Lecture #3
Developer Platform

Android Things 2020

Project Sample

- Prerequisites
  - SDK Tools at least 25.0.3.
  - SDK with API 27 or higher.

ANDROID PLATFORM VERSION	API LEVEL	CUMULATIVE DISTRIBUTION
4.0 Ice Cream Sandwich	15	
4.1 Jelly Bean	16	99.6%
4.2 Jelly Bean	17	98.1%
4.3 Jelly Bean	18	95.9%
4.4 KitKat	19	95.3%
5.0 Lollipop	21	85.0%
5.1 Lollipop	22	80.2%
6.0 Marshmallow	23	62.6%
7.0 Nougat	24	37.1%
7.1 Nougat	25	14.2%
8.0 Oreo	26	6.0%
8.1 Oreo	27	1.1%
9.0 Pie	28	

Project Sample

- Prerequisites
  - SDK Tools at least 25.0.3.
  - SDK with API 27 or higher.

ANDROID PLATFORM VERSION	API LEVEL	CUMULATIVE DISTRIBUTION
4.0 Ice Cream Sandwich	15	
4.1 Jelly Bean	16	99.6%
4.2 Jelly Bean	17	98.1%
4.3 Jelly Bean	18	95.9%
4.4 KitKat	19	95.3%
5.0 Lollipop	21	85.0%
5.1 Lollipop	22	80.2%
6.0 Marshmallow	23	62.6%
7.0 Nougat	24	37.1%
7.1 Nougat	25	14.2%
8.0 Oreo	26	6.0%
8.1 Oreo	27	1.1%
9.0 Pie	28	

Project Sample

- Prerequisites
  - SDK Tools at least 25.0.3.
  - SDK with API 27 or higher.

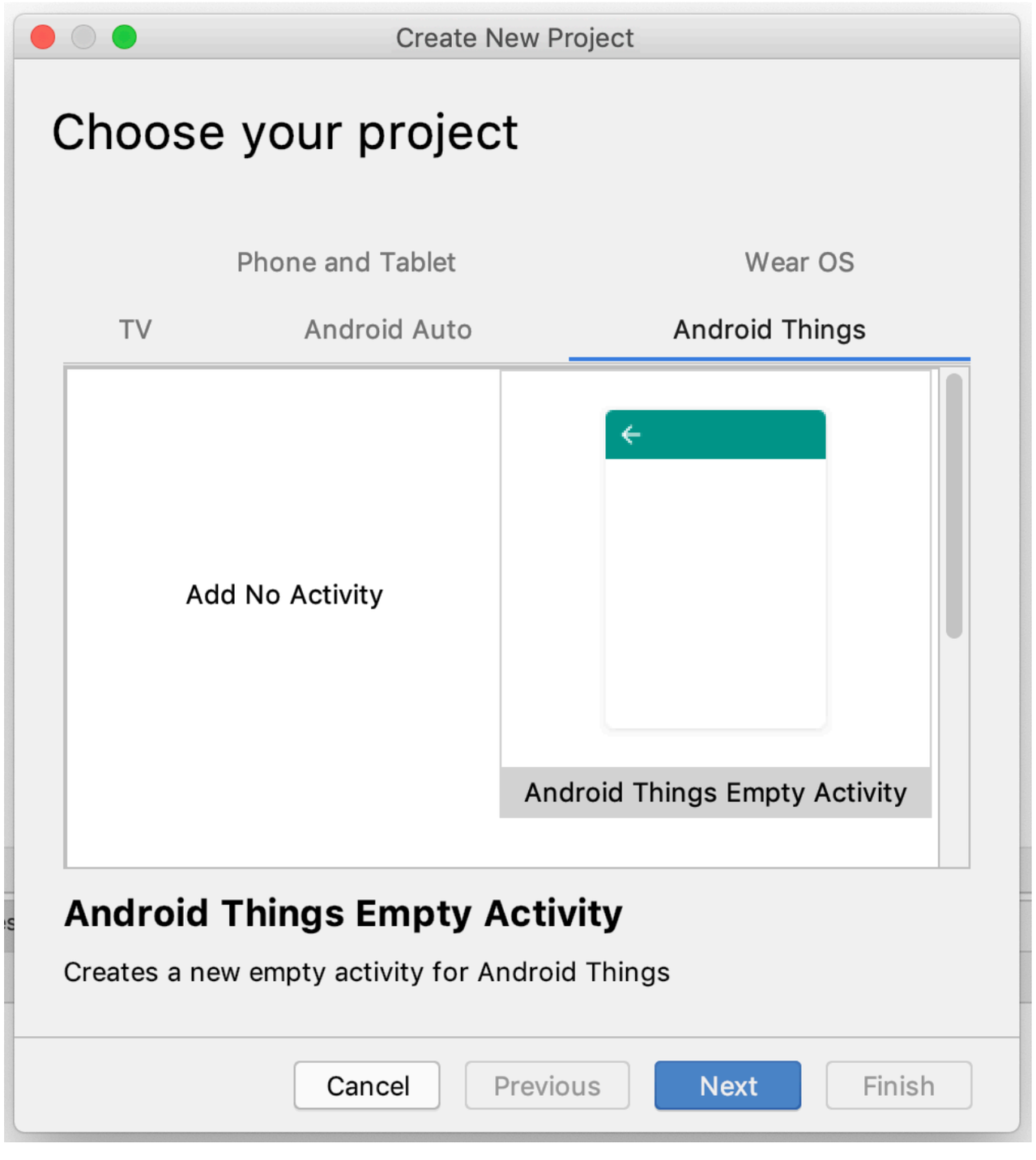
ANDROID PLATFORM VERSION	API LEVEL	CUMULATIVE DISTRIBUTION
4.0 Ice Cream Sandwich	15	
4.1 Jelly Bean	16	99.6%
4.2 Jelly Bean	17	98.1%
4.3 Jelly Bean	18	95.9%
4.4 KitKat	19	95.3%
5.0 Lollipop	21	85.0%
5.1 Lollipop	22	80.2%
6.0 Marshmallow	23	62.6%
7.0 Nougat	24	37.1% Minimum
7.1 Nougat	25	14.2%
8.0 Oreo	26	6.0%
8.1 Oreo	27	1.1%
9.0 Pie	28	

Project Sample

- Prerequisites
  - SDK Tools at least 25.0.3.
  - SDK with API 27 or higher.

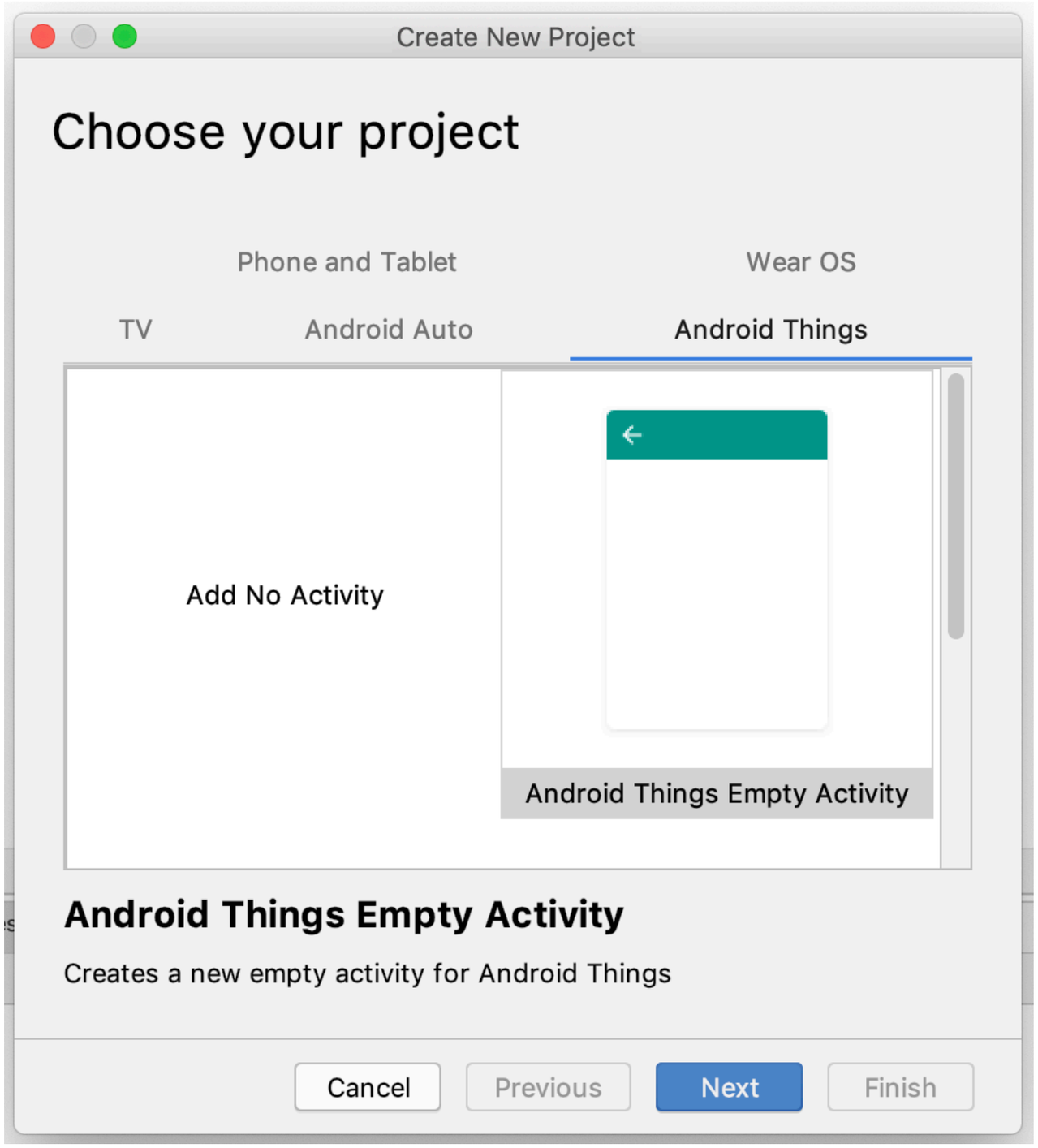
ANDROID PLATFORM VERSION	API LEVEL	CUMULATIVE DISTRIBUTION
4.0 Ice Cream Sandwich	15	
4.1 Jelly Bean	16	99.6%
4.2 Jelly Bean	17	98.1%
4.3 Jelly Bean	18	95.9%
4.4 KitKat	19	95.3%
5.0 Lollipop	21	85.0%
5.1 Lollipop	22	80.2%
6.0 Marshmallow	23	62.6%
7.0 Nougat	24	37.1% Minimum
7.1 Nougat	25	14.2%
8.0 Oreo	26	6.0%
8.1 Oreo	27	1.1% Recommended
9.0 Pie	28	

Create the Project



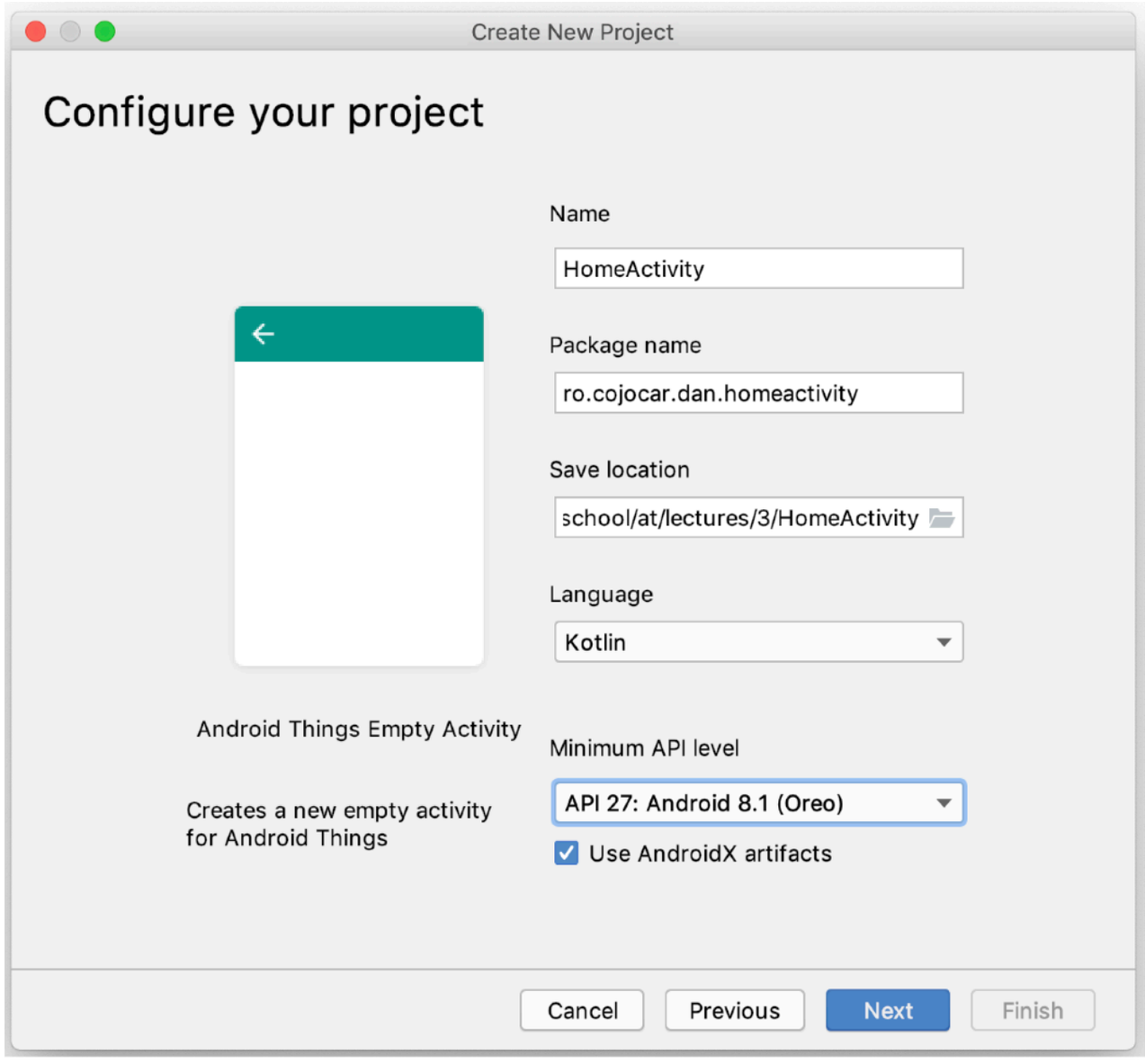
Create the Project

- Select **Android Things** as the only form factor.



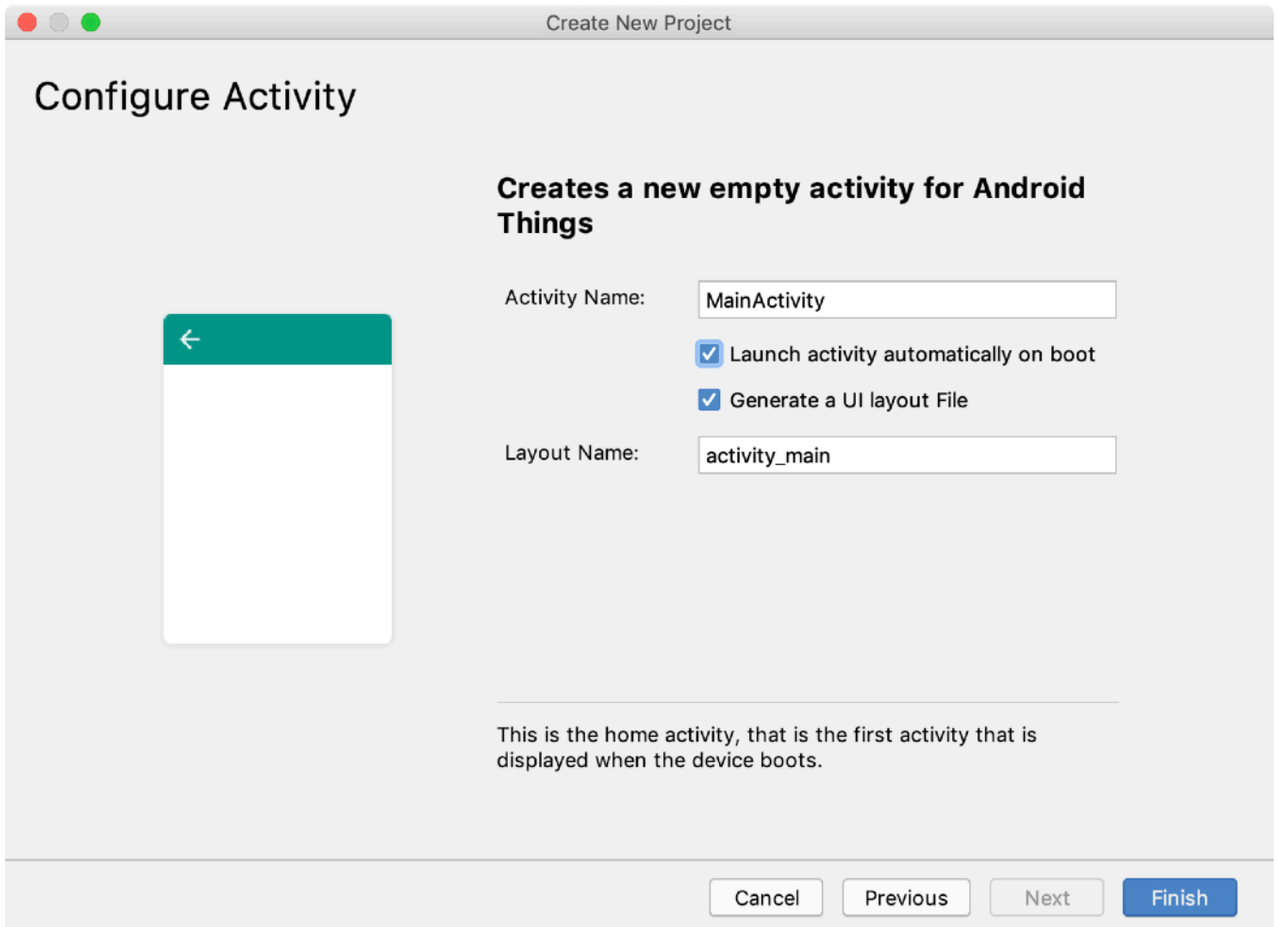
Create the Project

- Select **Android Things** as the only form factor.
- Ensure that API 27 is selected.
- Check AndroidX artifact.



Create the Project

- Select **Android Things** as the only form factor.
- Ensure that API 27 is selected.
- Check AndroidX artifact.
- Ensure that the activity will start automatically on boot



Key Generated Changes

- Build.gradle changes.

```
dependencies {
  ...
  compileOnly 'com.google.android.things:androidthings:+'
}
```

Key Generated Changes

- Manifest file changes.

```
<application>
  <uses-library android:name="com.google.android.things"/>
  <activity android:name=".HomeActivity">
    <!-- Launch activity as default from Android Studio -->
    <intent-filter>
      <action android:name="android.intent.action.MAIN"/>
      <category android:name="android.intent.category.LAUNCHER"/>
    </intent-filter>

    <!-- Launch activity automatically on boot,
         and re-launch if the app terminates. -->
    <intent-filter>
      <action android:name="android.intent.action.MAIN"/>
      <category android:name="android.intent.category.HOME"/>
      <category android:name="android.intent.category.DEFAULT"/>
    </intent-filter>
  </activity>
</application>
```

Key Generated Changes

- Manifest file changes.

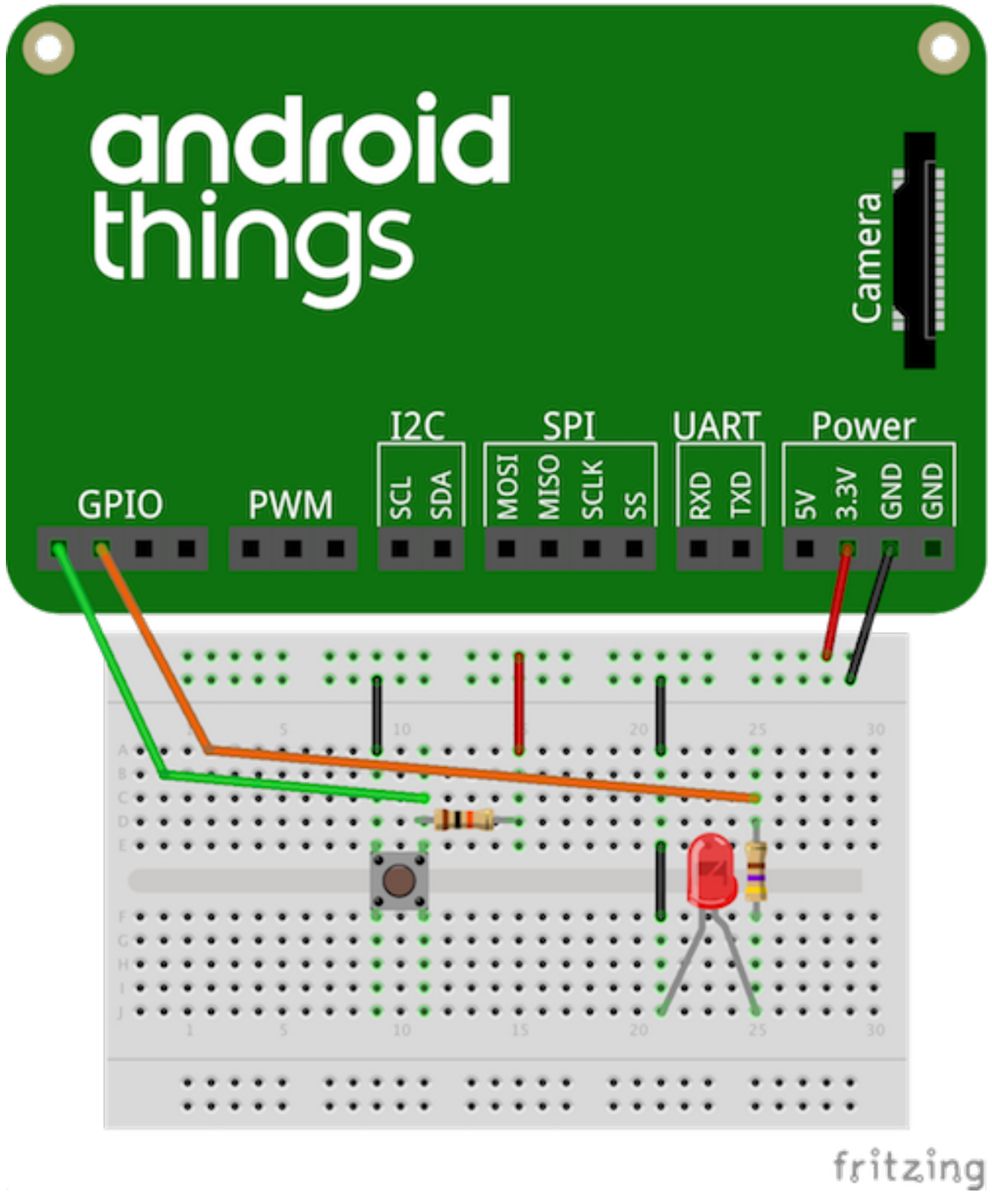
```
<application>
  <uses-library android:name="com.google.android.things"/>
  <activity android:name=".HomeActivity">
    <!-- Launch activity as default from Android Studio -->
    <intent-filter>
      <action android:name="android.intent.action.MAIN"/>
      <category android:name="android.intent.category.LAUNCHER"/>
    </intent-filter>

    <!-- Launch activity automatically on boot,
         and re-launch if the app terminates. -->
    <intent-filter>
      <action android:name="android.intent.action.MAIN"/>
      <category android:name="android.intent.category.HOME"/>
      <category android:name="android.intent.category.DEFAULT"/>
    </intent-filter>
  </activity>
</application>
```

Key Generated Changes

- Manifest file changes.

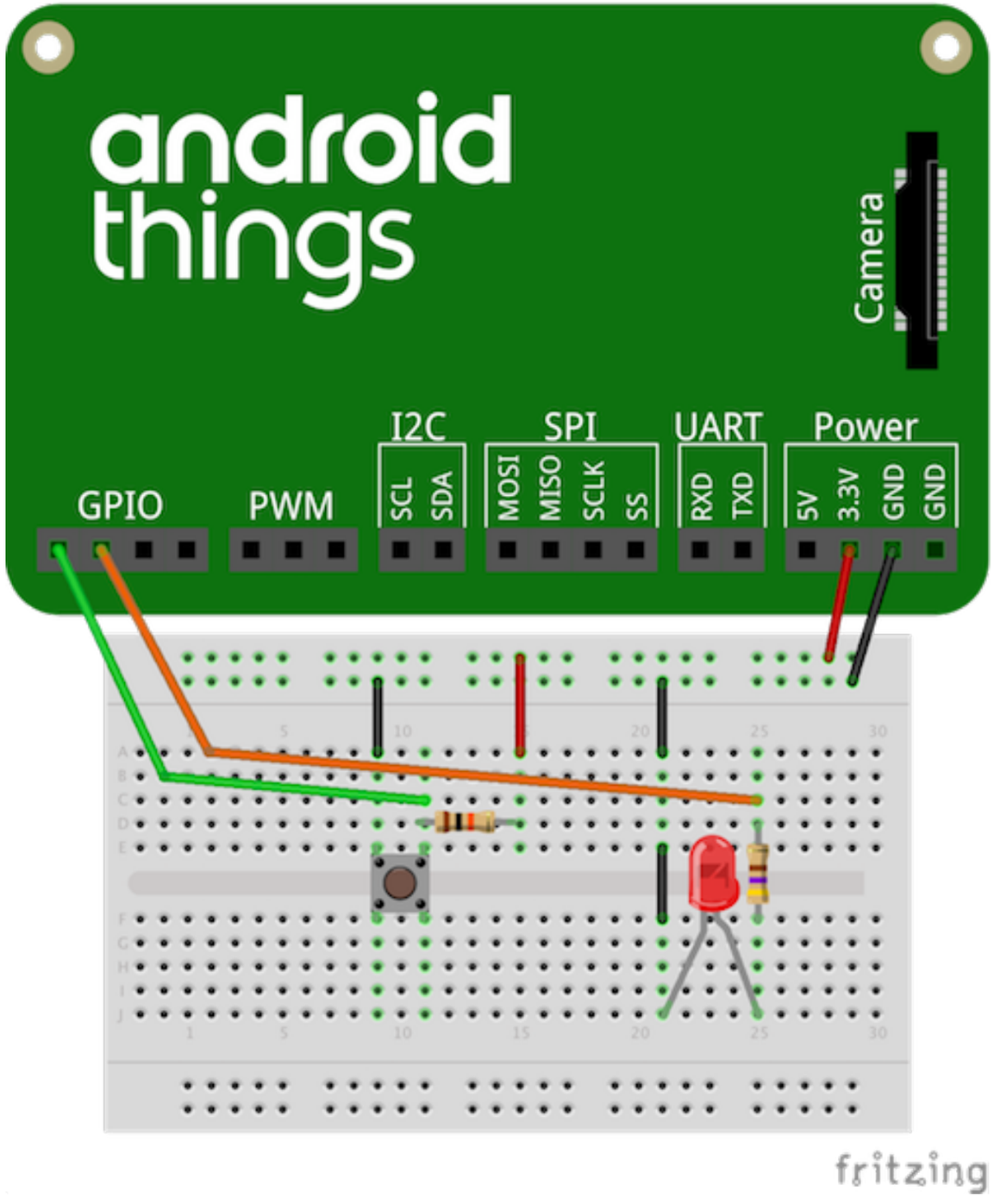
```
<application>
  <uses-library android:name="com.google.android.things"/>
  <activity android:name=".HomeActivity">
    <!-- Launch activity as default from Android Studio -->
    <intent-filter>
      <action android:name="android.intent.action.MAIN"/>
      <category android:name="android.intent.category.LAUNCHER"/>
    </intent-filter>

    <!-- Launch activity automatically on boot,
         and re-launch if the app terminates. -->
    <intent-filter>
      <action android:name="android.intent.action.MAIN"/>
      <category android:name="android.intent.category.HOME"/>
      <category android:name="android.intent.category.DEFAULT"/>
    </intent-filter>
  </activity>
</application>
```

Connect the Hardware



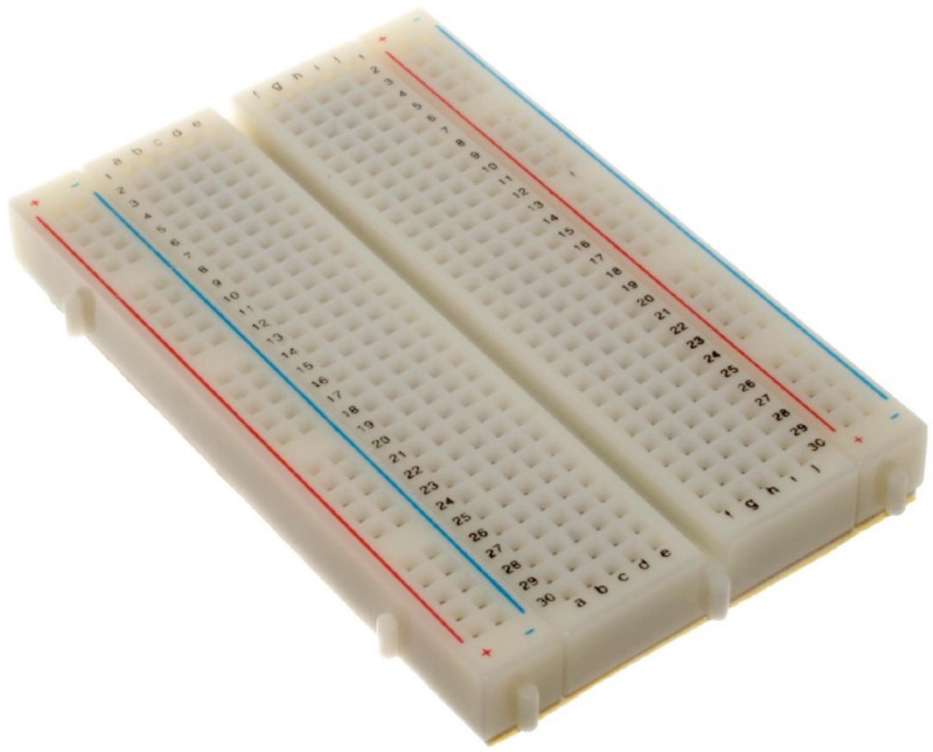
Connect the Hardware

- Requirements:
  - The AndroidThings board.



Connect the Hardware

- Requirements:
  - The AndroidThings board.
  - A breadboard.

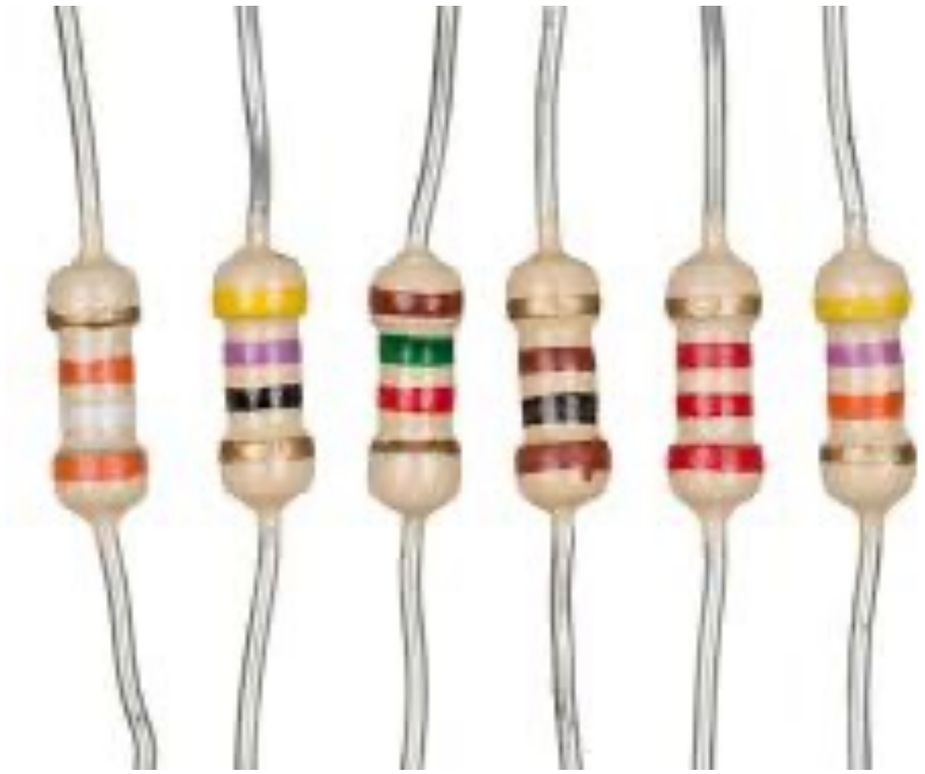


Connect the Hardware

- Requirements:
  - The AndroidThings board.
  - A breadboard.
  - A push button.

Connect the Hardware

- Requirements:
  - The AndroidThings board.
  - A breadboard.
  - A push button.
  - Two resistors.

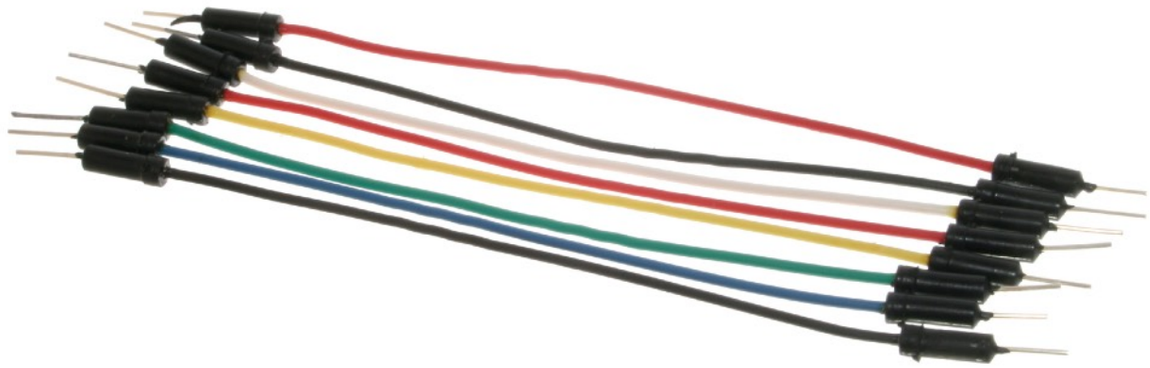


Connect the Hardware

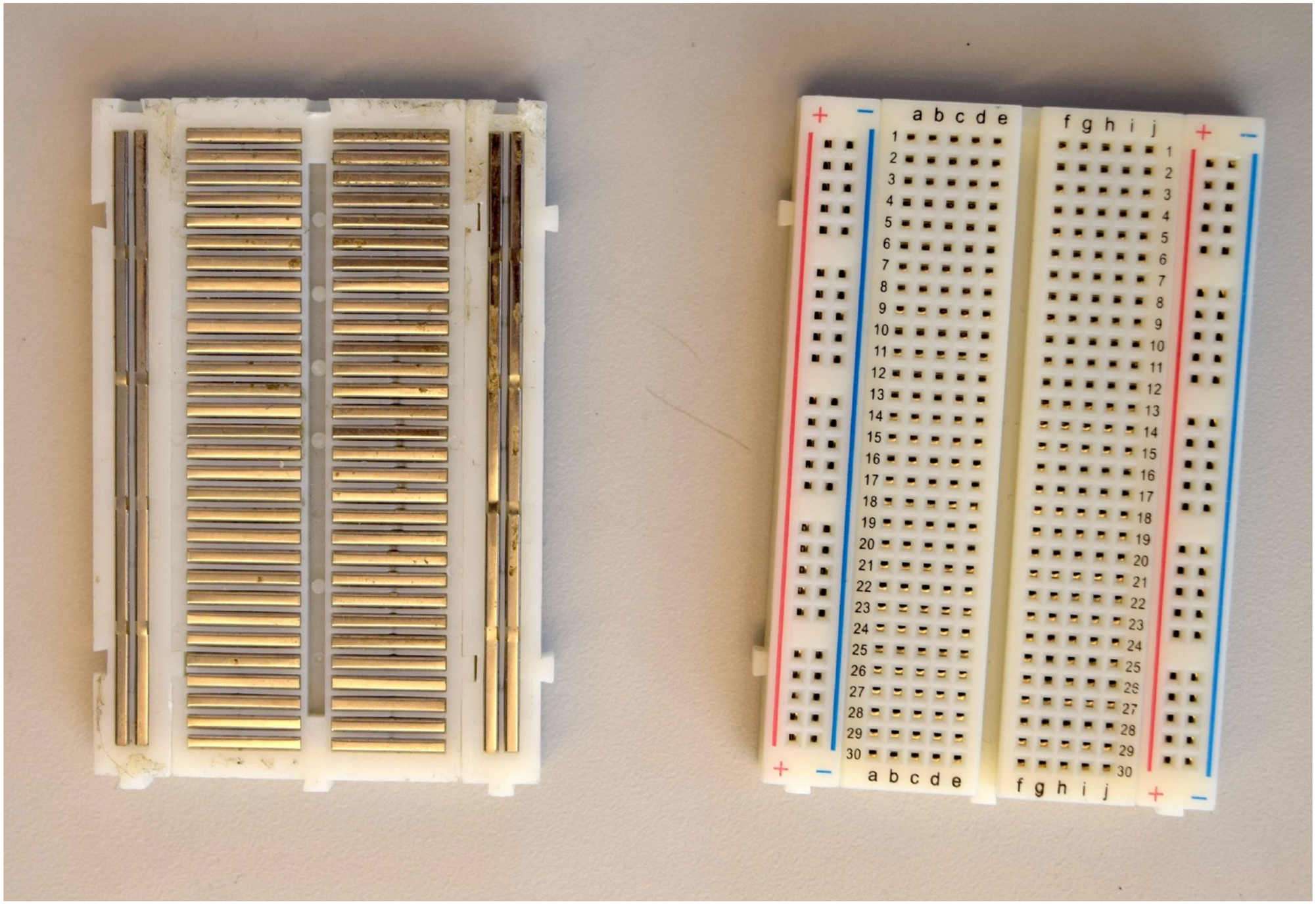
- Requirements:
  - The AndroidThings board.
  - A breadboard.
  - A push button.
  - Two resistors.
  - A LED.

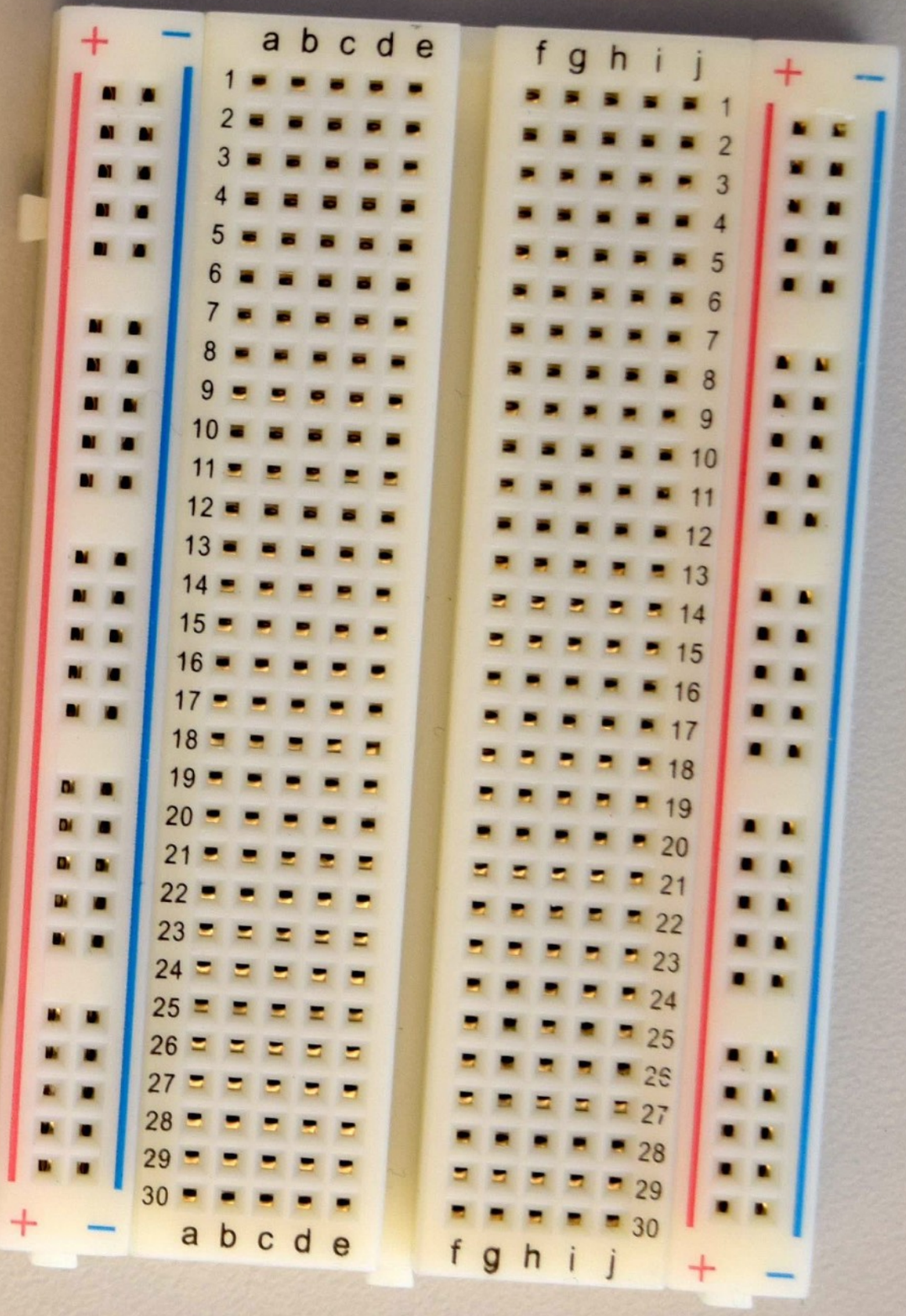
Connect the Hardware

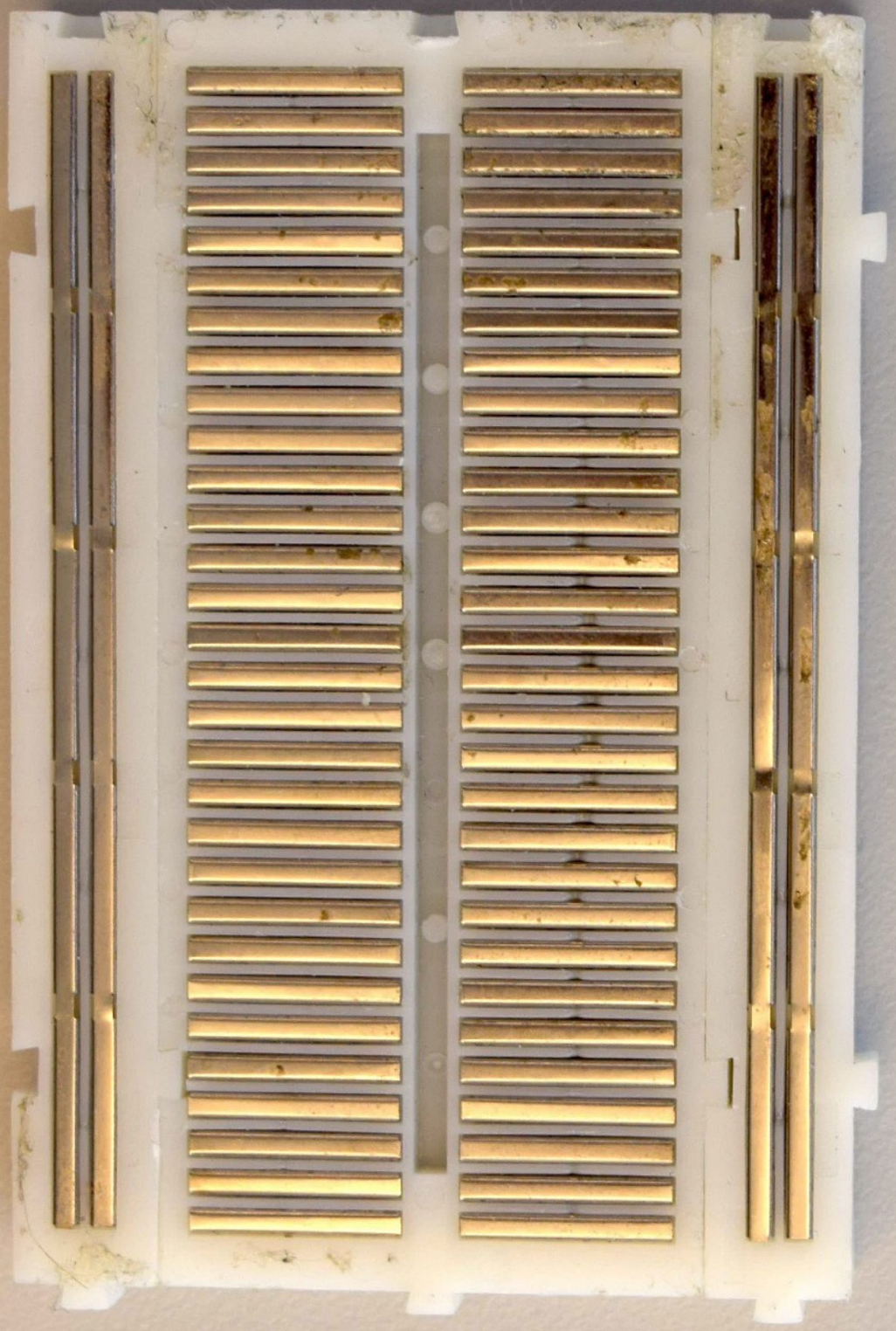
- Requirements:
  - The AndroidThings board.
  - A breadboard.
  - A push button.
  - Two resistors.
  - A LED.
  - Jumper wires.

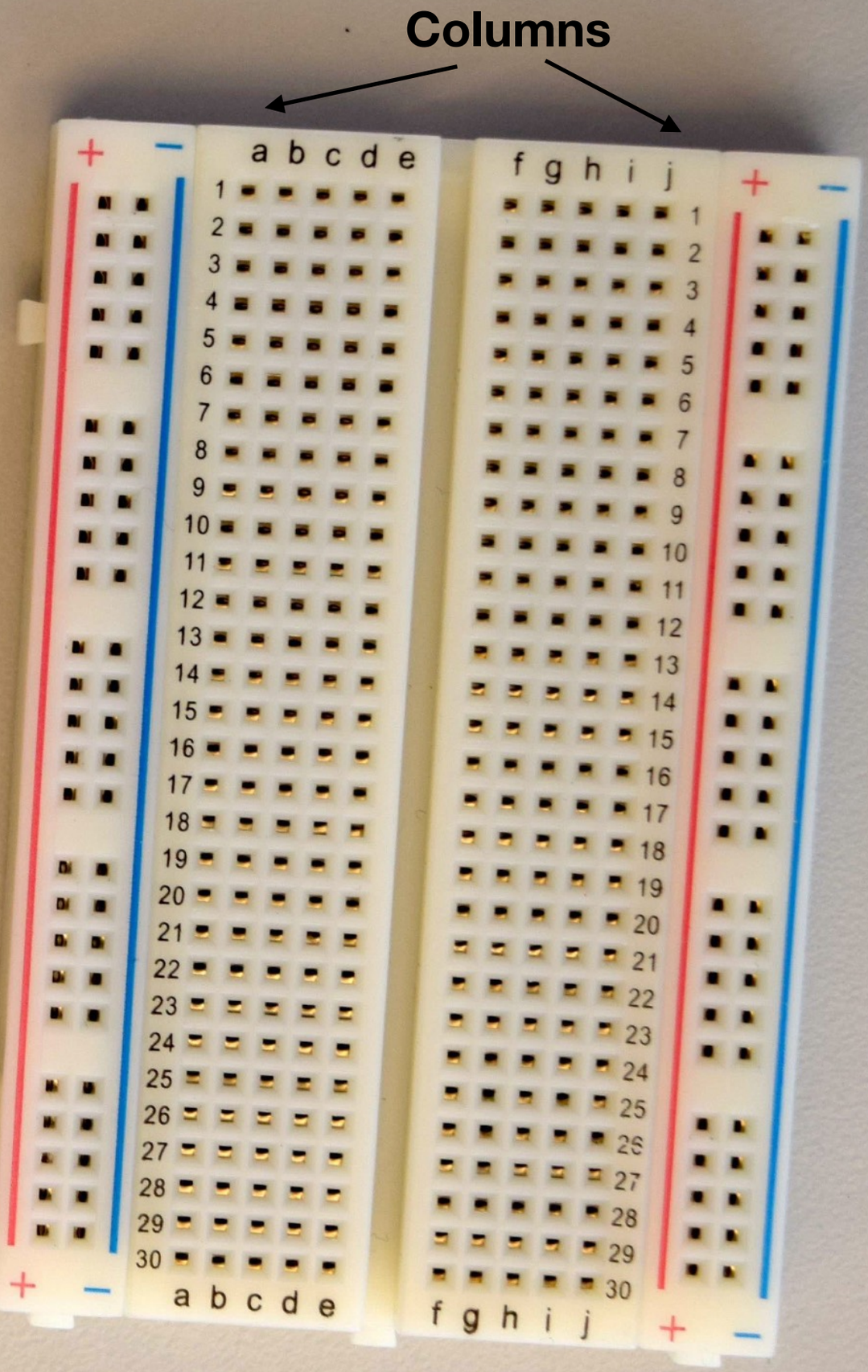


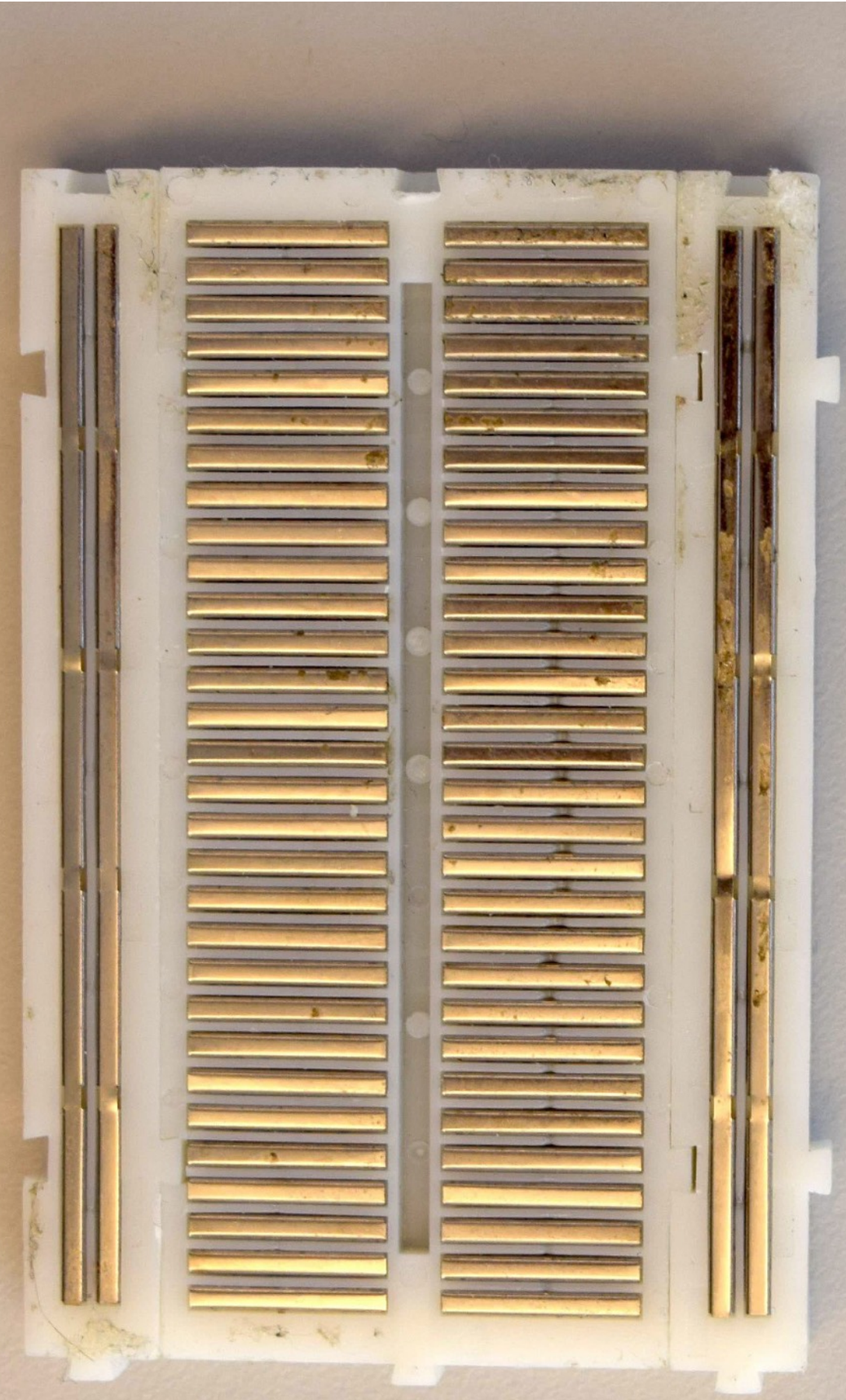
The breadboard



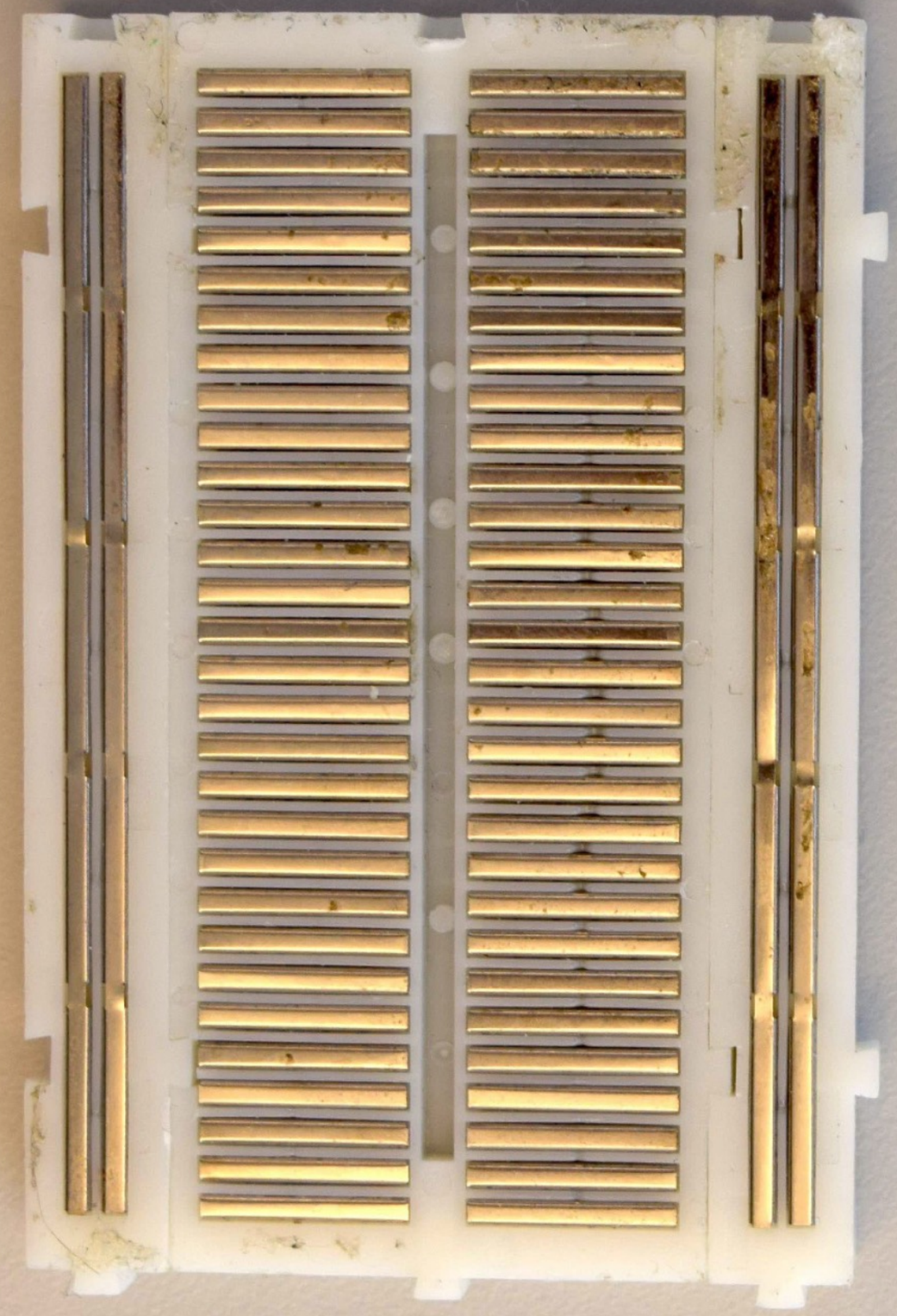

+ −
a b c d e
f g h i j
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30

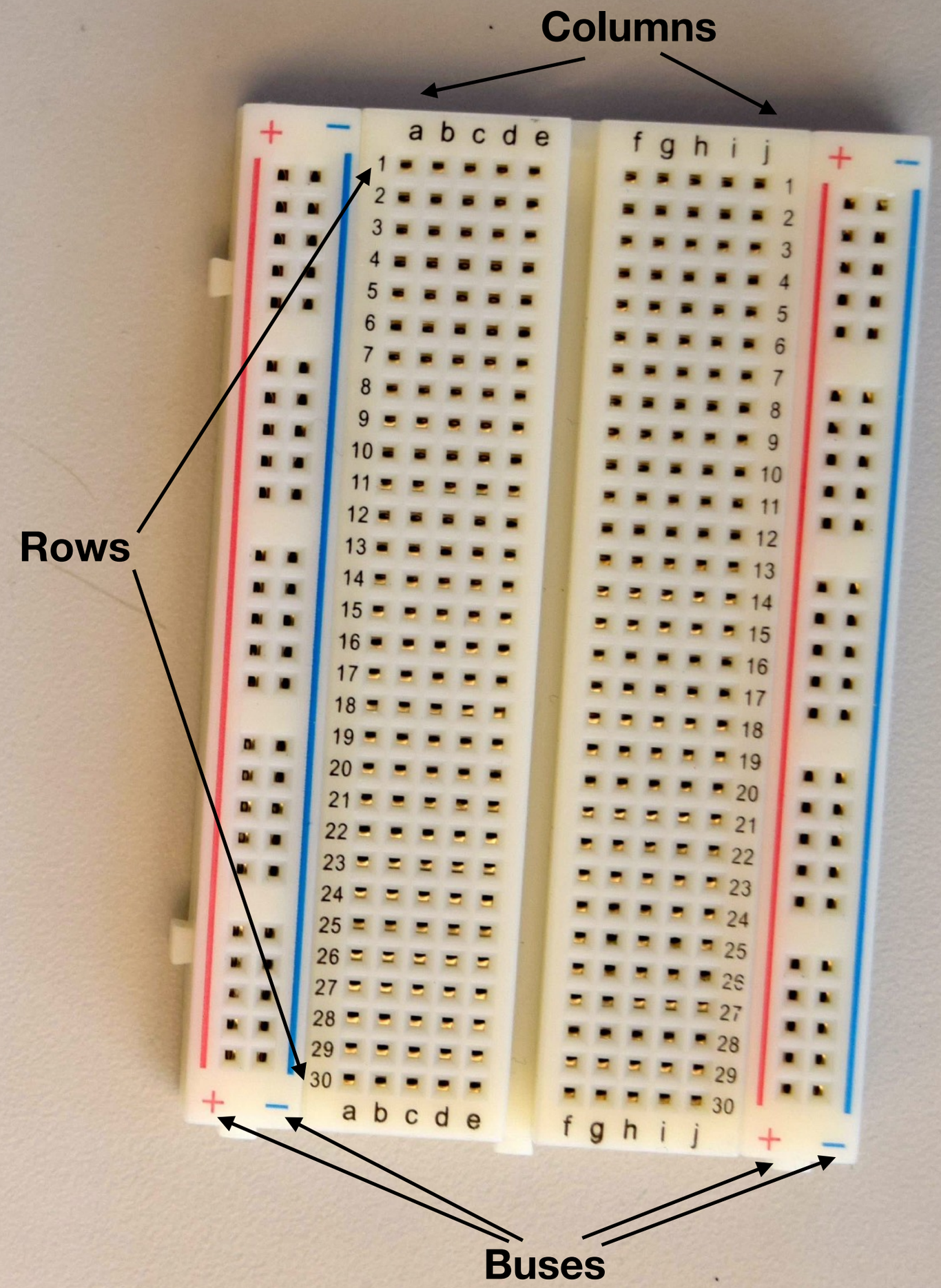



Columns
+
−
a b c d e
f g h i j
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30

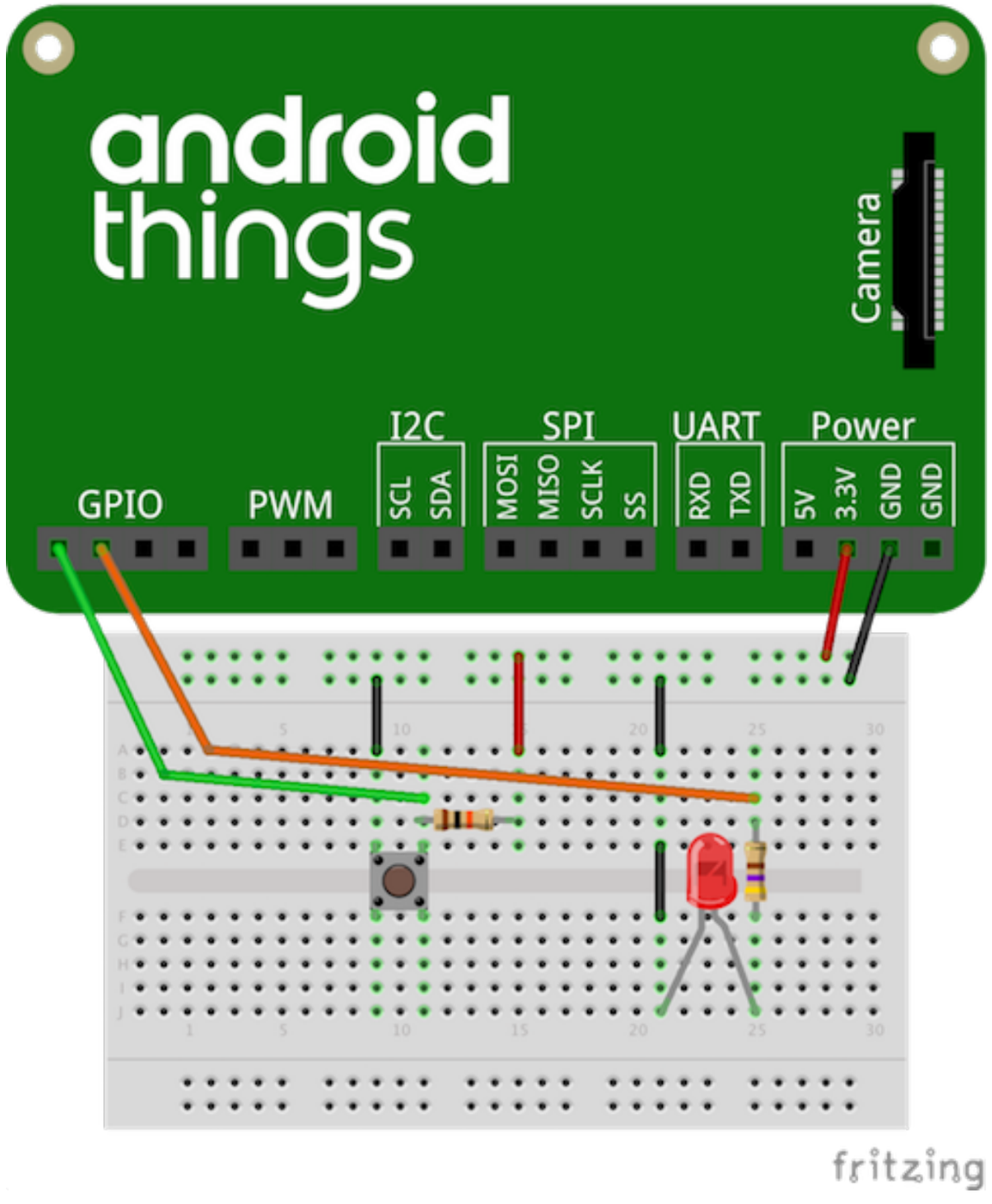


Columns
Rows
a b c d e
f g h i j
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30



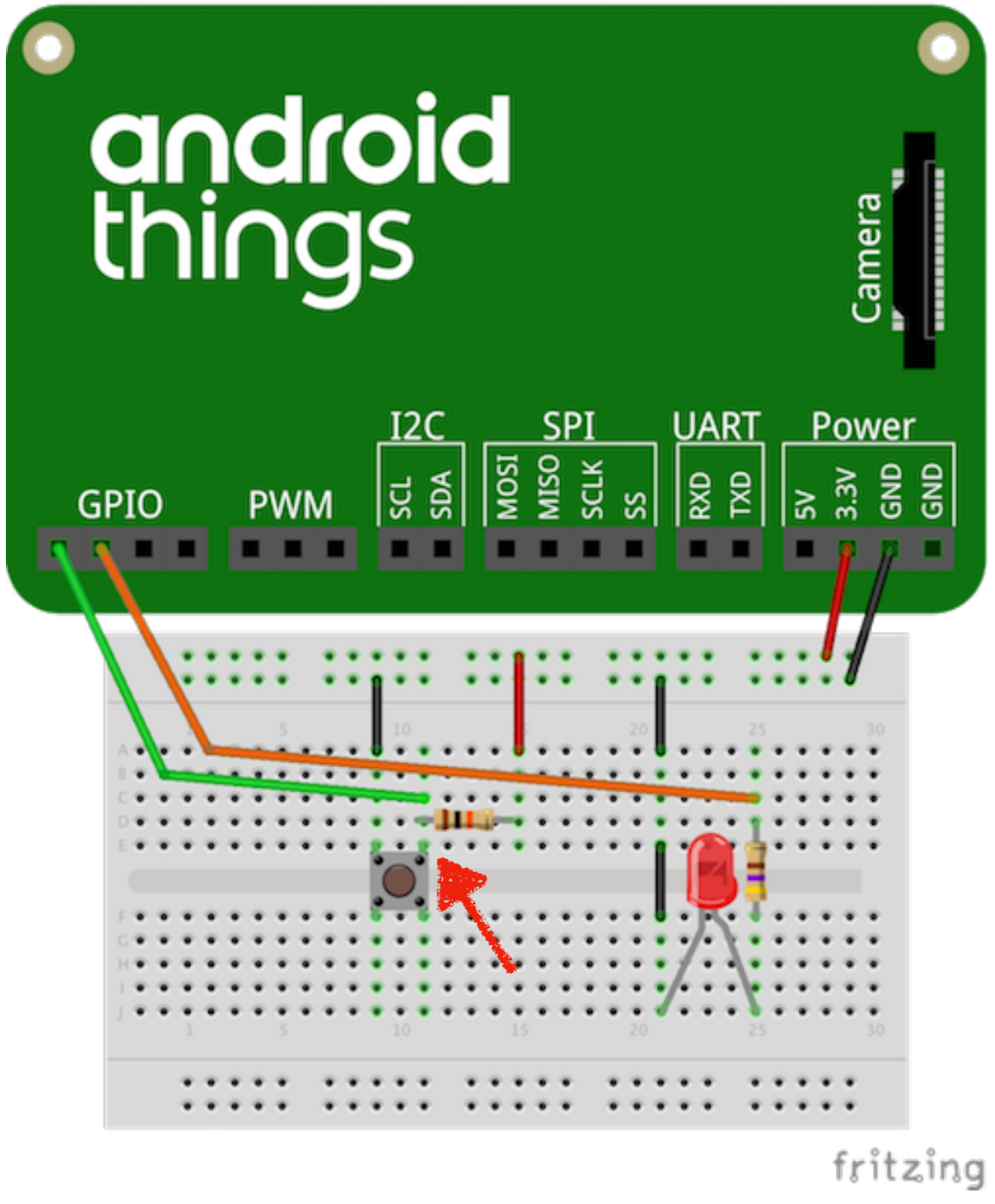

Columns
Rows
Buses
a b c d e
f g h i j
+
−

Create the Connections



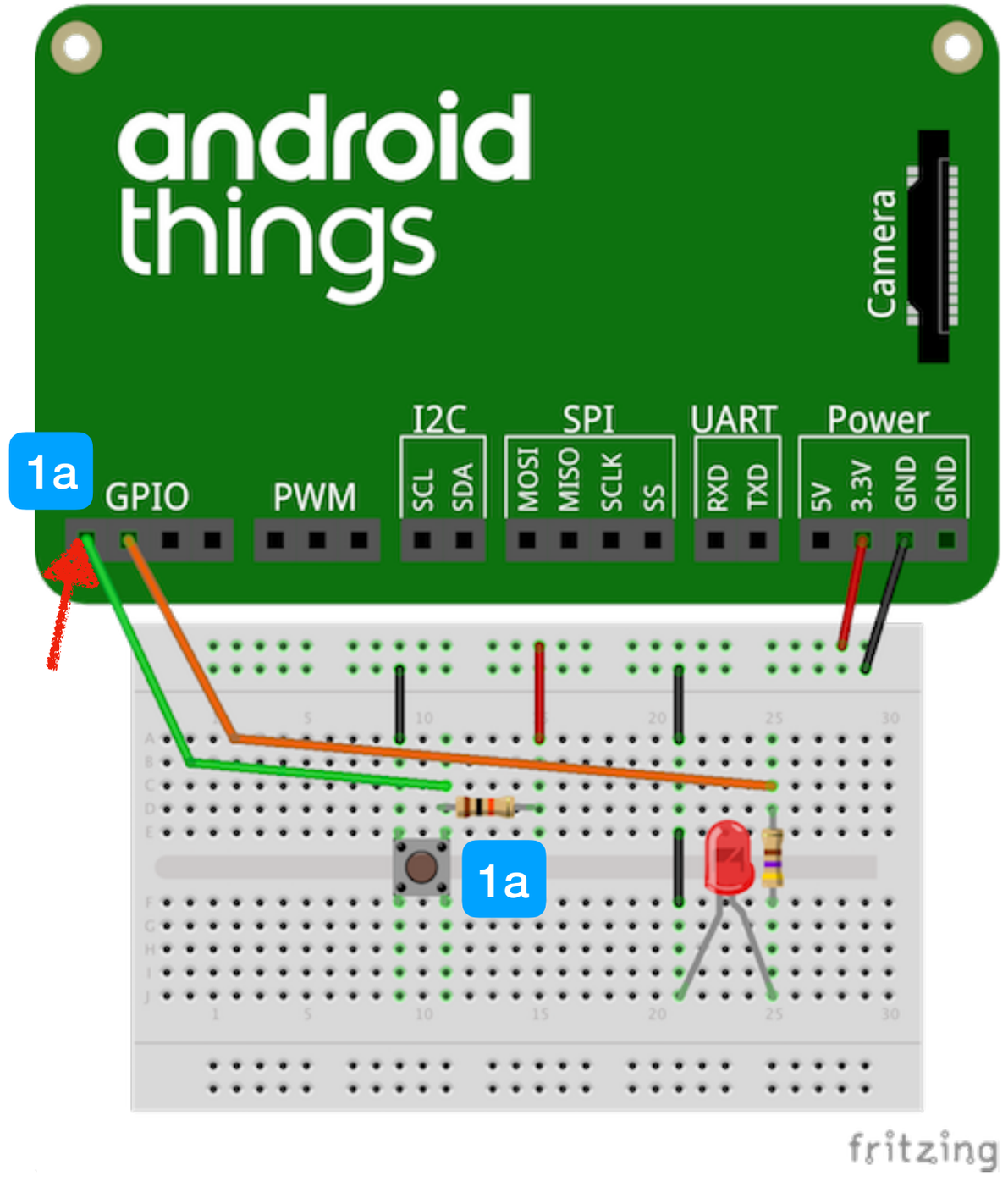
Create the Connections

1. Connect one side of the button to the chosen GPIO input pin, and the other side to ground.



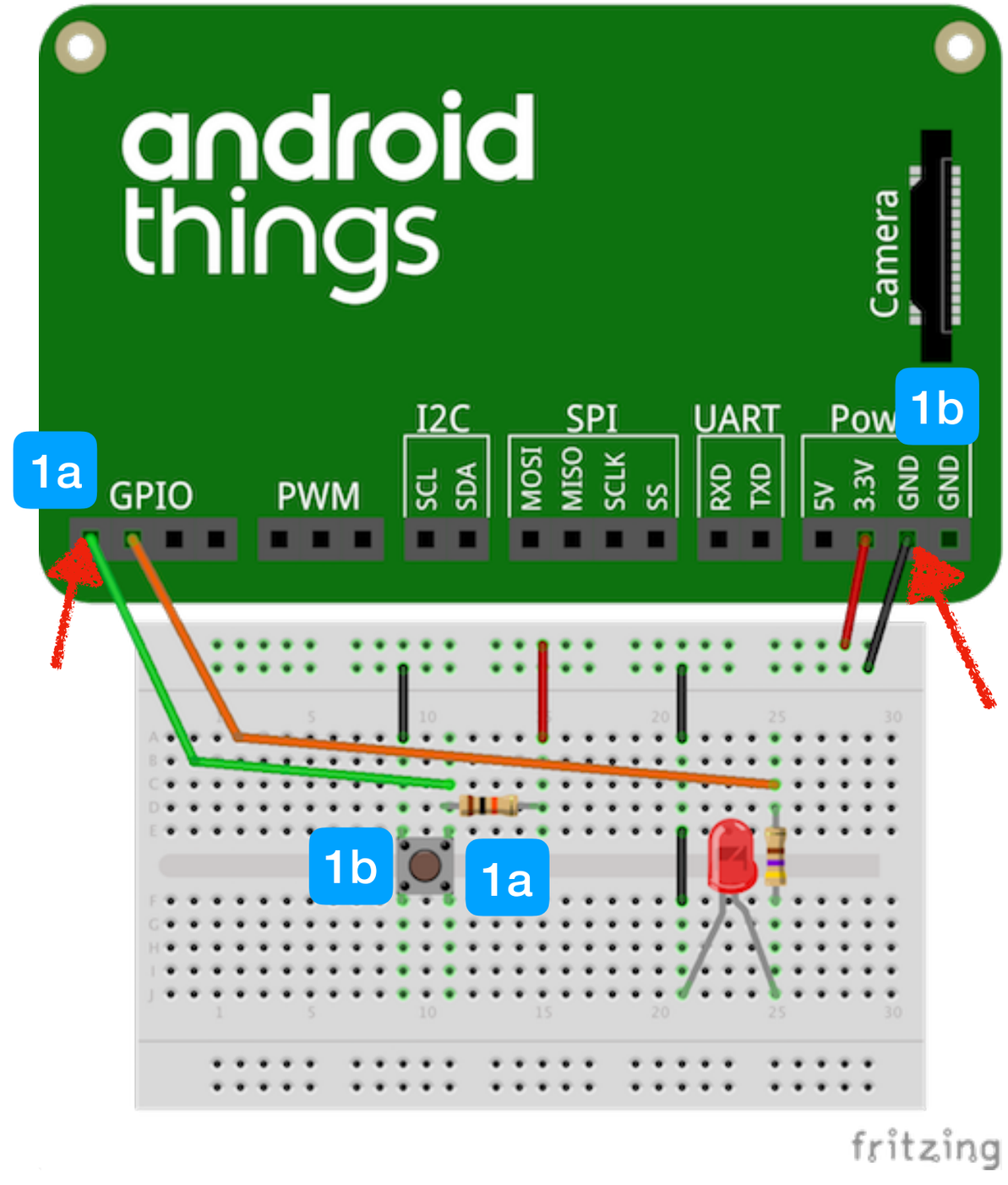
Create the Connections

1. Connect one side of the button to the chosen GPIO input pin, and the other side to ground.



Create the Connections

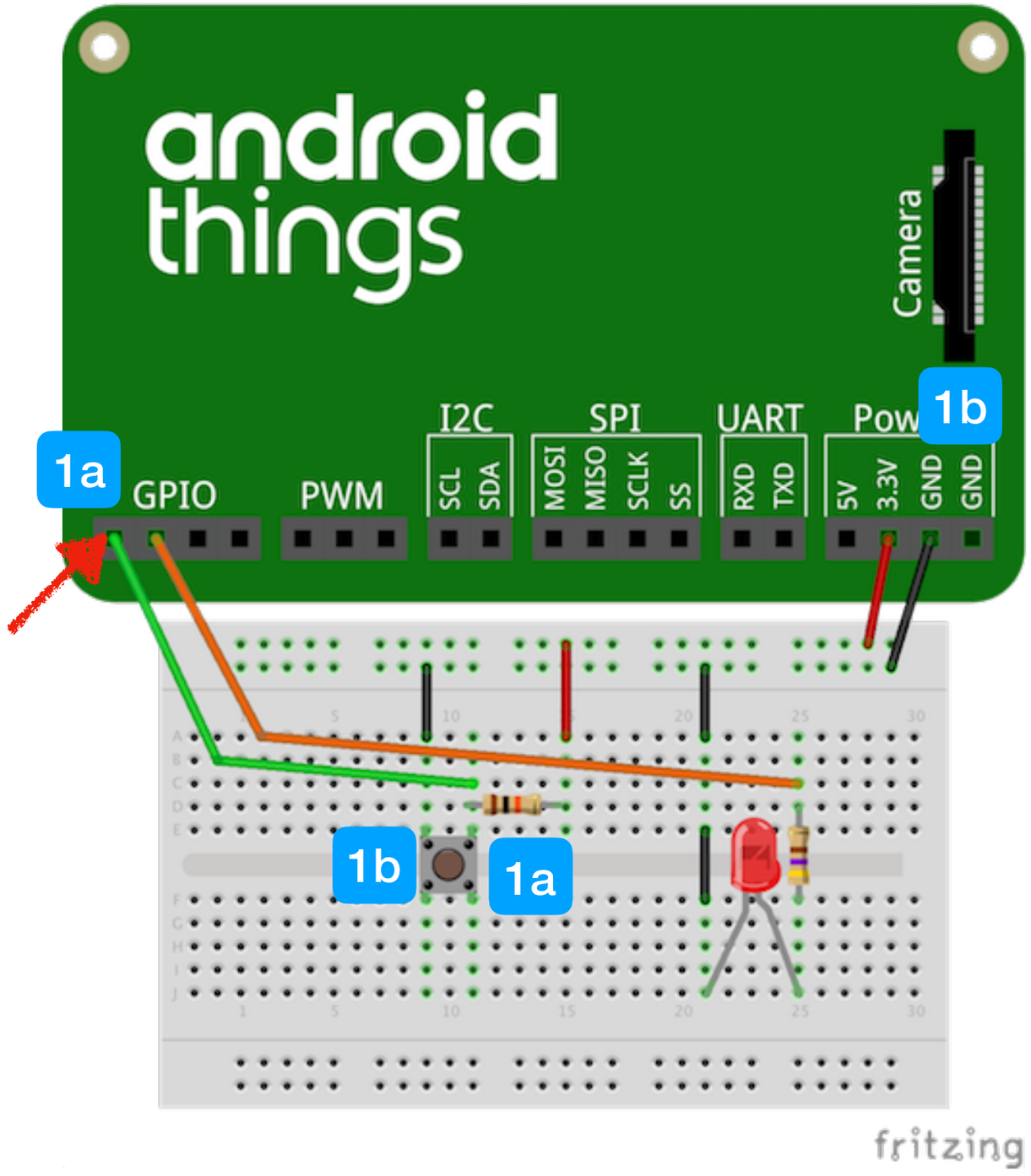
1. Connect one side of the button to the chosen GPIO input pin, and the other side to ground.



Create the Connections

1. Connect one side of the button to the chosen GPIO input pin, and the other side to ground.

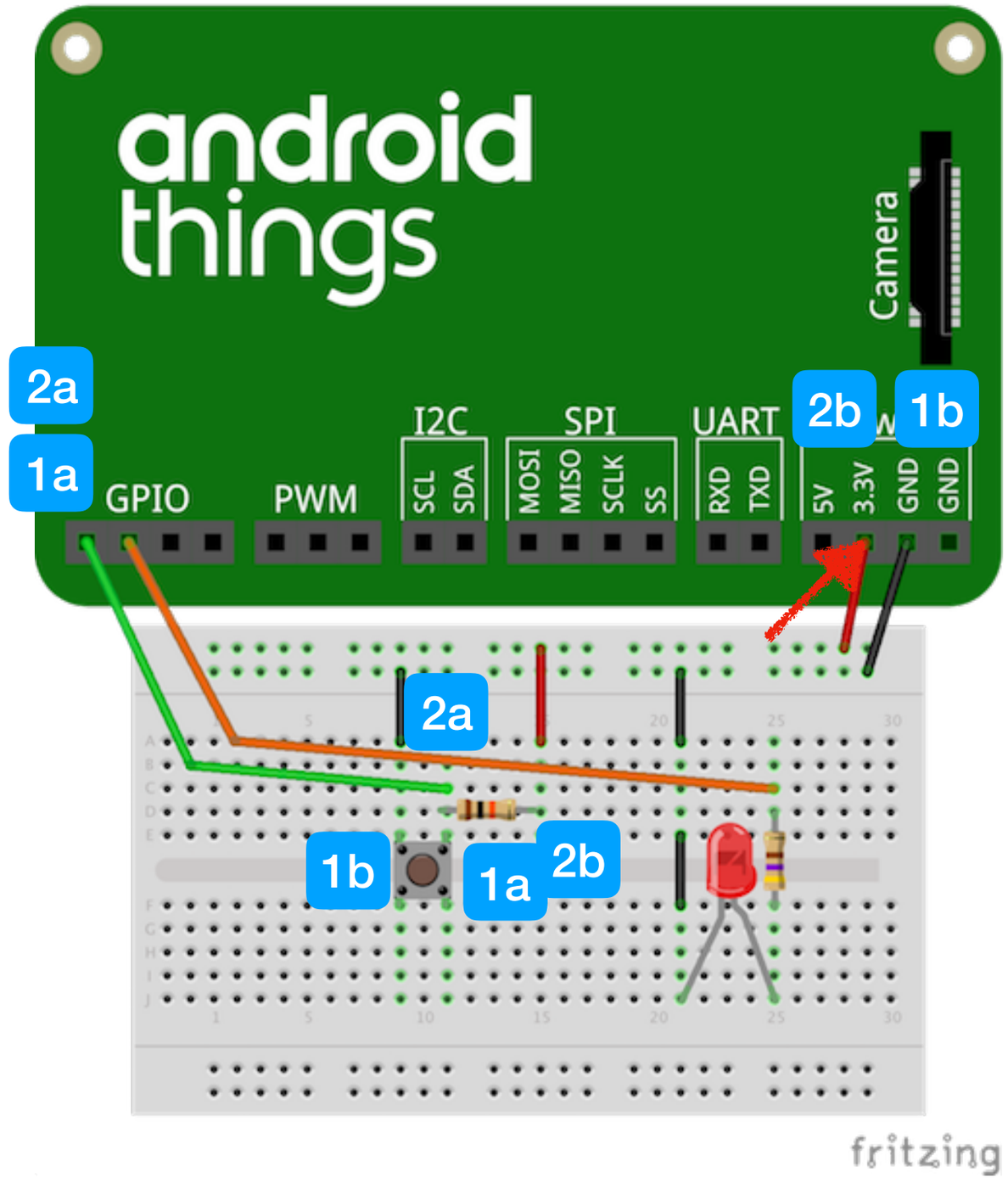
2. Connect the same GPIO input pin to +3.3V through a pull-up resistor.



Create the Connections

1 Connect one side of the button to the chosen GPIO input pin, and the other side to ground.

2. Connect the same GPIO input pin to +3.3V through a pull-up resistor.

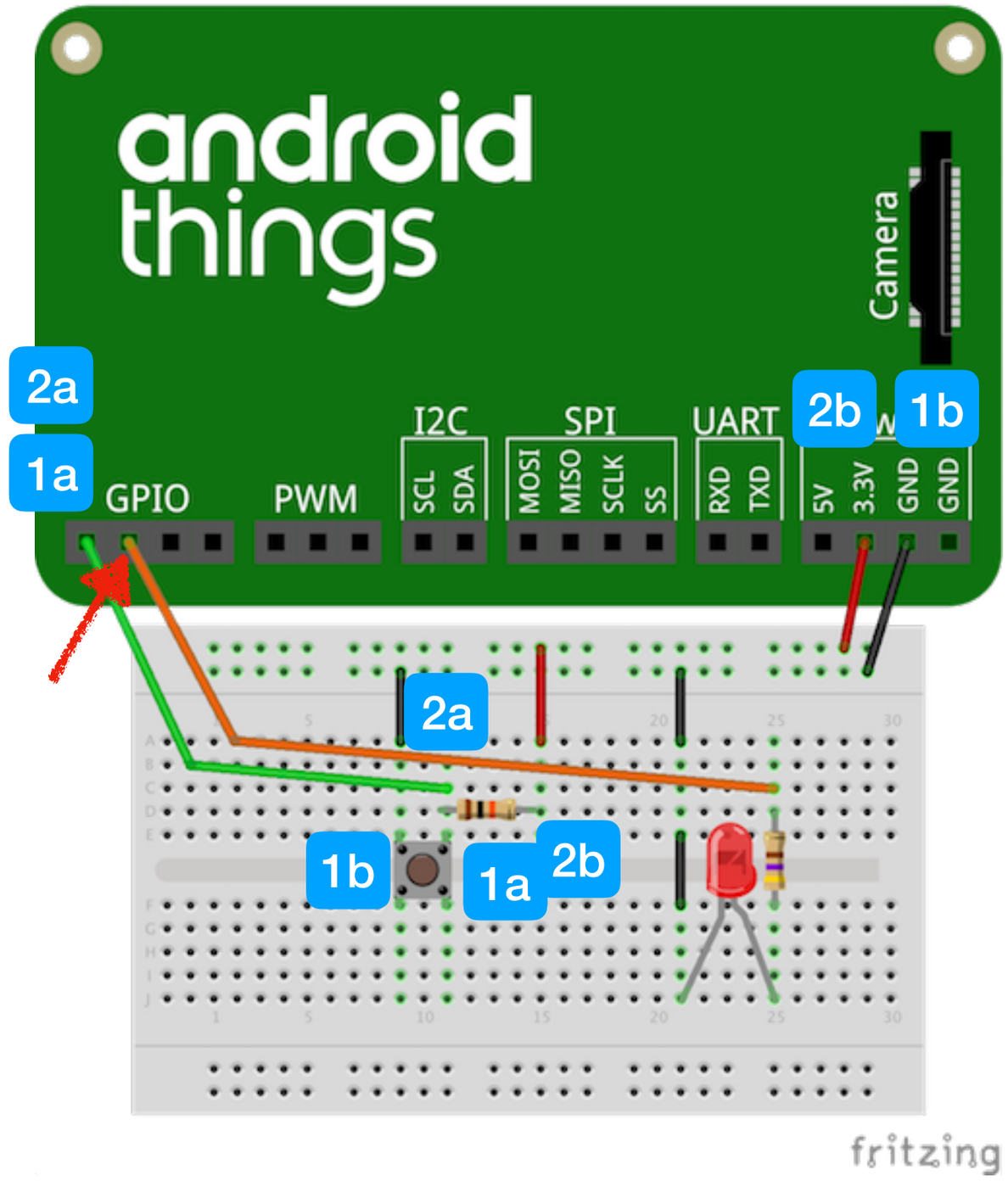


Create the Connections

1. Connect one side of the button to the chosen GPIO input pin, and the other side to ground.

2. Connect the same GPIO input pin to +3.3V through a pull-up resistor.

3. Connect the chosen GPIO output pin to one side of a series resistor.

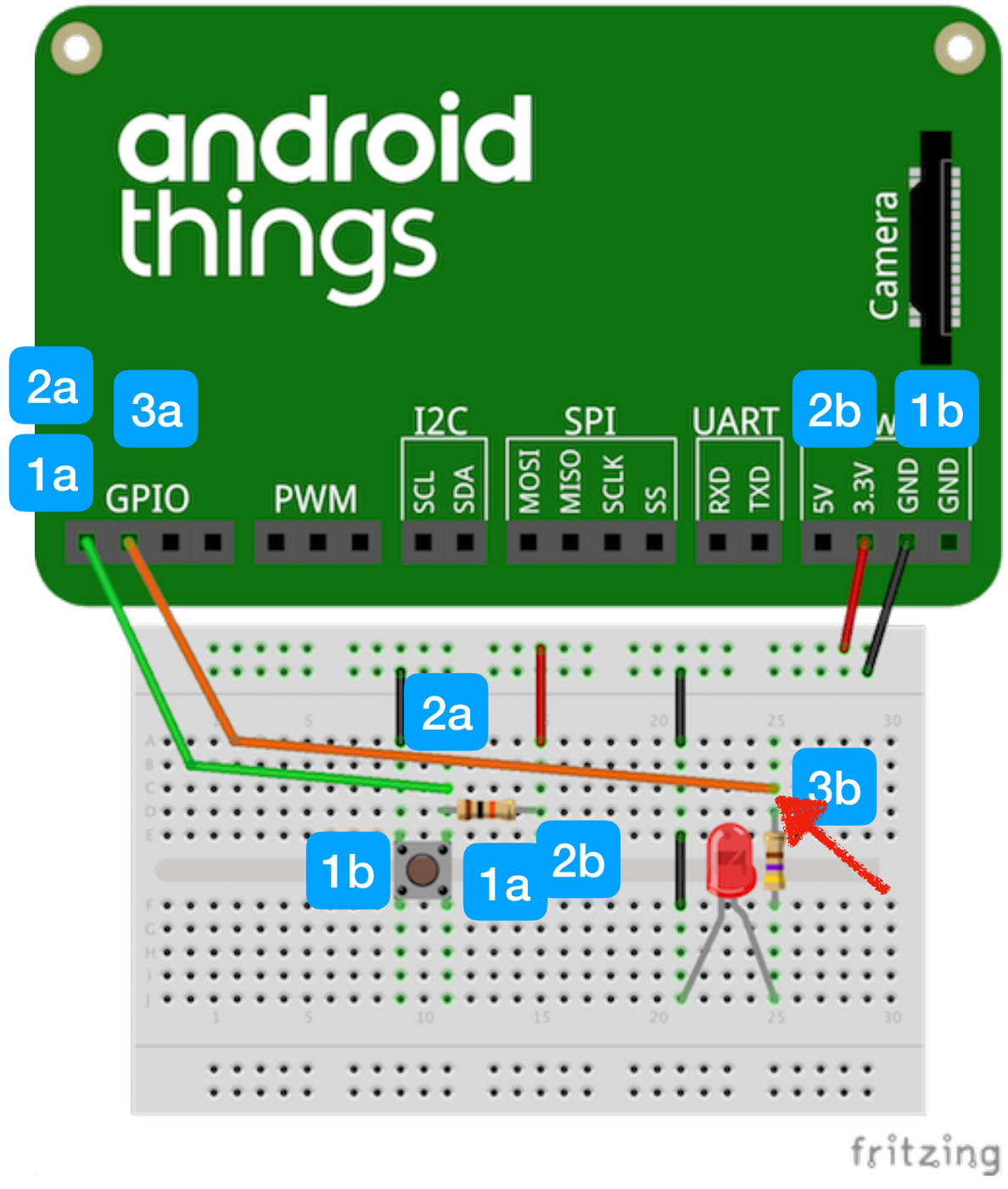


Create the Connections

1. Connect one side of the button to the chosen GPIO input pin, and the other side to ground.

2. Connect the same GPIO input pin to +3.3V through a pull-up resistor.

3. Connect the chosen GPIO output pin to one side of a series resistor.



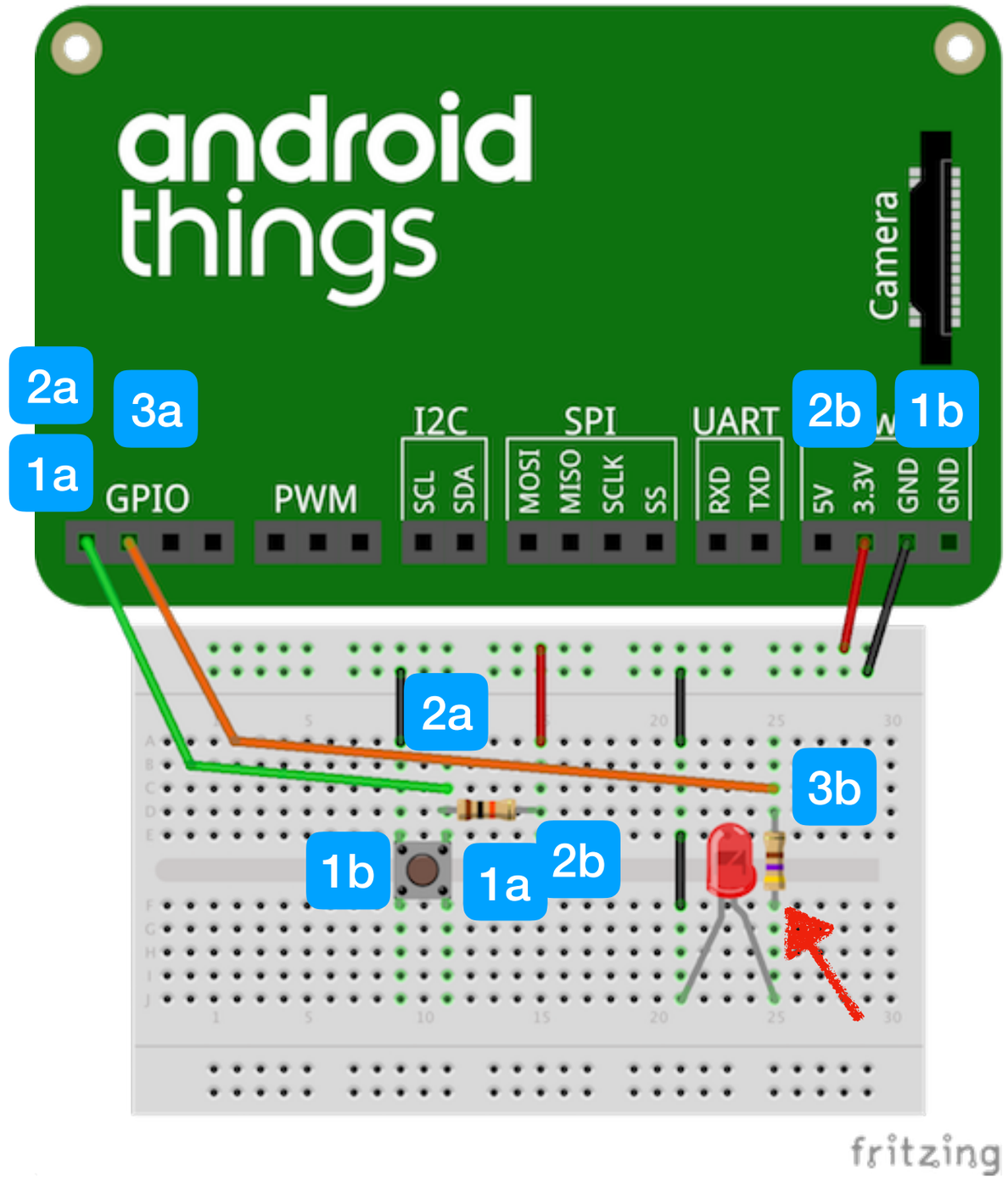
Create the Connections

1. Connect one side of the button to the chosen GPIO input pin, and the other side to ground.

2. Connect the same GPIO input pin to +3.3V through a pull-up resistor.

3. Connect the chosen GPIO output pin to one side of a series resistor.

4. Connect the other side of the resistor to the anode side (longer lead) of the LED.



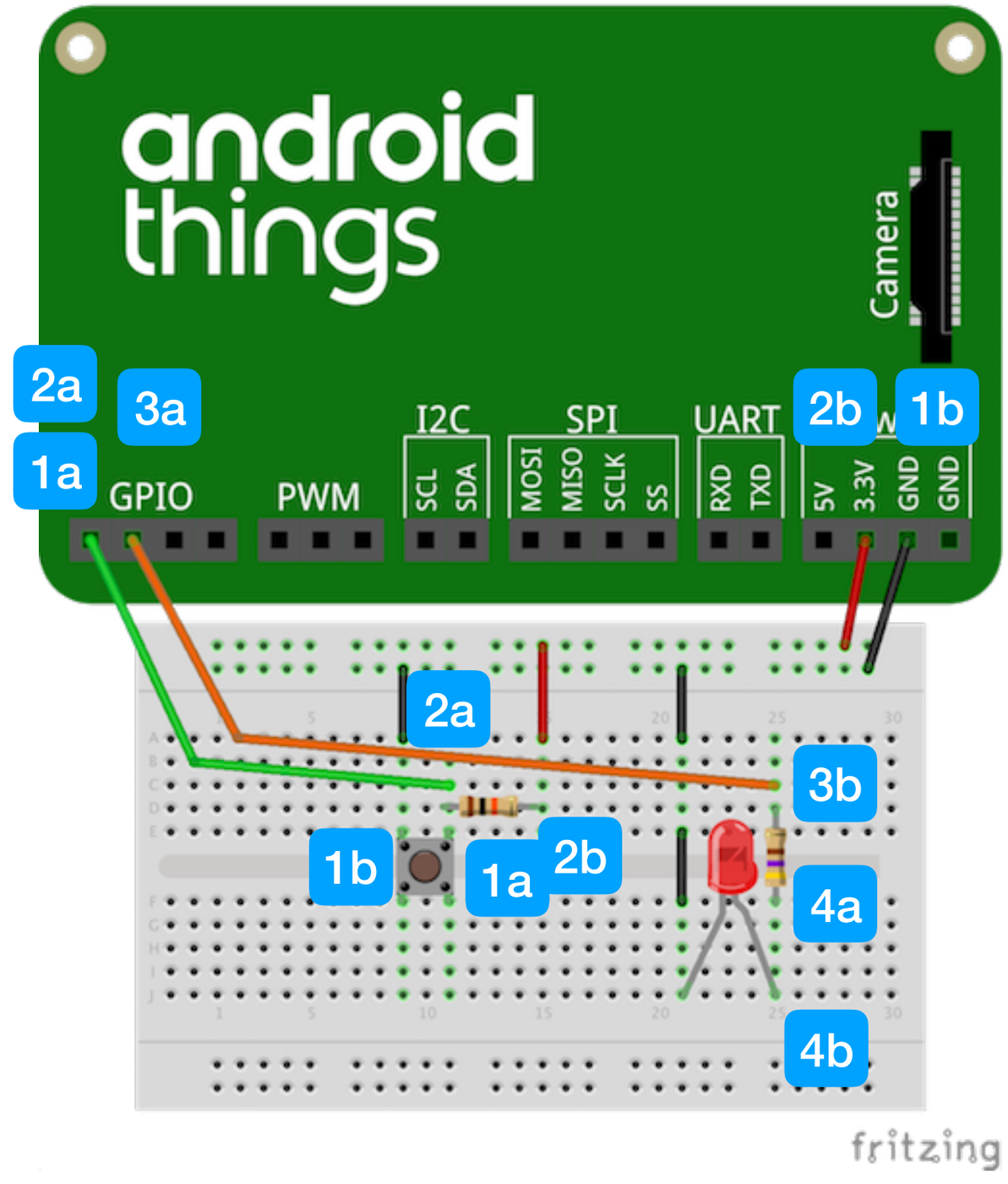
Create the Connections

1. Connect one side of the button to the chosen GPIO input pin, and the other side to ground.

2. Connect the same GPIO input pin to +3.3V through a pull-up resistor.

3. Connect the chosen GPIO output pin to one side of a series resistor.

4. Connect the other side of the resistor to the anode side (longer lead) of the LED.



Create the Connections

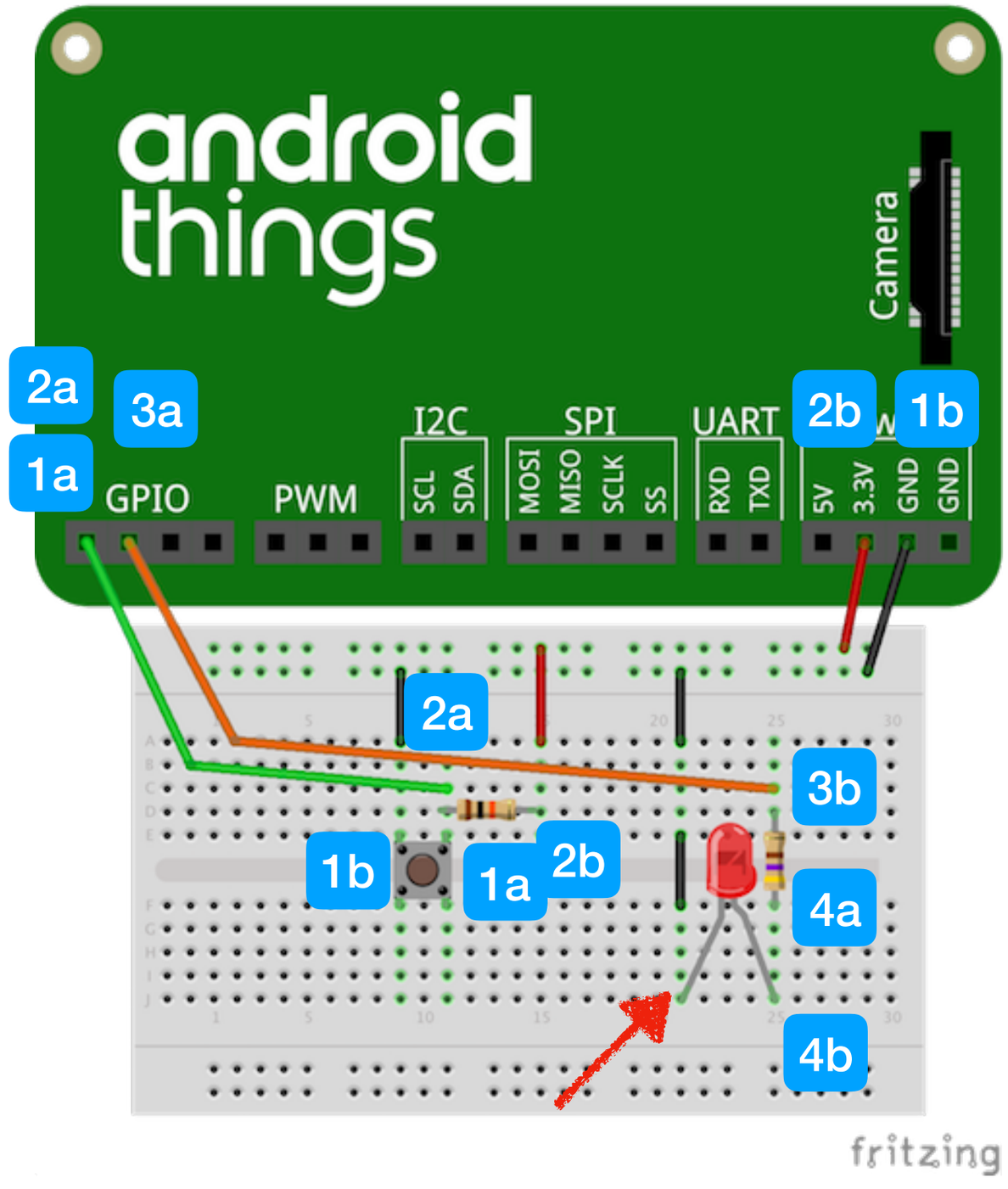
1. Connect one side of the button to the chosen GPIO input pin, and the other side to ground.

2. Connect the same GPIO input pin to +3.3V through a pull-up resistor.

3. Connect the chosen GPIO output pin to one side of a series resistor.

4. Connect the other side of the resistor to the anode side (longer lead) of the LED.

5. Connect the cathode side (shorter lead) of the LED to ground.



Create the Connections

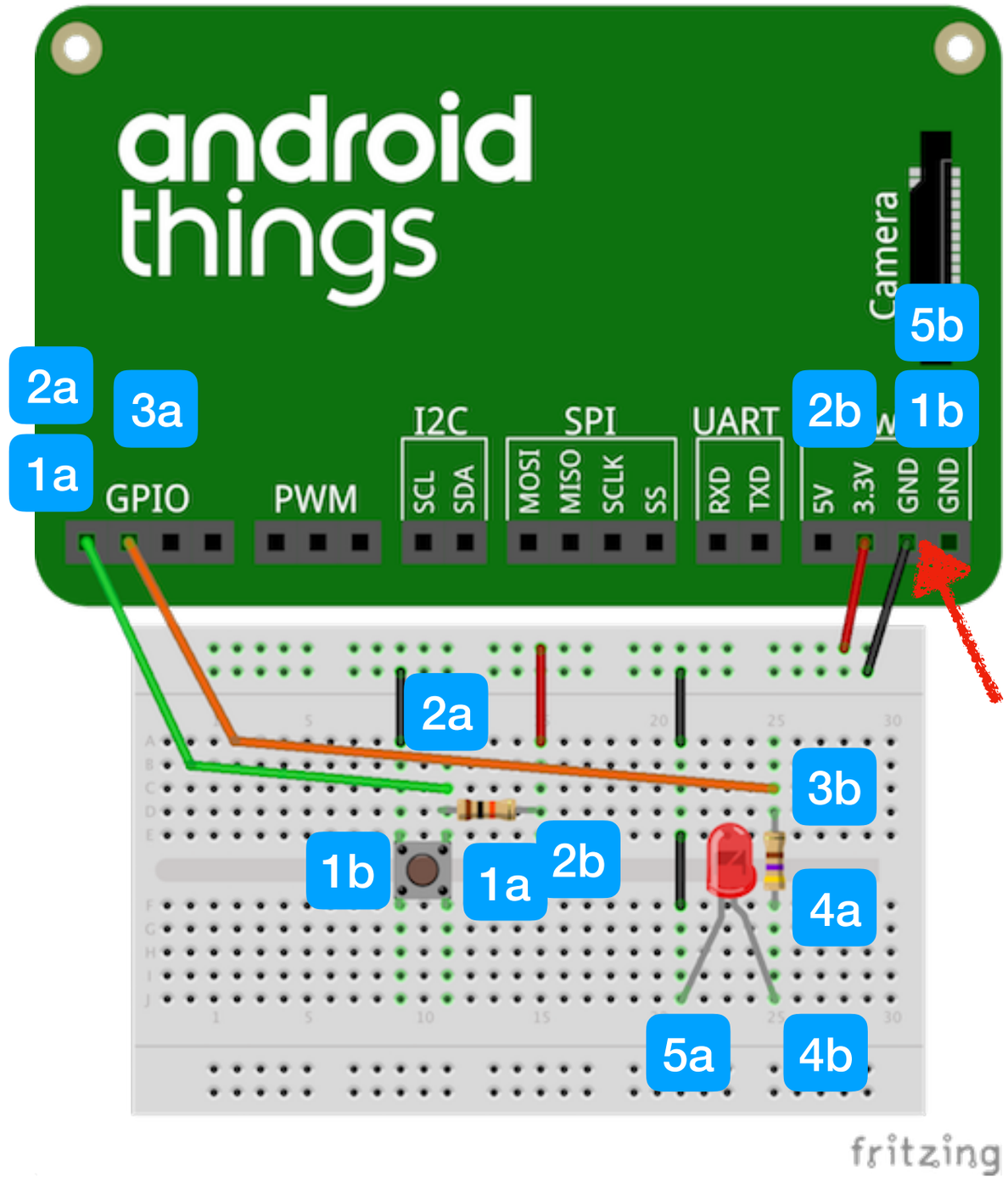
1. Connect one side of the button to the chosen GPIO input pin, and the other side to ground.

2. Connect the same GPIO input pin to +3.3V through a pull-up resistor.

3. Connect the chosen GPIO output pin to one side of a series resistor.

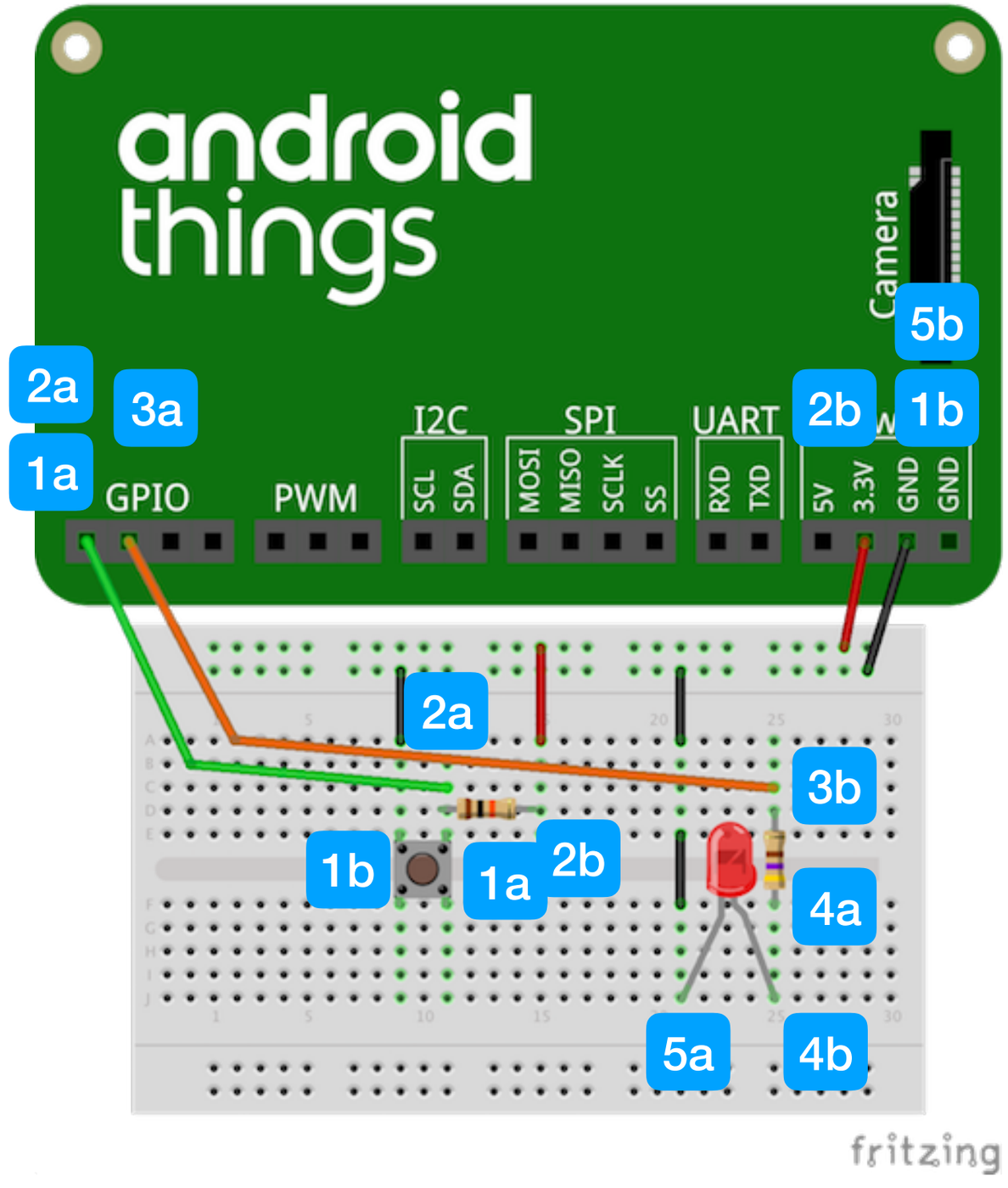
4. Connect the other side of the resistor to the anode side (longer lead) of the LED.

5. Connect the cathode side (shorter lead) of the LED to ground.

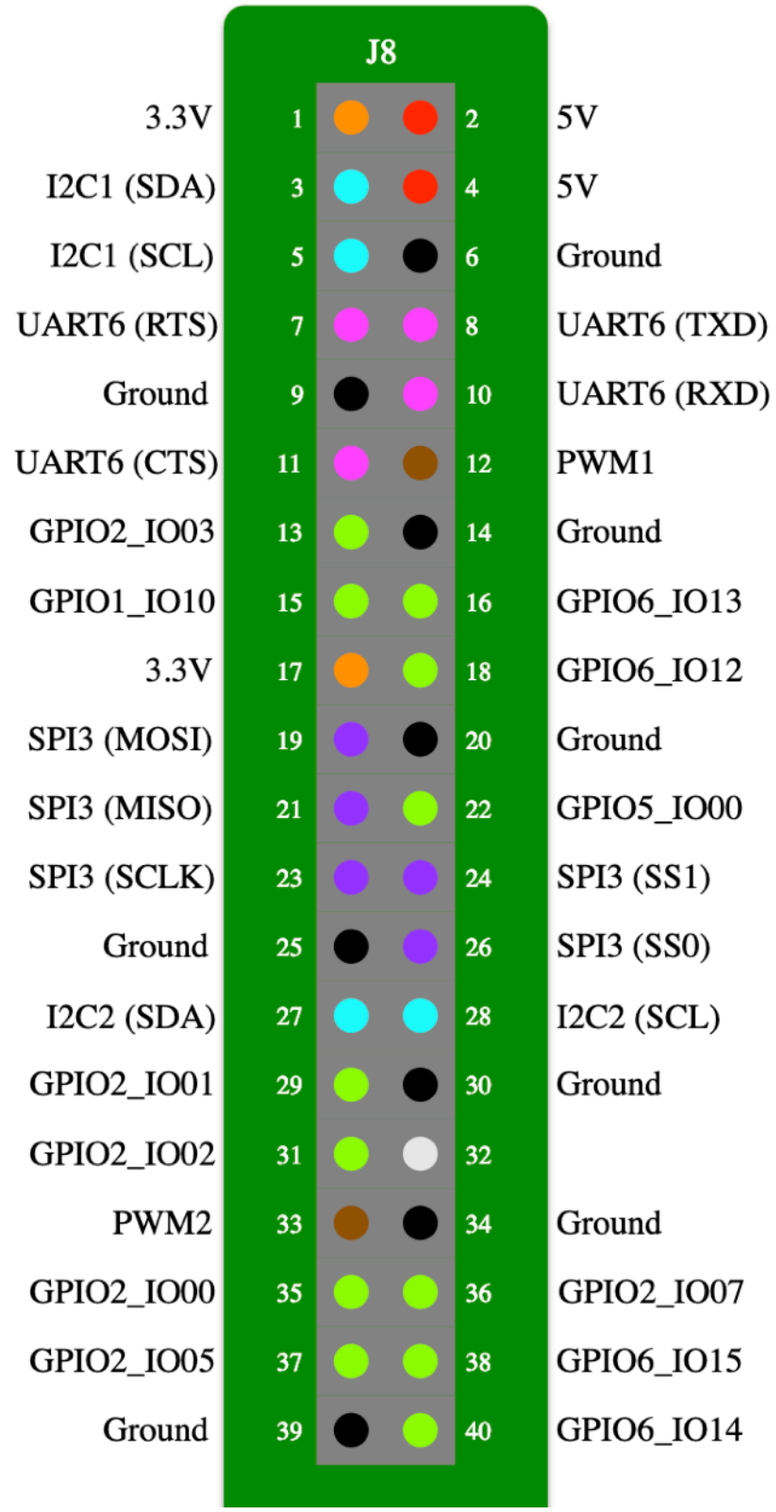


Create the Connections

1. Connect one side of the button to the chosen GPIO input pin, and the other side to ground.
2. Connect the same GPIO input pin to +3.3V through a pull-up resistor.
3. Connect the chosen GPIO output pin to one side of a series resistor.
4. Connect the other side of the resistor to the anode side (longer lead) of the LED.
5. Connect the cathode side (shorter lead) of the LED to ground.

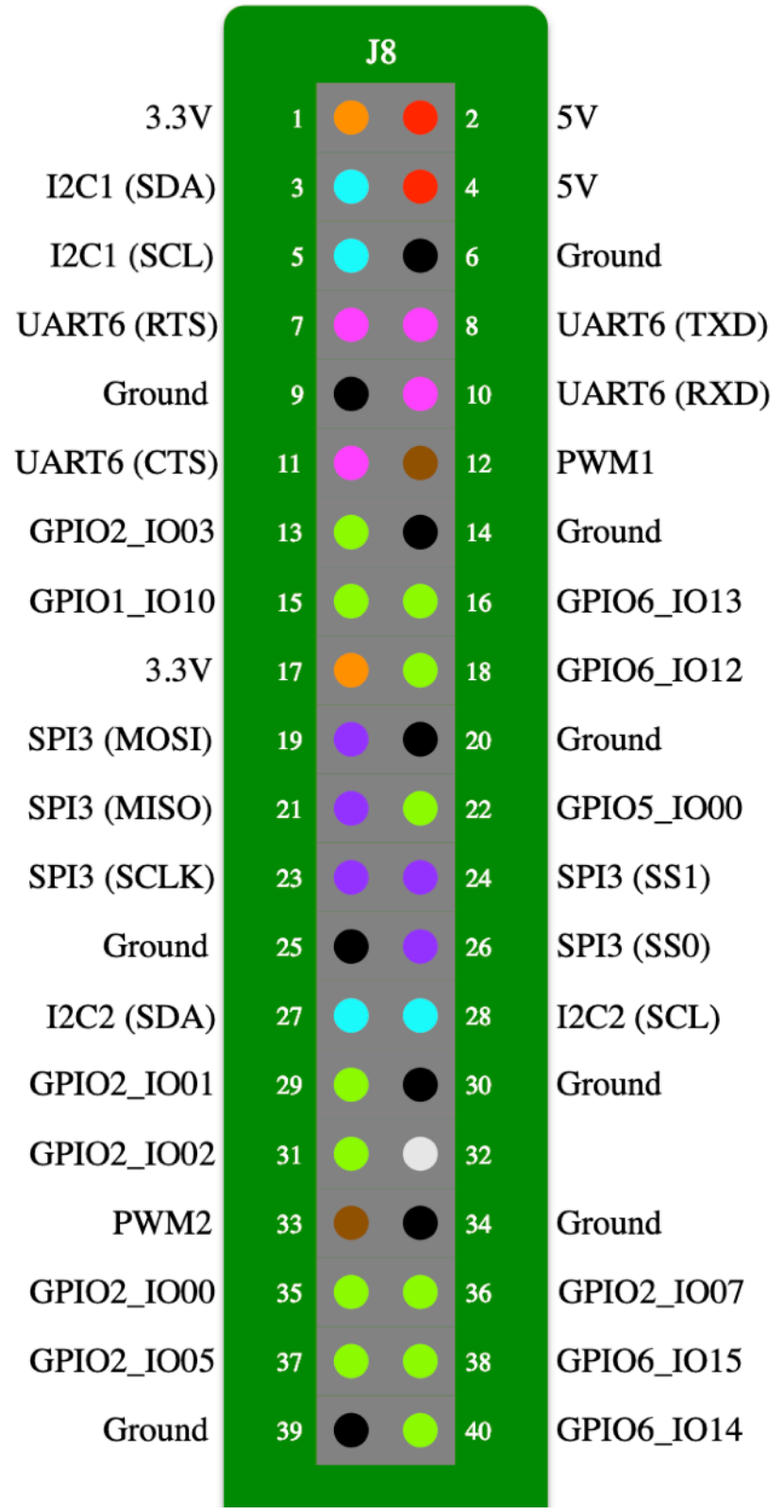


NXP i.MX7D



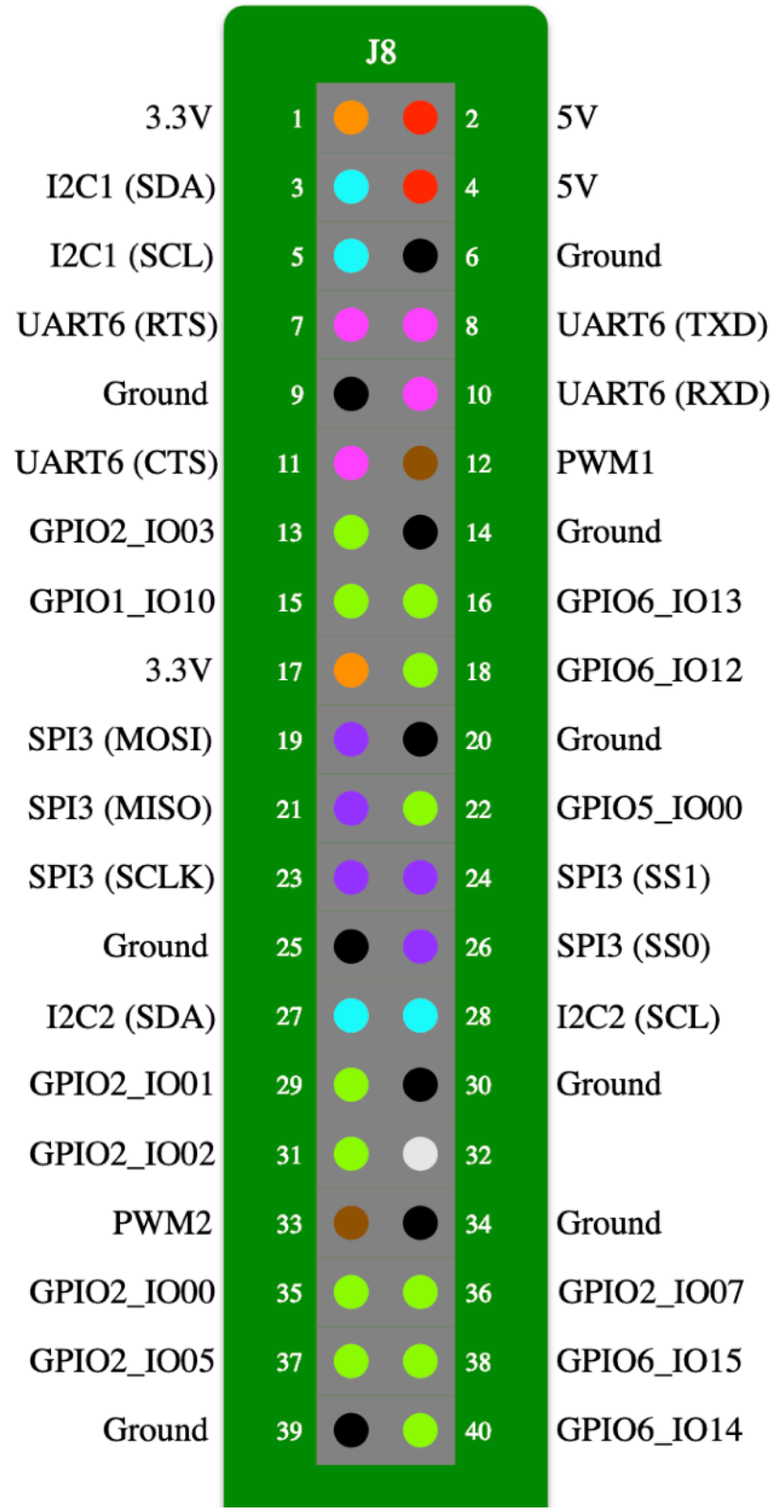
https://developer.android.com/things/hardware/imx7d#io-pinout

NXP i.MX7D



https://developer.android.com/things/hardware/imx7d#io-pinout

NXP i.MX7D

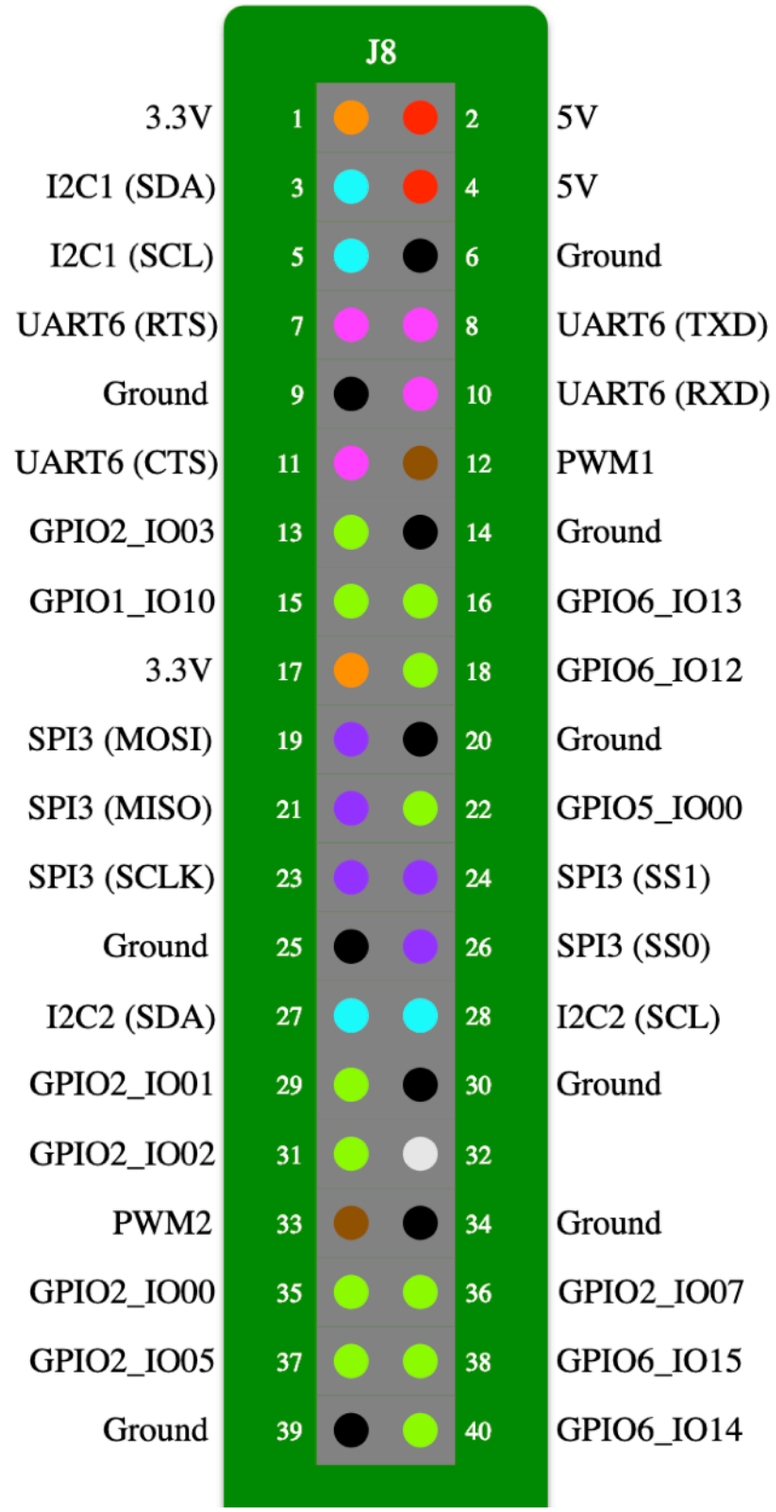


GPIO2_IO07 1a

Input 2a

https://developer.android.com/things/hardware/imx7d#io-pinout

NXP i.MX7D



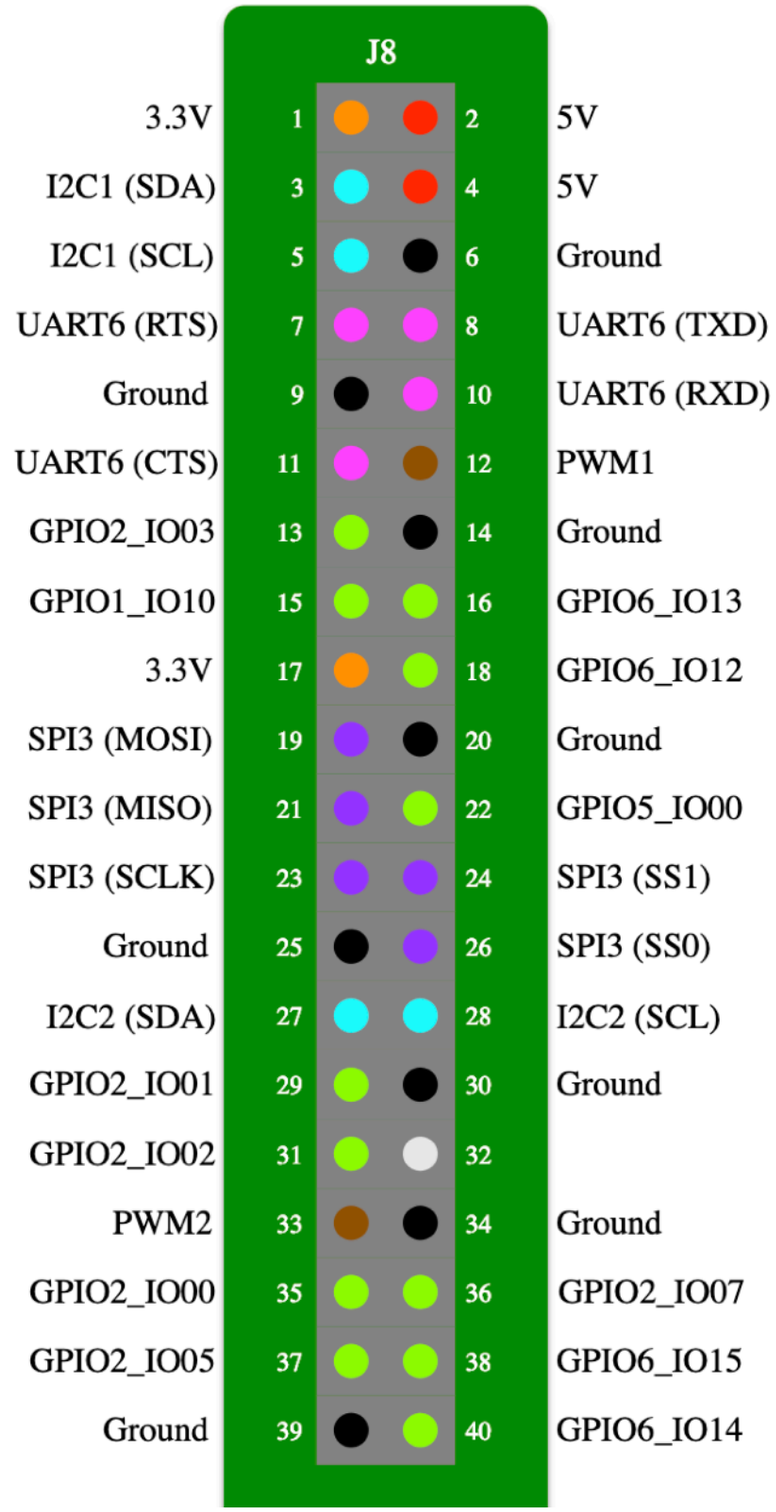
GPIO2_IO02

GPIO2_IO07 1a

Input 2a

https://developer.android.com/things/hardware/imx7d#io-pinout

NXP i.MX7D



3a **GPIO2_IO02**

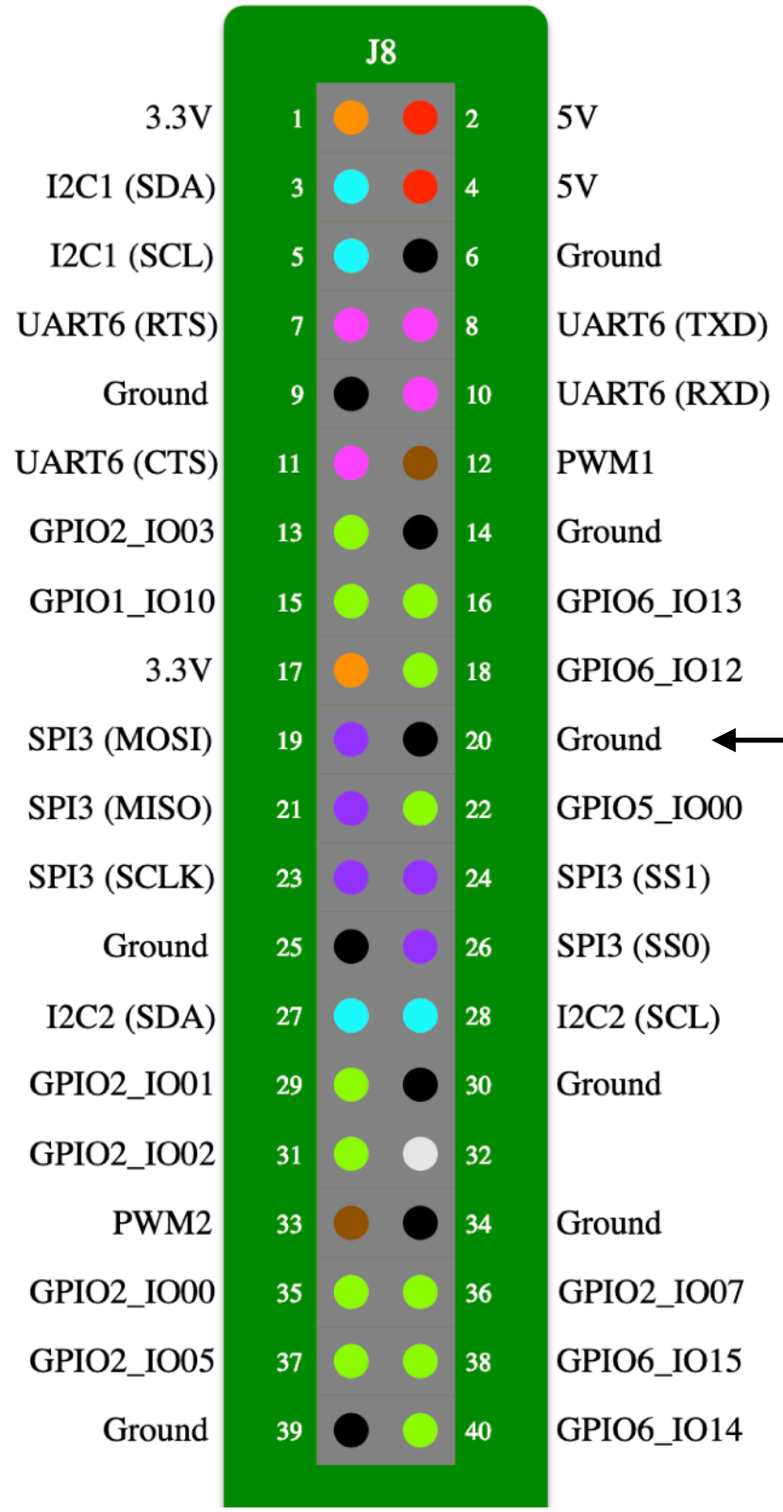
Output

GPIO2_IO07 1a

Input 2a

https://developer.android.com/things/hardware/imx7d#io-pinout

NXP i.MX7D



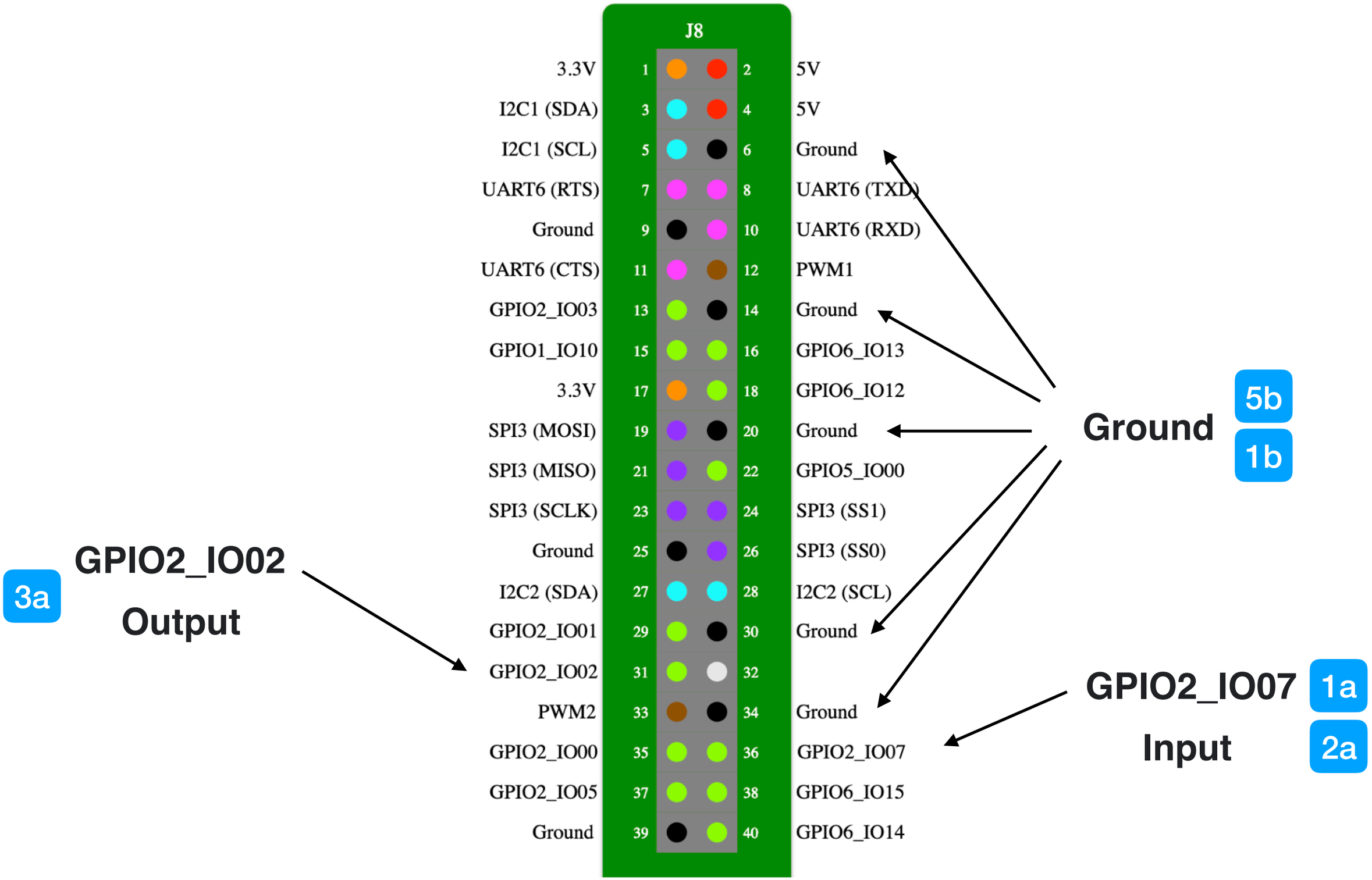
Ground

3a GPIO2_IO02 Output

GPIO2_IO07 1a Input 2a

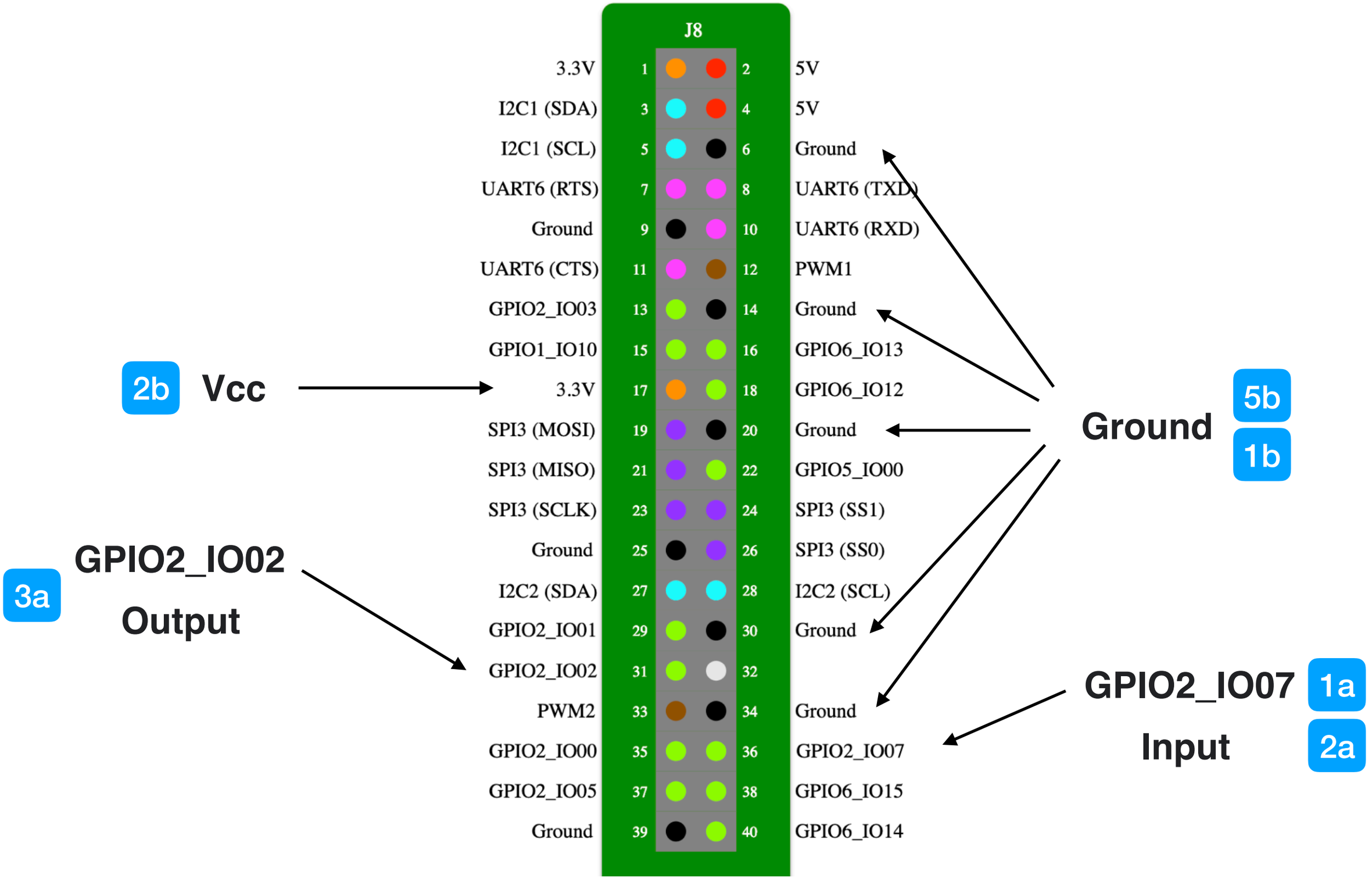
https://developer.android.com/things/hardware/imx7d#io-pinout

NXP i.MX7D



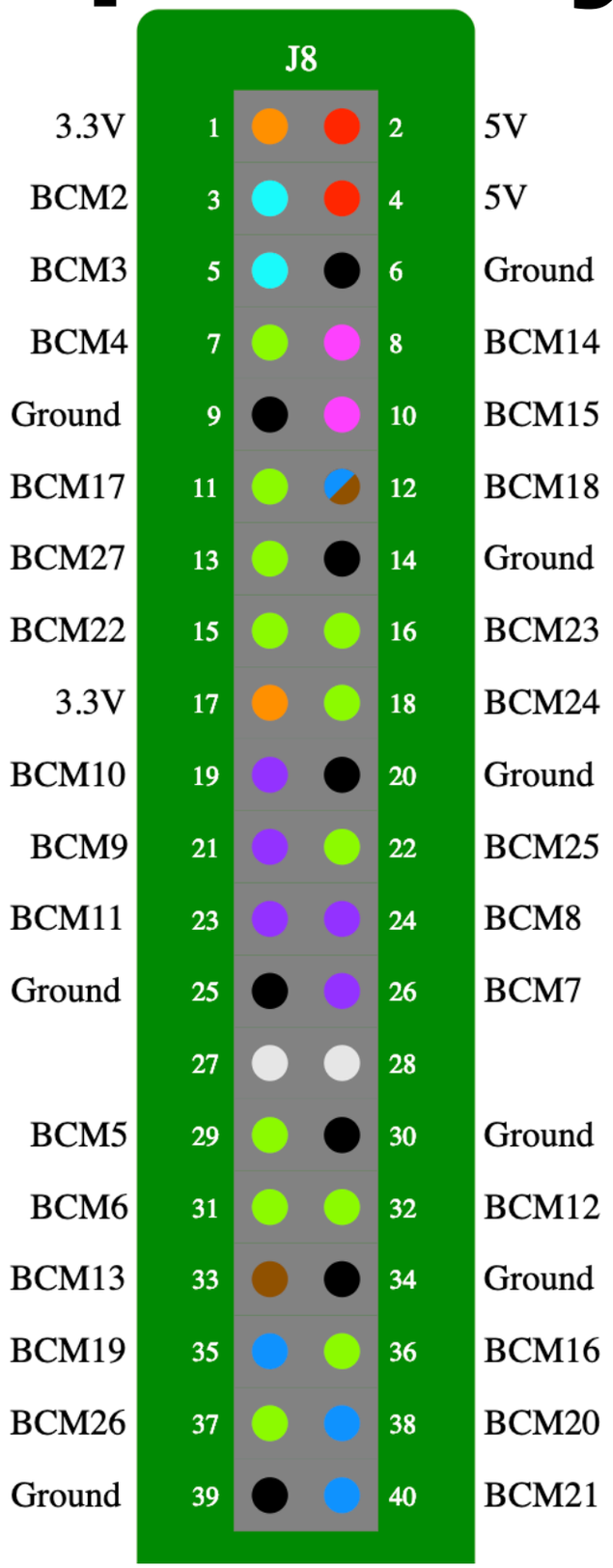
https://developer.android.com/things/hardware/imx7d#io-pinout

NXP i.MX7D



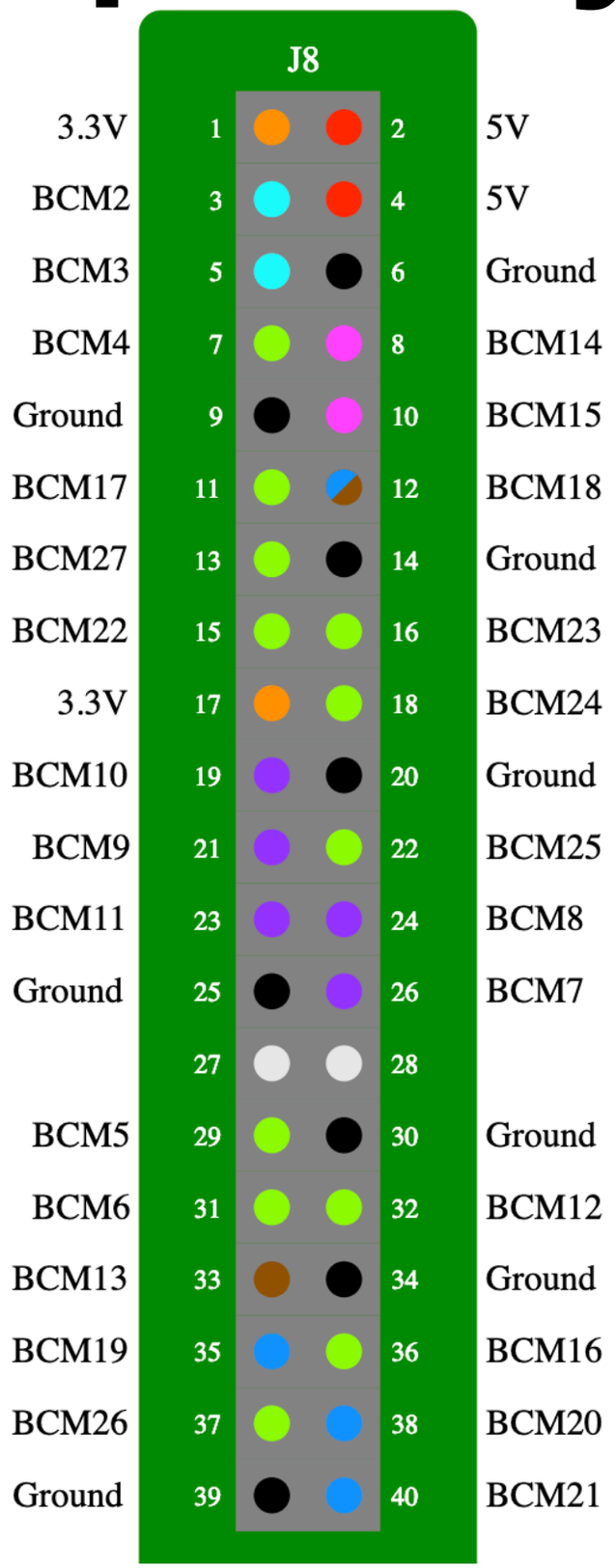
https://developer.android.com/things/hardware/imx7d#io-pinout

Raspberry Pi



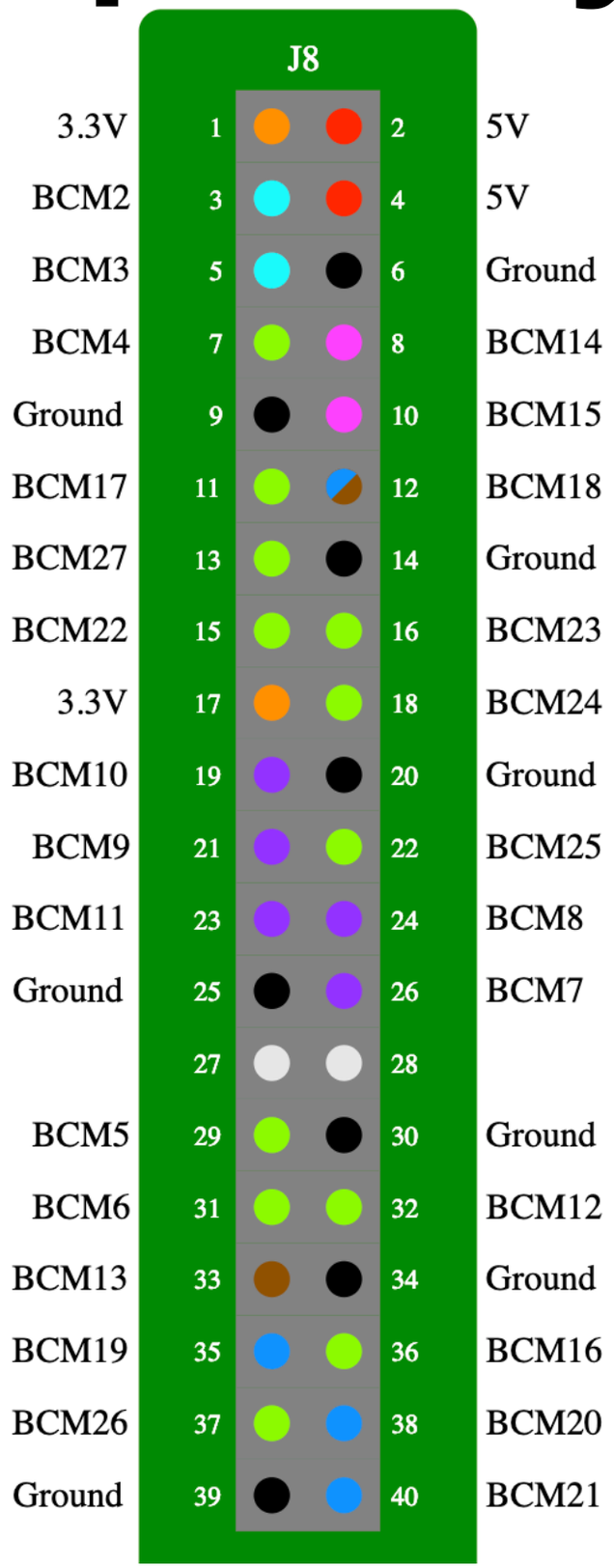
https://developer.android.com/things/hardware/raspberrypi#io-pinout

Raspberry Pi



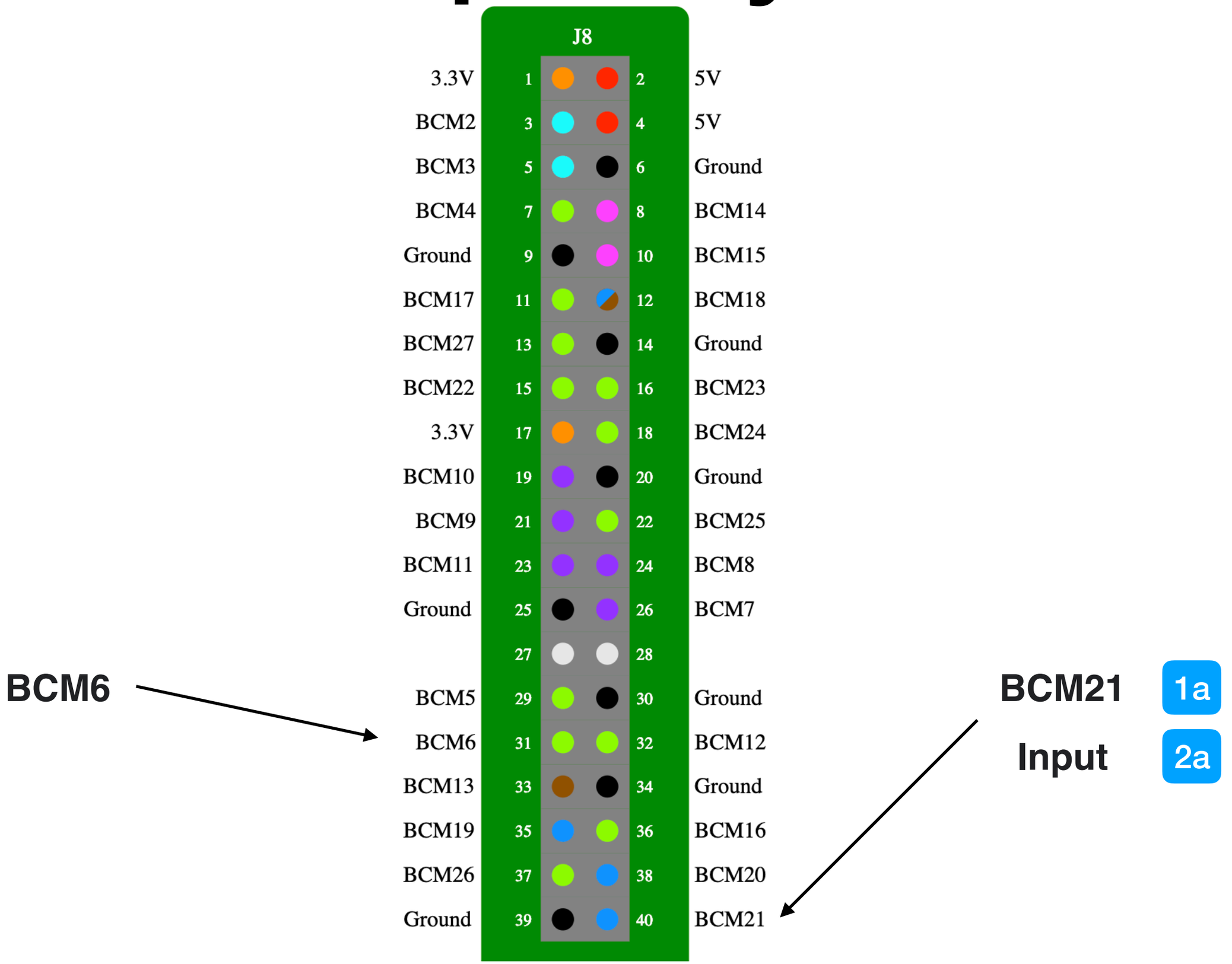
https://developer.android.com/things/hardware/raspberrypi#io-pinout

Raspberry Pi



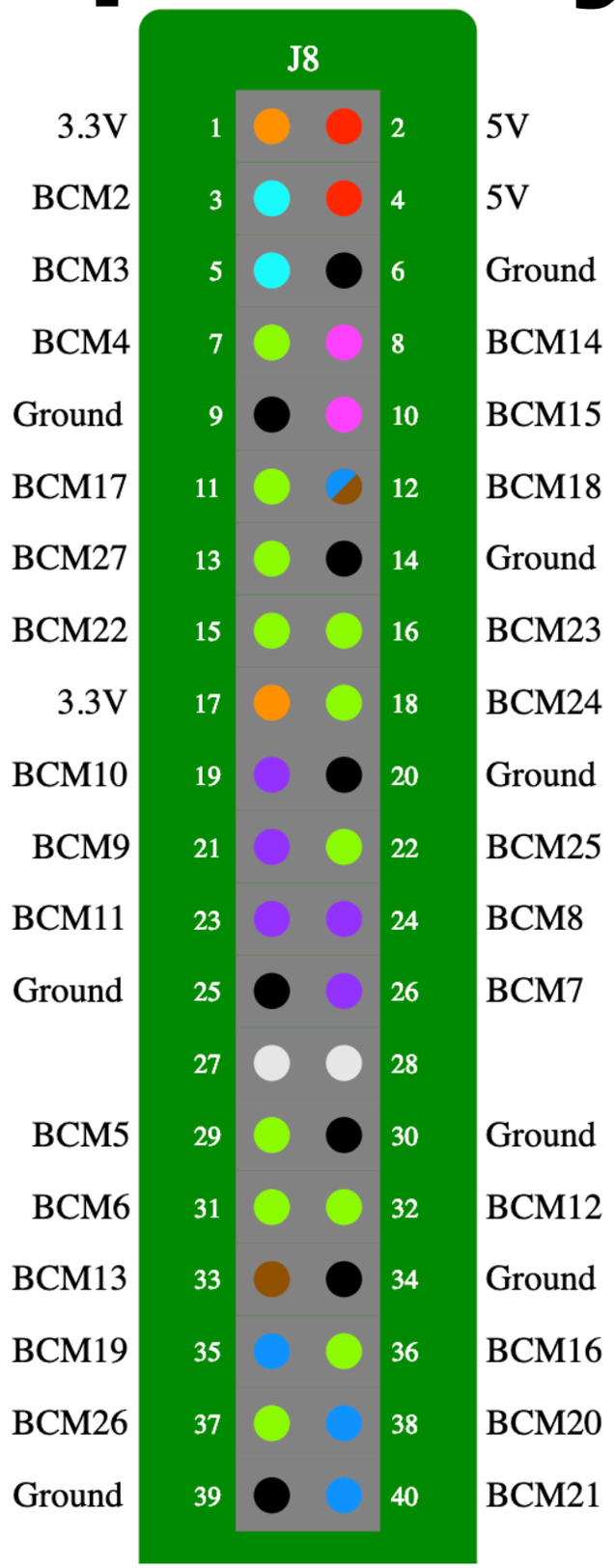
https://developer.android.com/things/hardware/raspberrypi#io-pinout

Raspberry Pi



https://developer.android.com/things/hardware/raspberrypi#io-pinout

Raspberry Pi

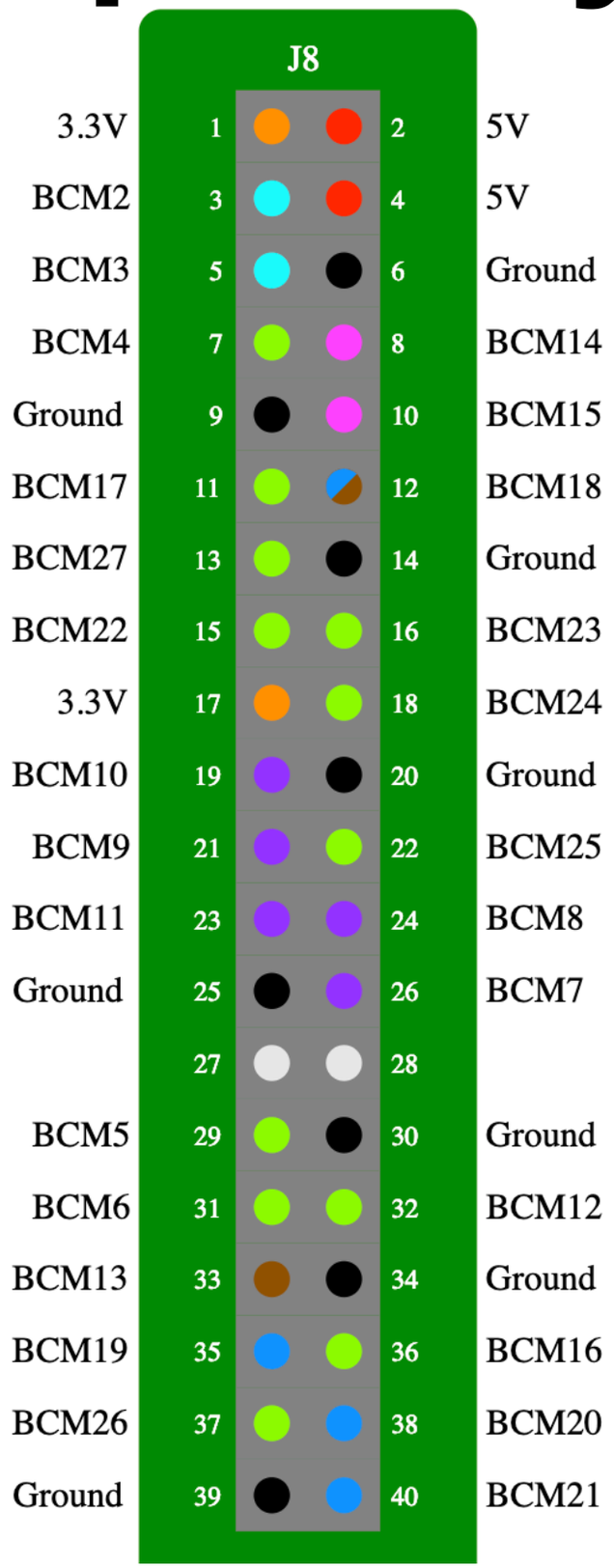


3a **BCM6** **Output**

BCM21 1a **Input** 2a

https://developer.android.com/things/hardware/raspberrypi#io-pinout

Raspberry Pi



Ground

3a

BCM6

Output

BCM21

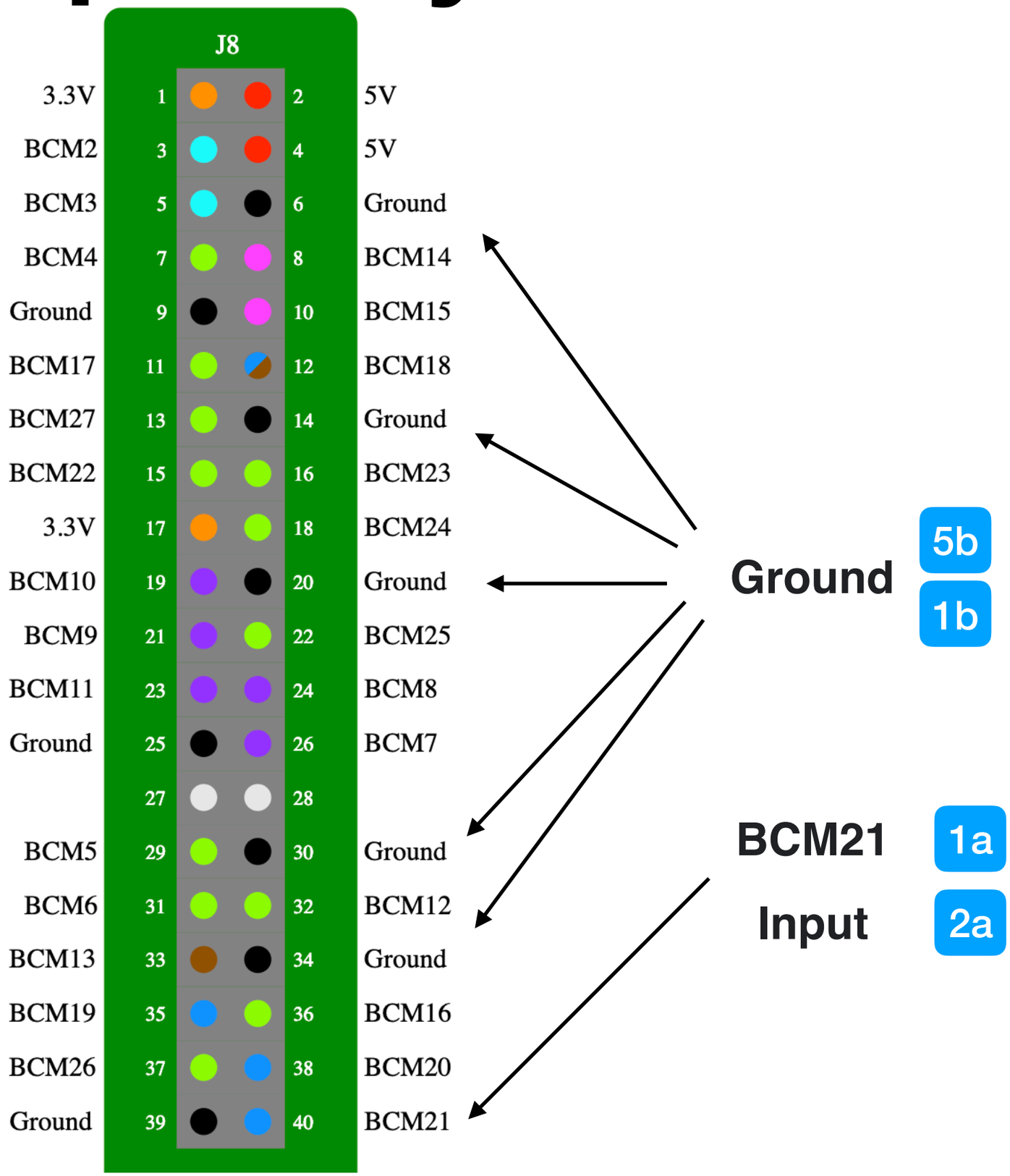
1a

Input

2a

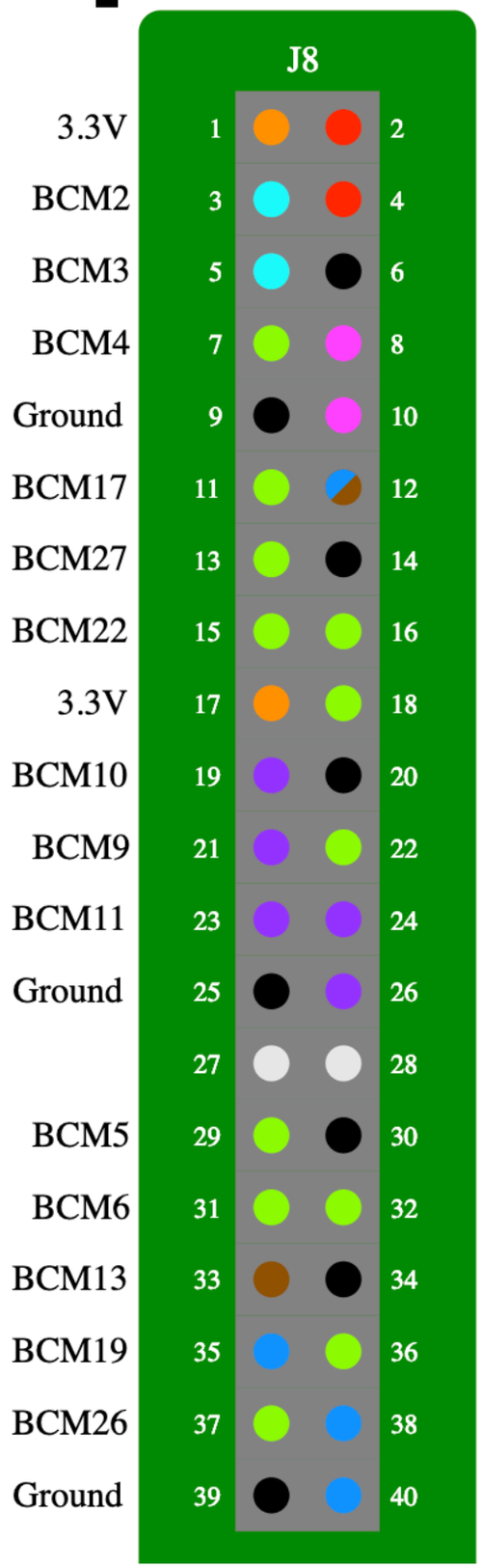
https://developer.android.com/things/hardware/raspberrypi#io-pinout

Raspberry Pi



https://developer.android.com/things/hardware/raspberrypi#io-pinout

Raspberry Pi



2b **Vcc**

Ground 5b 1b

3a **BCM6** **Output**

BCM21 **Input** 1a 2a

https://developer.android.com/things/hardware/raspberrypi#io-pinout

List available peripherals

```
import com.google.android.things.pio.PeripheralManager
...

class MainActivity : Activity() {

  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)

    val manager = PeripheralManager.getInstance()
    logd("Available GPIO: ${manager.gpioList}")
  }
}
```

List available peripherals

```
import com.google.android.things.pio.PeripheralManager
...

class MainActivity : Activity() {

  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)

    val manager = PeripheralManager.getInstance()
    logd("Available GPIO: ${manager.gpioList}")
  }
}
```

List available peripherals

```
import com.google.android.things.pio.PeripheralManager
...

class MainActivity : Activity() {

  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)

    val manager = PeripheralManager.getInstance()
    logd("Available GPIO: ${manager.gpioList}")
  }
}
```

https://developer.android.com/reference/com/google/android/things/pio/PeripheralManager.html

Handle button events

```
class ButtonActivity : Activity() {
  private companion object {
    const val BUTTON_PIN_NAME = "GPIO2_IO07" // GPIO port wired to the button
  }

  private lateinit var buttonGpio: Gpio

  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    val manager = PeripheralManager.getInstance()
    try {
      // Step 1. Create GPIO connection.
      buttonGpio = manager.openGpio(BUTTON_PIN_NAME)
      // Step 2. Configure as an input.
      buttonGpio.setDirection(Gpio.DIRECTION_IN)
      // Step 3. Enable edge trigger events.
      buttonGpio.setEdgeTriggerType(Gpio.EDGE_FALLING)
      // Step 4. Register an event callback.
      buttonGpio.registerGpioCallback(mCallback)
    } catch (e: IOException) {
      loge("Error on PeripheralIO API", e)
    }
}
```

Handle button events

```
class ButtonActivity : Activity() {
  private companion object {
    const val BUTTON_PIN_NAME = "GPIO2_IO07" // GPIO port wired to the button
  }

  private lateinit var buttonGpio: Gpio

  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    val manager = PeripheralManager.getInstance()
    try {
      // Step 1. Create GPIO connection.
      buttonGpio = manager.openGpio(BUTTON_PIN_NAME)
      // Step 2. Configure as an input.
      buttonGpio.setDirection(Gpio.DIRECTION_IN)
      // Step 3. Enable edge trigger events.
      buttonGpio.setEdgeTriggerType(Gpio.EDGE_FALLING)
      // Step 4. Register an event callback.
      buttonGpio.registerGpioCallback(mCallback)
    } catch (e: IOException) {
      loge("Error on PeripheralIO API", e)
    }
}
```

Handle button events

```
class ButtonActivity : Activity() {
  private companion object {
    const val BUTTON_PIN_NAME = "GPIO2_IO07" // GPIO port wired to the button
  }

  private lateinit var buttonGpio: Gpio

  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    val manager = PeripheralManager.getInstance()
    try {
      // Step 1. Create GPIO connection.
      buttonGpio = manager.openGpio(BUTTON_PIN_NAME)
      // Step 2. Configure as an input.
      buttonGpio.setDirection(Gpio.DIRECTION_IN)
      // Step 3. Enable edge trigger events.
      buttonGpio.setEdgeTriggerType(Gpio.EDGE_FALLING)
      // Step 4. Register an event callback.
      buttonGpio.registerGpioCallback(mCallback)
    } catch (e: IOException) {
      loge("Error on PeripheralIO API", e)
    }
}
```

Handle button events

```
class ButtonActivity : Activity() {
  private companion object {
    const val BUTTON_PIN_NAME = "GPIO2_IO07" // GPIO port wired to the button
  }

  private lateinit var buttonGpio: Gpio

  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    val manager = PeripheralManager.getInstance()
    try {
      // Step 1. Create GPIO connection.
      buttonGpio = manager.openGpio(BUTTON_PIN_NAME)
      // Step 2. Configure as an input.
      buttonGpio.setDirection(Gpio.DIRECTION_IN)
      // Step 3. Enable edge trigger events.
      buttonGpio.setEdgeTriggerType(Gpio.EDGE_FALLING)
      // Step 4. Register an event callback.
      buttonGpio.registerGpioCallback(mCallback)
    } catch (e: IOException) {
      loge("Error on PeripheralIO API", e)
    }
}
```

Handle button events

```
class ButtonActivity : Activity() {
  private companion object {
    const val BUTTON_PIN_NAME = "GPIO2_IO07" // GPIO port wired to the button
  }

  private lateinit var buttonGpio: Gpio

  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    val manager = PeripheralManager.getInstance()
    try {
      // Step 1. Create GPIO connection.
      buttonGpio = manager.openGpio(BUTTON_PIN_NAME)
      // Step 2. Configure as an input.
      buttonGpio.setDirection(Gpio.DIRECTION_IN)
      // Step 3. Enable edge trigger events.
      buttonGpio.setEdgeTriggerType(Gpio.EDGE_FALLING)
      // Step 4. Register an event callback.
      buttonGpio.registerGpioCallback(mCallback)
    } catch (e: IOException) {
      loge("Error on PeripheralIO API", e)
    }
}
```

Handle button events

```
class ButtonActivity : Activity() {
  private companion object {
    const val BUTTON_PIN_NAME = "GPIO2_IO07" // GPIO port wired to the button
  }

  private lateinit var buttonGpio: Gpio

  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    val manager = PeripheralManager.getInstance()
    try {
      // Step 1. Create GPIO connection.
      buttonGpio = manager.openGpio(BUTTON_PIN_NAME)
      // Step 2. Configure as an input.
      buttonGpio.setDirection(Gpio.DIRECTION_IN)
      // Step 3. Enable edge trigger events.
      buttonGpio.setEdgeTriggerType(Gpio.EDGE_FALLING)
      // Step 4. Register an event callback.
      buttonGpio.registerGpioCallback(mCallback)
    } catch (e: IOException) {
      loge("Error on PeripheralIO API", e)
    }
}
```

```
    // Step 1. Create GPIO connection.
    buttonGpio = manager.openGpio(BUTTON_PIN_NAME)
    // Step 2. Configure as an input.
    buttonGpio.setDirection(Gpio.DIRECTION_IN)
    // Step 3. Enable edge trigger events.
    buttonGpio.setEdgeTriggerType(Gpio.EDGE_FALLING)
    // Step 4. Register an event callback.
    buttonGpio.registerGpioCallback(mCallback)
  } catch (e: IOException) {
    loge("Error on PeripheralIO API", e)
  }
}

// Step 4. Register an event callback.
private val mCallback = GpioCallback {
  logi("GPIO changed, button pressed")
  // Step 5. Return true to keep callback active.
  true
}

override fun onDestroy() {
  super.onDestroy()

  // Step 6. Close the resource
  buttonGpio.unregisterGpioCallback(mCallback)
  try {
    buttonGpio.close()
  } catch (e: IOException) {
    loge("Error on PeripheralIO API", e)
  }
}
}
```

```
    // Step 1. Create GPIO connection.
    buttonGpio = manager.openGpio(BUTTON_PIN_NAME)
    // Step 2. Configure as an input.
    buttonGpio.setDirection(Gpio.DIRECTION_IN)
    // Step 3. Enable edge trigger events.
    buttonGpio.setEdgeTriggerType(Gpio.EDGE_FALLING)
    // Step 4. Register an event callback.
    buttonGpio.registerGpioCallback(mCallback)
  } catch (e: IOException) {
    loge("Error on PeripheralIO API", e)
  }
}

// Step 4. Register an event callback.
private val mCallback = GpioCallback {
  logi("GPIO changed, button pressed")
  // Step 5. Return true to keep callback active.
  true
}

override fun onDestroy() {
  super.onDestroy()

  // Step 6. Close the resource
  buttonGpio.unregisterGpioCallback(mCallback)
  try {
    buttonGpio.close()
  } catch (e: IOException) {
    loge("Error on PeripheralIO API", e)
  }
}
}
```

DEMO

```
        // Step 1. Create GPIO connection.
        buttonGpio = manager.openGpio(BUTTON_PIN_NAME)
        // Step 2. Configure as an input.
        buttonGpio.setDirection(Gpio.DIRECTION_IN)
        // Step 3. Enable edge trigger events.
        buttonGpio.setEdgeTriggerType(Gpio.EDGE_FALLING)
        // Step 4. Register an event callback.
        buttonGpio.registerGpioCallback(mCallback)
    } catch (e: IOException) {
        loge("Error on PeripheralIO API", e)
    }
}

// Step 4. Register an event callback.
private val mCallback = GpioCallback {
    logi("GPIO changed, button pressed")
    // Step 5. Return true to keep callback active.
    true
}

override fun onDestroy() {
    super.onDestroy()

    // Step 6. Close the resource
    buttonGpio.unregisterGpioCallback(mCallback)
    try {
        buttonGpio.close()
    } catch (e: IOException) {
        loge("Error on PeripheralIO API", e)
    }
}
```

Blink an LED

```
class BlinkActivity : Activity() {
  private companion object {
    const val LED_PIN_NAME = "GPIO2_IO02" // GPIO port wired to the LED
    const val INTERVAL_BETWEEN_BLINKS_MS = 1000L
  }
  private val mHandler = Handler()
  private lateinit var ledGpio: Gpio

  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)

    // Step 1. Create GPIO connection.
    val manager = PeripheralManager.getInstance()
    try {
      ledGpio = manager.openGpio(LED_PIN_NAME)
      // Step 2. Configure as an output.
      ledGpio.setDirection(Gpio.DIRECTION_OUT_INITIALLY_LOW)

      // Step 4. Repeat using a handler.
      mHandler.post(blinkRunnable)
    } catch (e: IOException) {
      loge("Error on PeripheralIO API", e)
    }
  }
```

```
        // Step 4. Repeat using a handler.
        mHandler.post(blinkRunnable)
      } catch (e: IOException) {
        loge("Error on PeripheralIO API", e)
      }
    }

    private val blinkRunnable = object : Runnable {
      override fun run() {
        try {
          // Step 3. Toggle the LED state
          ledGpio.value = !ledGpio.value
          // Step 4. Schedule another event after delay.
          mHandler.postDelayed(this, INTERVAL_BETWEEN_BLINKS_MS)
        } catch (e: IOException) {
          loge("Error on PeripheralIO API", e)
        }
      }
    }

    override fun onDestroy() {
      super.onDestroy()
      // Step 4. Remove handler events on close.
      mHandler.removeCallbacks(blinkRunnable)
      // Step 5. Close the resource.
      try {
        ledGpio.close()
      } catch (e: IOException) {
        loge("Error on PeripheralIO API", e)
      }
    }
}
```

Bluetooth

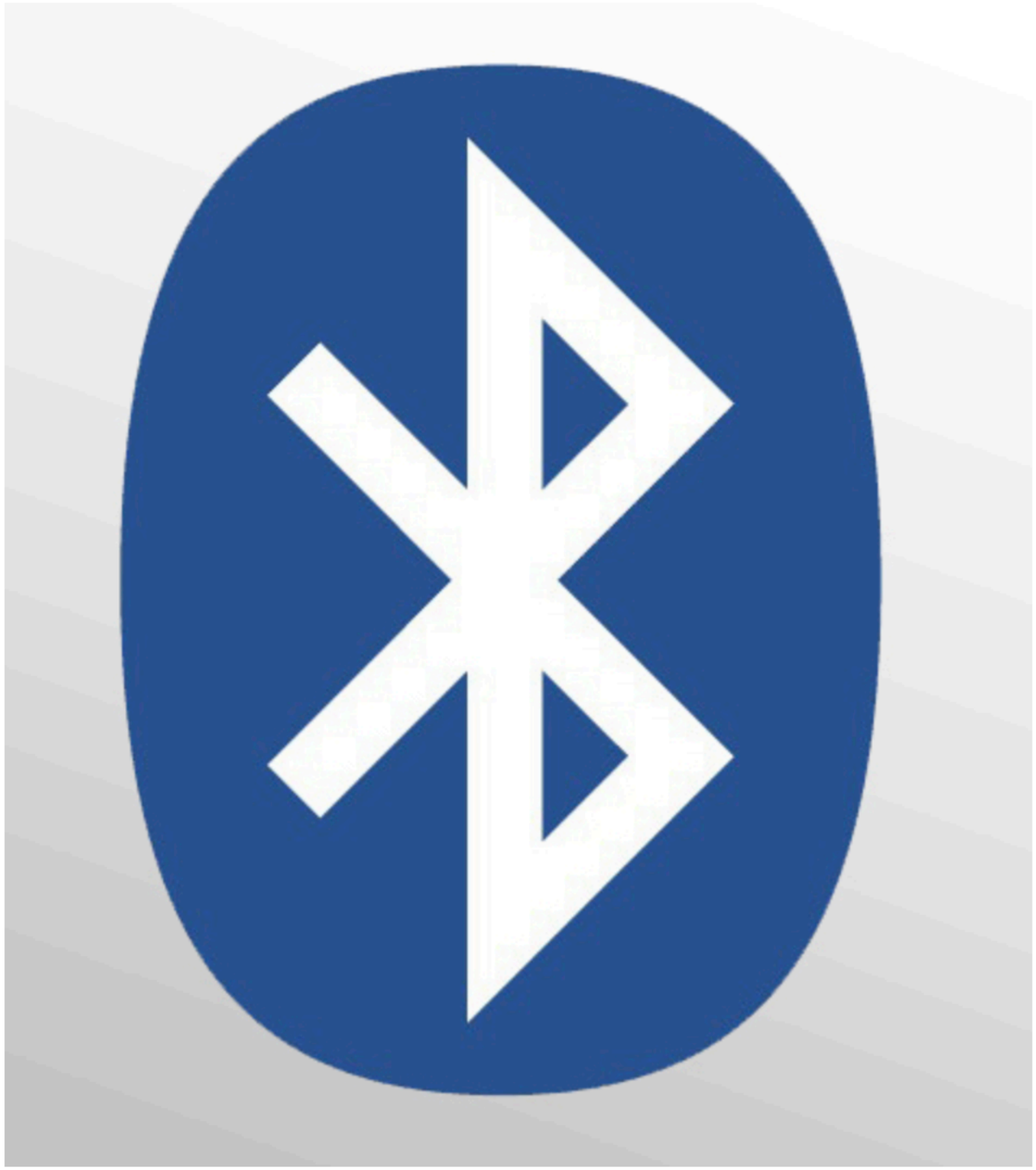


Image source: http://blog.lenovo.com/

Permissions

```
<uses-permission
        android:name="com.google.android.things.permission.MANAGE_BLUETOOTH"/>
```

Permissions

```
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>

<uses-permission
        android:name="com.google.android.things.permission.MANAGE_BLUETOOTH"/>
```

Permissions

Allows applications to connect to paired bluetooth devices.

```
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>

<uses-permission
        android:name="com.google.android.things.permission.MANAGE_BLUETOOTH"/>
```

Permissions

Allows applications to connect to paired bluetooth devices.

```
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>
```

Allows applications to discover and pair bluetooth devices.

```
<uses-permission
        android:name="com.google.android.things.permission.MANAGE_BLUETOOTH"/>
```

Configuring device attributes

```
import android.bluetooth.BluetoothClass
import com.google.android.things.bluetooth.BluetoothClassFactory
import com.google.android.things.bluetooth.BluetoothConfigManager
...

val manager = BluetoothConfigManager.getInstance()
// Report the local Bluetooth device class as a speaker
manager.bluetoothClass = BluetoothClassFactory.build(
  BluetoothClass.Service.AUDIO,
  BluetoothClass.Device.AUDIO_VIDEO_LOUDSPEAKER
)
```

Configuring device attributes

```
import android.bluetooth.BluetoothClass
import com.google.android.things.bluetooth.BluetoothClassFactory
import com.google.android.things.bluetooth.BluetoothConfigManager
...

val manager = BluetoothConfigManager.getInstance()
// Report the local Bluetooth device class as a speaker
manager.bluetoothClass = BluetoothClassFactory.build(
  BluetoothClass.Service.AUDIO,
  BluetoothClass.Device.AUDIO_VIDEO_LOUDSPEAKER
)
```

Configuring device attributes

`BluetoothClassFactory()`

Public methods	
`static BluetoothClass`	`build(int service, int device)` Construct the Bluetooth Class of Device (CoD) using the Service and Device values.

https://developer.android.com/reference/com/google/android/things/bluetooth/BluetoothClassFactory.html

Configuring device attributes

Constants	
int	AUDIO
int	CAPTURE
int	INFORMATION
int	LIMITED_DISCOVERABILITY
int	NETWORKING
int	OBJECT_TRANSFER
int	POSITIONING
int	RENDER
int	TELEPHONY

https://developer.android.com/reference/android/bluetooth/BluetoothClass.Service.html

Configuring device attributes

Constants	
int	AUDIO_VIDEO_CAMCORDER
int	AUDIO_VIDEO_CAR_AUDIO
int	AUDIO_VIDEO_HANDSFREE
int	AUDIO_VIDEO_HEADPHONES
int	AUDIO_VIDEO_HIFI_AUDIO
int	AUDIO_VIDEO_LOUDSPEAKER
int	AUDIO_VIDEO_MICROPHONE
int	AUDIO_VIDEO_PORTABLE_AUDIO
int	AUDIO_VIDEO_SET_TOP_BOX
int	AUDIO_VIDEO_UNCATEGORIZED
int	AUDIO_VIDEO_VCR
int	AUDIO_VIDEO_VIDEO_CAMERA

int	AUDIO_VIDEO_PORTABLE_AUDIO
int	AUDIO_VIDEO_SET_TOP_BOX
int	AUDIO_VIDEO_UNCATEGORIZED
int	AUDIO_VIDEO_VCR
int	AUDIO_VIDEO_VIDEO_CAMERA
int	AUDIO_VIDEO_VIDEO_CONFERENCING
int	AUDIO_VIDEO_VIDEO_DISPLAY_AND_LOUDSPEAKER
int	AUDIO_VIDEO_VIDEO_GAMING_TOY
int	AUDIO_VIDEO_VIDEO_MONITOR
int	AUDIO_VIDEO_WEARABLE_HEADSET
int	COMPUTER_DESKTOP
int	COMPUTER_HANDHELD_PC_PDA
int	COMPUTER_LAPTOP
int	COMPUTER_PALM_SIZE_PC_PDA
int	COMPUTER_SERVER
int	COMPUTER_UNCATEGORIZED
int	HEALTH_GLUCOSE
int	HEALTH_PULSE_OXIMETER

int	HEALTH_GLUCOSE
int	HEALTH_PULSE_OXIMETER
int	HEALTH_PULSE_RATE
int	HEALTH_THERMOMETER
int	HEALTH_UNCATEGORIZED
int	HEALTH_WEIGHING
int	PHONE_CELLULAR
int	PHONE_CORDLESS
int	PHONE_ISDN
int	PHONE_MODEM_OR_GATEWAY
int	PHONE_SMART
int	PHONE_UNCATEGORIZED
int	TOY_CONTROLLER
int	TOY_DOLL_ACTION_FIGURE
int	TOY_GAME
int	TOY_ROBOT
int	TOY_UNCATEGORIZED

int	PHONE_MODEM_OR_GATEWAY
int	PHONE_SMART
int	PHONE_UNCATEGORIZED
int	TOY_CONTROLLER
int	TOY_DOLL_ACTION_FIGURE
int	TOY_GAME
int	TOY_ROBOT
int	TOY_UNCATEGORIZED
int	TOY_VEHICLE
int	WEARABLE_GLASSES
int	WEARABLE_HELMET
int	WEARABLE_JACKET
int	WEARABLE_PAGER
int	WEARABLE_UNCATEGORIZED
int	WEARABLE_WRIST_WATCH

https://developer.android.com/reference/android/bluetooth/BluetoothClass.Device.html

I/O capabilities

Bluetooth with setIoCapability()

Bluetooth Low Energy (LE) with setLeIoCapability()

I/O capabilities

Bluetooth with setIoCapability()
Bluetooth Low Energy (LE) with setLeIoCapability()

- **IO_CAPABILITY_NONE**: Device has no input or output capabilities. **This is the default value.**
- **IO_CAPABILITY_OUT**: Device has a display only.
- **IO_CAPABILITY_IN**: Device can accept keyboard user input only.
- **IO_CAPABILITY_IO**: Device has a display and can accept basic (yes/no) input.
- **IO_CAPABILITY_KBDISP**: Device has a display and can accept keyboard user input.

https://developer.android.com/reference/com/google/android/things/bluetooth/BluetoothConfigManager.html

I/O capabilities

```
import com.google.android.things.bluetooth.BluetoothConfigManager
...
val manager = BluetoothConfigManager.getInstance()
// Report full input/output capability for this device
manager.ioCapability = BluetoothConfigManager.IO_CAPABILITY_IO
```

https://developer.android.com/reference/com/google/android/things/bluetooth/BluetoothConfigManager.html

I/O capabilities

Warning: Currently Android Things devices reporting **`IO_CAPABILITY_IN`** or **`IO_CAPABILITY_KBDISP`** are not capable of pairing with a remote device that also reports **`IO_CAPABILITY_KBDISP`**, such as an Android mobile device.

https://developer.android.com/reference/com/google/android/things/bluetooth/BluetoothConfigManager.html

Enabled Profiles

```
import com.google.android.things.bluetooth.BluetoothProfileManager;
import com.google.android.things.bluetooth.BluetoothProfile;
...

val manager = BluetoothProfileManager.getInstance()
val enabledProfiles = manager.enabledProfiles
if (!enabledProfiles.contains(BluetoothProfile.A2DP_SINK)) {
  Log.d(TAG, "Enabling A2DP sink mode.")
  val toDisable = listOf(BluetoothProfile.A2DP)
  val toEnable = listOf(BluetoothProfile.A2DP_SINK,
                        BluetoothProfile.AVRCP_CONTROLLER)
  manager.enableAndDisableProfiles(toEnable, toDisable)
}
```

https://developer.android.com/reference/com/google/android/things/bluetooth/BluetoothProfile.html

Pairing with a remote device

```
import android.bluetooth.BluetoothDevice
import com.google.android.things.bluetooth.BluetoothConnectionManager
import com.google.android.things.bluetooth.BluetoothPairingCallback
import com.google.android.things.bluetooth.PairingParams
...

class PairingActivity : Activity() {
  private lateinit var bluetoothConnectionManager: BluetoothConnectionManager

override fun onCreate(savedInstanceState: Bundle?) {
  super.onCreate(savedInstanceState)
  bluetoothConnectionManager = BluetoothConnectionManager.getInstance().apply {
    registerPairingCallback(bluetoothPairingCallback)
  }
}

override fun onDestroy() {
  super.onDestroy()
  bluetoothConnectionManager.unregisterPairingCallback(bluetoothPairingCallback)
}

private fun startPairing(remoteDevice: BluetoothDevice) {
  bluetoothConnectionManager.initiatePairing(remoteDevice)
}
```

```
private fun startPairing(remoteDevice: BluetoothDevice) {
  bluetoothConnectionManager.initiatePairing(remoteDevice)
}

private val bluetoothPairingCallback = object : BluetoothPairingCallback {
  override fun onPairingInitiated(
      bluetoothDevice: BluetoothDevice,
      pairingParams: PairingParams
    ) {
    // Handle incoming pairing request or confirmation of outgoing pairing request
    handlePairingRequest(bluetoothDevice, pairingParams)
  }

  override fun onPaired(bluetoothDevice: BluetoothDevice) {
    // Device pairing complete
  }

  override fun onUnpaired(bluetoothDevice: BluetoothDevice) {
    // Device unpaired
  }

  override fun onPairingError(
      bluetoothDevice: BluetoothDevice,
      pairingError: BluetoothPairingCallback.PairingError
    ) {
    // Something went wrong!
  }
}
}
```

```
private fun handlePairingRequest(
    bluetoothDevice: BluetoothDevice,
    pairingParams: PairingParams) {
  when (pairingParams.pairingType) {
    PairingParams.PAIRING_VARIANT_DISPLAY_PIN,
    PairingParams.PAIRING_VARIANT_DISPLAY_PASSKEY -> {
      // Display the required PIN to the user
      Log.d(TAG, "Display Passkey - ${pairingParams.pairingPin}")
    }
    PairingParams.PAIRING_VARIANT_PIN,
    PairingParams.PAIRING_VARIANT_PIN_16_DIGITS -> {
      // Obtain PIN from the user
      val pin = ...
      // Pass the result to complete pairing
      bluetoothConnectionManager.finishPairing(bluetoothDevice, pin)
    }
    PairingParams.PAIRING_VARIANT_CONSENT,
    PairingParams.PAIRING_VARIANT_PASSKEY_CONFIRMATION -> {
      // Show confirmation of pairing to the user
      ...
      // Complete the pairing process
      bluetoothConnectionManager.finishPairing(bluetoothDevice)
    }
  }
}
```

https://developer.android.com/reference/com/google/android/things/bluetooth/BluetoothConnectionManager.html

Connecting to a remote device

```
import android.bluetooth.BluetoothDevice
import com.google.android.things.bluetooth.BluetoothConnectionManager
import com.google.android.things.bluetooth.BluetoothConnectionCallback
import com.google.android.things.bluetooth.BluetoothProfile
import com.google.android.things.bluetooth.ConnectionParams
...

class ConnectActivity : Activity() {
  private lateinit var bluetoothConnectionManager: BluetoothConnectionManager
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    bluetoothConnectionManager = BluetoothConnectionManager.getInstance().apply {
      registerConnectionCallback(bluetoothConnectionCallback)
    }
  }

  override fun onDestroy() {
    super.onDestroy()
    bluetoothConnectionManager.unregisterConnectionCallback(
        bluetoothConnectionCallback)
  }

  private fun connectToA2dp(bluetoothDevice: BluetoothDevice) {
    bluetoothConnectionManager.connect(bluetoothDevice, BluetoothProfile.A2DP_SINK)
```

```
        bluetoothConnectionCallback)
}

private fun connectToA2dp(bluetoothDevice: BluetoothDevice) {
  bluetoothConnectionManager.connect(bluetoothDevice, BluetoothProfile.A2DP_SINK)
}

// Set up callbacks for the profile connection process.
private val bluetoothConnectionCallback = object : BluetoothConnectionCallback {
  override fun onConnectionRequested(
      bluetoothDevice: BluetoothDevice,
      connectionParams: ConnectionParams) {
    // Handle incoming connection request
    handleConnectionRequest(bluetoothDevice, connectionParams)
  }

  override fun onConnectionRequestCancelled(
      bluetoothDevice: BluetoothDevice,
      requestType: Int) {
    // Request cancelled
  }

  override fun onConnected(bluetoothDevice: BluetoothDevice, profile: Int) {
    // Connection completed successfully
  }

  override fun onDisconnected(bluetoothDevice: BluetoothDevice, profile: Int) {
    // Remote device disconnected
  }
 }
}
```

Connecting to a remote device

DEMO

```
private fun handleConnectionRequest(
    bluetoothDevice: BluetoothDevice,
    connectionParams: ConnectionParams) {
  // Determine whether to accept the connection request
  val accept = connectionParams.requestType ==
               ConnectionParams.REQUEST_TYPE_PROFILE_CONNECTION

  // Pass that result on to the BluetoothConnectionManager
  bluetoothConnectionManager.confirmOrDenyConnection(
      bluetoothDevice,
      connectionParams,
      accept)
}
```

https://developer.android.com/guide/topics/connectivity/bluetooth.html#ConnectingDevices

Lecture outcomes

- Connect the hardware components.
- Send and receive data.
- Use the bluetooth stack.