

- `t2 run <file>` copy the file and its dependencies into Tessel's RAM & run immediately. Use this during development of your device application.
 - `[--lan]` deploy over [LAN](#) connection
 - `[--usb]` deploy over USB connection
 - `[--slim]` true by default, copy only files needed by the program to run
 - `[--full]` the opposite of `--slim`, copy all the files in the project directory
 - `[--compress]` true by default, compress and minify the project code prior to copying it to the device

- `--binopts` Arguments sent to the binary (e.g. Node.js, Python). For example, to enable ES modules in Node use `--binopts="--experimental-modules"`
- `t2 push <file>` copy the file and its dependencies into Tessel's Flash memory & run immediately. Once deployed with `push` command, the device application will automatically run every time the Tessel restarts.
 - `--lan` deploy over LAN connection
 - `--usb` deploy over USB connection
 - `--slim` true by default, copy only files needed by the program to run
 - `--full` the opposite of `--slim`, copy all the files in the project directory
 - `--compress` true by default, compress and minify the project code prior to copying it to the device
 - `--binopts` Arguments sent to the binary (e.g. Node.js, Python). For example, to enable ES modules in Node use `--binopts="--experimental-modules"`
- `t2 erase` erase any code pushed using the `t2 push` command

Using Wifi

- `t2 wifi` show details about an existing WiFi connection
 - `-l` lists the available networks
 - `-n SSID` required, connects to the provided SSID
 - `-p PASS` optional, connects with the given password
 - `-s SECURITY` connects with the given security type, valid options:
 - `none` open network, no need for a password
 - `wep` WEP network, password required
 - `psk` WPA Personal, password required
 - `psk2` WPA2 Personal, password required
 - `wpa` WPA Enterprise, password required
 - `wpa2` WPA2 Enterprise, password required
 - `--off` disconnects from the current network
 - `--on` connects to the last configured network

Create An Access Point

- `t2 ap`
 - `-n SSID` required, creates a network with the given ssid
 - `-p PASS` optional for open networks, creates a network with the given password
 - `-s SECURITY` creates a network with the given security, valid options:
 - `none` open network, default if no password is given
 - `wep` WEP network, password required
 - `psk` WPA Personal, password required

- `psk2` WPA2 Personal, recommended, password required & default if password is given without security
- `--off` turn off the current [access point](#)
- `--on` turn on the most recently used [access point](#)

Tessel 2 Hardware API

- [Ports and pins](#)
 - [Modules](#)
 - [Pin mapping](#)
 - [Pull-up and Pull-down pins](#)
 - [Digital pins](#)
 - [Analog pins](#)
 - [Interrupts](#)
 - [PWM pins](#)
 - [Terms used](#)
 - [I2C](#)
 - [SPI](#)
 - [UART/Serial](#)
 - [Power management](#)
- [LEDs](#)
- [Button](#)
- [USB ports](#)

When you `require('tessel')` within a script which is executed on Tessel 2, this loads a library which interfaces with the Tessel 2 hardware, including [pins](#), [ports](#), and LEDs, just like Tessel 1 ([Tessel 1 hardware documentation](#)). The code for Tessel 2's hardware object can be found [here](#).

Ports and pins

Tessel has two [ports](#), A and B. They are referred to as `tessel.port.B` . `tessel.port['B']` is also an acceptable reference style.

Tessel's [ports](#) can be used as module [ports](#) as in Tessel 1 (e.g.

```
accelerometer.use(tessel.port.B) ), or used as flexible GPIO pins (e.g. myPin = tessel.port.A.pin[0] ).
```

Modules

Tessel 2's module [ports](#) can be used with [Tessel modules](#) much as in [Tessel 1](#).

Here is an example of using the Tessel Climate module on Tessel's port B:

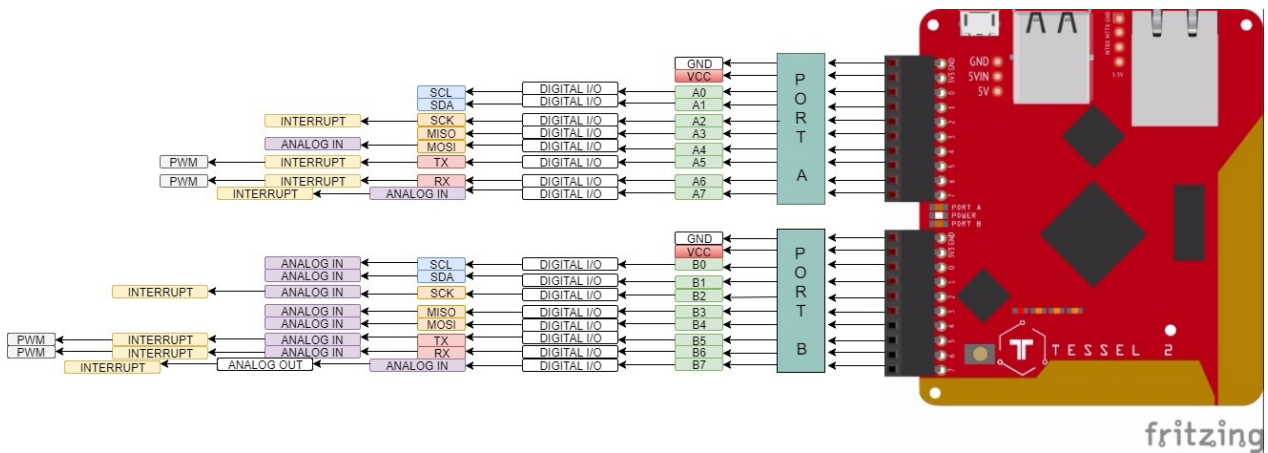
```
var tessel = require('tessel');
var climatelib = require('climate-si7020').use(tessel.port.B);
```

Pin mapping

The module `ports` are not just for modules! They can also be used as flexible, simply addressable General purpose input-output (GPIO) pins. GPIO pins provide access for digital and analog signal lines. Available pins are exposed through the `.pin`, `.digital`, `.analog`, and `.pwm` arrays.

The pin capabilities for ports A and B are as follows:

Port	Pin	Digital I/O	SCL	SDA	SCK	MISO	MOSI	TX	RX	Analog In
A	0	✓	✓							
A	1	✓		✓						
A	2	✓			✓					
A	3	✓				✓				
A	4	✓					✓			✓
A	5	✓						✓		
A	6	✓							✓	
A	7	✓								✓
B	0	✓	✓							✓
B	1	✓		✓						✓
B	2	✓			✓					✓
B	3	✓				✓				✓
B	4	✓					✓			✓
B	5	✓						✓		✓
B	6	✓							✓	✓
B	7	✓								✓



Tessel 2 Pin Diagram

If you're newer to hardware and these functions look like alphabet soup to you, take a look at our [communication protocols documentation](#) to get an idea of how these [pins](#) should be used.

By default, all of the [pins](#) are pulled high if not specifically set.

Pull-up and Pull-down pins

[Pins 2-7](#) on both [Ports](#) are available for [pull-up](#) and [pull-down](#).

The basic function of a [pull-up](#) resistor is to ensure that given no other input, a circuit assumes a default value. The 'pullup' mode pulls the line high and the 'pulldown' mode pulls it low.

Usage

```
// Invoke this method to set the pull mode. pullType can be - pullup, pulldown or none.

pin.pull(pullType, callback);
```

Example :

```
var tessel= require('tessel'); // Import Tessel

var pin = tessel.port.A.pin[2]; // Select pin 2 on port A

pin.pull('pullup', (error, buffer) => {
  if (error) {
    throw error;
  }

  console.log(buffer.toString('hex')); // The output would be 88. Which is an OK Signal from SAMD21
})
```

[Learn more about using the Pullup / Pulldown API.](#)

Digital pins

A [digital](#) pin is either high (on/3.3V) or low (off/0V). Any pin on both [ports](#) A and B, other than 3.3V and GND, can be used as a [digital](#) pin. All [pins](#) are pulled high to 3.3V by default.

pin.write:

Set the [digital](#) value of a pin.

```
pin.write(number, callback(error, buffer));
```

Option	Type	Description	Required
number	Number	0 for "low", 1 for "high"	Yes
callback	Function	Called when write is complete	Yes
error	Error	Contains information if an error occurred, otherwise <code>null</code>	Yes
buffer	Buffer	Value written to the pin	Yes

Example:

```
var tessel = require('tessel'); // Import tessel
var pin = tessel.port.A.pin[2]; // Select pin 2 on port A
pin.write(1, (error, buffer) => {
  if (error) {
    throw error;
  }

  console.log(buffer.toString()); // Log the value written to the pin
});
```

pin.read:

Read the **digital** value of a pin.

```
pin.read(callback(error, number));
```

Option	Type	Description	Required
callback	Function	Called when read is complete	Yes
error	Error	Contains information if an error occurred, otherwise <code>null</code>	Yes
number	Number	1 if "high", 0 if "low"	Yes

Example:

```
var tessel = require('tessel'); // Import tessel
var pin = tessel.port.A.pin[2]; // Select pin 2 on port A
pin.read(function(error, number) {
  if (error) {
    throw error;
  }

  console.log(number); // 1 if "high", 0 if "low"
});
```

pin.toggle:

Toggle the value of a **digital** pin. When the value of a pin is high, then it is set to low, and vice versa.

```
pin.toggle(callback(error, buffer));
```

Option	Type	Description	Required
callback	Function	Called when action is complete	Yes
error	Error	Contains information if an error occurred, otherwise <code>null</code>	Yes
buffer	Buffer	Value written to the pin	Yes

```
var tessel = require('tessel'); // Import tessel
var pin = tessel.port.A.pin[3]; // Select pin 3 on port A
pin.toggle((error, buffer) => {
  if (error) {
    throw error;
  }

  console.log(buffer.toString()); // Log the value written to the pin
});
```

Analog pins

An [analog](#) pin is a pin whose value can vary in the range between 0V and 3.3V. The [analog pin API](#) represents this value as a percentage, between 0 and 1, where 1 stands for 3.3V.

pin.analogWrite:

Set the [analog](#) value of a pin. Enabled on pin 7 of Port B.

```
pin.analogWrite(number);
```

Option	Type	Description	Required
number	Number	Minimum value 0, maximum value 1	Yes

Example:

```
var tessel = require('tessel'); // Import tessel
var pin = tessel.port.B.pin[7]; // Select pin 7 on port B
pin.analogWrite(0.6); // Turn pin to 60% of high voltage
```

pin.analogRead:

Read the [analog](#) value of a pin. Enabled on [pins](#) 4 and 7 on Port A, and all [pins](#) on Port B. On both [ports](#), all [pins](#) are pulled high by default, which means a pin not connected to any sort of sensor or resistor can still return a value over 0.

```
pin.analogRead(callback(error, number));
```

Option	Type	Description	Required
callback	Function	Called when read is complete	Yes
error	Error	Contains information if an error occurred, otherwise <code>null</code>	Yes
number	Number	Minimum value 0, maximum value 1	Yes

Example:

```
var tessel = require('tessel'); // Import tessel
var pin = tessel.port.A.pin[4]; // Select pin 4 on port A
pin.analogRead((error, number) => {
  if (error) {
    throw error;
  }

  console.log(number); // The number is a value between 0 and 1
});
```

Interrupt Pins

[Pins 2, 5, 6, 7](#) on both [Ports](#) are available for interrupts.

Interrupts allow us to [register](#) events based on state changes in a pin.

Usage

```
// Invoke the callback whenever the event occurs
pin.on(eventName, callback);

// Invoke the callback the first time the event occurs
pin.once(eventName, callback);
```

Available Events

Event Name	Description	Notes
rise	Pin voltage rises	
fall	Pin voltage falls	
change	Pin voltage changes	Passes current value of pin to callback
high*	Pin voltage goes high	Only available via pin.once()
low*	Pin voltage goes low	Only available via pin.once()

* Only one of these events may be registered at a time.

[Learn more about using the interrupt API.](#)

PWM pins

Tessel has four **PWM pins**: pins 5 and 6 on each **module port**.

Usage

tessel.pwmFrequency:

Set the signal frequency in hertz. 1 hertz equals 1 cycle of signal per second.

```
tessel.pwmFrequency(number);
```

Option	Type	Description	Required
number	Number	Minimum value 1, maximum value 5000	Yes

Note: the `pwmFrequency` function *must* be called before `pwmDutyCycle`. Re-setting `pwmFrequency` will disable **PWM** output until `pwmDutyCycle` is called again.

pin.pwmDutyCycle:

Set the percentage amount of time the pin is active each cycle of signal.

```
pin.pwmDutyCycle(number);
```

Option	Type	Description	Required
number	Number	Minimum value 0, maximum value 1, e.g. 0.6	Yes

[Learn more about using the PWM API.](#)

Terms Used

The following sections use industry standard technical terms that are considered non-inclusive. They are used here in an explicitly technical manner and only to be accurate in describing these wire [communication protocols](#). The terms are used strictly as defined here:

Master: A machine or device directly controlling another (source: [Oxford Dictionary](#))

Slave: A device, or part of one, directly controlled by another (source: [Oxford Dictionary](#))

I2C

An [I2C](#) channel uses the [SCL](#) and [SDA pins](#) (0 and 1 on Tessel 2). If you are unfamiliar with the [I2C](#) protocol, please see the [communication protocols tutorial](#).

Usage

There are three different commands you can send over [I2C](#): transfer, send, and read.

[i2c.transfer](#)

Write data to the [slave](#) device and then subsequently read data from the [slave](#) device. Can be used to read from a specific [register](#) (depending on the communication details of the [slave](#) device— read the [datasheet!](#)).

A common use of this method is to send the address of a [register](#), then receive back the information contained in that [register](#).

```
var tessel = require('tessel');

// Connect to device
var port = tessel.port.A; // Use the SCL/SDA pins of Port A
var slaveAddress = 0xDE; // Specific to device
var i2c = new port.I2C(slaveAddress); // Initialize I2C communication

// Details of I2C transfer
var bytesToSend = [0xde, 0xad, 0xbe, 0xef]; // An array, can be the address of a register or data to write (depends on device)
var numBytesToRead = 4; // Read back this number of bytes

// Send/receive data over I2C using i2c.transfer
i2c.transfer(new Buffer(bytesToSend), numBytesToRead, function (error, dataReceived) {
  // Print data received (buffer of hex values)
  console.log('Buffer returned by I2C slave device ('+slaveAddress.toString(16)+'):', dataReceived);
});
```

i2c.send

Write data over **I2C** to the **slave** device without reading data back.

```
var tessel = require('tessel');

// Connect to device
var port = tessel.port.A; // Use the SCL/SDA pins of Port A
var slaveAddress = 0xDE; // Specific to device
var i2c = new port.I2C(slaveAddress); // Initialize I2C communication

// Details of I2C transfer
var bytesToSend = [0xde, 0xad, 0xbe, 0xef]; // An array, can be the address of a register or data to write (depends on device)

// Send data over I2C using i2c.send
i2c.send(new Buffer(bytesToSend), function (error) {
  // Print confirmation
  console.log('Wrote', bytesToSend, 'to slave device', slaveAddress.toString(16), 'over I2C.');
```

i2c.read

Read data over **I2C** but do not write to the **slave** device.

```
var tessel = require('tessel');

// Connect to device
var port = tessel.port.A; // Use the SCL/SDA pins of Port A
var slaveAddress = 0xDE; // Specific to device
var i2c = new port.I2C(slaveAddress); // Initialize I2C communication

// Details of I2C transfer
var numBytesToRead = 4; // Read back this number of bytes

// Read data over I2C using i2c.transfer
i2c.read(numBytesToRead, function (error, dataReceived) {
  // Print data received (buffer of hex values)
  console.log('Buffer returned by I2C slave device ('+slaveAddress.toString(16)+'):', dataReceived);
});
```

SPI

A **SPI** channel uses the **SCK**, **MISO**, and **MOSI** pins (2, 3, and 4 on Tessel 2). If you are unfamiliar with the **SPI** protocol, please see the [communication protocols tutorial](#).

Here is an example using Tessel's **SPI** protocol:

```
var port = tessel.port.A;
var spi = new port.SPI({
  clockSpeed: 4*1000*1000, // 4MHz
  polarity: 0, // Polarity - optional
  phase: 0, // Clock phase - optional
  chipSelect: port.pin[7] // Chip select - optional
});

spi.transfer(new Buffer([0xde, 0xad, 0xbe, 0xef]), function (error, dataReceived) {
  console.log('buffer returned by SPI slave:', dataReceived);
});
```

Wikipedia's article on [SPI](#) has an excellent [section](#) on polarity (cpol) and clock phase (cpha).

UART/Serial

A [UART](#) ([serial](#)) channel uses the [TX](#) and [RX pins](#) (5 and 6 on Tessel 2). If you are unfamiliar with the [UART](#) protocol, please see the [communication protocols tutorial](#).

Here is an example using Tessel's [UART](#) protocol:

```
var port = tessel.port.A;
var uart = new port.UART({
  baudrate: 115200
});

uart.write('ahoy hoy\n')
uart.on('data', function (data) {
  console.log('received:', data);
})

// UART objects are streams!
// pipe all incoming data to stdout:
uart.pipe(process.stdout);
```

Power management

Board

tessel.reboot:

Power cycle the board. This will clear RAM and restart the program once reboot is complete. This can be done with the `t2 reboot` command as well.

```
tessel.reboot();
```

Ports

Tessel 2's firmware turns off power to the port if a program does not require the `tessel` module. However, if `tessel` is required in the code, both will be turned on by default. It is possible to control the power to module `ports` with an explicit call:

```
var Tessel = require('tessel-export');

var tessel = new Tessel({
  ports: {
    A: false,
    B: false,
  }
});
```

tessel.open:

Allow power to one or both Tessel `ports`.

```
tessel.open(portName);
```

Option	Type	Description	Required
portName	String	Either 'A' or 'B'	No

Leaving out the port name will turn on power to both `ports`, i.e. `tessel.open()` .

tessel.close:

Disallow power to one or both Tessel `ports`.

```
tessel.close(portName);
```

Option	Type	Description	Required
portName	String	Either 'A' or 'B'	No

Leaving out the port name will turn off power to both `ports`, i.e. `tessel.close()` .

LEDs

There are 4 LEDs available on the Tessel 2, you may see them labeled on the board as `ERR` , `WLAN` , `LED0` , and `LED1` . They are available through the `tessel.led` object.

```
// an array of available LEDs
var leds = tessell.leds;

// ERR - Red
var red = leds[0];

// WLAN - Amber
var amber = leds[1];

// LED0 - Green
var green = leds[2];

// LED1 - Blue
var blue = leds[3];
```

Each **LED** has a few methods for controlling it.

```
// Green LED
var led = tessel.led[2];

/*
 * Property: isOn
 * Returns: boolean (true or false) if led is on
 *
 * Checks the led to see if it is on or not.
 */
if (led.isOn) {
  console.log('The green LED is currently on.');
```



```
/*
 * Method: on
 * Returns: the led object (this makes it a chainable function)
 *
 * Turns the led ON.
 */
led.on();

/*
 * Method: off
 * Returns: the led object (this makes it a chainable function)
 *
 * Turns the led OFF.
 */
led.off();

/*
 * Method: toggle
 * Arguments:
 * - callback: function to call once the led's value has been set and is passed an error object if one occurred
 *
 * Toggles the current state of the led and calls the callback function once that is done.
 */
led.toggle(function (error) {
  if (error) {
    console.log('There was an error with toggling the led.', error);
  } else {
    console.log('The green led is now on OR off!');
  }
});
```

Button

Tessel 2's button is not programmable.

During operation of a script, pressing the button will restart the script.

If you hold down the button while plugging in power, Tessel 2 boots into [DFU](#) mode. This is typically used to load firmware built from source.

USB Ports

USB modules do not need to be accessed through the Tessel object. See [tessel-av](#) for an example USB module.

Tessel 2 Modules

Tessel uses a system of modules to add functionalities to the ecosystem.

A [Tessel module](#) consists of some piece of hardware, matched with a software package which explains and provides [API](#) access to the hardware.

There are three types of modules:

- Tenpin: purchaseable built-for Tessel modules following the 10-pin module format
- USB: any USB device with an [API](#) Tessel can use
- Community: any piece of hardware with an [API](#) Tessel can use, with instructions on how to connect the hardware to Tessel

The best place to find [API](#) documentation for a [Tessel module](#) is through the links on the [modules page](#).

Quick links

- [See all the modules and find their API documentation](#)
- [Learn how to use modules on the start page](#)
- [Metadata about tenpin, USB, and community modules, including compatibility](#)
- [Hardware docs for Tessel's first-party, tenpin modules](#)
- [How to make a module](#)
- [Communication protocols 101](#)

Module design philosophy

Modules should be devices with clear-cut functionality. That is to say, they should have a single, well-defined purpose or a set of closely related functions, rather than an eclectic mix of capabilities onboard. This requirement is designed to reduce complexity, cost, and power consumption and maximize reusability in hardware and software.

Tessel 2 Network API

- [Wifi](#)
- [Access Point](#)

When you `require('tessel')` within a script which is executed on Tessel 2, this loads a library which interfaces with the Tessel 2 hardware, including the wireless radios. The code for Tessel 2's hardware object can be found [here](#).

While [the cli](#) can be used to configure the Tessel to connect to local wifi networks and to create a custom wireless network, the `tessel` package, via the Network [API](#), can be used in the same way. This functionality can be useful for programs that want to create on-demand access points or connect to a local network chosen by someone interacting with a web app served by a Tessel. Having some control of the networking abilities make Tessel a powerful, web-enabled device.

Wifi

The following methods and properties are available through `tessel.network.wifi`.

Methods

- `connect`
- `connection`
- `disable`
- `enable`
- `findAvailableNetworks`
- `reset`

connect

Attempts to connect to a local wifi network using the passed-in configuration object.

```
tessel.network.wifi.connect(config, callback);
```

argument	type	description	required
config	Object	{ ssid, password, security } configuration used to connect to local network	yes
callback	Function	called when done connecting or if an error occurred	no

Example

```
tessel.network.wifi.connect({
  ssid: 'WifiNetworkName', // required
  password: 'WifiNetworkPassword', // required if network is password-protected
  security: 'psk2' // available values - none, wep, psk, psk2, wpa, wpa2, default is '
  none' if no password needed, default is 'psk2' otherwise. See https://tessel.io/docs/cli#usage for more info
}, function (error, settings) {
  if (error) {
    // handle error if it exists
  }

  console.log(settings); // object containing ssid, security, IP address
});
```

connection

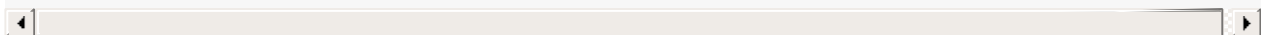
Returns the settings for the current connection, includes ssid (network name), security, and IP address.

```
tessel.network.wifi.connection(callback);
```

Example

```
tessel.network.wifi.connection(function(error, settings) {
  if (error) {
    throw error;
  }

  console.log(settings.ssid); // logs the name of the wifi network being used by Tessel
});
```



disable

Disables Tessel's connection to wifi.

```
tessel.network.wifi.disable(callback);
```

argument	type	description	required
callback	Function	called when done disabling or if an error occurred	no

Example

```
tessel.network.wifi.disable(function (error) {  
  if (error) {  
    // handle error if it exists  
  }  
  
  // Tessel is now disconnected from wifi  
});
```

enable

Enables Tessel's connection to wifi.

```
tessel.network.wifi.enable(callback);
```

argument	type	description	required
callback	Function	called when done enabling or if an error occurred	no

Example

```
tessel.network.wifi.enable(function (error) {  
  if (error) {  
    // handle error if it exists  
  }  
  
  // Tessel is now connected to wifi  
});
```

findAvailableNetworks

Scans for local wifi networks available to Tessel.

```
tessel.network.wifi.findAvailableNetworks(callback);
```

argument	type	description	required
callback	Function	called when done scanning or if an error occurred	no

Example

```
tessel.network.wifi.findAvailableNetworks(function (error, networks) {  
  if (error) {  
    // handle error if it exists  
  }  
  
  console.log(networks) // 'networks' is an array of objects containing { ssid, security, quality }, quality is a string comparing the signal strength out of 70 - '49/70'  
});
```

reset

Resets the Tessel's wifi connection, first disabling, then enabling the connection.

```
tessel.network.wifi.reset();
```

argument	type	description	required
callback	Function	called when Tessel is done resetting	no

Example

```
tessel.network.wifi.reset(function (error) {  
  if (error) {  
    // handle error if it exists  
  }  
  
  // Tessel's wifi connection has been successfully reset  
});
```

Events

The `Wifi` Class inherits from the Node.js `EventEmitter` Class, so methods such as `on`, `once`, and `removeListener` can be used with the following events:

- `connect` : this event is emitted when Tessel has successfully **connected** to a wifi network
- `disconnect` : this event is emitted when Tessel has successfully **disconnected** from a wifi network
- `error` : this event is emitted when an error occurs during any of the wifi functions

Example

```
tessel.network.wifi.on('connect', function (settings) {  
  // the 'connect' event has been called and happens to return a 'settings' object  
});
```

Access Point

The following methods and properties are available through `tessel.network.ap` .

Methods

- `create`
- `disable`
- `enable`
- `reset`

create

Attempts to create a custom [access point](#) using the passed-in configuration object.

```
tessel.network.ap.create(config, callback);
```

argument	type	description	required
config	Object	{ ssid, password, security } configuration used to create a custom access point	yes
callback	Function	called when done creating or if an error occurred	no

Example

```
tessel.network.ap.create({
  ssid: 'AccessPointName', // required
  password: 'CustomSecurePassword', // required if network is password-protected
  security: 'psk2' // available values - none, wep, psk, psk2, default is 'none' if no
  password needed, default is 'psk2' otherwise. See https://tessel.io/docs/cli#usage fo
  r more info
}, function (error, settings) {
  if (error) {
    // handle error if it exists
  }

  console.log(settings); // object containing ssid, security, IP address
});
```

disable

Disables the created [access point](#).

```
tessel.network.ap.disable(callback);
```

argument	type	description	required
callback	Function	called when done disabling or if an error occurred	no

Example

```
tessel.network.ap.disable(function (error) {
  if (error) {
    // handle error if it exists
  }

  // the access point has been successfully disabled
});
```

enable

Enables a previously configured [access point](#).

```
tessel.network.ap.enable(callback);
```

argument	type	description	required
callback	Function	called when done enabling or if an error occurred	no

Example

```
tessel.network.ap.enable(function (error) {  
  if (error) {  
    // handle error if it exists  
  }  
  
  // the access point has been successfully enabled  
});
```

reset

Resets the active [access point](#), first disabling, then enabling the network.

```
tessel.network.ap.reset();
```

argument	type	description	required
callback	Function	called when Tessel is done resetting	no

Example

```
tessel.network.ap.reset(function (error) {  
  if (error) {  
    // handle error if it exists  
  }  
  
  // Tessel's wifi connection has been successfully reset  
});
```

Events

The `AP` Class inherits from the Node.js `EventEmitter` Class, so methods such as `on`, `once`, and `removeListener` can be used with the following events:

- `create`: this event is emitted when an [access point](#) is successfully created, includes `settings` object in the callback
- `disable`: this event is emitted when an [access point](#) is successfully disabled

- `error` : this event is emitted when an error occurs during any of the [access point](#) functions
- `enable` : this event is emitted when an [access point](#) is successfully enabled
- `reset` : this event is emitted when an [access point](#) is preparing to reset

Example

```
tessel.network.ap.on('create', function (settings) {  
  // the 'create' event has been called and happens to return a 'settings' object  
});
```

Supported languages

JavaScript/Node.js

Tessel 2 has full support for JavaScript and Node.js (LTS versions). The relevant repos can be found here:

- [Firmware support for JavaScript/Node](#)
- [CLI support for JavaScript/Node](#)

Binary modules

There is support for precompiled binary modules on Tessel 2. The best way to find out whether the module you want is available is to try deploying it. The module has been precompiled, it will just work!

For modules that have not been precompiled (you can see the list at packages.tessel.io), you will see an error message explaining why it cannot be loaded:

```
Pre-compiled module is missing: {name of module}.
This might be caused by any of the following:
1. The binary is platform specific and cannot be compiled for OpenWRT.
2. A pre-compiled binary has not yet been generated for this module.
3. The binary didn't compile correctly for the platform that you're developing on.
    It's possible that the binary is Linux-only or even OpenWRT specific,
    try npm installing with "--force" and rerun your deployment command.
Please file an issue at https://github.com/tessel/t2-cli/issues/new
```

Submit an issue and we will look into precompiling it. Our precompilation server lives in the [t2-compiler](#) [repo](#).

Rust (work in progress)

The Tessel team is working toward first-class support for Rust. If you're interested in the state of that project, check out the [tessel-rust](#) [repo](#).

How to add support for more languages

Interested in adding support for a new language to Tessel 2's [CLI](#)? Here is a detailed blog post on the six components you will need: [Interfacing with the Language Plugin API for Tessel 2](#)

(It is worth noting that [support of Python for Tessel 2](#) was originally planned, but is no longer in active development. [Read the blog post](#) explaining this decision.)

Debugging LAN Discovery

Introduction to MDNS

The Tessel 2 [CLI](#) is able to detect Tessel 2s on the same wireless network as your host computer using a protocol called [mDNS](#). If you run `t2 list` and don't see your Tessel labeled [LAN](#) like the example below, then either the Tessel is not advertising properly, your wifi network infrastructure blocks [mDNS](#) traffic, or your computer is not picking it up properly.

```
→ t2 list
INFO Searching for nearby Tessels...
Frank LAN
```

An issue with detection

First, you should confirm your host machine is on the same wifi network as your Tessel. You can run `t2 wifi` to get the network your Tessel 2 is connected to. If it's not connected, you'll need to do that and run `t2 list` again.

Next, let's check if your computer is able to detect the Tessel at all (to determine if it's an issue with the [CLI](#) or with the Tessel). We'll use a utility that we absolutely know works.

I only have an example for OSX - if you know how to do [mDNS](#) discovery on Windows, please add to this file and submit a PR!

OSX

```
dns-sd -B _tessel._tcp .
```

This tells the `dns-sd` (DNS Service Discovery) tool to start searching for any devices advertising as a `_tessel._tcp` device. If your device doesn't show up in the results, then it is an issue with detection, then your computer isn't detecting it up properly.

There is also issue a peculiar but uncommon issue where OSX just stops picking up the advertisements for some reason. The cause is unknown thus far but you can fix it with:

```
sudo killall -HUP mDNSResponder
```

This command basically clears the cache of [mDNS](#) devices so the next time you run the `dns-sd` utility or `t2 list`, Tessel will show up (if this was the root cause).

Linux

```
avahi-browse _tessel._tcp -t
```

This displays a list of DNS-SD entries that match `_tessel._tcp` and quits afterwards.

An issue with network configuration

Attempts to connect to your Tessel 2 via a network that is configured to block [mDNS](#) traffic will always fail. This type of configuration is often used for secured company networks and open public networks to prevent unwanted peer-to-peer discovery and access. If this is your home network, change the configuration via the router's administration control panel. If this is a network that you do not control, you may have to request the change from an IT department (or similar resource).

An issue with advertisement

To determine if it's an issue with Tessel not advertising properly, you'll need to get [root access](#) to your Tessel. Then run `ps` to check if the [mDNS](#) service is actually running on Tessel:

```
1259 root      904 S      dns-sd -R Frank _tessel._tcp local 22
```

If you don't see this service running (note that the name of your Tessel will be different), you can restart it with:

```
/etc/init.d/tessel-mdns restart
```

Once it's restarted, check if it's running with the `dns-sd` utility on your host machine or `t2 list`.

Node.js

Installation instructions for the Tessel [CLI](http://tessel.github.io/t2-start/) can be found here: <http://tessel.github.io/t2-start/>

This document will address common problems related to Node.js usage.

NVM

The official installation instructions assume Node is installed globally. For those of you using NVM this will not be the case. As indicated [here](#) NVM does not install globally by design.

Linux

Install `t2-cli` module as normal. To install the drivers, perform the following:

Find where the `node` executable is located:

```
$ which node
/home/username/.nvm/versions/node/v4.5.0/bin/node
```

In the example above, the path is `/home/username/.nvm/versions/node/v4.5.0/bin` (referred to as `$nodePath` below).

Install the drivers.

```
$ sudo PATH="$PATH:$nodePath" $nodePath/t2 install-drivers
```

Accessing root on Tessel 2

If you wish to access root on Tessel 2 and explore its Linux filesystem, you have a few options.

The simplest option is `t2 root`, which gives SSH access to a Tessel 2 you are authorized with.

However, while you're building tools for Tessel, you may need to access Tessel in ways not exposed by the [CLI](#). Here are two options:

SSH

If you can connect to your Tessel 2 over [LAN](#), SSH or `t2 root` are your best options.

Setup

Make sure your Tessel is [online](#). Check that it shows up on [LAN](#) with `t2 list`.

Run `t2 provision` to authorize your computer with your Tessel.

SSH in

Use the command `ssh root@<tesselname>.local -i ~/.tessel/id_rsa`.

The `-i` command lets you specify the filepath to the RSA keys written by `t2 provision`.

Getting out

When you need out, type `exit`.

dterm

Dterm lets you talk to Tessel over USB.

Setup

OS X

Unfortunately, dterm isn't supported on El Capitan and later. You may be able to use [screen](#) instead.

Install dterm with `brew install dterm` .

Figure out what USB port your Tessel is plugged into by typing `ls /dev/tty.usbmodem` – see whether it auto-completes to e.g. `/dev/tty.usbmodem1412` or a different number.

Linux

Download and build dterm from <http://www.knossos.net.nz/resources/free-software/dterm/>, or use screen, which comes with most Linux distros.

The path to the [serial](#) port is `/dev/ttyACM0` or `/dev/serial/by-id/usb-Technical_Machine_Tessel_2_<serial number>-if01` - find out by typing `ls` for those locations.

Windows

Not sure, sorry... if you know how to access [serial](#) on Windows, please PR.

dterm in

Run `dterm /dev/tty.usbmodem1412` (or a different port number). You will have to press enter twice.

If you're going to do this a lot, it might be a good idea to alias the command: `alias v2="dterm /dev/tty.usbmodem1412"` .

If you need your ip address (for example if you'd like to ssh in next time), you can run `ifconfig` to get it.

Getting out

`CTRL+]`

then

`CTRL+C`

Tessel 2 Technical Overview

This guide is intended to help code contributors understand how relevant system components of Tessel 2 work, where to find code and design files, and accelerate the development process through technical transparency.

- [T2 Github Repositories](#)
 - [Core](#)
 - [Command Line Interface](#)
 - [OpenWRT OS](#)
 - [Co-processor Firmware](#)
 - [Tools](#)
 - [Virtual Machine](#)
 - [Binary Compiler](#)
 - [Hardware Designs](#)
 - [Tessel 2 Hardware Design Files](#)
 - [Tessel 2 Parts Library](#)
 - [Modules](#)
 - [10-Pin Modules](#)
 - [USB Modules](#)
- [System Architecture](#)
 - [Hardware Overview](#)
 - [Software Overview](#)
- [Code Deploy Walkthrough](#)
- [Getting Questions Answered](#)
- [Links to more information](#)

Relevant Github Repositories

Quick guide to the Tessel stack (in order of complexity):

Level	Stack	Repo
User / Module Code	JS	(Many)
CLI	JS	t2-cli
Tessel Library	Linux / JS / Comms	t2-firmware
OpenWRT	Linux / C / Shell / More	t2-firmware
SSH/ GPIO Daemons	Linux / C	t2-firmware
Firmware	C / Comms	t2-firmware

Tessel 2 Core Repos:

You can find all of the Tessel repositories [on the organization page](#) but this section provides some guidance over relevant Tessel 2-specific repositories.

[Tessel 2 Command Line Interface \(CLI\)](#)

The [CLI](#) is the primary method of interacting with Tessel for users and developers. It provides useful functions like listing available Tessels, deploying code, and setting wifi credentials on the device.

[Tessel 2 Operation System \(OpenWRT\)](#)

The primary processor of the Tessel 2 runs a very lightweight version of Linux called OpenWRT. OpenWRT provides all of the TCP/IP drivers, threading/schedule support, and runs Node, Rust or whatever other language you're using with Tessel 2. This repo contains all of the source files of the Tessel build of OpenWRT.

[Tessel 2 Co-processor Firmware](#)

Tessel 2 features a SAMD21 co-processor that manages a handful of responsibilities like routing USB communication from the [CLI](#) and the realtime operations on the [GPIO pins](#) of the two module [ports](#). This repo contains not only the firmware that drives this microcontroller but hardware-related scripts that define the interface to other languages (like [tessel.js](#) for Node scripts) so that the available hardware functionality doesn't get out of sync with what is exposed to the OpenWRT processor.

Tessel 2 Tools

[Tessel 2 Virtual Machine \(VM\)](#)

The VM is used primarily by Tessel developers who either don't have hardware available or want to develop on a faster computer. The repo provides the ability to create and run these Tessel virtual machines. You can use the [CLI](#) (see above) to interact with the running VM just as you would an actual Tessel 2. The VM has the limitation of not being able to set wifi credentials (it passes through the host network interface). It is not able to use 10-pin hardware modules but USB devices do get passed through.

Tessel 2 Compiler

The Tessel 2 compiler is another virtual machine with all the build tools needed to develop [native add-ons](#) to Node modules. These add-ons are built to perform faster or interact with lower level hardware than JS alone would provide. The compiler is being used, for example, to develop the [audio-video Node module](#) for webcams and recording audio from microphones.

Hardware Designs

Tessel 2 Hardware Design Files

These are the production schematic and assembly files for the Tessel 2 hardware. You can find information about the parts that are used and how they are all connected to each other. These files were created with [KiCad](#), a completely open source electronics design tool.

Tessel Parts Library

This repo contains all of the KiCad part models for all Tessel hardware (not just Tessel 2). You will need this library in order to recreate the schematic and PCB of the hardware design files above.

Modules

10-Pin Modules

There are 9 compatible 10-pin hardware modules for Tessel 2 available at the time of this writing. Each of these modules have a repo that contains the JavaScript driver. These repos can also be found on NPM under their respective Node modules names.

- [Ambient \(Light & Sound\) Module](#)
- [RFID/NFC Module](#)
- [Accelerometer Module](#)
- [Bluetooth Low Energy Module](#)
- [Climate \(temperature & humidity\) Module](#)
- [GPS Module](#)

- [Infrared Module](#)
- [Relay Module](#)
- [Servo Module](#)

USB Modules

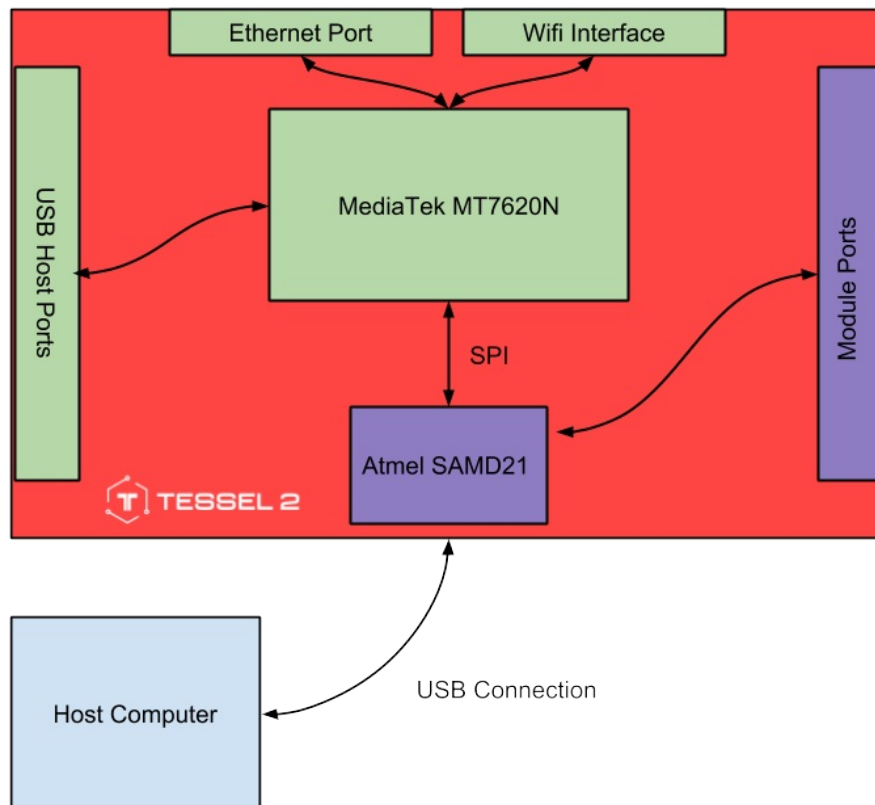
Tessel supports interfacing with USB devices. Here are libraries we support for Tessel to interoperate with USB modules, available from many vendors:

- [Bluetooth Low Energy Dongles](#)
- Cellular (3G/4G) Dongles (no Github module, it just works through the Node `http` module and similar for other languages)
- [Audio Recording/Playback](#)
- [Video Cameras \(specifically, webcams\)](#)
- Flash Storage Devices (no Github module, it just works through the Node `fs` module and similar for other languages)

System Architecture

High Level Look At Hardware

Below is a high level diagram of the hardware architecture:



As you can see from the diagram, the Atmel co-processor manages the USB connection and can communicate with the MediaTek through a [SPI](#) bus. The MediaTek can execute user scripts, and when it interprets hardware commands (like pulling a [GPIO](#) high or sending data over [I2C](#)), it packages it up and sends it over to the Atmel over a pre-defined protocol on that [SPI](#) bus.

Software Architecture

The MediaTek is responsible for managing the two USB Host [ports](#), the ethernet port, and the WiFi network interface directly from the OpenWRT Operating System.

Let's take a look at the software architecture of Tessel 2: ☐

The USB Port on the Atmel actually has three interfaces to the to the MediaTek:

1. A [UART](#) connection that acts a [serial](#) terminal to the MediaTek. You can use a program like `dterm` to access the console through this connection.
2. A direct connection to the [SPI](#) Flash bus of the MediaTek. This allows the Atmel to rewrite the OpenWRT image and helps prevent the Tessel from becoming bricked by any corruptions to OS.

3. A [SPI](#) bus connection to the MediaTek for arbitrary data.

A Code Deploy Example

A good method of understanding all the components of the system is to go through an example of running a simple script on Tessel 2:

The user code to light up an LED

The user might have code like the example below to light up [LED 0](#) on Tessel 2.

```
var tessel = require('tessel');
tessel.led[0].high();
```

Deploy with the CLI

First, the user will use the [command line interface](#) to deploy the script from the host computer:

```
t2 run index.js
```

The [CLI](#) will compress the code and break this simple `run` command into a [number of bash commands](#) required to deploy the code like [untarring the code](#) and [running it with Node](#).

Accepting USB Communication On The Atmel

The utf-8 encoding of each command is sent over the USB connection via a [USB pipe](#). The USB pipe driver hands off the data to the general purpose [SPI bridge](#) (labeled as "[SPI SLAVE SERCOMMx](#)" in the diagram above) which transfers the command to the MediaTek.

Relaying Data to the MediaTek SPI Daemon

On the MediaTek, the [SPI Daemon](#) is constantly waiting for data over the [SPI](#) bus and checks whether this data is from one of the module [ports](#) (for example, if an [analog](#) voltage was previously requested and was now returning the result) or from the USB port.

The [shell script](#) that starts up the [SPI Daemon](#) on boot also creates three unix domain sockets: one for [Module Port A](#) (`/var/run/tessel/port_a`), one for [Module Port B](#) (`/var/run/tessel/port_b`), and one for USB (`/var/run/tessel/usb`). We'll get to the two

module [ports](#) in a moment, but first, the data from the [CLI](#) is passed into the USB domain socket.

One More Relay Into The USB Daemon

Waiting on the other end of the domain socket is [the USB Daemon](#). This program has the thrilling task of accepting shell commands, initializing new threads to execute them, and routing stdin, stdout, and stderr of those threads back to the [CLI](#).

So by this point, the [CLI](#) has executed [a handful of bash commands](#) that transferred the code onto the Tessel 2 and started a Node process to run it. The next question is how we actually do anything with the hardware.

Accessing Hardware Through `tessel.js`

For JavaScript files, the answer comes from the `var tessel = require('tessel')` line of the user script. When Node executes this on the MediaTek, it finds the `tessel` module in `/usr/lib/node/tessel.js`. When this file is executed, [it connects to a domain socket for each module port](#): two of the three domain sockets created by the [SPI Daemon](#) earlier! That means that whenever you run a command on a port like `tessel.led[0].high()`, it's [packaging that command into a array of bytes](#) and sending it back over the [SPI](#) bus to the Atmel bridge firmware! That firmware then parses the command as pulling the specific [GPIO](#) high, and executes it. It can then return the result of that operation back through the [SPI](#) bus to the MediaTek [SPI Daemon](#).

Why Domain Sockets?

The beauty of this architecture with domain sockets is that most legitimate programming languages have the capability to utilize domain sockets. This is how Tessel 2 can be used by any language, not just JavaScript (the code that packages up the commands into array bytes just has to be ported to that language first).

How Does Deploying Code over LAN Work Into This?

In the case of a [LAN](#) connection over WiFi or Ethernet, the procedure is much the same except all of the shell commands are executed over SSH. All of the [module port](#) communication between the [SPI Daemon](#) and the USB Bridge firmware is entirely the same. That's why the [CLI](#) is able to abstract out USB and [LAN](#) interfaces [into a single Connection object](#) that simply accepts shell commands and returns a process object with the three streams (stdin, stdout, stderr) regardless of physical connection.

What is UCI/ubus on the diagram above?

UCI and ubus are unique features of OpenWRT that essentially let you access system configuration programmatically and receive results in a JSON format. It's very handy for the command line interface. You can see that the CLI makes use of these features heavily, especially for wifi related settings.

Questions?

Ask us on [Slack](#) (join [here](#) if you haven't already) or ask on [the forums](#).

More Useful Links

- [Current CLI Development Roadmap](#)
- [OpenWRT website](#)
- [MediaTek MT7620 datasheet](#)
- [Atmel SAMD21 datasheet](#)
- [Tessel Project Blog](#)
- [Getting Started with Node](#)

Debugging USB Communications

Introduction to USB

For an in-depth description of how USB works, read [about a code deploy example](#).

Otherwise, the TL;DR is that USB communication from the [CLI](#) gets sent to the atmel samd21 coprocessor which then routes the packets over [SPI](#) to a daemon running on OpenWRT which then passes it on to another daemon whose only job is to create processes requested over USB and pipe the standard streams back to it. If that description caused your eyes to glaze over, that's okay because the only thing you need to take out of that is that for USB comms to be working, you need:

- The coprocessor to be powered and have valid firmware
- The [SPI](#) Daemon process running on OpenWRT
- The USB Daemon process running on OpenWRT.

The coprocessor should pretty much never be a problem unless you're tinkering with the firmware or the hardware. The [SPI](#) Daemon is relatively simple and has been well battle tested so that's unlikely to be broken. The USB Daemon is very often the culprit for USB issues.

How to debug?

The easiest way to debug USB communication issues is to check out the status of the USB and [SPI](#) daemons on OpenWRT. To do that, you need to get a root shell. Check out this [short tutorial](#) for getting a root shell over SSH or USB. I recommend using SSH because some of the programs used to route [serial](#) traffic over dterm is shared with the USB communications from the [CLI](#) so if you're already having a problem with USB comms, dterm may not work due to the same cause.

Once you have a root shell, look for the following two processes by running `ps` :

```
1169 root      772 S      spid /dev/spidev32766.1 2 1 /var/run/tessel
1185 root      780 S      usbexecd /var/run/tessel/usb
```

The top line is the [SPI](#) Daemon and the bottom line is the USB Daemon. If neither of them are running, you will have an issue. To start (or restart) one or both of them, run:

```
/etc/init.d/spid restart
/etc/init.d/usbexecd restart
```

Once you restart them, you can try talking to Tessel from your host machine to see if that resolves the issue (`t2 list` is a good litmus test).

Hardware documentation for Tessel modules

Scope

This document provides hardware overviews for each [Tessel module](#):

- [Accelerometer](#)
- [Ambient](#)
- [Climate](#)
- [GPS](#)
- [Infrared](#)
- [Relay](#)
- [RFID](#)
- [Servo](#)

Accelerometer

Measures acceleration in the x, y, and z directions.

Quick overview:

Parameter	Value
Tessel part #	TM-01-XX
Latest version	TM-01-02
Key components	MMA8452Q
Current consumption (rated max)	165 microamps
Current consumption (average)	165 microamps
Communication protocol	I2C , GPIO

Notes

- [I2C](#) address can be modified by shorting J3 – see [MMA8452Q datasheet](#) 5.11.1 Table 10.
- +x = towards the Tessel, +y = G3 --> GND, +z = up

Links to Hardware Information:

- [Code: firmware, docs, examples](#)
- [Schematic\(PDF\)](#)
- [Schematic \(DipTrace\)](#)
- [Layout \(DipTrace\)](#)
- [Layout support file for silkscreen](#)

Ambient

Measures ambient light and sound. Sound gain adjustable via R6.

Quick overview:

Parameter	Value
TM part #	TM-08-XX
Latest version	TM-08-03
Key components	PT15-21C/TR8 photodiode, SPU0410HR5H-PB microphone
Current consumption (rated max)	10 mA
Current consumption (average)	10 mA
Communication protocol	SPI , GPIO

Notes

- Light signal path includes low pass filters with $f_c = 1.6$ Hz
- Sound signal path measures loudness via a half wave rectifier

Links to Hardware Information:

- [Code: firmware, docs, examples](#)
- [Schematic\(PDF\)](#)
- [Schematic \(DipTrace\)](#)
- [Layout \(DipTrace\)](#)
- [Layout support file for silkscreen](#)

Climate

Measure temperature and humidity

Quick overview:

Parameter	Value
TM part #	TM-02-XX
Latest version	TM-02-02
Key components	SI7020-A20 sensor
Current consumption (rated max)	565 microamps
Current consumption (average)	320 microamps
Communication protocol	I2C , GPIO

Notes

Temperature of the climate module can be affected by proximity to the Tessel processor.

Links to Hardware Information:

- [Code: firmware, docs, examples](#)
- [Schematic\(PDF\)](#)
- [Schematic \(DipTrace\)](#)
- [Layout \(DipTrace\)](#)
- [Layout support file for silkscreen](#)

GPS

Global positioning system: position reference, movement speed & heading, time reference

Quick overview:

Parameter	Value
TM part #	TM-09-XX
Latest version	TM-09-02
Key components	A2235-H GPS module (user manual , product page)
Current consumption (rated max)	42 mA
Current consumption (average)	36 mA
Communication protocol	UART , GPIO

Notes

- The A2235-H is based on CSR SiRFstarIV chipset, which supports a novel GPS fix mode and a variety of low power states.
- The A2235-H supports two different data modes: NMEA and SiRF binary mode (also called OSP mode). Tessel GPS's default APIs use the module in NMEA mode.
- A-GPS is supported with client generated extended ephemeris up to 3 days.

Links to Hardware Information:

- [Code: firmware, docs, examples](#)
- [Schematic\(PDF\)](#)
- [Schematic \(DipTrace\)](#)
- [Layout \(DipTrace\)](#)
- [Layout support file for silkscreen](#)

Infrared

Communicate with and command household electronics using IR light

Quick overview:

Parameter	Value
TM part #	TM-11-XX
Latest version	TM-11-03
Key components	TSOP38238 38 kHz IR demodulator, SFH-4646-Z IR LED
Current consumption (rated max)	95 mA
Current consumption (average)	5 mA
Communication protocol	UART , GPIO

Notes

- Receiving IR takes very little power. Transmission, on the other hand, uses a good deal of power.
- Receiver frequency is fixed at 38 kHz, although pin-compatible parts which operate at other frequencies presumably exist. Check your appliance's manual and/or the internet to make sure your hardware is compatible.

Links to Hardware Information:

- [Code: firmware, docs, examples](#)
- [Schematic\(PDF\)](#)
- [Schematic \(DipTrace\)](#)
- [Layout \(DipTrace\)](#)
- [Layout support file for silkscreen](#)

Relay

Switch large, high voltage loads

Quick overview:

Parameter	Value
TM part #	TM-04-XX
Latest version	TM-04-04
Key components	IM02DGR (DigiKey page)
Current consumption (rated max)	90 mA
Current consumption (average)	10 mA
Communication protocol	GPIO

Notes

- Relays are normally open (load disconnected) and will open should the module/Tessel lose power
- Current draw increases when the relays close ($I_{draw} = 10\text{mA} + 40\text{mA} * [\text{number of relays on}]$)
- Rated to 240 V, 5 A (but please be careful)
- Insert and remove wires by pressing down on the connectors with a ballpoint pen (or similar).

Links to Hardware Information:

- [Code: firmware, docs, examples](#)
- [Schematic\(PDF\)](#)
- [Schematic \(DipTrace\)](#)
- [Layout \(DipTrace\)](#)
- [Layout support file for silkscreen](#)

RFID

Read and write from/to 13.56 MHz RFID cards and tags

Quick overview:

Parameter	Value
TM part #	TM-10-XX
Latest version	TM-10-03
Key components	PN532 (datasheet , user manual , MiFARE tag IC/protocol docs) 13.56 MHz RFID transceiver
Current consumption (rated max)	150 mA
Current consumption (average)	60 mA
Communication protocol	I2C , GPIO

Notes

- Module can be switched to communicate over [SPI](#) with appropriate resistor pop option swaps
- Current is highest when transmitting and much lower most of the time
- A MiFare Classic card ships with the module
- Hardware-compatible with [ISO-14443 RFID tags](#), including but not limited to: MiFare (Classic 1k, Classic 4k, Ultralight), DesFire, and FeliCa. Further information in the link.
- Compatible cards (consumer): Charlie, Clipper

Links to Hardware Information:

- [Code: firmware, docs, examples](#)
- [Schematic\(PDF\)](#)
- [Schematic \(DipTrace\)](#)
- [Layout \(DipTrace\)](#)
- [Layout support file for silkscreen](#)

Servo

Control up to 16 servos

Quick overview:

Parameter	Value
TM part #	TM-03-XX
Latest version	TM-03-02
Key components	PCA9685 LED driver
Current consumption (rated max)	50 mA
Current consumption (average)	10 mA
Communication protocol	I2C , GPIO

Notes

- An [I2C LED PWM](#) driver makes for a great servo driver too!
- 16 channels
- Tessel is a 3.3V device, so V_{out} on the [PWM](#) channels is 3.3 V (works for standard hobby servos)
- 5.5 mm barrel jack (center positive) for powering servos
- Can be used to control anything that needs [PWM](#) (speed controllers, gate drivers-to-FETs-big LEDs, etc.)

Links to Hardware Information:

- [Code: firmware, docs, examples](#)
- [Schematic\(PDF\)](#)
- [Schematic \(DipTrace\)](#)
- [Layout \(DipTrace\)](#)
- [Layout support file for silkscreen](#)

Hardware Overview of Tessel 2

Quick specs

At a glance, the main hardware components of Tessel 2 are:

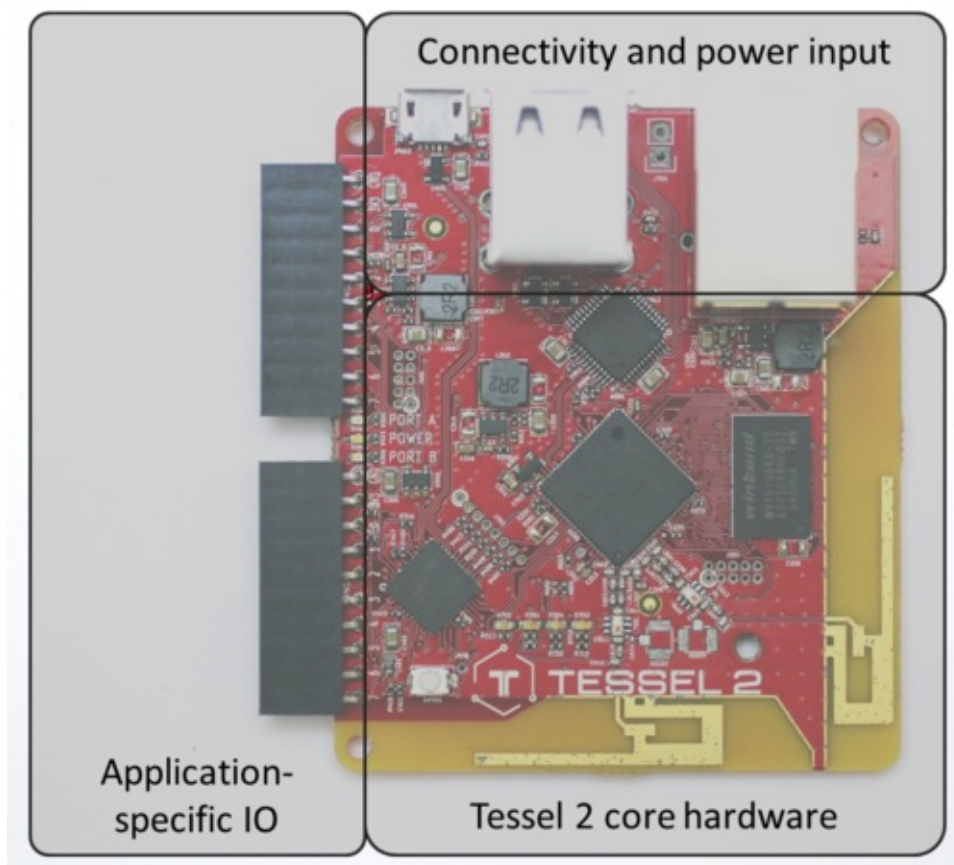
- 580MHz WiFi router system on chip ([Mediatek MT7620n](#)) running linux ([OpenWRT](#))
- 64 MB of DDR2 RAM
- 32 MB of flash storage
- 2 High-speed USB 2.0 [ports](#)
- micro USB port
- 10/100 Ethernet port (RJ-45 jack)
- 48MHz ARM Cortex M0 microcontroller ([Atmel SAMD21](#))
- Two flexible module [ports](#)
- 4 LEDs
- Reset button

Above-and-beyond features:

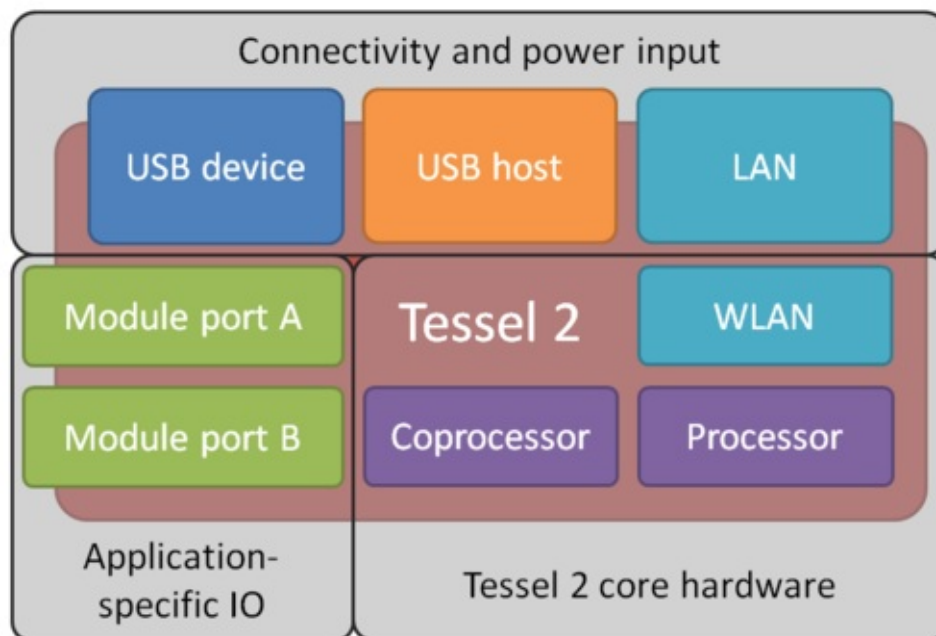
- Router-grade 802.11b/g/n WiFi (2.4 GHz), including [access point](#) mode ([Tessel can be a router](#))
- 16 [GPIO](#) broken out as a pair of multi-purpose module [ports](#)
- Individual control over and protection for all outward-facing power buses (USB and module [ports](#))
- A form factor designed for abstraction and flexibility in the hardware, software, and mechanical worlds as you scale from prototype to production

Tessel 2 Layout

On the board, Tessel 2 is divided into three functional blocks:



These can be abstracted as follows:



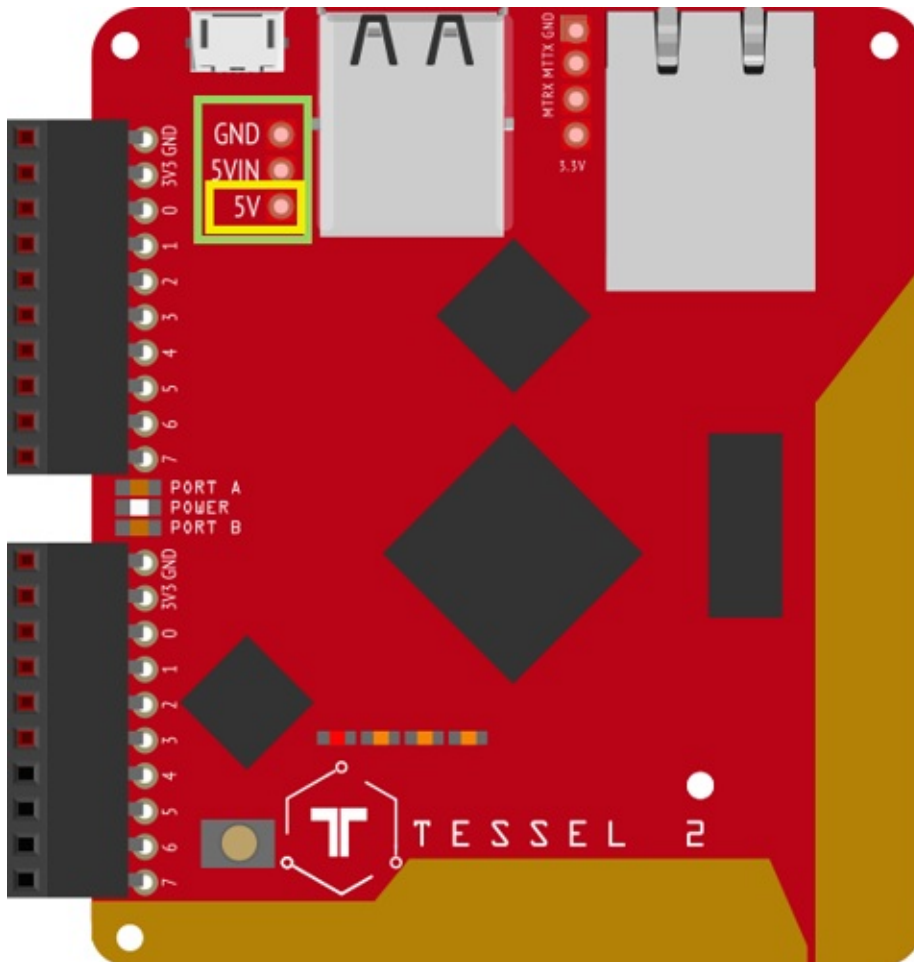
Abstraction boundaries of Tessel 2

The board employs a processor/coprocessor architecture. The Mediatek runs your user code, hosts USB devices, handles the network connections (be they wired, wireless, or cellular over USB), and communicates with the SAMD21.

The SAM acts as a coprocessor and handles real-time, low-level IO through the module [ports](#), USB comms through the Micro USB port, and programming the device as a whole.

The two chips are connected by a [SPI](#) bridge that also includes the onboard flash (the [readme for Tessel 2's firmware repo goes into more detail here](#)).

The whole system is powered from the single micro USB (device) port. Alternately, power can be supplied over the 5VIN pin:



The additional 5V pin shown in the yellow box is a source of 5V power.

Power architecture of Tessel 2

The below table explains how the major components of Tessel 2 are powered.

Item	Powered by	Enabled or reset by
5V	USB cable	Presence of USB cable with upstream power
3.3V Mediatek	5V	GPIO on coprocessor, pull resistor to ENABLE
Mediatek chip	3.3V Mediatek	GPIO on coprocessor, pull resistor to !RESET
Flash chip	3.3V Mediatek	GPIO on both the SAM and the Mediatek
1.8V (not exposed)	5V	3.3V Mediatek
RAM for Mediatek	1.8V	Mediatek (involuntary)
3.3V coprocessor	5V	5V
Coprocessor chip	3.3V Coprocessor	GPIO on Mediatek, pull resistor to !RESET
3.3V module port A	3.3V Coprocessor	GPIO on coprocessor, pull resistor to OFF
3.3V module port B	3.3V Coprocessor	GPIO on coprocessor, pull resistor to OFF
USB hub IC	5V	3.3V Mediatek
Power to USB ports	5V	Mediatek via USB hub IC, pull resistor to off

What this means in practice is that the coprocessor (SAMD21) has control over both the "power plug" and the reset switch for the Mediatek, while the Mediatek only has control over the SAM's reset line. The rationale here is that the SAM uses a negligible amount of power, but it might be beneficial to be able to shut down the Mediatek in order to save power.

That said, there is no true "off switch" in the hardware that enables a permanent shutdown so long as external power is applied. Quiescent draw is currently estimated to be <10mW.

Triggering a Mediatek reset from the SAM results in a clean reboot. I believe the SAM does currently save some state, but it can be reset by the Mediatek, so this isn't really a concern.

Because the flash must be on in order for the Mediatek to boot (that's where [our build of OpenWRT](#) lives), it is powered by the 3.3V Mediatek bus. In order for the SAM to program the flash, it must hold the Mediatek in reset. Note that this is most useful for programming the board the first time...and other custom, low-level work, like what you would need to write in order to successfully save state so you could power down the Mediatek and bring it back up again elegantly. The protocol and architecture involved between the Mediatek and the coprocessor is explained more [in the T2 Firmware readme](#).

Port power management

Various pieces of the Tessel 2 have hardware support for individual power management.

Module port power

Tessel 2's architecture gives you individual control over the 3.3V (at least 250 mA) rails on each port, so you can turn modules off when they're not in use to save power.

Tessel 2's firmware turns off power to the port if a program does not require the `tessel` module. However, if `tessel` is required in the code, both will be turned on by default. It is possible to turn off the other [module port](#) with an explicit call, detailed [here](#).

USB ports

Though not currently broken out into the user layer, it is possible to control power to each USB port via `SET_FEATURE(USB_PORT_FEAT_POWER)` requests.

Module port goodies

Each pin on the two 10-pin headers is unique and can be reconfigured to do almost anything from speaking alternate comms protocols to clock generation. For example, if you decide you don't want [SPI](#), feel free to give yourself another [I2C](#) or [UART](#) with minimal changes to the SAMD21's firmware. Touching only JS, you can forgo the fancy comms in favor of just plain [GPIO](#), which gives you access to as many as 16 of them.

Last but not least, all eight [pins](#) on Port B are also inputs to a 12-bit, 350ksps ADC, with adjustable gain that can operate in differential mode.

More information on the module [ports](#) can be found in the [Atmel SAMD21 datasheet](#), such as this chart of I/O pin characteristics (p962):

Table 37-13. Normal I/O Pins Characteristics

Symbol	Parameter	Conditions	Min.	Typ.	Max.	Units
R_{PULL}	Pull-up - Pull-down resistance	I	20	40	60	k Ω
V_{IL}	Input low-level voltage	$V_{DD}=1.62V-2.7V$	-	-	$0.25 \cdot V_{DD}$	V
		$V_{DD}=2.7V-3.63V$	-	-	$0.3 \cdot V_{DD}$	
V_{IH}	Input high-level voltage	$V_{DD}=1.62V-2.7V$	$0.7 \cdot V_{DD}$	-	-	
		$V_{DD}=2.7V-3.63V$	$0.55 \cdot V_{DD}$	-	-	
V_{OL}	Output low-level voltage	$V_{DD} > 1.6V$, I_{OL} maxI	-	$0.1 \cdot V_{DD}$	$0.2 \cdot V_{DD}$	
V_{OH}	Output high-level voltage	$V_{DD} > 1.6V$, I_{OH} maxII	$0.8 \cdot V_{DD}$	$0.9 \cdot V_{DD}$	-	

Symbol	Parameter	Conditions	Min.	Typ.	Max.	Units
I_{OL}	Output low-level current	$V_{DD}=1.62V-3V$, PORT.PINCFG.DRVSTR=0	-	-	1	mA
		$V_{DD}=3V-3.63V$, PORT.PINCFG.DRVSTR=0	-	-	2.5	
		$V_{DD}=1.62V-3V$, PORT.PINCFG.DRVSTR=1	-	-	3	
		$V_{DD}=3V-3.63V$, PORT.PINCFG.DRVSTR=1	-	-	10	
I_{OH}	Output high-level current	$V_{DD}=1.62V-3V$, PORT.PINCFG.DRVSTR=0	-	-	0.70	
		$V_{DD}=3V-3.63V$, PORT.PINCFG.DRVSTR=0	-	-	2	
		$V_{DD}=1.62V-3V$, PORT.PINCFG.DRVSTR=1	-	-	2	
		$V_{DD}=3V-3.63V$, PORT.PINCFG.DRVSTR=1	-	-	7	
t_{RISE}	Rise time ⁽¹⁾	PORT.PINCFG.DRVSTR=0load = 5pF, $V_{DD} = 3.3V$	-	-	15	ns
		PORT.PINCFG.DRVSTR=1load = 20pF, $V_{DD} = 3.3V$	-	-	15	
t_{FALL}	Fall time ⁽¹⁾	PORT.PINCFG.DRVSTR=0load = 5pF, $V_{DD} = 3.3V$	-	-	15	ns
		PORT.PINCFG.DRVSTR=1load = 20pF, $V_{DD} = 3.3V$	-	-	15	
I_{LEAK}	Input leakage current	Pull-up resistors disabled	-1	+/-0.015	1	μA

Note: 1. These values are based on simulation. These values are not covered by test limits in production or characterization.

Tessel 2 Module Communication Protocols

Each of the two [ports](#) on Tessel 2 exposes ten [pins](#) for use in creating custom modules. Two of the [pins](#) are for power (3.3V and ground) and the other eight are configurable [GPIO](#) (General Purpose Input and Output) [pins](#) that can be used to communicate with your module.

All eight communication [pins](#) allow for communication through changes in voltage: some through variable voltage between 0V and 3.3V ([analog pins](#)), some through simple on/off (0V/3.3V) states ([digital pins](#)), and some through special [digital](#) toggling referred to by their [communication protocols](#). [Communication protocols](#) (such as [SPI](#), [I2C](#), and [UART](#)) involve a pre-defined set of [pins](#) used in a specific way. These protocols are used to encode complex messages. It's a lot like [Morse code](#), but for electronics.

In embedded electronics, there are four common protocols and Tessel [supports them all in JavaScript](#).

- [GPIO](#)
- [SPI](#)
- [I2C](#)
- [UART](#)

This guide will provide a brief overview of the protocols and cover some of the strengths and weaknesses of each.

You can see a mapping of which [pins](#) can be used for which protocols [on the Tessel docs page](#).

Quick Reference

Most of the time, you will choose your protocol based on the parts you are using when designing your module. Other things to consider are the [pins](#) you have available, as well as your communication speed requirements. The following table can be used as a quick reference for the more detailed explanations of each protocol below.

Protocol	# Pins Required	Max Speed
GPIO	1	1kHz
SPI	3+ (MOSI , MISO , SCK + 1 GPIO pin)	25MBit/s
I2C	2 (SCL and SDA)	100kHz or 400kHz configurable by module port
UART	2 (TX and RX)	8Mbit/s

A Note on Level Shifting

All of the diagrams and discussions below regarding [communication protocols](#) assume that the hardware modules you are communicating with operate at 3.3V, just like the Tessel.

If you have a device on your custom module that operates at 5V, 1.8V, or any other non-3.3V voltage, be careful! Directly connecting components with different operating voltages can damage the Tessel and/or your device.

You can use devices which operate at different voltages by employing a technique called 'level shifting'. Sparkfun has a [nice writeup on voltage dividers and level shifting](#) that can be used as a starting point.

The easiest way to avoid this complication is by trying to find module components that natively work at 3.3V.

GPIO

Pros: Simple, Requires a single pin

Cons: Not good for sending complex data

By far the simplest form of communication is via General Purpose Input/Output ([GPIO](#)). [GPIO](#) isn't really a 'protocol'. It is a rudimentary form of communication where you manually (in code) turn a pin on and off or read its state.

By default, Tessel's [GPIO pins](#) are configured to be inputs.

[See which pins support this protocol.](#)

Input

When acting as a [digital](#) input, a pin can be queried in software and will return a value indicating the current state of the pin: high (1 or true) or low (0 or false).

The following code snippet is an example of querying pin 2 on port A.

```
const tessel = require('tessel'); // Import tessel
const pin = tessel.port.A.pin[2]; // Select pin 2 on port A
pin.read((error, value) => {
  // Print the pin value to the console
  console.log(value);
});
```

This is great for connecting things like [switches](#), [buttons](#), and even [motion detectors](#). Just remember that the Tessel can only handle signals that are 3.3V or lower.

Determining digital state— a note for the curious:

It's pretty clear that if an input pin sees 3.3V it would be interpreted as a high state and if the pin is connected to ground it would recognize that as a low state. But what if the pin senses something in between, like 2V?

Your first thought might be that a high state is anything 1.65V (halfway between 0 and 3.3) or higher, and anything lower than that would be considered the low state. However, this is not always the case.

The high/low threshold is always determined by the processor. In the case of the Tessel 2, that's the SAM D21. The [documentation on the SAM D21](#) tells us that the Tessel will consider an input to be high if it is at least 55% of the Tessel's supply voltage (VDD) which is 3.3V. It also tells us that any signal that is 30% of VDD or lower is guaranteed to be read as a low state. That means anything 1.815V (referenced as VIH) or higher would be considered high, and anything .99V (referenced as VIL) or lower is guaranteed to be interpreted as a low state.

37.8. I/O Pin Characteristics

37.8.1. Normal I/O Pins

Table 37-14. Normal I/O Pins Characteristics

Symbol	Parameter	Conditions	Min.	Typ.	Max.	Units
R_{PULL}	Pull-up - Pull-down resistance	I	20	40	60	k Ω
V_{IL}	Input low-level voltage	$V_{DD}=1.62V-2.7V$	-	-	$0.25 \cdot V_{DD}$	V
		$V_{DD}=2.7V-3.63V$	-	-	$0.3 \cdot V_{DD}$	
V_{IH}	Input high-level voltage	$V_{DD}=1.62V-2.7V$	$0.7 \cdot V_{DD}$	-	-	
		$V_{DD}=2.7V-3.63V$	$0.55 \cdot V_{DD}$	-	-	
V_{OL}	Output low-level voltage	$V_{DD}>1.6V$, $I_{OL} \text{ maxI}$	-	$0.1 \cdot V_{DD}$	$0.2 \cdot V_{DD}$	
V_{OH}	Output high-level voltage	$V_{DD}>1.6V$, $I_{OH} \text{ maxII}$	$0.8 \cdot V_{DD}$	$0.9 \cdot V_{DD}$	-	

A screen capture from the electrical characteristics section of [Atmel's SAM D21 [datasheet](http://www.atmel.com/Images/Atmel-42181-SAM-D21_Datasheet.pdf)] (http://www.atmel.com/Images/Atmel-42181-SAM-D21_Datasheet.pdf)

What about the voltages between .99V and 1.815V? The read behavior is undefined and you are not guaranteed to get an accurate result. That's one reason why it's important to make sure that any module you connect to a Tessel input pin provides a high voltage that is between V_{IH} and 3.3V and a low voltage between ground and V_{IL} .

[More GPIO example code and information](#)

Output

When acting as a [digital](#) output, a pin can be set to one of two states: high (on/1/true) or low (off/0/false). High means the main Tessel board will set that pin to be 3.3V and low means it will set it to 0V.

[Digital](#) output is useful for connected hardware that understands simple on/off states. The following code snippet shows how you can set the state of the pin 2 on port A.

```
const tessel = require('tessel');
const pin = tessel.port.A.pin[2]; // Select pin 2 on port A
pin.output(1); // Turn pin high (on)
pin.read((error, value) => {
  // Print the pin value to the console
  console.log(value);
  pin.output(0); // Turn pin low (off)
});
```

Some examples of using a [GPIO](#) pin as an output are [simple LEDs](#) and for turning appliances on and off with a [relay](#).



An output pin is perfect for controlling an [LED](#). Image is licensed under the [Creative Commons Attribution-Share Alike 2.0 Generic](<http://creativecommons.org/licenses/by-sa/2.0/deed.en>) license.

[More GPIO example code and information](#)

SPI

Pros: Fast, supports multiple devices on a single bus, allows two-way communication

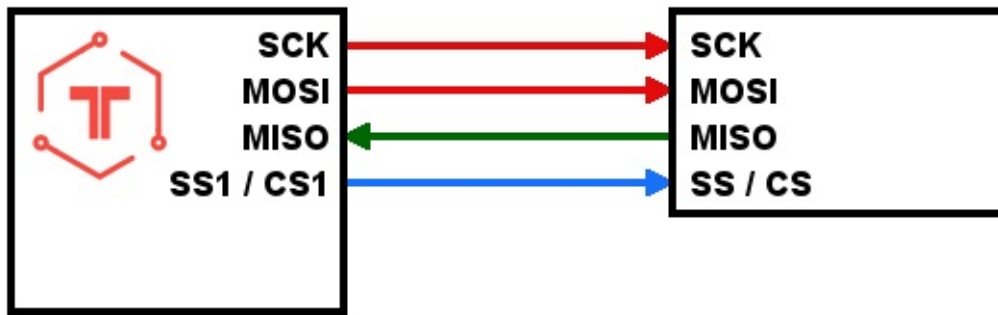
Cons: Requires at least 3 [pins](#)

[SPI](#) stands for [Serial Peripheral Interface](#). The [SPI](#) protocol allows data to be exchanged one byte at a time between the Tessel and a module via two communication lines. This is great for transferring data like sensor readings or sending commands to a module.

The [SPI](#) protocol is known as a [Master/Slave](#) protocol, which means that there is always a single [master](#) device which controls the flow of communication with one or more [slave](#) devices. Think of the [master](#) as a traffic cop. It directs all of the connected [slave](#) devices so they know when it's their turn to communicate.

When you are creating modules, the Tessel will always act as the [master](#) device, and your custom module will be a [slave](#) device.

The [SPI](#) protocol requires a minimum of three signal connections and usually has four (this is in addition to the power connections). The following diagram shows the connections (arrows indicate flow of data).



*The red lines constitute the shared bus connections used for talking to the **slave** devices. The green wire is the shared bus connection used by the slaves to talk to the **master**. The blue line is the chip select for signaling each **slave** individually.*

SCK

This is the clock signal that keeps the Tessel and the module synchronized while transferring data. The two devices need to have a mutual understanding of how fast data is to be transferred between them. This is sometimes referred to as the baud or bitrate. The clock signal provides that reference signal for the devices to use when exchanging data.

Without a clock signal to synchronize the devices, the devices would have no way to interpret the signal on the data lines.

One bit of data is transferred with each clock cycle (see the diagram below).

MOSI

MOSI stands for **Master Out Slave In** and is the connection used by the Tessel to send data to the module. It's on this line that the Tessel will encode its data.

MISO

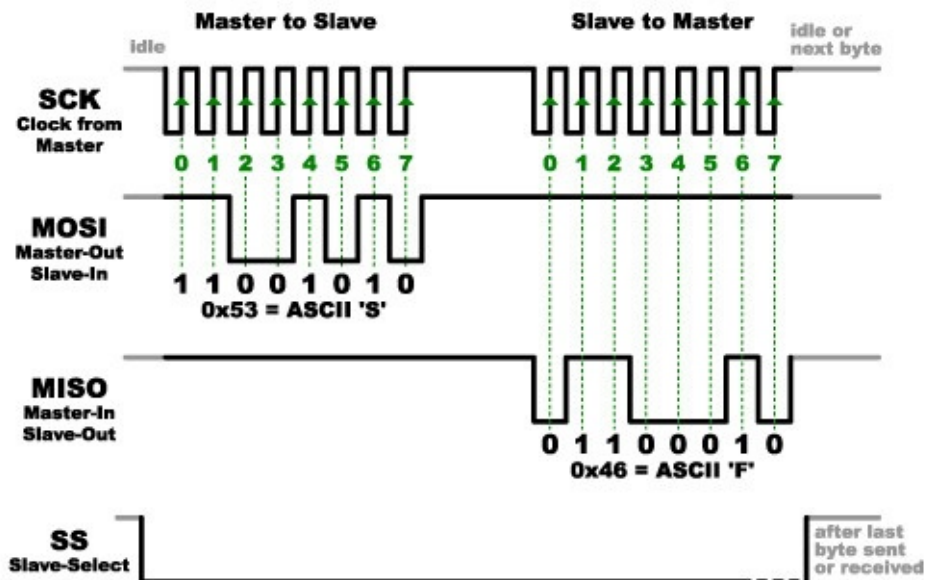
MISO stands for **Master In Slave Out** and is the connection used by the module to send data to the Tessel.

SS or CS

This line, normally referred to as the **Slave Select (SS)** or **Chip Select (CS)** line, is used by the **master** device to notify a specific **slave** device that it is about to send data. We normally call it CS, but you may see it either way in datasheets and other references.

When you create a **Tessel module** which uses the **SPI** protocol, the CS connection will be handled by one of the **GPIO pins** on the Tessel port.

The following diagram shows how the various pins in the SPI protocol are toggled to create meaningful data. In this case, the master sends the ASCII character 'S', and the slave responds with 'F'.



Timing diagram of SPI data exchange. Modified [image]

(<https://dlnmh9ip6v2uc.cloudfront.net/assets/c/7/8/7/d/52ddb2dcce395fed638b4567.png>)
 from Sparkfun is [CC BY-NC-SA 3.0](<http://creativecommons.org/licenses/by-nc-sa/3.0/>)

Remember that the master initiates all communication. When it is ready, the first thing it will do is pull the CS/SS pin low to let the slave device know that a data transmission is about to begin. The master holds this pin low for the duration of the data exchange as seen in the diagram above.

With the CS/SS pin low, the master will start to toggle the clock pin (SCK) while simultaneously controlling the MOSI to represent the bits of information it wishes to send to the slave. The numbers in green on the diagram above delineate each bit in the byte being transferred.

It sounds complicated, but remember that the Tessel takes care of all of this pin manipulation for you. All you have to do is write some Javascript like this code snippet, which demonstrates the use of the SPI protocol on port A.


```
'use strict';
const tessel = require('tessel');
const port = tessel.port.A;
const spi = new port.SPI({
  clockSpeed: 4000000 // 4MHz
});

spi.transfer(new Buffer([0xde, 0xad, 0xbe, 0xef]), (error, buffer) => {
  console.log(`buffer returned by SPI slave: <${[...buffer]}>`);
});
```

[More SPI example code and information](#)

I2C

Pros: Only requires 2 [pins](#), multiple devices on a single bus, allows two-way communication

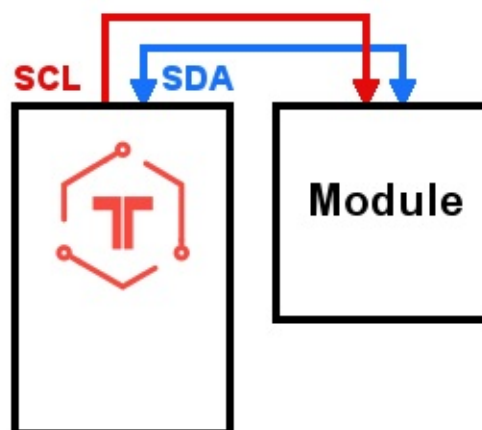
Cons: Devices can have address conflicts, not as fast as [SPI](#)

[I2C](#) stands for [Inter-Integrated Circuit](#) and is pronounced "I squared C", "I two C" or "I-I-C".

[I2C](#) is a protocol that allows one device to exchange data with one or more connected devices through the use of a single data line and clock signal.

[I2C](#) is a [Master/Slave](#) protocol, which means that there is always a single [master](#) device which controls the flow of communication with one or more [slave](#) devices.

[I2C](#) only requires two communication connections:



SCL

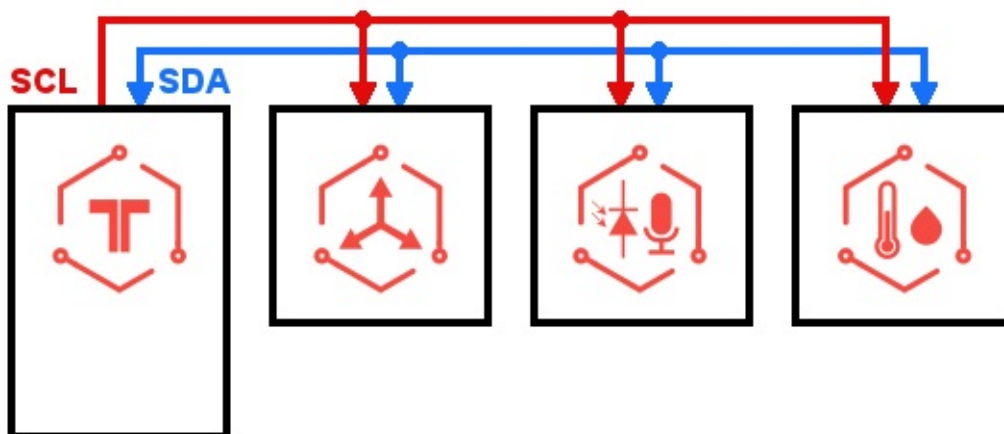
This is the clock signal that keeps the Tessel and the module synchronized while transferring data. The two devices need to have a mutual understanding of how fast data is to be transferred between them. This is sometimes referred to as the baud or bitrate. The clock signal provides that reference signal for the devices to use when exchanging data. Without a clock signal to synchronize the devices, they would have no way to interpret the signal on the data lines.

SDA

This is the data line used for exchanging data between the [master](#) and slaves. Instead of having separate communication lines for the [master](#) and [slave](#) devices, they both share a single data connection. The [master](#) coordinates the usage of that connection so that only one device is "talking" at a time.

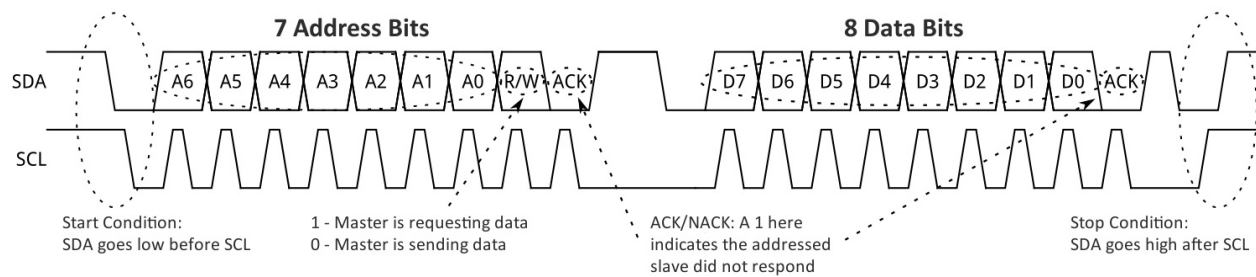
Since multiple [slave](#) devices can use the same [SDA](#) line, the [master](#) needs a way to distinguish between them and talk to a single device at a time. The [I2C](#) protocol uses the concept of **device addressing** to coordinate traffic on the data line.

Every single [I2C](#) device connected to the Tessel will have an internal address that cannot be the same as any other module connected to the Tessel. This address is usually determined by the device manufacturer and listed in the [datasheet](#). Sometimes you can configure the address through device-specific tweaks defined by the manufacturer. The Tessel, as the [master](#) device, needs to know the address of each [slave](#) and will use it to notify a device when it wants to communicate with it before transferring data.



Flow of data between Tessel and multiple [I2C](#) devices.

The following diagram illustrates how the [SDA](#) and [SCL](#) pins are toggled when transferring data with the [I2C](#) protocol.



To begin a data transaction, the **master** creates what is called a start condition by pulling the **SDA** pin low before the **SCL** pin.

The **master** then broadcasts the address of the device it wishes to communicate with by sending each bit of the 7 bit address. Notice the clock signal (**SCL**) is toggled for each bit. This toggling is how the slaves know when to read each bit of the address so they can determine with which device the **master** wants to communicate.

Right after the address, the **master** sends a read/write bit which signals whether it will be sending data to the **slave** or reading data from the **slave**.

After broadcasting the address, the **master** either transmits data to the **slave** or sends the address of a **register** (internal storage) on the **slave** from which it wishes to retrieve data.

Finally, the **master** will issue a stop condition on the bus by pulling **SCL** high, followed by **SDA**.

It's a little complicated, but the Tessel takes care of all the details for you. Using the **I2C pins** on port A looks like this:

```
'use strict';
const tessel = require('tessel');
const port = tessel.port.A;
// This is the address of the attached module/sensor
const address = 0xDE;
const i2c = new port.I2C(address);

i2c.send(new Buffer([0xde, 0xad, 0xbe, 0xef]), (error) => {
  console.log("I'm done sending the data");
  // Can also use err for error handling
})
```

[More I2C example code and information](#)

UART

Pros: Widely supported, allows two-way communication

Cons: Can't share communication lines, slower than [SPI](#) and [I2C](#)

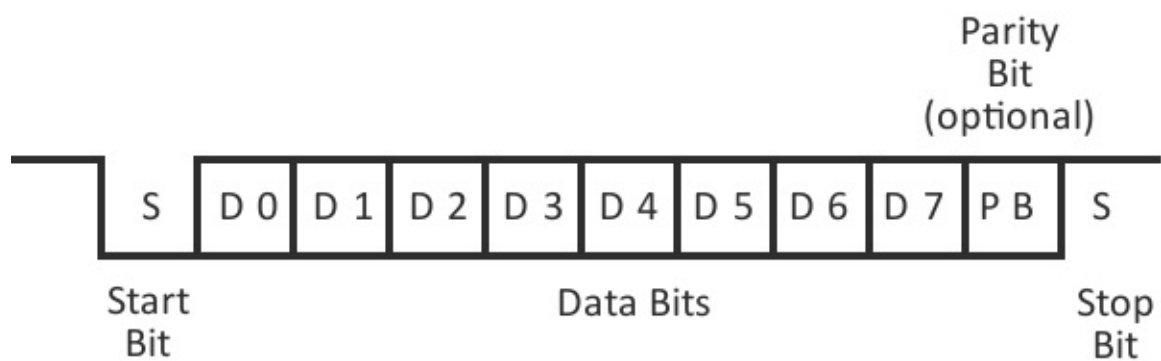
[UART](#) stands for [Universal Asynchronous Receiver/Transmitter](#) and is really just a fancy way of referring to a [serial](#) port. It is really easy to understand as it only requires two lines: a transmission line ([TX](#)) and a receiving line ([RX](#)). The Tessel sends data to connected modules on the [TX](#) line and gets data back on the [RX](#) line.

TX

Used by the Tessel to send data to the module.

RX

Used by the module to send data to the Tessel.



A [UART](#) data transmission.

[UART](#) transmissions begin with a start bit where the appropriate line ([TX](#) or [RX](#)) is pulled low by the sending party. Then 5 to 8 data bits are sent. The diagram above shows a scenario where 8 bits are sent.

Following the data, an optional [parity bit](#) is sent, followed by 1 or 2 stop bits, where the sending module pulls the pin high.

For this protocol to work, the sender and receiver have to agree on a few things.

1. How many data bits are sent with each packet (5 to 8)?
2. How fast should the data be sent? This is known as the baud rate.
3. Is there a parity bit after the data, and is it high or low?

4. How many stop bits will be sent at the end of each transmission?

When you want to interact with a specific module via [UART](#), the answers to these questions are found in the module's [datasheet](#). Using that information you can configure the [UART](#) in Javascript like this:

```
'use strict';
const tessell = require('tessell');
const port = tessell.port.A;
const uart = new port.UART({
  dataBits: 8,
  baudrate: 115200,
  parity: "none",
  stopBits: 1
});

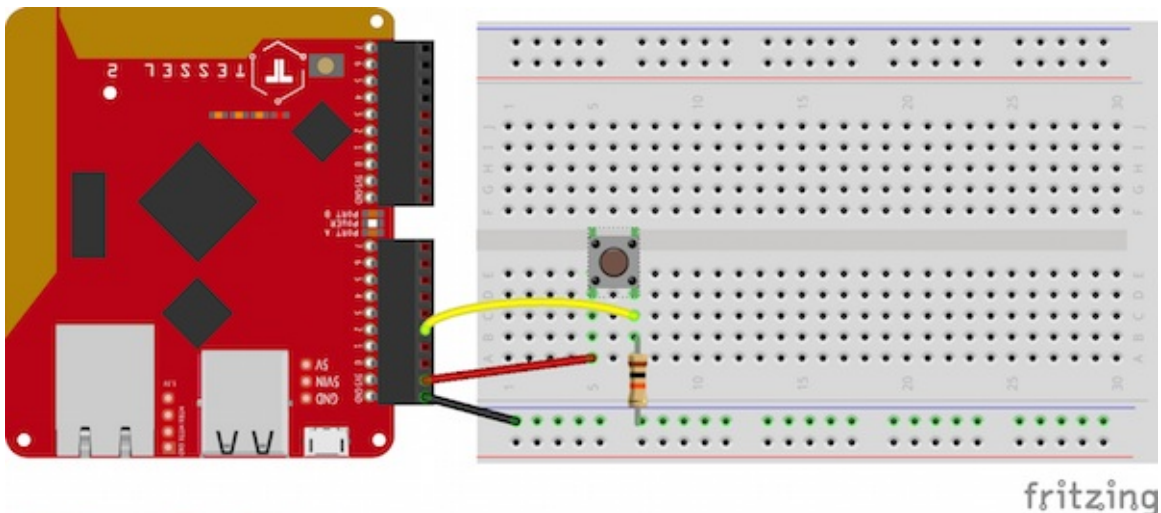
uart.write('Hello UART');
```

[More example code using a UART](#)

Interrupt Pins

Interrupts allow us to [register](#) events based on state changes in a pin. [Pins](#) 2, 5, 6, 7 on both [Ports](#) are available for interrupts.

Take a look at the following circuit. While the switch is open, pin2 will be low. When the button is pressed we'll complete the circuit and the voltage at pin2 will rise to 3.3V.



Let's turn on the green [LED](#) when the button is pressed. Here's one way of doing it.

```
var tessel = require('tessel'); // Import tessel

// Initialize our pins
var green = tessel.led[2];
var pin2 = tessel.port.A.pin[2];

// Turn off the green LED
green.off();

// Every 10 milliseconds check to see if the button was pressed
setInterval(function() {
  pin2.read(function(error, value) {

    // button was pressed
    if (value === 1) {
      green.on();
    }
  });
}, 10);
```

This is pretty inefficient. We're spending a lot of time (every 10 milliseconds) manually checking to see if the button was pressed. Let's just tell the processor to run some code when the voltage rises.

```
var tessel = require('tessel'); // Import tessel

// Initialize our pins
var green = tessel.led[2];
var pin2 = tessel.port.A.pin[2];

// Turn off the green LED
green.off();

// Register an event. When the voltage on pin2 rises, turn on the green LED.
pin2.on('rise', function() {
  green.on();
});
```

We tell the processor to invoke our callback function when pin2 rises (button has been pressed). We don't have to continuously poll to see if the voltage has risen. We can just [register](#) our listener and move on.

[More information on interrupts.](#)

Creating Your Own Tessel Module

What Is a Module?

Modules should be devices with clear-cut functionality. That is to say, they should have a single, well-defined purpose or a set of closely related functions, rather than an eclectic mix of capabilities onboard. This requirement is designed to reduce complexity, cost, and power consumption and maximize reusability in hardware and software.

—Tessel hardware docs

One of the main goals of the Tessel platform is "connected hardware as easy as npm install." If you need an accelerometer, Bluetooth Low Energy connection, SIM card capability, or any of the other [first-party or USB modules available](#), you can literally plug, npm install, and play.

There is also a growing library of community-contributed [third-party modules](#): npm libraries paired with some simple wiring instructions, built for specific pieces of hardware.

But what if your project needs functionality that can't be provided by one of the existing first- or third-party modules? You make your own, of course.

This guide will walk you through the basics of creating your own [Tessel module](#) using our new proto-module boards.

A Quick Note of Encouragement

Making your own module might seem like an overwhelming task best left to those who know things like Ohm's Law, how to solder, and why licking a 9V doesn't feel very good. But while having some electronics knowledge doesn't hurt, it's not a hard and fast prerequisite. If you know how to program, you are smart enough to figure out the basic concepts and make awesome things happen. Just trust us and read on.

Module Design Basics

Before you venture into the world of custom module creation, we need to cover some basics that will help guide you along the way.

Every module created for the Tessel involves 5 parts:

1. Power
2. Communication
3. Software
4. Documentation
5. Sharing

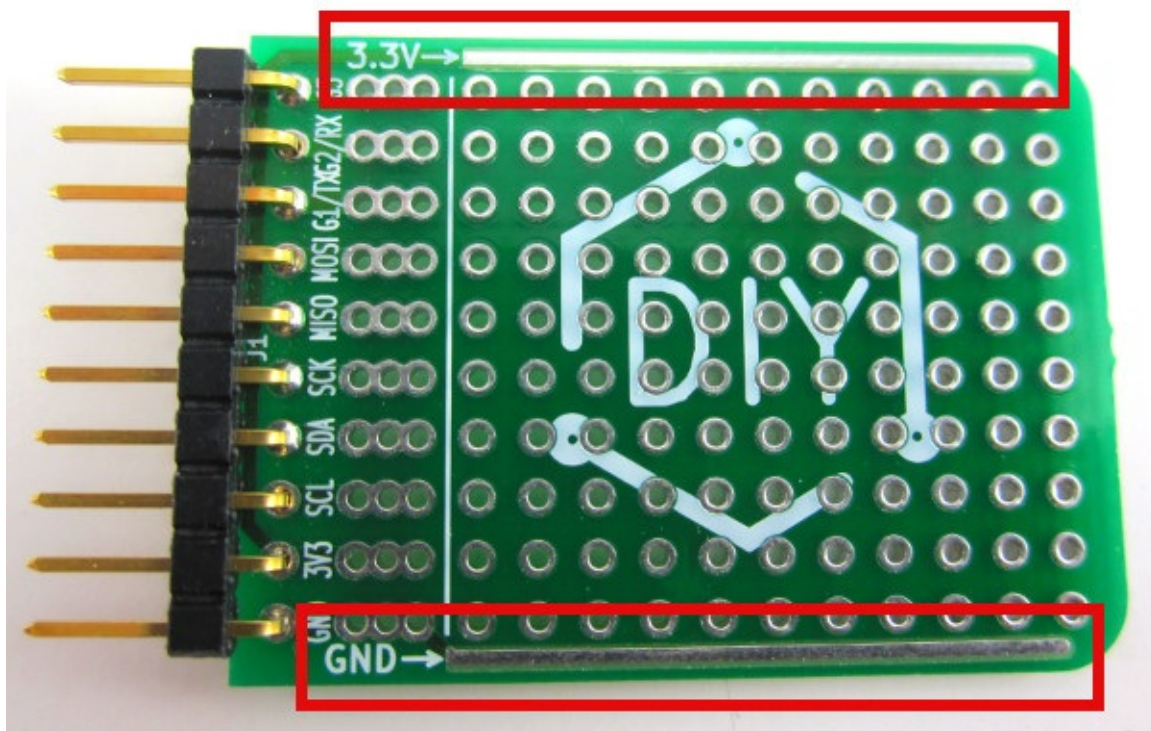
If you understand how each of these fit into the module creation process, you will be well on your way to creating your own custom module. Let's start with power.

Power

When dealing with anything in electronics, whether it be sensors, displays, buttons, or servos, you have to provide power. Everything needs a power source, and Tessel modules are no exception. In its simplest form, you can think of a power source as two connections; a positive voltage and a ground connection. These connections are provided on each Tessel port.

The main Tessel board can be powered several ways, but regardless of how you provide power to the main board it ultimately turns that source into 3.3V. That's the native "voltage language" of the Tessel. It likes to speak 3.3V to everything if possible.

One of the nice things about the proto-module is that the 3.3V and ground connections are exposed as two rails that run along each side of the module as seen below. This allows you to easily power your module components.

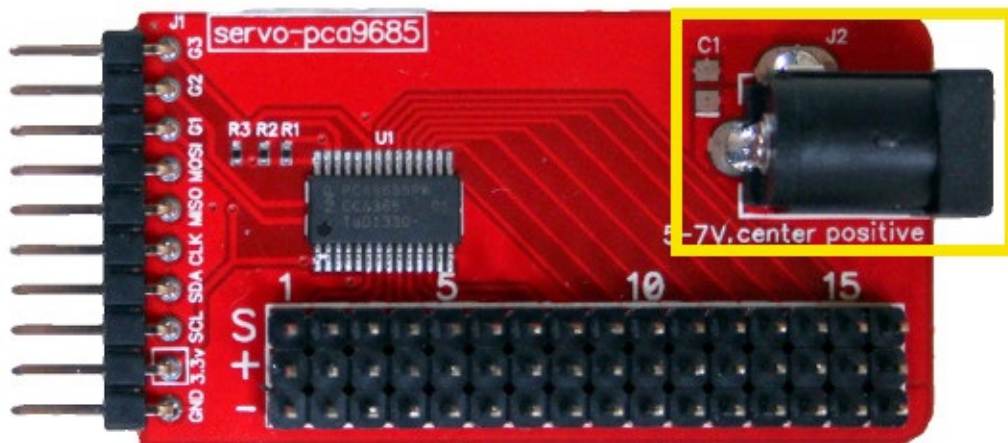


Proto-module power rails

Special Considerations

If all of the components on your custom module operate at 3.3V, then your power design is extremely simple. You just use the 3.3V and ground rails and connect your components accordingly (the [custom screen module](#) below is a good example). Sometimes, however, you may encounter a situation where 3.3V is not what you need, like in the case of the [servo module](#).

Many servos like to operate at 5V. That's their native "voltage language" and so the 3.3V provided by the Tessel isn't ideal and, in many cases, just won't work. Servos can also draw a lot of current, which may overwhelm the Tessel's power supply. To solve this problem, you'll notice that the servo module has a DC barrel jack on it that allows you to plug in a 5V adapter to provide sufficient power to the connected servos.



DC Barrel Jack on the Servo Module

From the [servo module schematic](#), we can see that communication is accomplished with the normal I²C lines, which operate at 3.3V, but servo power is provided via schematic component J2, which is the barrel jack.

This guide isn't meant to be a comprehensive power reference, but we just want to point out that if you have any components on your custom module that work outside of the 3.3V realm, you will [need to design for it](#). To simplify your module design, we recommend using 3.3V components where possible.

The Power Warnings

Here are some items to remember when working with power in electronics.

- **ALWAYS** unplug your Tessel and any external power before making or altering connections.
- Don't mix voltages unless you know what you're doing. For example, if you put 5V on any of the module [pins](#), you can ruin your Tessel.
- Never connect the positive voltage directly to ground. This is called a [short circuit](#) and can ruin components and your day.
- Always exercise caution and verify that you have hooked everything up correctly before plugging in your Tessel.

Communication

Once you have decided how you are going to power your custom module, it's time to decide how the main Tessel board will talk to it.

In the world of web communication, there are standards like HTTP, HTTPS, and FTP that allow different systems to talk to each other in well-defined ways. The same concept exists in the hardware world and the Tessel supports four standard [communication protocols](#) for talking to modules.

- [GPIO](#)
- [SPI](#)
- [I2C](#)
- [UART](#)

Because the Tessel [does most of the heavy lifting](#) for all of these, you don't need to be an expert to use them in your custom module. However, if you'd like to learn a little more, we've [provided a simple overview of each](#).

So Which Communication Protocol Should I Use?

Knowing that there are four communication options available to you, which should you use for your custom module? The good news is that this will usually be decided for you based on the type of module you are creating. For example, most [PIR sensor modules](#) will set a pin high when motion is detected, which can be read with a simple [digital](#) input ([GPIO](#)). The same applies to sensors. For example, the Si7020 temperature and humidity sensor on the [Climate Module](#) communicates via the [I2C protocol](#). Usually sensors will only support one protocol— so the decision is easy, you use that one.

You will find some modules that support both [SPI](#) and [I2C](#), and either will work just fine with the Tessel. As a general rule of thumb, we recommend you favor the [SPI](#) protocol in these scenarios as it eliminates the risk of [I2C](#) address collisions with other connected [I2C](#) modules.

Software

Once you have the power and communication all worked out and connected, it's time to start writing JavaScript to talk to your module. This is where the open-source nature of the Tessel really comes in handy. We've already used all of the possible [communication protocols](#) in [our modules](#) and the [code is free to look at](#) and copy.

Design an [API](#) for working with your module so that it's easy for others to integrate into their projects. As a general rule, top priority is intuitive interaction. Second priority is exposing as many features as you can. You can find a lot of great information about organizing your project and writing an [API](#) in [Making a Tessel-style Library for Third-Party Hardware](#).

Documentation and Sharing

Once you have a working module, it's time to share the good news with everyone so other people can build amazing things with your work. We recommend doing a few things to share it with the community as outlined below. This helps create a consistent feel across all Tessel modules, whether they are official modules or submitted by the community.

Create A Git Repo

Having your code in a git repo and available online makes it easy for others to grab your code and start using it in their projects. To help you get started we've created a template repository that you can use as a starting point.

[Custom Module Git Repo Template](#)

Document It

You may have just created the world's most amazing [Tessel module](#), but how is anybody going to know about it or use it? Once you've hashed out the [API](#) and everything is working, it's important to document its use so others can easily apply your work to their projects. The best way to do this is to use the [README template](#), which includes things like how to install the module, example usage, and [API](#) information.

Publish Your Code on NPM

Once your git repo is ready and you've documented your module, this step is really easy and makes your module fit the Tessel motto of "connected hardware as easy as npm install." If you've never published code to NPM before, you can get started with just four lines of code (run in a shell window).

```
npm set init.author.name "Your Name"
npm set init.author.email "you@example.com"
npm set init.author.url "http://yourblog.com"

npm adduser
```

This sets up your NPM author info. Now you're ready to create your package.json file. There is one in the repo template but we suggest creating it by running npm init from within the project directory.

```
npm init
```

Edit your package.json file to include a name, version, and description. We also highly recommend adding "tessel" as a keyword so that other Tessel users can easily find your work. Most of the package.json file is self-explanatory and follows the [npm package.json standard](#) with the exception of the **hardware** member.

```
1 //This an example package.json taken from the Relay module.
2 {
3   "name": "relay-mono",
4   "version": "0.1.2",
5   "description": "Library to run the Tessel Relay module.",
6   "main": "index.js",
7   "scripts": {
8     "test": "tinytap -e 'tessel run {}' ${RELAY_PORT}' test/*.js"
9   },
10  "hardware": {
11    "./examples": false,
12    "./test": false
13  },
14  "dependencies": {
15    "tinytap": "0.0.2"
16  },
```

*****hardware**** section of package.json*

This is a Tessel-specific item that you must add manually and is a list of files and folders in your project that you would like to exclude when code is pushed to your Tessel.

Once your `package.json` file is complete you're ready to publish your code. Run the following command from the top level directory of your project.

```
npm publish ./
```

Create a Project Page

The [Tessel Projects page](#) is a way to share your module directly with the Tessel community. You simply provide a few pieces of information, a picture, and can even use your `README.md` file from your Git repo as the contents.

Submit Your Module

We're always looking to add modules to our [third-party module list](#) so if you'd like your custom module to be listed at tessel.io/modules then fill out this form and we'd be happy to review it.

[Third-Party Module Submission Form](#)

A great example of using this module-creation pattern can be found in [Making a Tessel-style Library for Third-Party Hardware](#).

Your First Custom Module

So now that we've described the pattern for making a custom module, let's walk through creating a very simple module using that pattern. The Tessel has a spare button on the main board, but maybe you'd like to add one as a module. Kelsey did [a great writeup on adding a button to the GPIO bank](#) so let's use her work to take it one step further using a proto-module.

Power

You might not think of a button as needing power, and you're right, sort of. While the button itself doesn't need power to function, we can connect our button in such a way that it uses the power connections to create high and low states on a [GPIO](#) pin.

[GPIO pins](#) on the Tessel will always read high/truthy with nothing connected, because internally (inside the main Tessel chip), they are pulled up to the 3.3V supply. That's our positive connection.

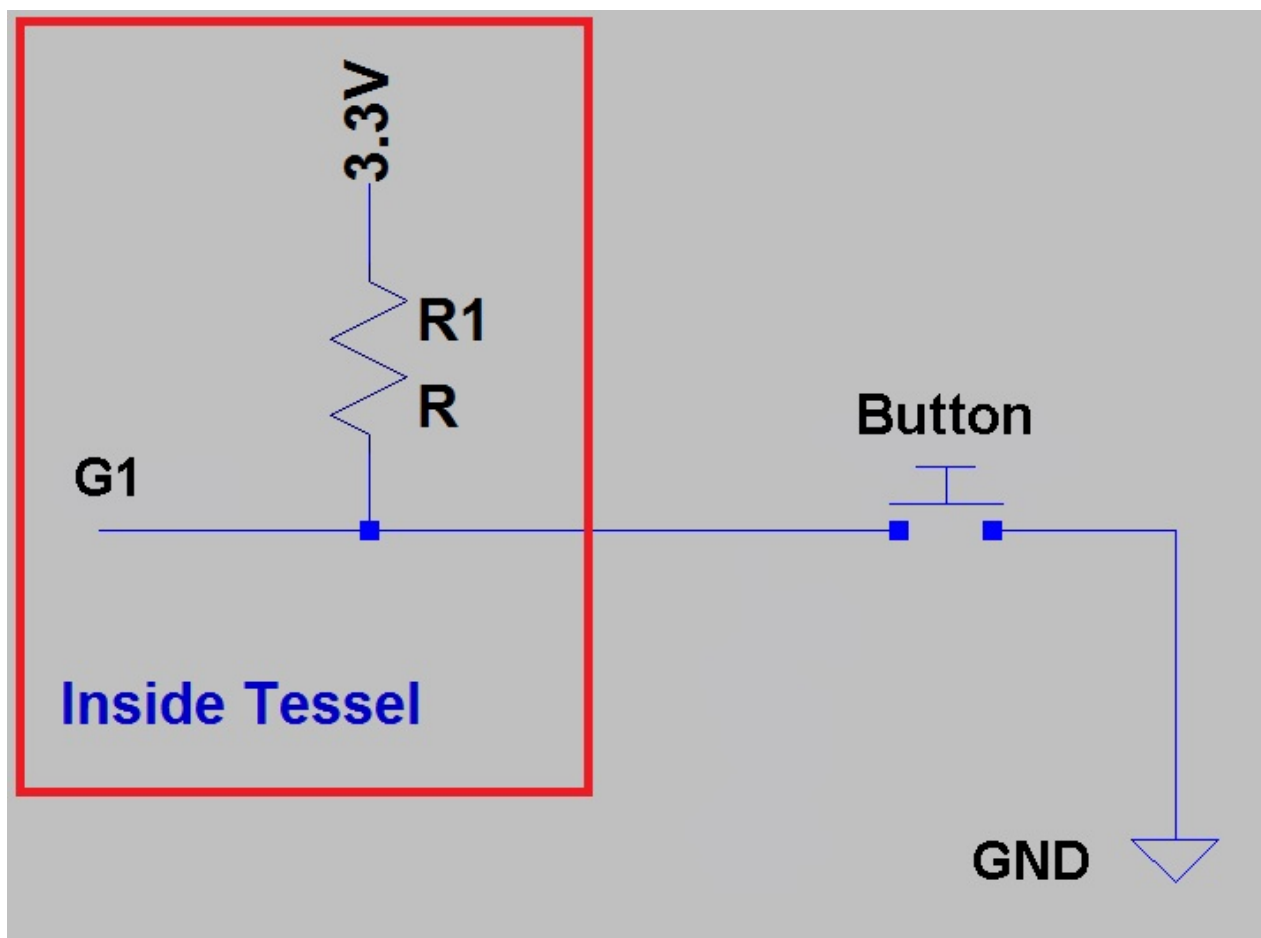
The other power connection is ground, which we'll connect to one side of our button. It doesn't matter which side, because a button is just a momentary switch that creates and breaks a connection. You can't hook it up backward. We'll get to connecting the other side of the button in a minute.

Communication

As mentioned above, normally your communication protocol is determined by your module. In the case of a button, we use a [digital GPIO](#) pin because we want to read the state of the button.

Each port on the Tessel has [several digital I/O pins that can serve this purpose](#), and you are free to pick any one you like because it doesn't matter.

We're going to choose port A's pin 1, which is what we will hook up to the other side of the button. When the button is not pressed, our input pin will read high, or true. When we press it, we are making a connection between our [GPIO](#) pin and ground, which will cause a low state to be present on the input pin. This is what the design looks like.



Schematic of button connections

Don't let the soldering part scare you. Soldering components like this onto a proto-module is a little harder than learning to use a hot glue gun, but not a lot harder. [This tutorial from Sparkfun](#) is a great place to start learning soldering.

Software

Actually, we're going to reuse the code from Kelsey and modify it just a bit. Since she followed the style guidelines and shared her work on NPM, we actually don't have to write the bulk of the code. She's even provided a [Quick Start guide](#) in her documentation, so we'll use that.

- ```
npm install tessel-gpio-button
```



2. Create a file named **myButton.js** and copy her Quick Start code into it. It should look like this:

```
// examples/button.js
// Count button presses

var tessel = require('tessel');
var buttonLib = require('../');
var myButton = buttonLib.use(tessel.port['GPIO'].pin['G3']);

var i = 0;

myButton.on('ready', function () {
 myButton.on('press', function () {
 i++;
 console.log('Press', i);
 });

 myButton.on('release', function () {
 i++;
 console.log('Release', i);
 });
});
```

This almost works right out of the box. We just need to make two small adjustments. Do you see them?

First, to include her module we won't use "require('../')." Instead we'll include the module directly with `require('tessel-gpio-button')`.

Second, she hooked her button up to the G3 pin on the Tessel 1's **GPIO** bank, but we've hooked our proto-module up to Port A on Tessel 2 and used pin 1. So all we have to change is the line where *myButton* is defined. We'll change it to be:

```
var myButton = buttonLib.use(tessel.port['A'].pin[1]);
```

Save your changes and test it out.

```
tessel run myButton.js
```

Every time you push your button it should log to the console.

Congratulations! You just created your first custom module for the Tessel.

## Documentation and Sharing

We sort of cheated for our first module; Kelsey had already created an NPM package that we could reuse, so there wasn't really anything to document or share on the software side. There is nothing wrong with that. In fact, the less code you have to write, the better. This is a great example of how taking the time to document and share your work benefits the entire community.

What we can do though is create [a project page](#) showing how we took Kelsey's button to the next level in the form of a plug-in module. We created the physical module. It's a simple module, but we should document it in case someone else wants a button module like ours.

[Custom Button Module Project Page](#)

## Custom Screen Module

Now that you have a simple module under your belt, it's time to level up. To date, the module that people have requested the most is a screen module. Displays are tricky because they come in so many flavors. There are simple 7-segment displays, LCD displays, OLEDs, resistive touchscreens, capacitive touchscreens, and more. This is a great use case for a custom module.

One of the popular screen modules in embedded projects is the Nokia 5110, because of its simple interface and low cost. Let's see how we'd create a module for it by following the same pattern as before.

For this example we'll use the [Nokia 5110 breakout from Sparkfun](#), but you could also use the [Adafruit version of the screen](#) or just try to snag one [on Ebay](#)

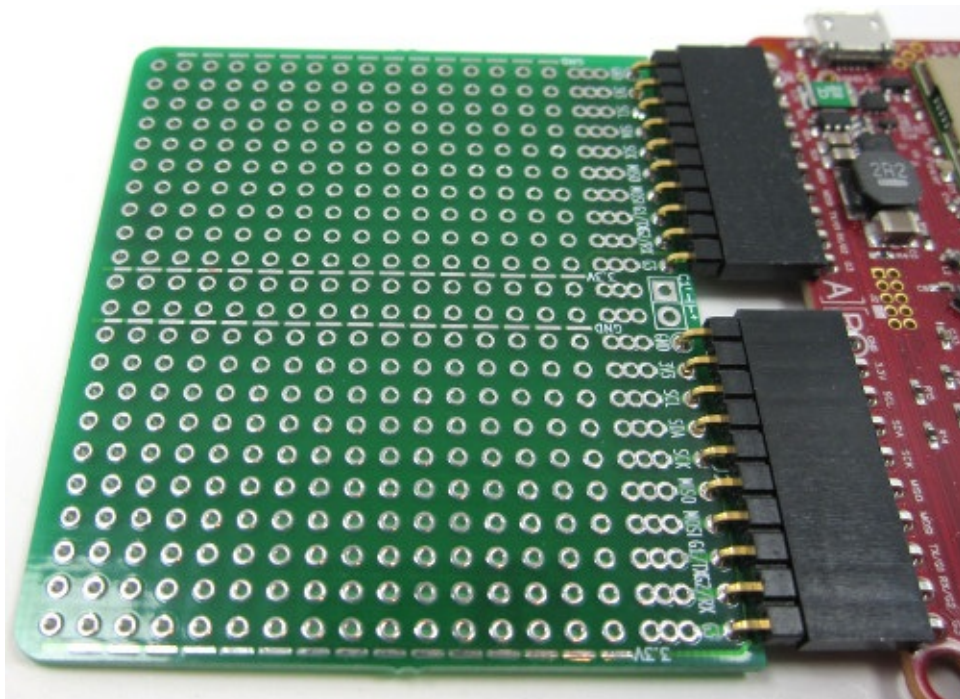


## Nokia 5110 Graphic LCD

### Power

The 5110 has a listed power supply range of 2.7V to 3.3V, which means any voltage in between (inclusive) is sufficient to power the screen. Since the Tessel [ports](#) have a 3.3V supply pin we don't have to do anything special to hook it up. All we need to do is connect the screen VCC, or positive pin, to a 3.3V rail on the proto-module and the GND on the screen to a GND rail.

Because of the screen's size, we'll use one of the double-wide proto-modules this time, even though we'll only use a single port to connect everything.



*Double-Wide Proto-Module*

### Communication

Just like in the button example, the communication protocol for the screen has already been chosen for us. The Nokia 5110 uses a slightly modified version of [SPI](#) to communicate with a parent controller, namely the Tessel in our case.

In addition to the normal [SPI](#) protocol, the 5110 has an extra pin involved (**D/C**) that tells the screen whether the data we are sending via [SPI](#) is a special command or actual screen data.

The D/C pin is controlled by a simple high or low signal, which is a perfect job for one of the [GPIO pins](#).

The following table shows all of the communication connections available on our screen and how we'll attach them to the Tessel port.

| Nokia 5110 Pin       | Proto-Module Connection                  |
|----------------------|------------------------------------------|
| SCE                  | G1                                       |
| RST                  | Connected to 3.3V through 10K resistor   |
| D/C                  | G2                                       |
| <a href="#">MOSI</a> | <a href="#">MOSI</a>                     |
| SCLK                 | <a href="#">SCK</a>                      |
| <a href="#">LED</a>  | Connected to G3 through 330 ohm resistor |

## Design Note

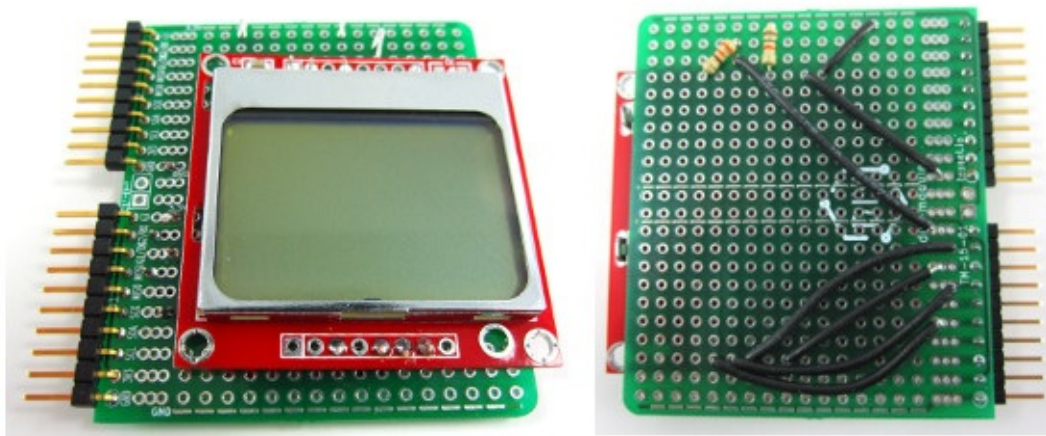
The Nokia 5110 has 4 connections that can utilize [GPIO pins](#) for functionality. The D/C (data/command) and SCE (chip select) [pins](#) have to be used to get data to the screen. That leaves just one [GPIO](#) pin on the port with RST and [LED](#) left unconnected. You have a few options here.

1. Wire RST to 3.3V through a 10K resistor which prevents you from resetting the screen in code. This allows you to control the backlight with the free [GPIO](#) pin.
2. Wire [LED](#) to 3.3V through a 330 ohm resistor (to limit current) which will permanently turn on the backlight. This leaves a [GPIO](#) free that can be used to reset the screen via Javascript.
3. Since we're using the double-wide, you could use a [GPIO](#) pin from the adjacent port and have use of both [LED](#) and RST
4. Connect SCE (chip select) to ground, which frees up a [GPIO](#) so you can control both [LED](#) and RST. Holding the chip select low, however, makes it so that **no other SPI device** (including other Tessel modules that use [SPI](#) e.g., the Camera module) can be connected to the Tessel on any other port.

We decided to go with option 1 because there isn't really a need to reset the screen in most cases and it allows control of the backlight with a [GPIO](#) pin. This is another great thing about custom modules. You can design it however you want to fit your project needs.

We hooked everything up using the [Graphic LCD Hookup Guide](#). We recommend testing everything with a [breadboard](#) before you solder everything in place, just to make sure it works the way you expect it to.

Here is what the module looks like soldered to the double-wide proto-module.



*Nokia 5110 soldered to a large proto-module board*

## Software

With the screen hooked up, it's time to start writing code. We'll follow the pattern found in the [Git Repo Template](#) and start by creating a directory called **tessel-nokia5110** and cd into that directory. Next, we'll use `t2 init` to create **index.js** which is where we'll write our [API](#) using [the example index.js template](#) as a guide.

Because this screen is so popular, there are lots of code examples and libraries online for interacting with it. We don't need to reinvent the wheel; we just want to control the screen with JavaScript.

We took a [simple Arduino library](#) for this screen and [ported it to JavaScript](#). Our [API](#) is very simple and exposes just one event and a few methods.

## Event

Nokia5110.**on**('ready', callback(error, screen)) - Emitted when the screen object is first initialized

## Methods

Nokia5110.**gotoXY**(x,y,[callback(error)]) - Sets the active cursor location to (x,y)

Nokia5110.**character**(char, [callback(error)]) - Writes a single character to the display

Nokia5110.**string**(data, [callback(error)]) - Writes a string to the display

Nokia5110.**bitmap**(bitmapData, [callback(error)]) - Draws a monochrome bitmap from *bitmapData*

Nokia5110.**clear**([callback(error)]) - Clears the display

Nokia5110.**setBacklight**(state) - Turns the backlight on if *state* is truthy, off otherwise

## Documentation

Now that the module is connected up and the software is working, it's time to document its use.

We can't stress enough how important this is, and it really only takes a few minutes once you've defined everything. Just think of all the times you've needed a piece of code and found a beautifully documented example that had you up and running in minutes. Share that love with others when you create your own modules, no matter how trivial you think they are.

In our case, we'll take the [template README.md file](#) and [add some notes for getting started as well as document our API](#).

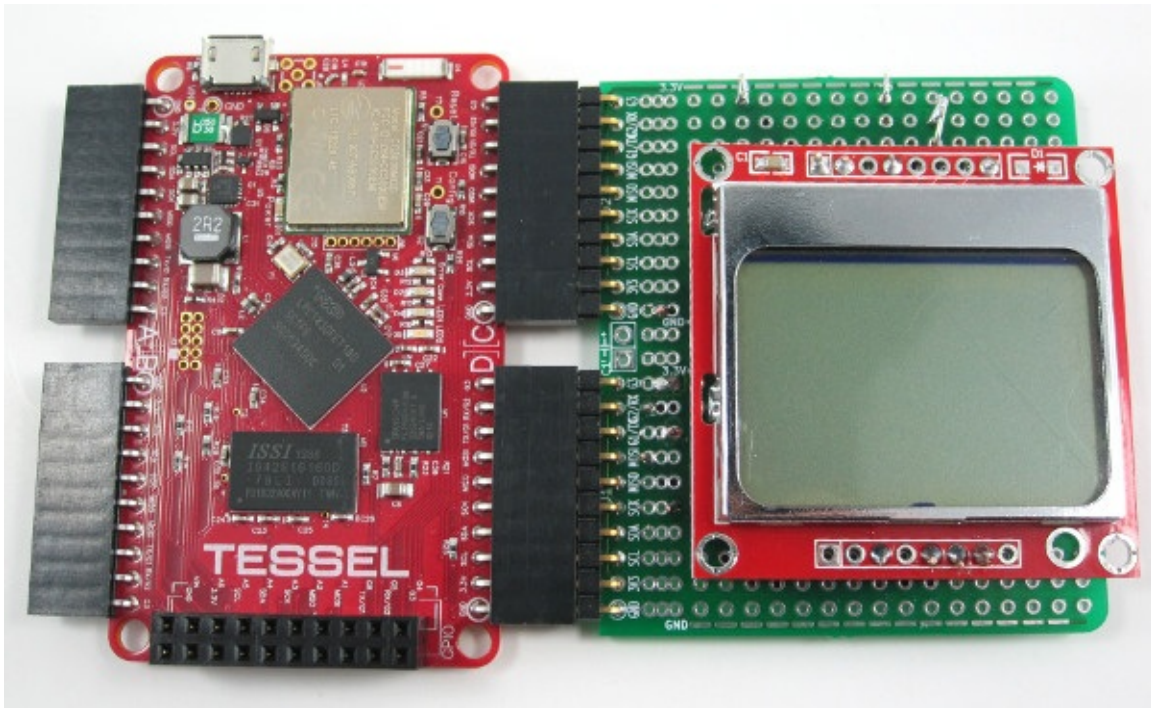
We'll also create an **examples** folder to show how the module can be used.

## Sharing

Now it's time to share our new creation with the world by:

- Creating a [git repo and pushing the code online](#)
- [Publishing the module to NPM](#)
- Creating a [project page](#) for it
- [Submitting](#) it to the [third-party module list](#)





*Finished screen module*

## Resource List

To help you get started creating your own custom modules, here is a list of the resources we used to put this tutorial together.

### Power

- [Powering Your Tessel](#)
- [Level Shifting](#)

### Communication

- [Tessel Module Communication Protocols](#)

### Software

- [Making a Tessel-style Library for Third-Party Hardware](#)
- [Tessel Hardware API](#)
- [All first-party module code on Github](#)

### Documentation

- [Git Repo Template](#)

- [README.md Template](#)

## Sharing

- [Publishing to NPM Tutorial](#)
- [package.json Standard](#)
- [Tessel Project Page](#)
- [Third-Party Module Submission Form](#)

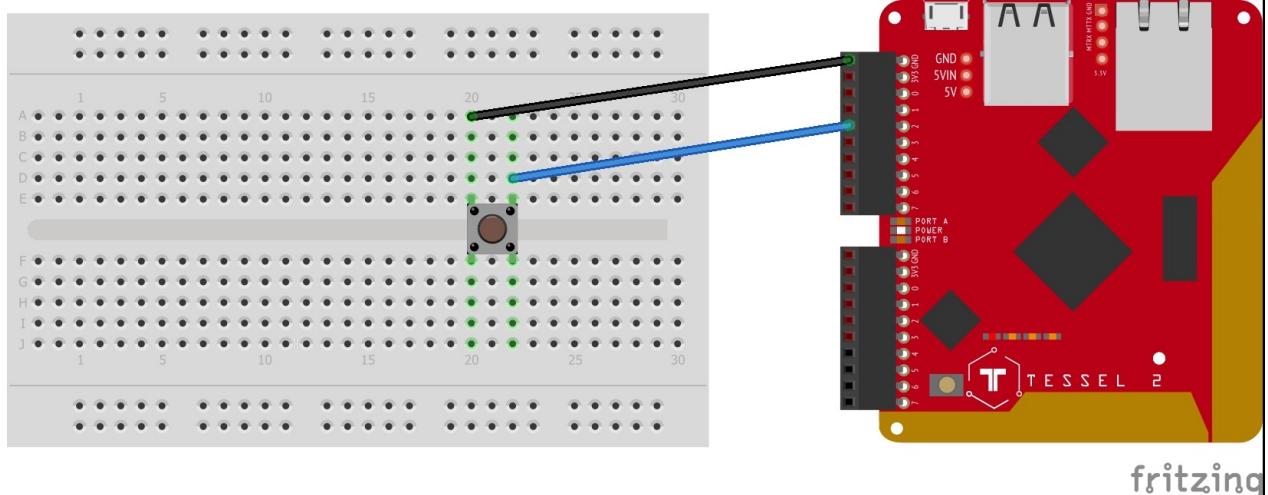


# Pull- Pins

Suppose a pin is configured as an input. If nothing is connected to the pin and the program tries to read the state of the pin, it would be in a 'floating' state i.e an unknown state. To prevent this, a **pull-up** or a **pull-down** state is defined. They are often used in the case of Buttons and Switches.

**Pins 2-7** on both the **Ports** are available for **pull-up** and **pull-down**.

Take a look at the following circuit. The code example given below turns on the Blue **LED** of the **Tessel module** when the pushbutton is pressed and turns off the Blue **LED** when the pushbutton is released.



```
var tessel= require('tessel'); // Import Tessel

var pin = tessel.port.A.pin[2]; // Select pin 2 on port A

var pullType = "pullup"; // Set the mode of `pin.pull` to pullup

var led = tessel.led[3]; // Blue LED of Tessel

pin.pull(pullType,(error, buffer) => { // Pin 2 pulled high

 if (error){
 throw error;
 }

 setInterval(() => {
 pin.read(function(error, valReturned){
 // valReturned is the digital value that is returned from the pin

 if (error) {
 throw error;
 }

 // Pin 2 reads high when the pushbutton is not pressed since it is pulled up
 console.log(valReturned);
 if (valReturned == 1){
 led.off();
 }

 // Pin 2 reads low when the pushbutton is pressed since its connection with gr
 ound is completed
 else{
 led.on();
 }

 });

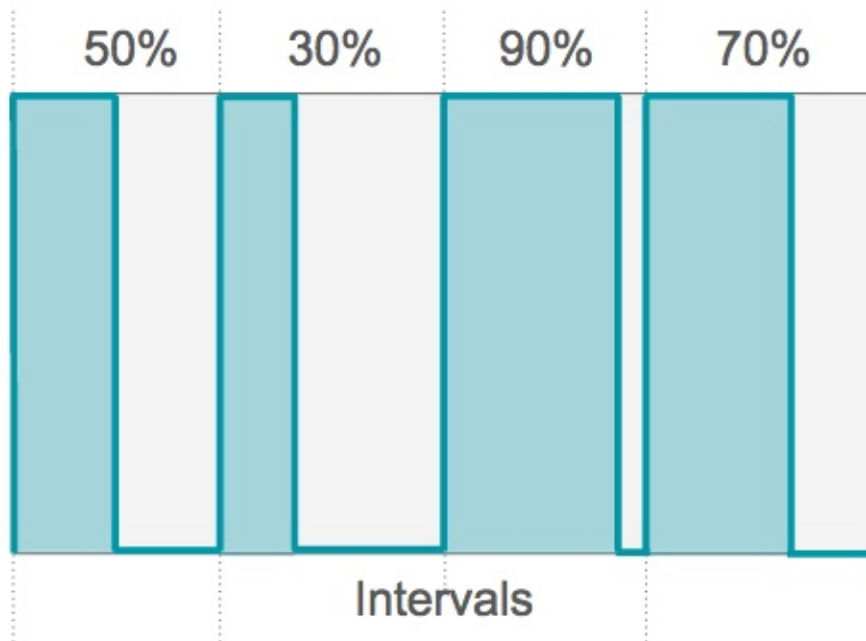
 }, 500);

});
```

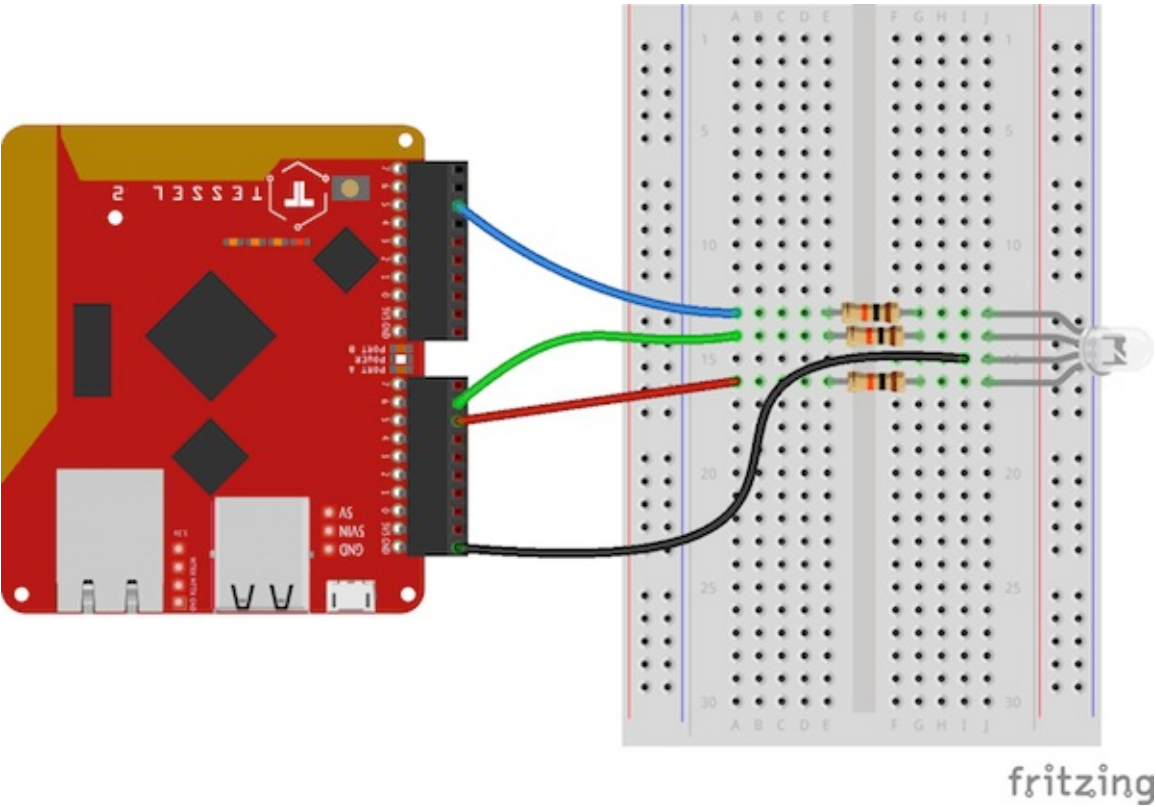
[More information on Pull-up pins and resistors \(Sparkfun Tutorials\)](#)

## Pulse-width Modulation (PWM)

**PWM pins** are pulse-width modulated **pins**. Essentially, **PWM** is a **digital** signal that spends between 0% and 100% of its time pulled high/on (this is its "duty cycle"). You can set the **PWM pins** to any value between 0 (0%) and 1 (100%) to approximate an **analog** signal. **PWM** is often used to control servo speeds or **LED** brightness.



In the following example, the Tessel will change the color of an RGB **LED** by changing the duty cycle sent to each color pin over a set interval of time. By changing the brightness of each internal **LED** (red, green, blue) per cycle, the combined color of the RGB component is changed.



```
var tessel = require('tessel'); // Import tessel

var portA = tessel.port.A; // Select port A
var portB = tessel.port.B; // Select port B

var redPin = portA.pwm[0]; // Select the first PWM pin on port A, equivalent to portA.
pin[5]
var greenPin = portA.pwm[1];
var bluePin = portB.pwm[0];

// The starting values of each color out of 255
var red = 0;
var green = 0;
var blue = 0;

// Use this to increment the color value without exceeding 255
function stepColor (value, step) {
 value += step; // Add the step count to the existing value

 // If the value exceeds 255, then reset it to 0
 if (value > 255) {
 value = 0;
 }

 return value;
}

// Set the signal frequency to 1000 Hz, or 1000 cycles per second
// This the rate at which Tessel will send the PWM signals
// This is program specific
tessel.pwmFrequency(1000);

// Create a loop to run a function at a set interval of time
setInterval(function() {
 // Increment each color at a unique step
 red = stepColor(red, 10);
 blue = stepColor(blue, 5);
 green = stepColor(green, 20);

 // Set how often each pin is turned on out of 100%
 // Divide the value by 255 to get a value between 0 and 1
 redPin.pwmDutyCycle(red / 255);
 greenPin.pwmDutyCycle(blue / 255);
 bluePin.pwmDutyCycle(green / 255);
}, 500); // Set this function to be called every 500 milliseconds, or every half a second
```

Note: the `pwmFrequency` function *must* be called before `pwmDutyCycle`. Re-setting `pwmFrequency` will disable PWM output until `pwmDutyCycle` is called again.

[More information on pulse-width modulation.](#)

