

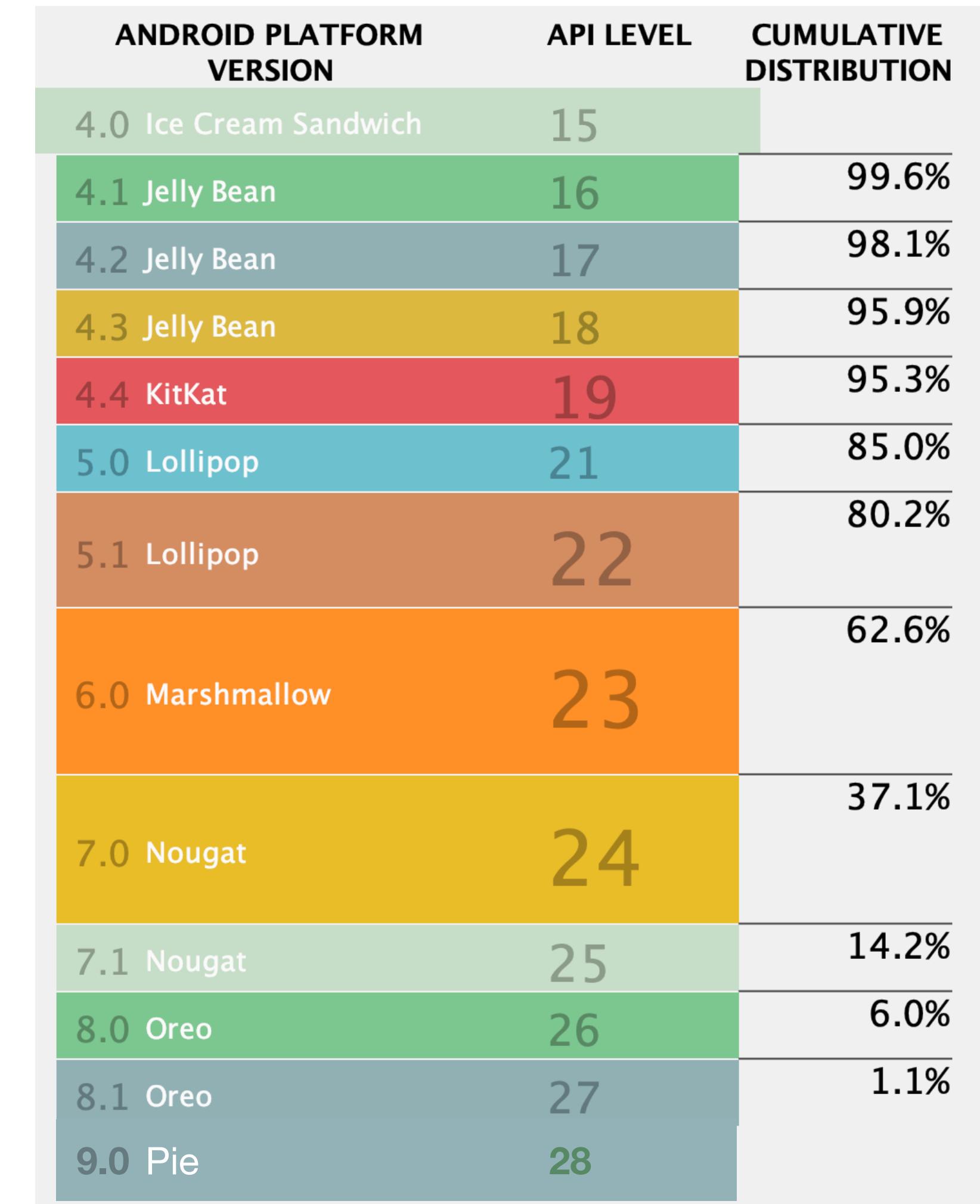
Lecture #3

Developer Platform

Android Things 2021

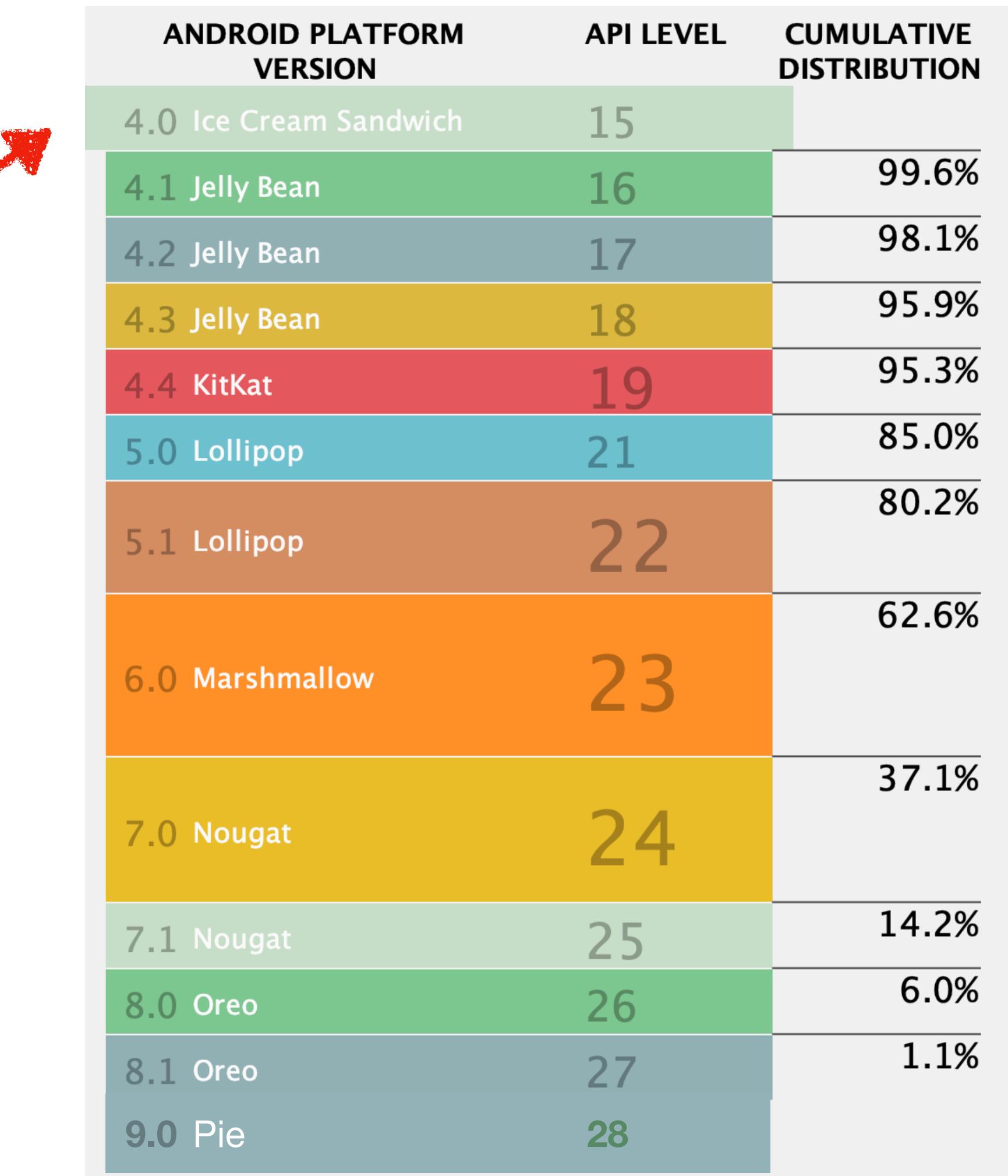
Project Sample

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



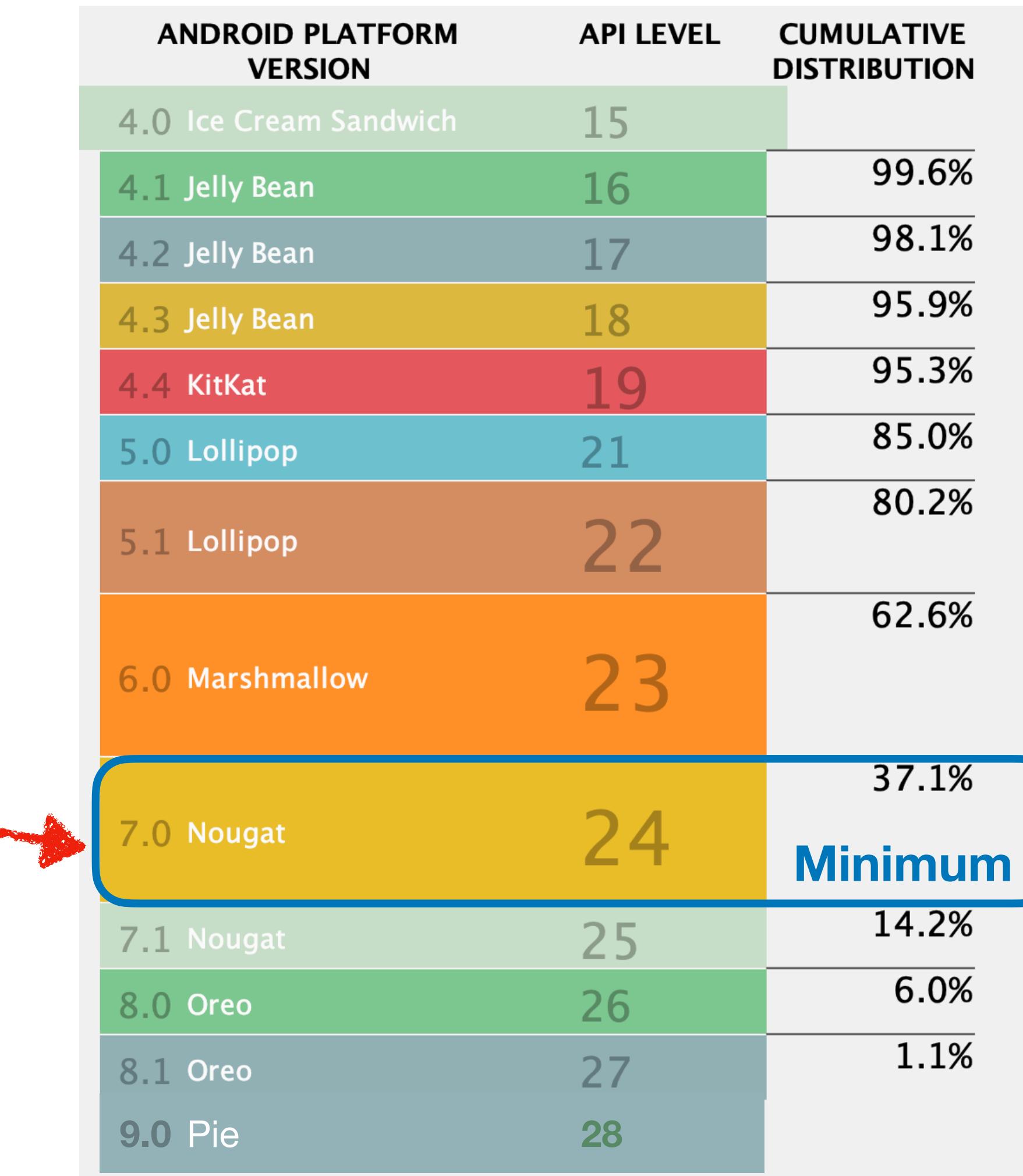
Project Sample

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



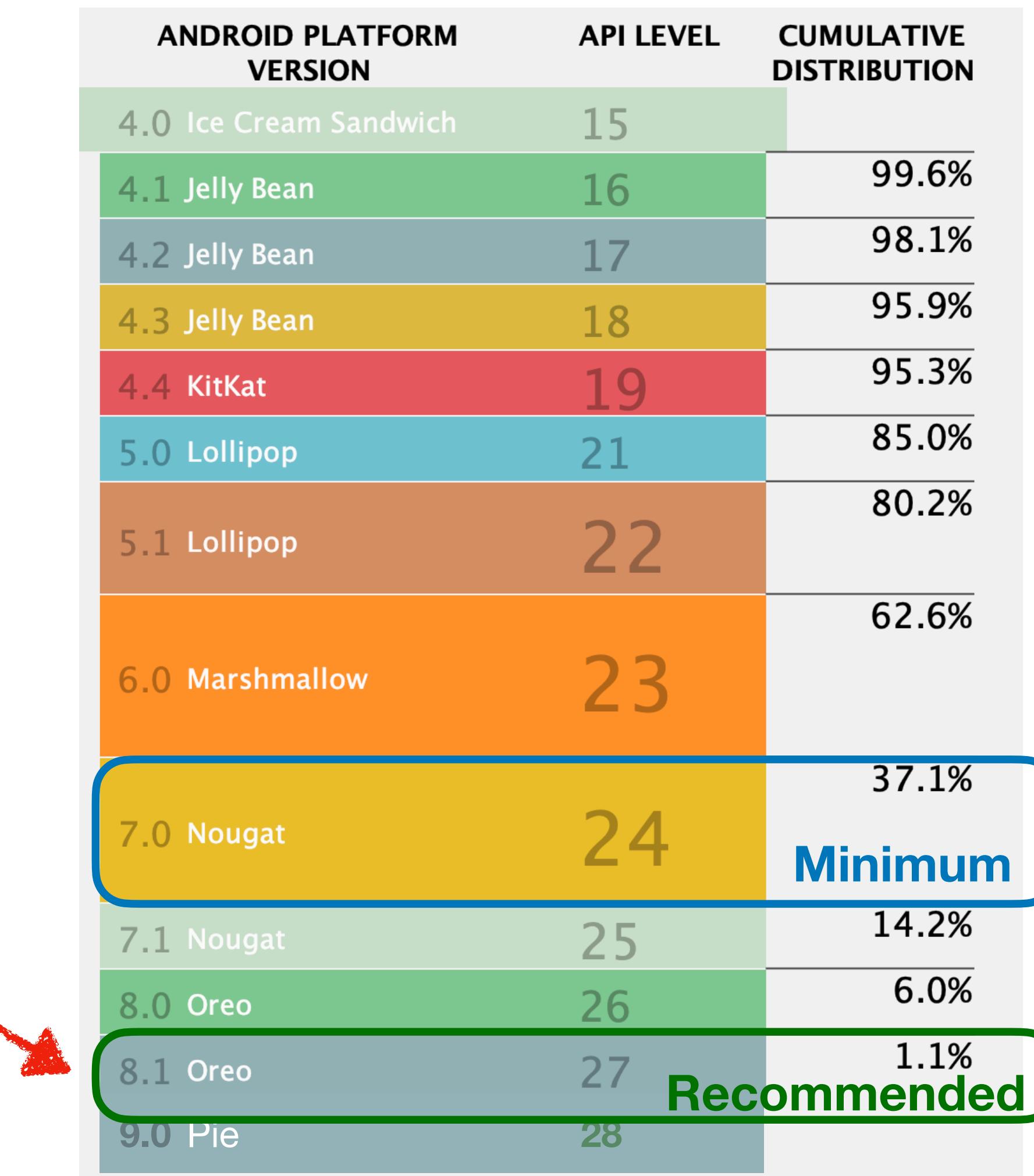
Project Sample

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

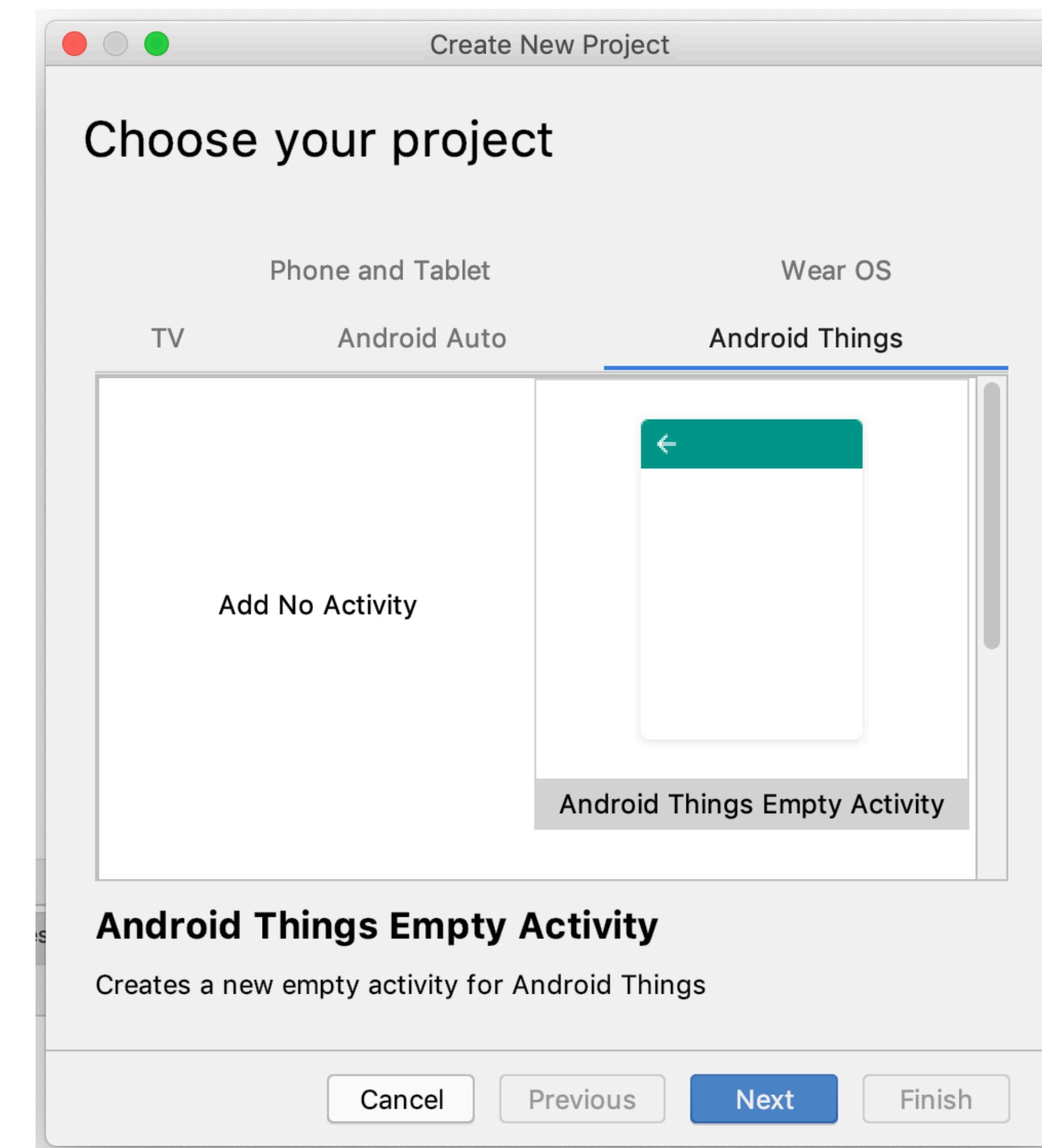


Project Sample

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

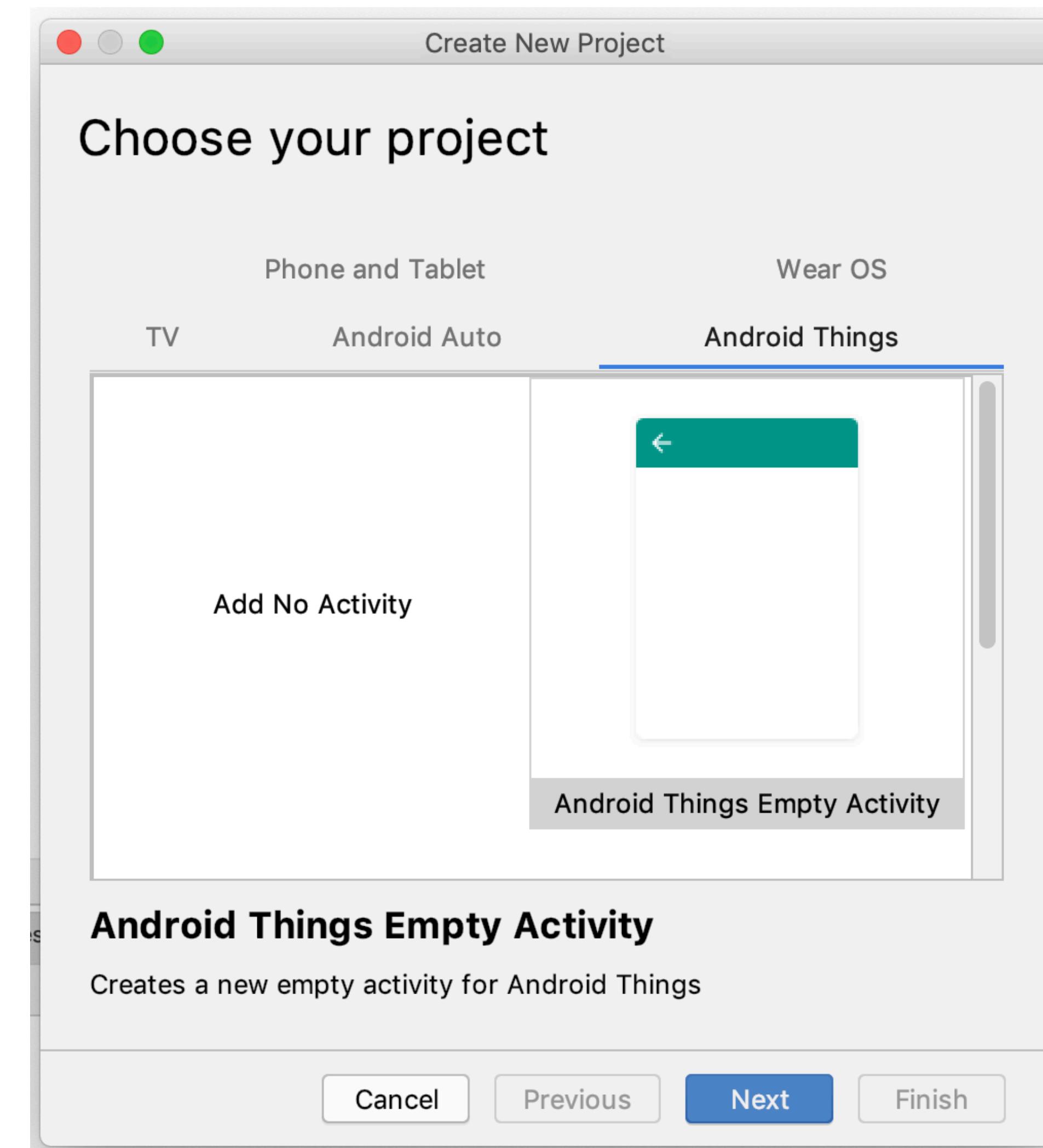


Create the Project



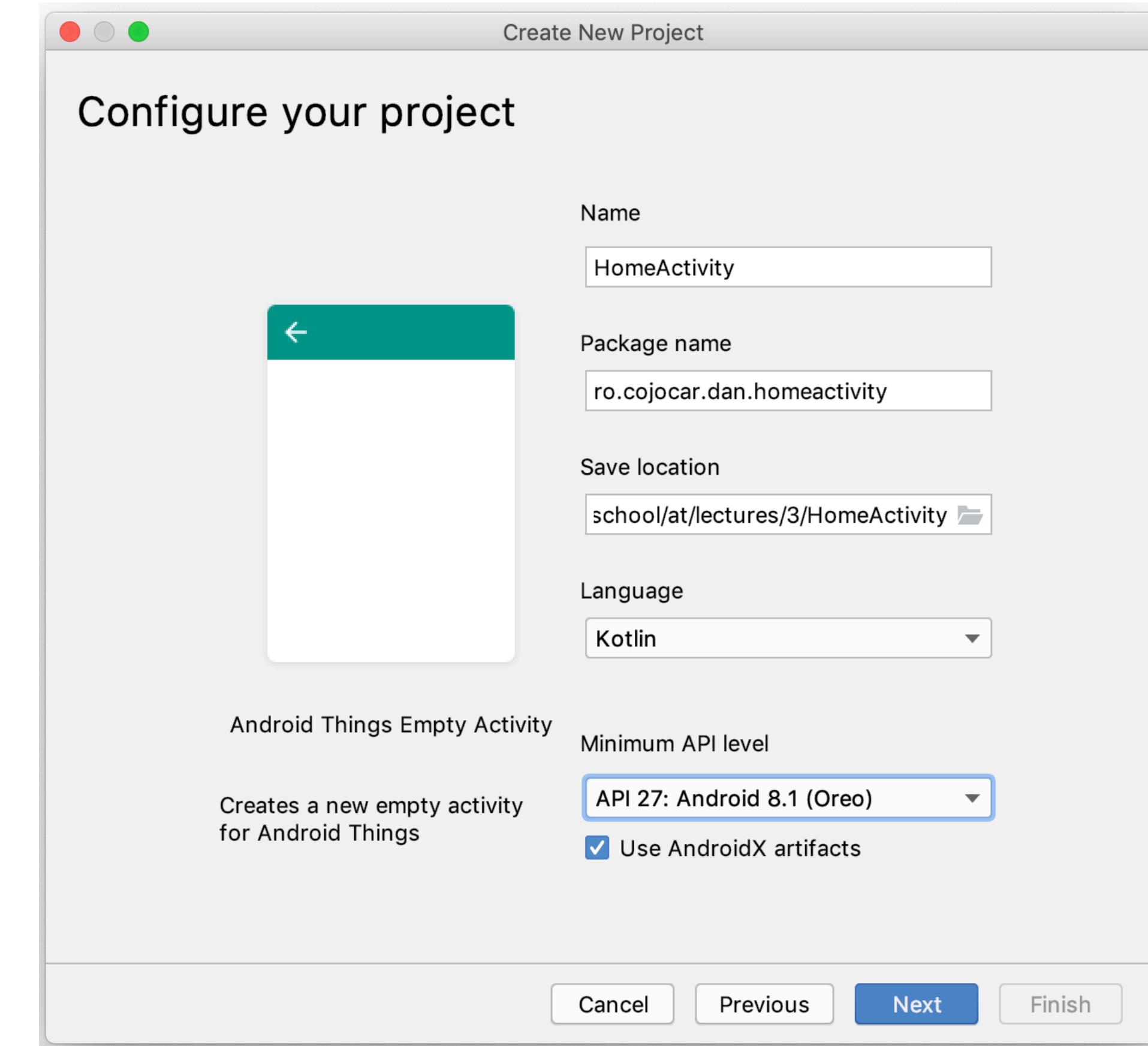
Create the Project

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



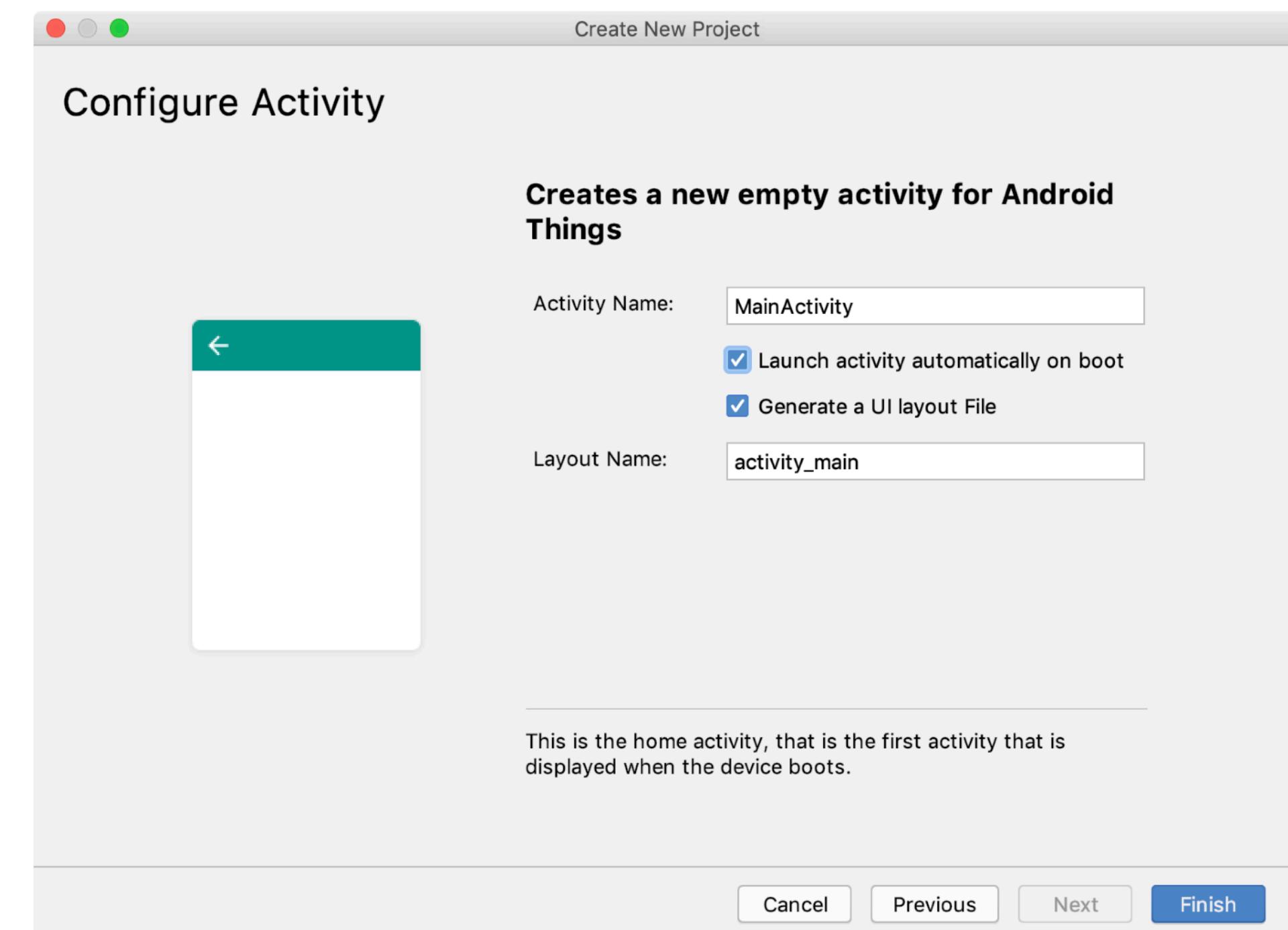
Create the Project

- 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>
```

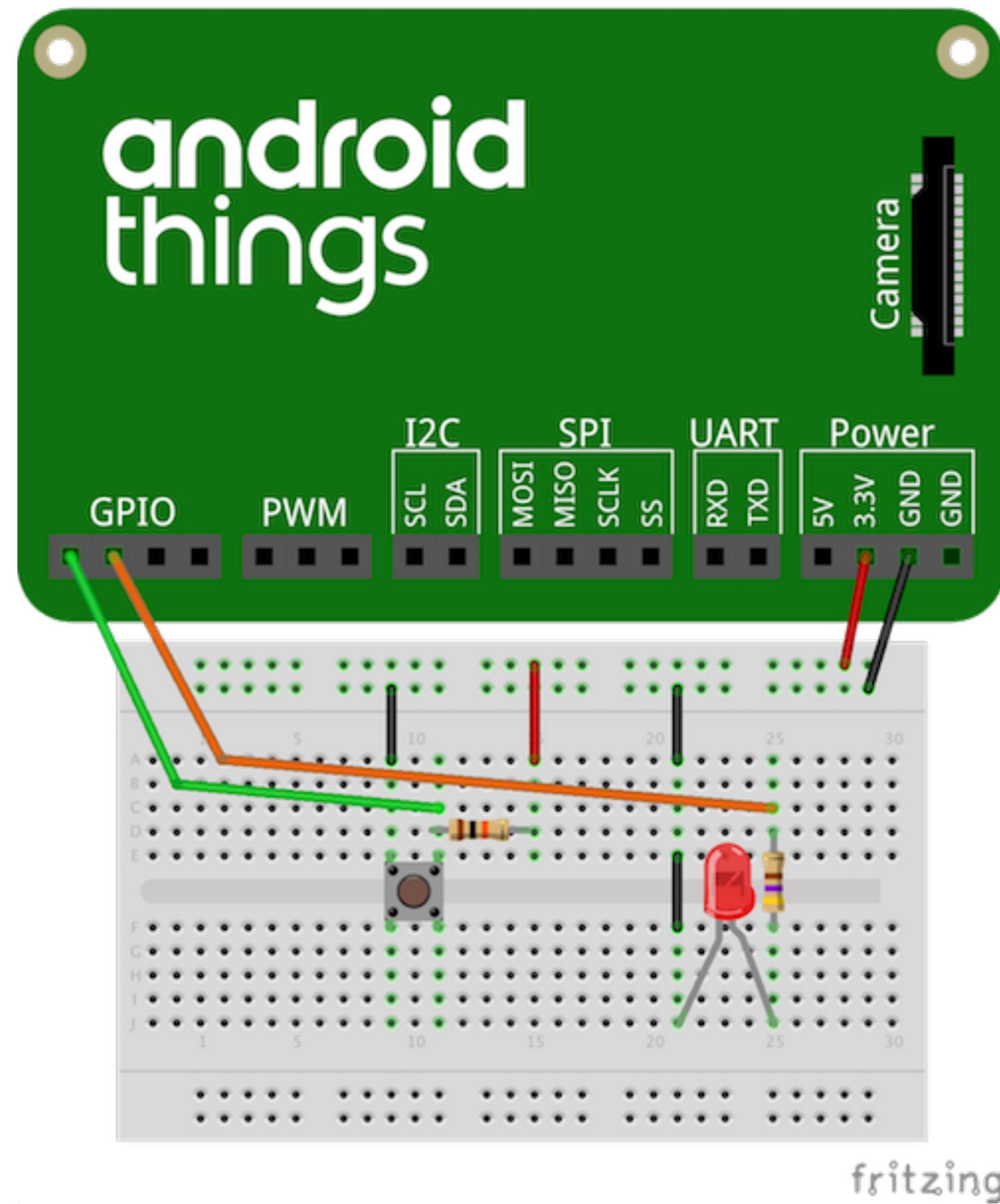
Key Generated Changes

- 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>
```

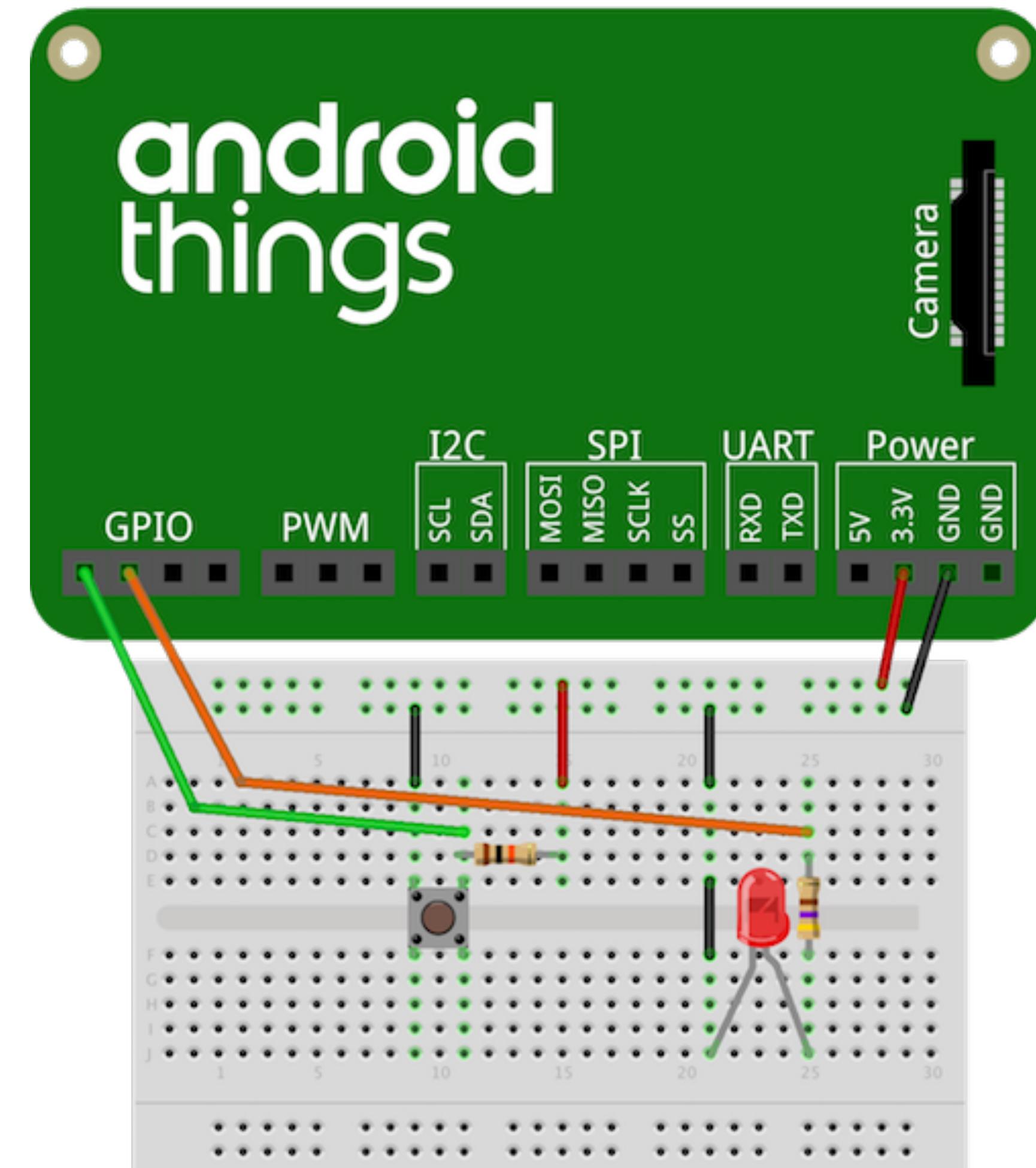
Connect the Hardware



fritzing

Connect the Hardware

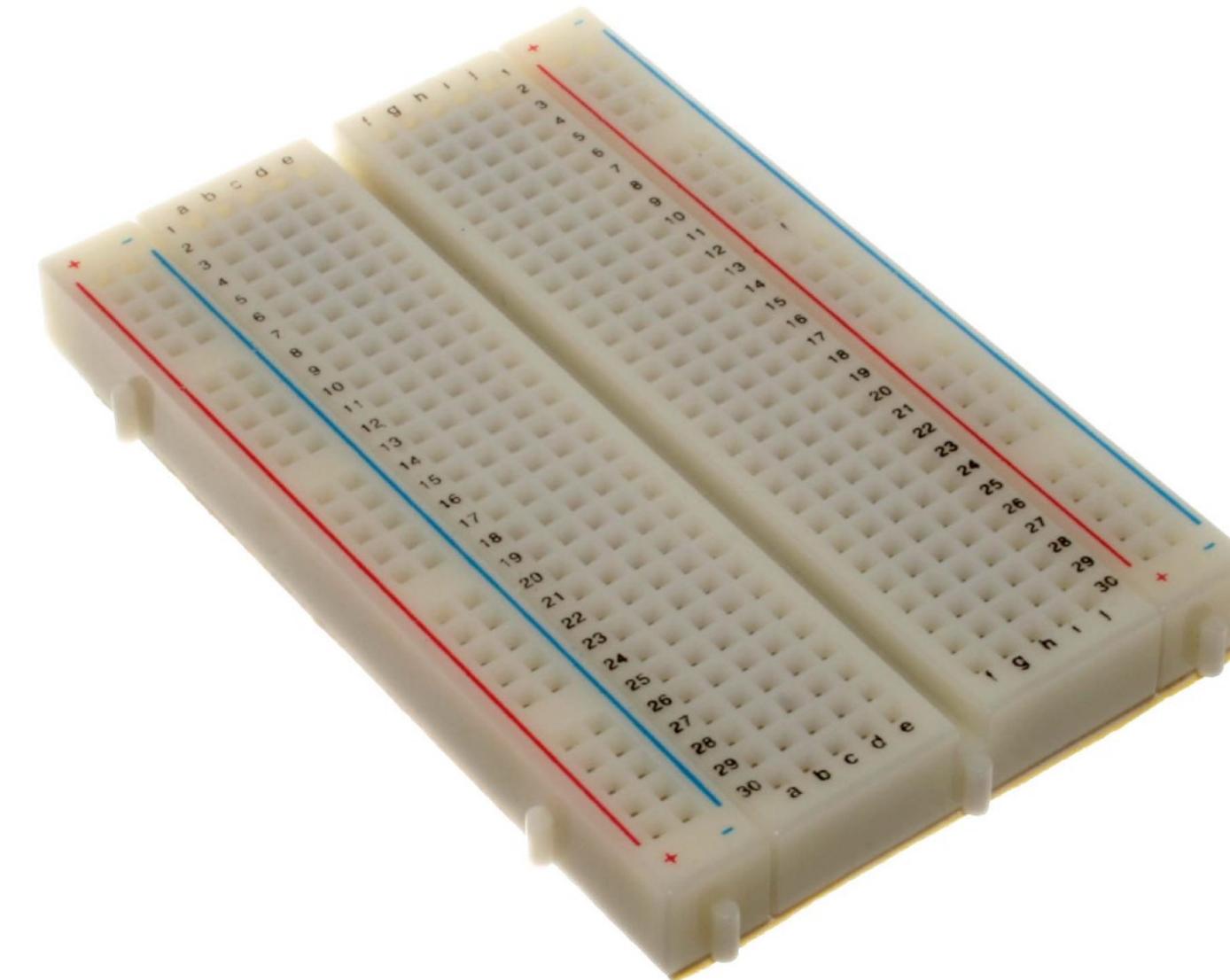
- Requirements:
 - The AndroidThings board.



fritzing

Connect the Hardware

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



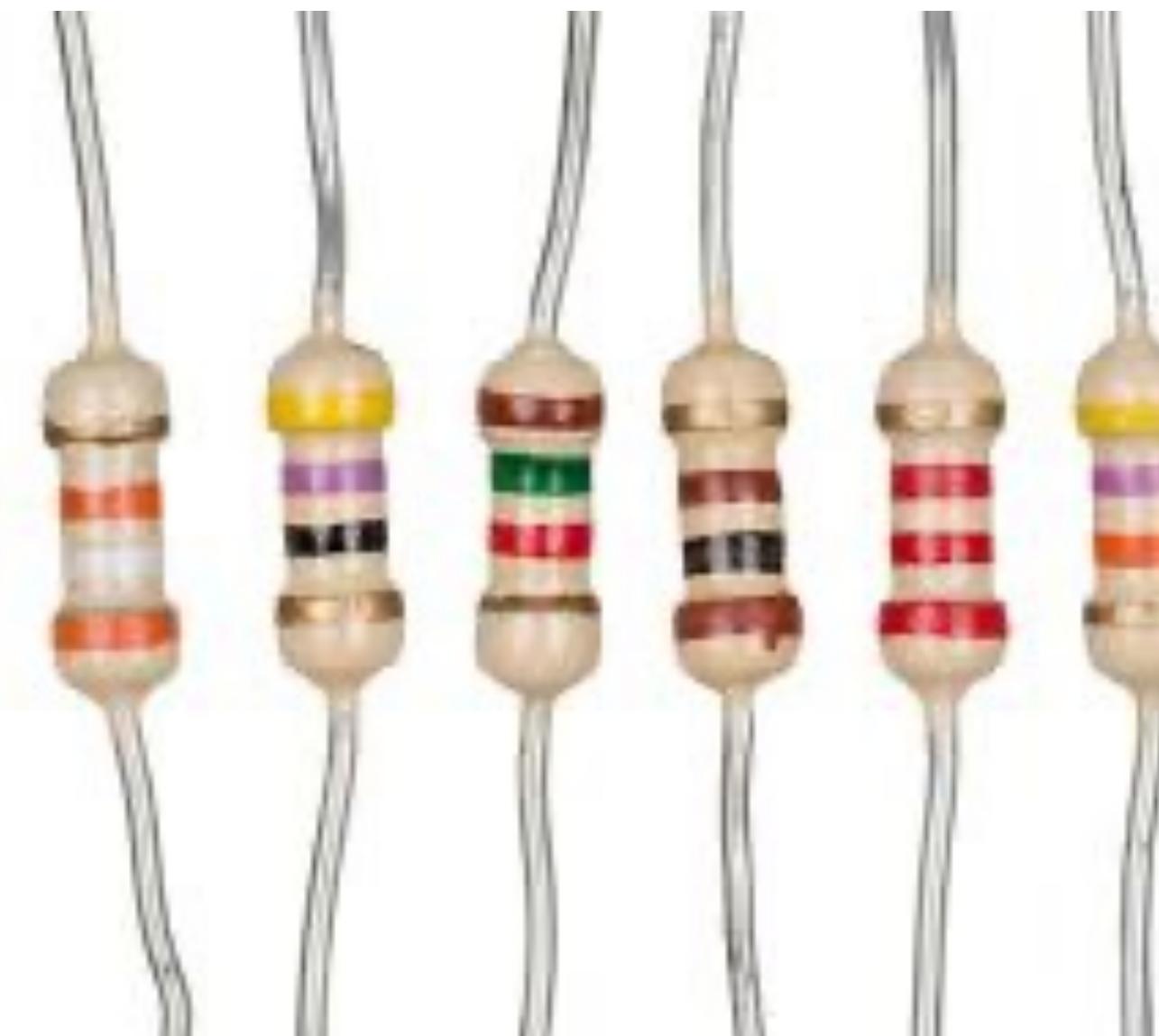
Connect the Hardware

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



Connect the Hardware

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



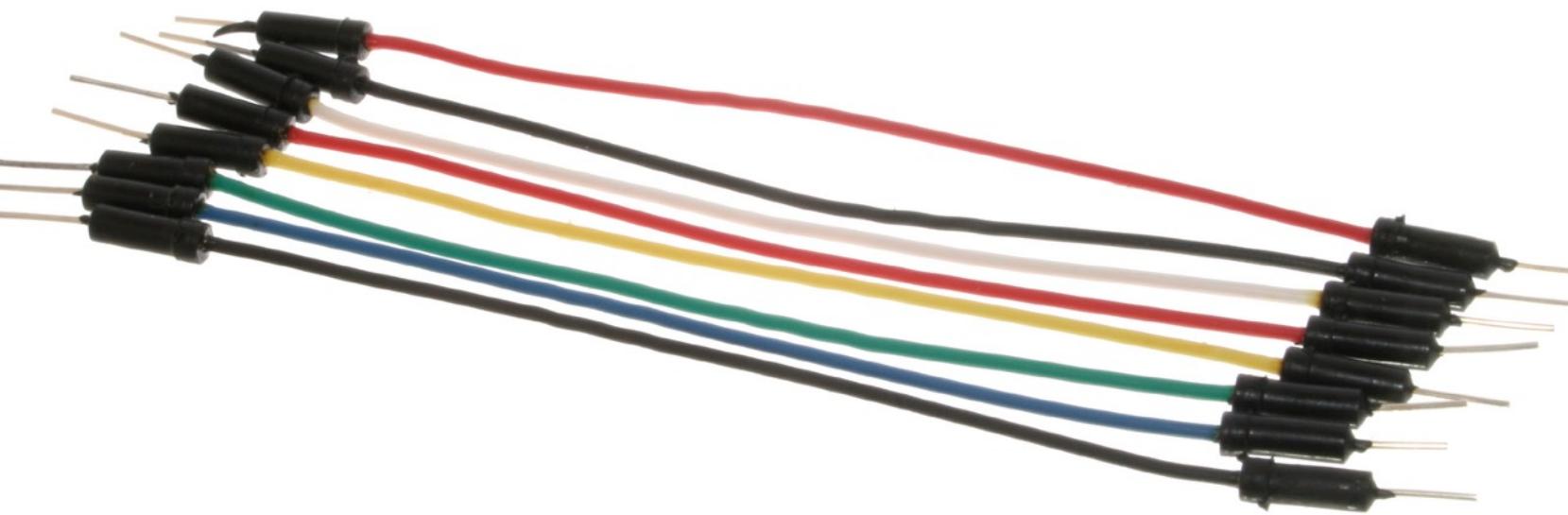
Connect the Hardware

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

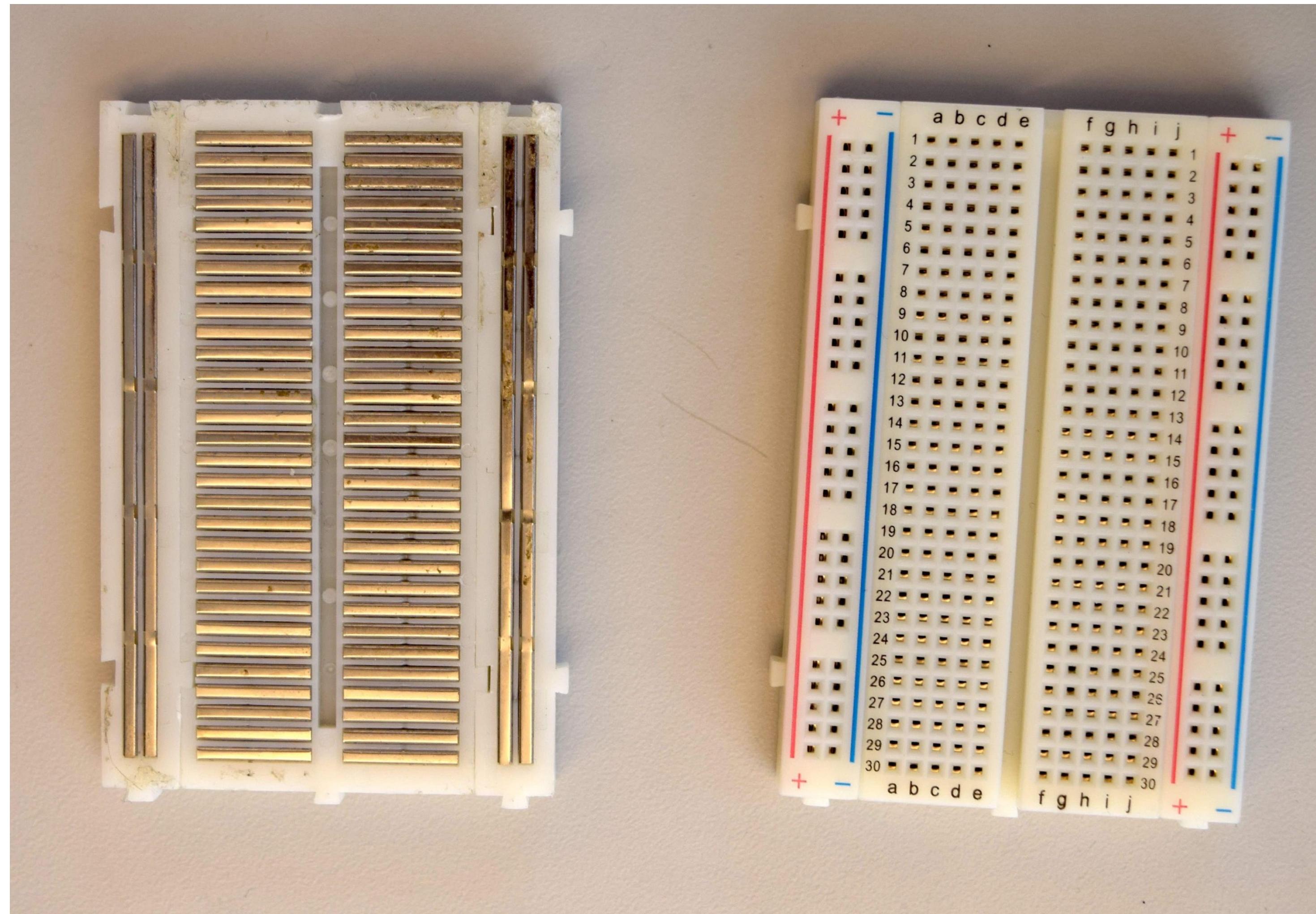


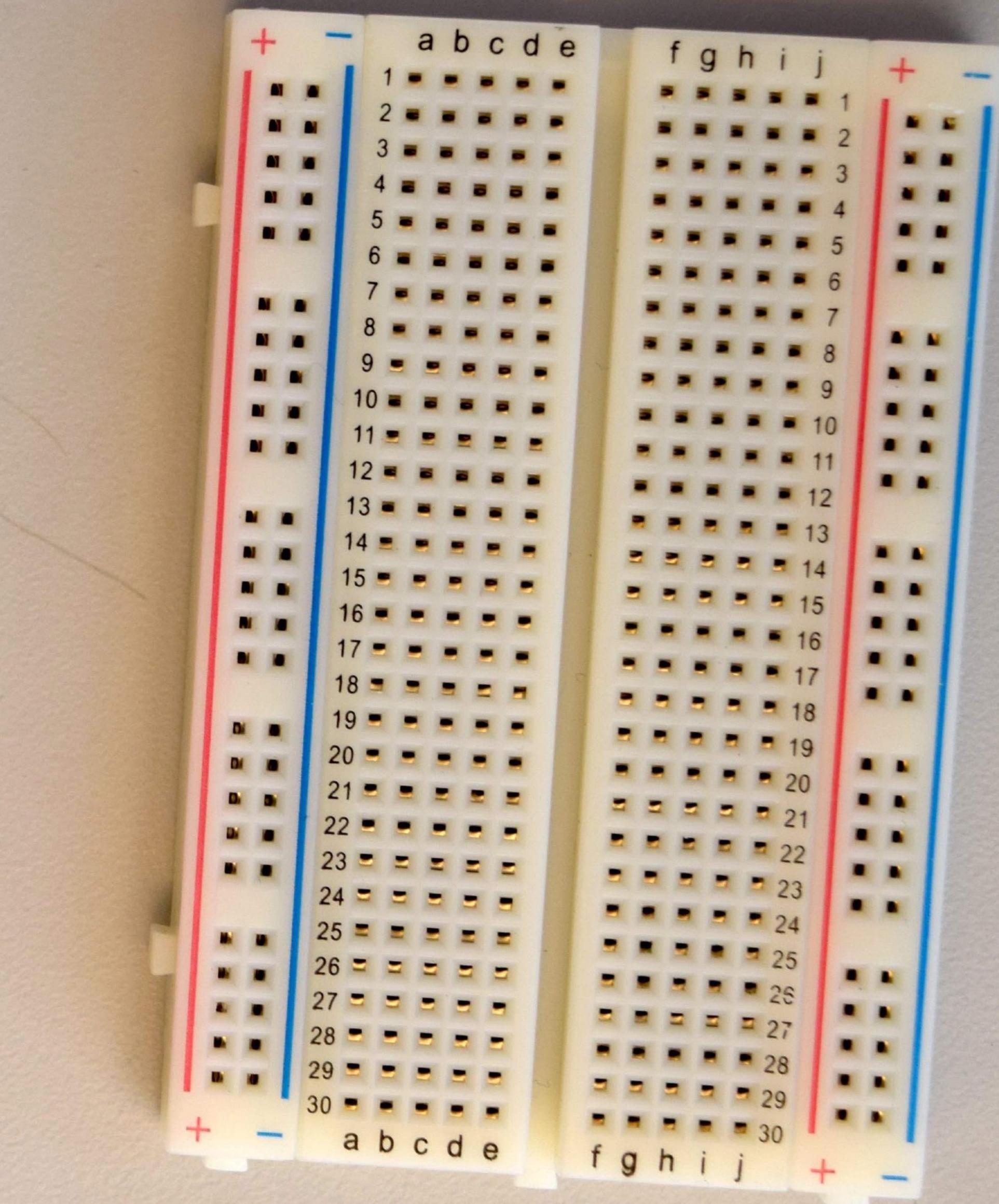
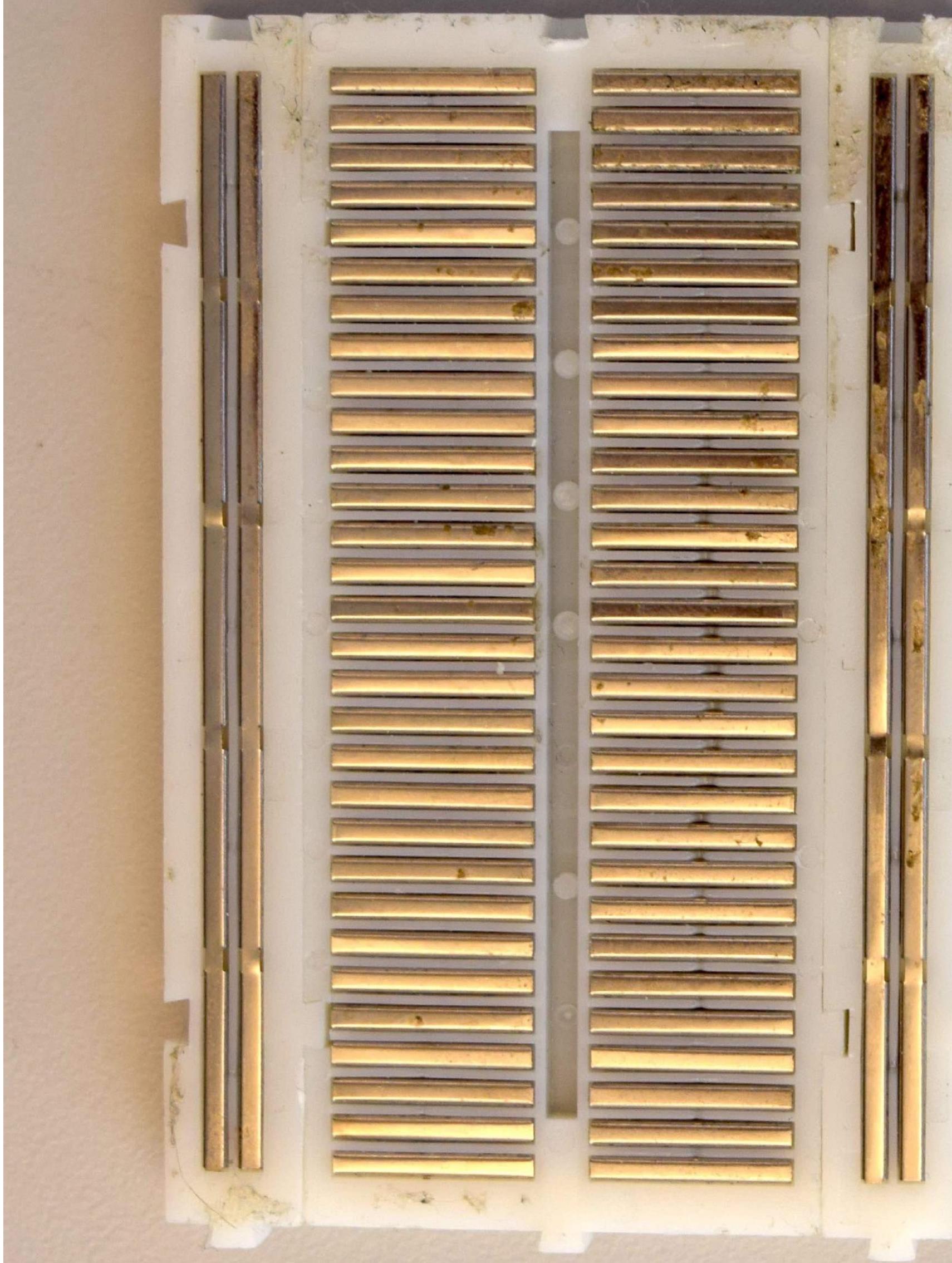
Connect the Hardware

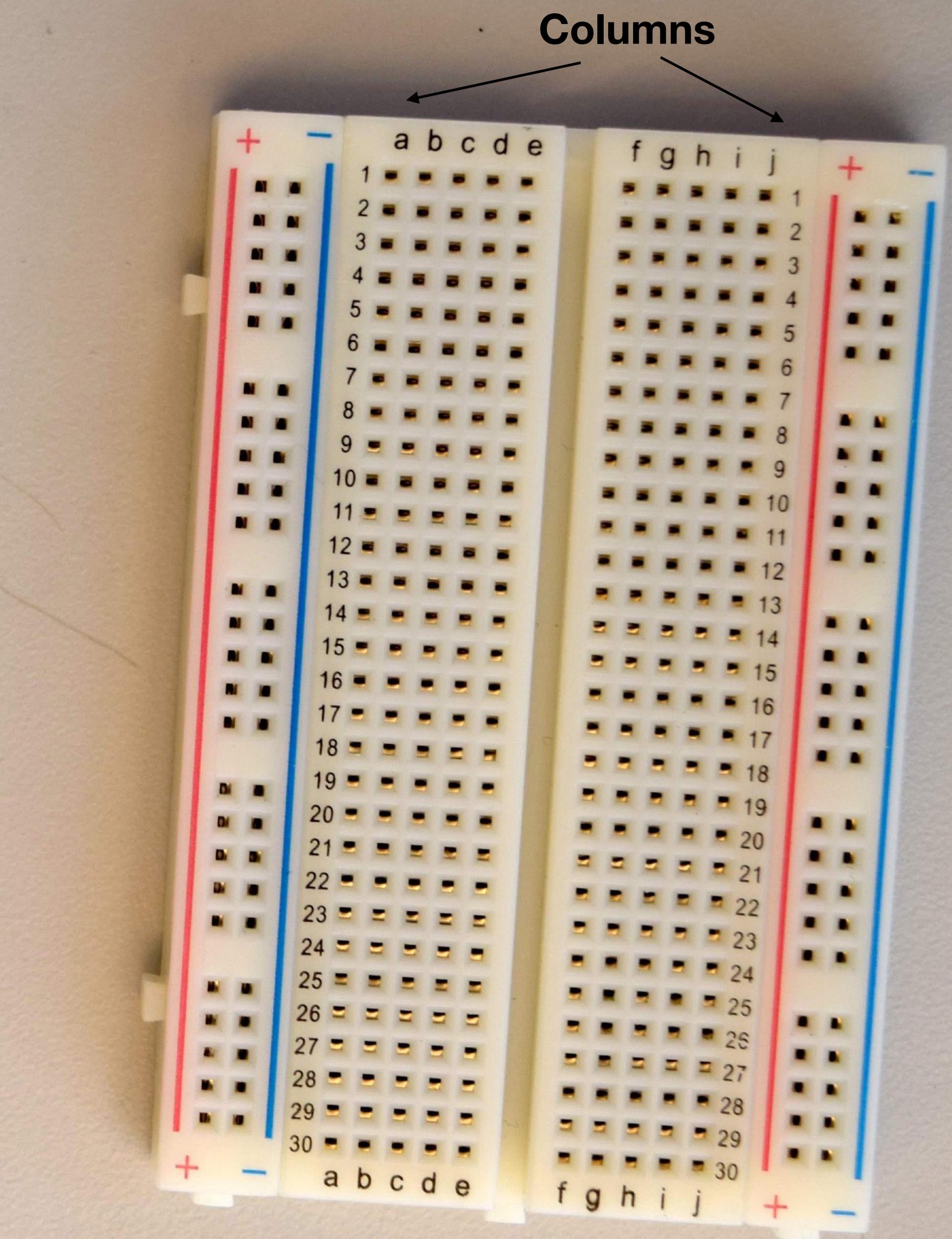
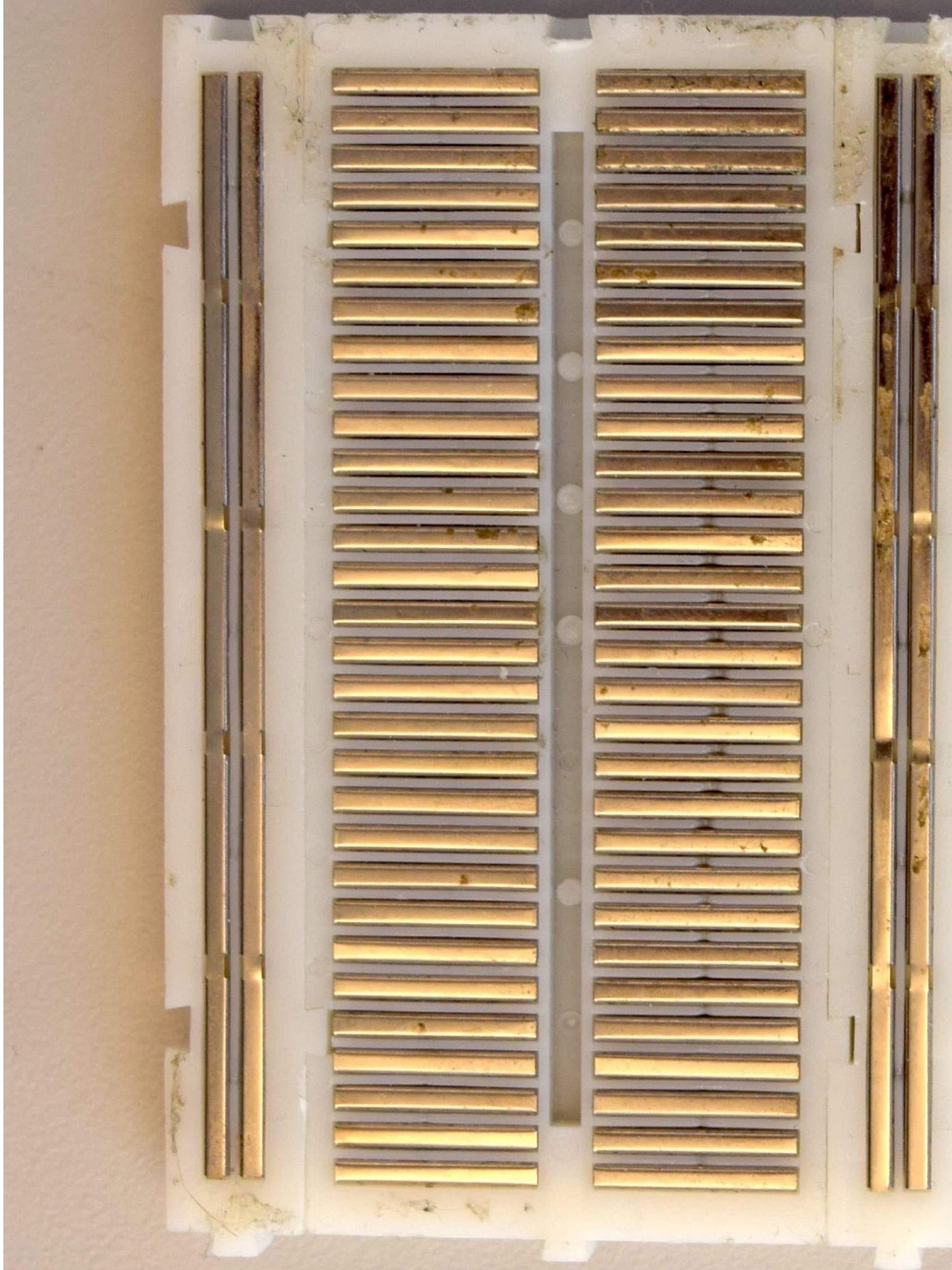
- Requirements:
 - The AndroidThings board.
 - A breadboard.
 - A push button.
 - Two resistors.
 - A LED.
 - Jumper wires.

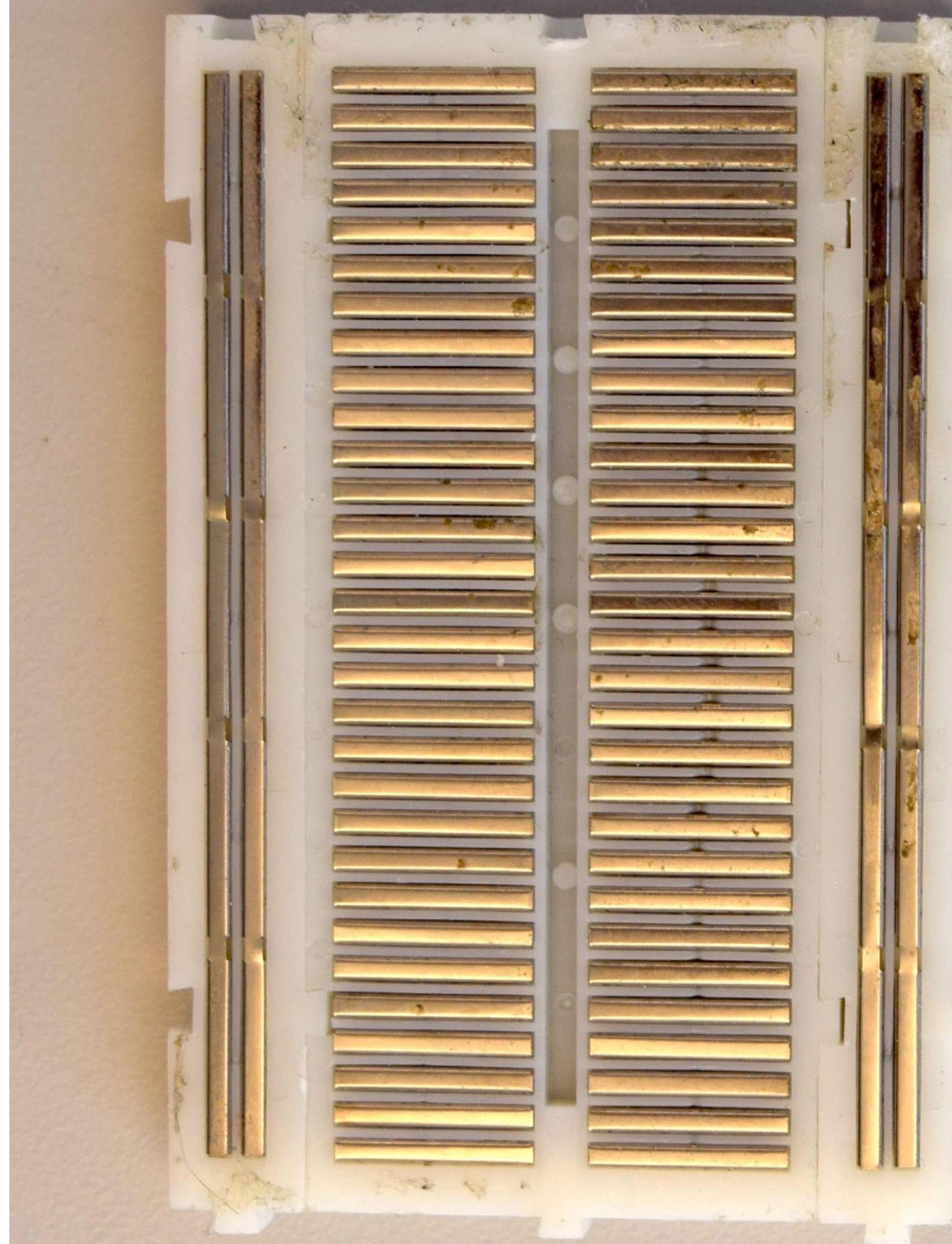


The breadboard





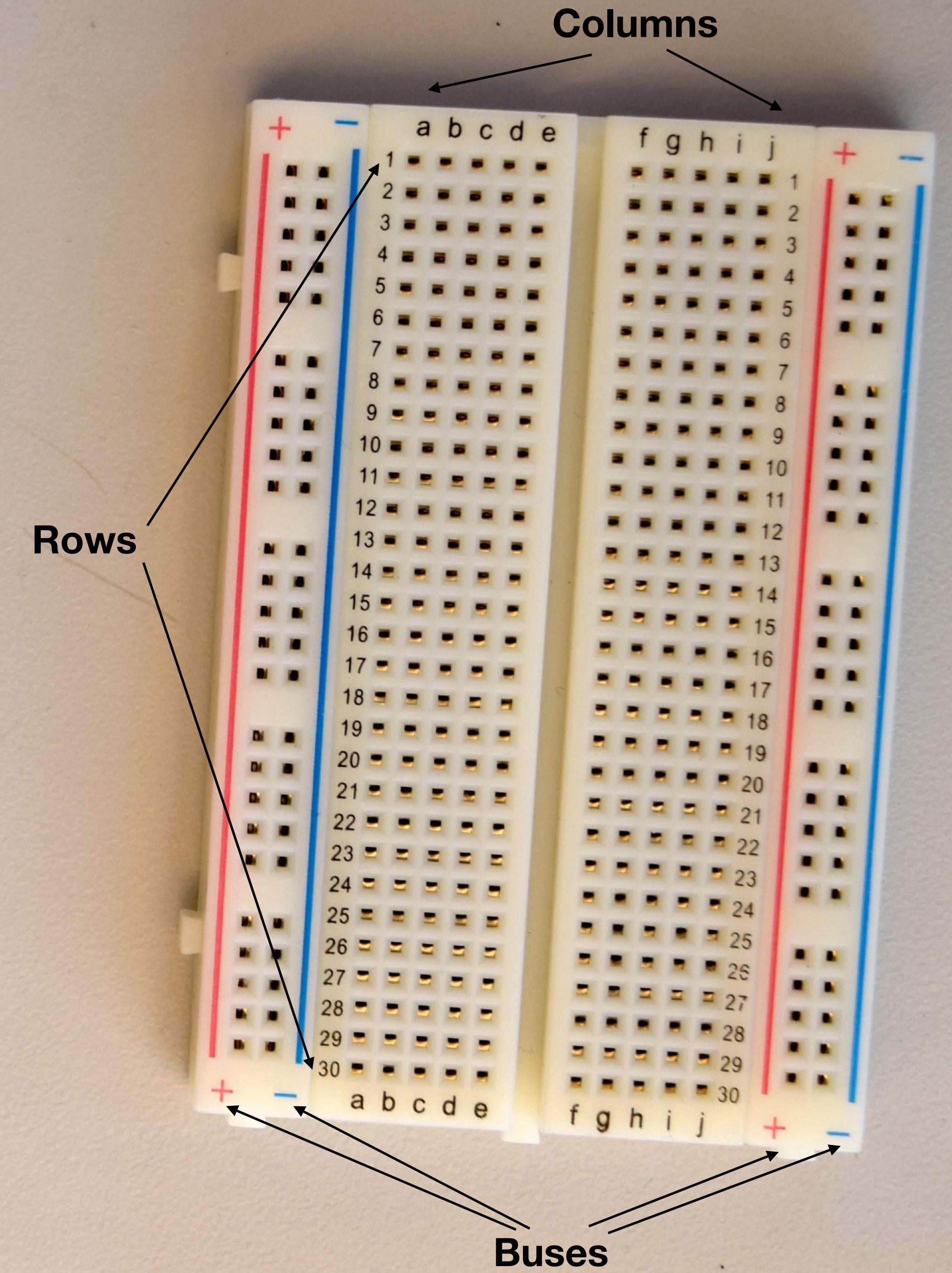
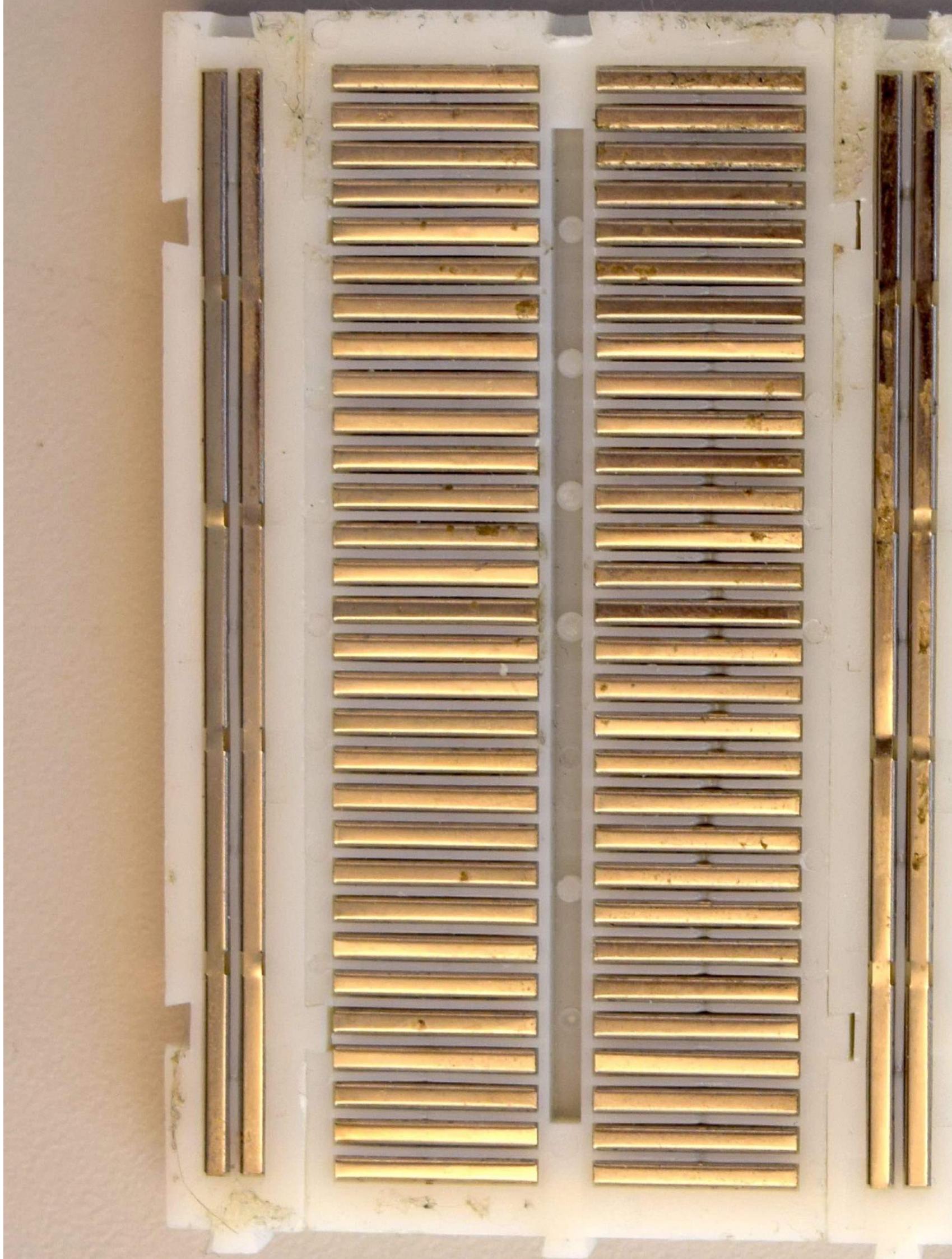




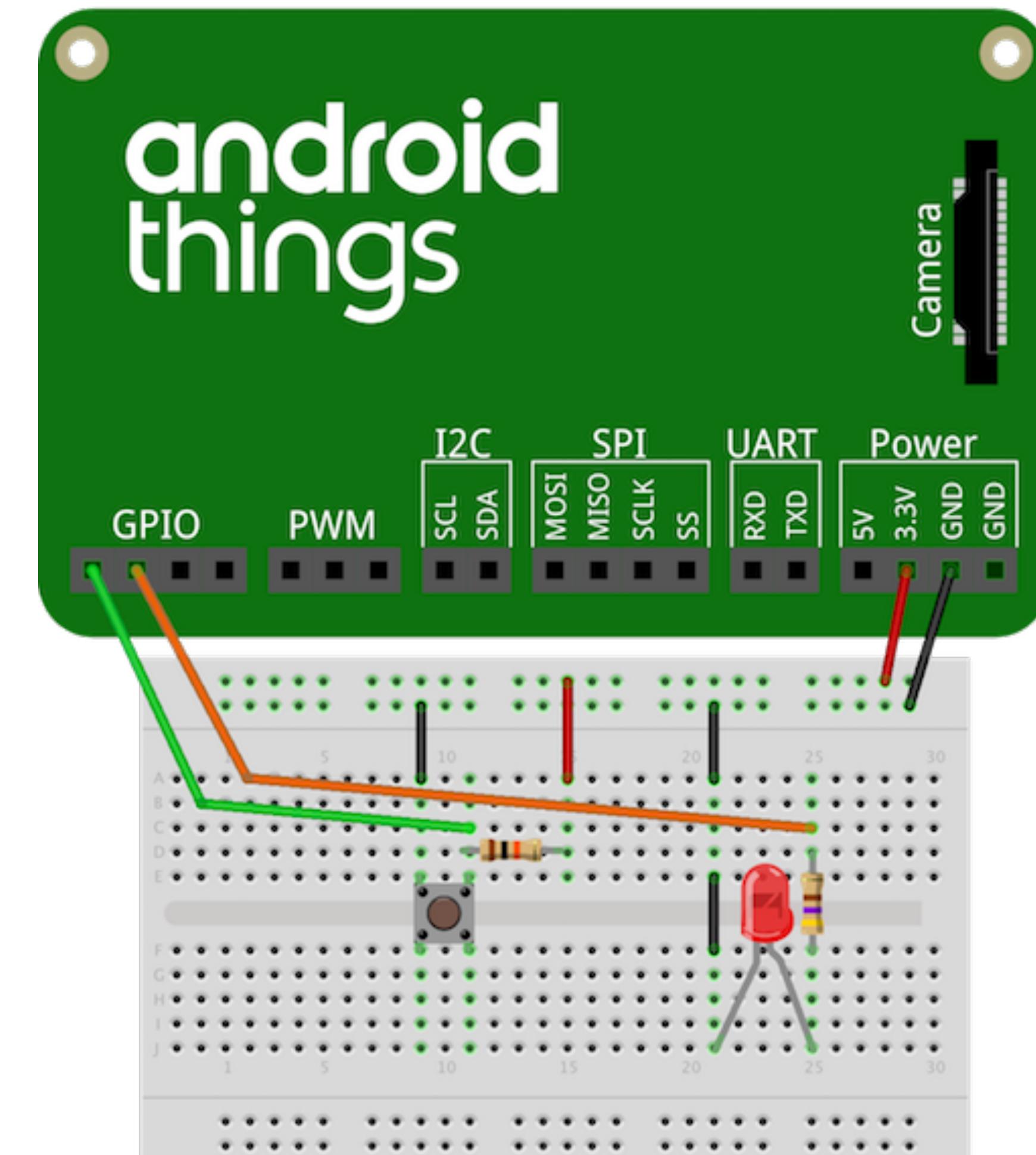
Row

Columns

The image shows two white breadboards side-by-side. The left breadboard has its top row of pins labeled with '+' and '-' terminals. A red vertical line runs along its left edge, ending in a red '+' terminal at the top and a red '-' terminal at the bottom. A blue vertical line runs along its right edge, ending in a blue '+' terminal at the top and a blue '-' terminal at the bottom. A black diagonal line with an arrow points from the red '+' terminal on the left breadboard to the blue '+' terminal on the right breadboard. The right breadboard also has its top row of pins labeled with '+' and '-' terminals. Both breadboards have columns of pins labeled 'a' through 'j' across their top and bottom rows. The left breadboard's columns are labeled 'a' through 'e', and the right breadboard's columns are labeled 'f' through 'j'. Each column is numbered from 1 to 30 vertically. The breadboards are connected by a single jumper wire that spans the gap between them, connecting the blue '+' terminal of the left board to the blue '+' terminal of the right board.



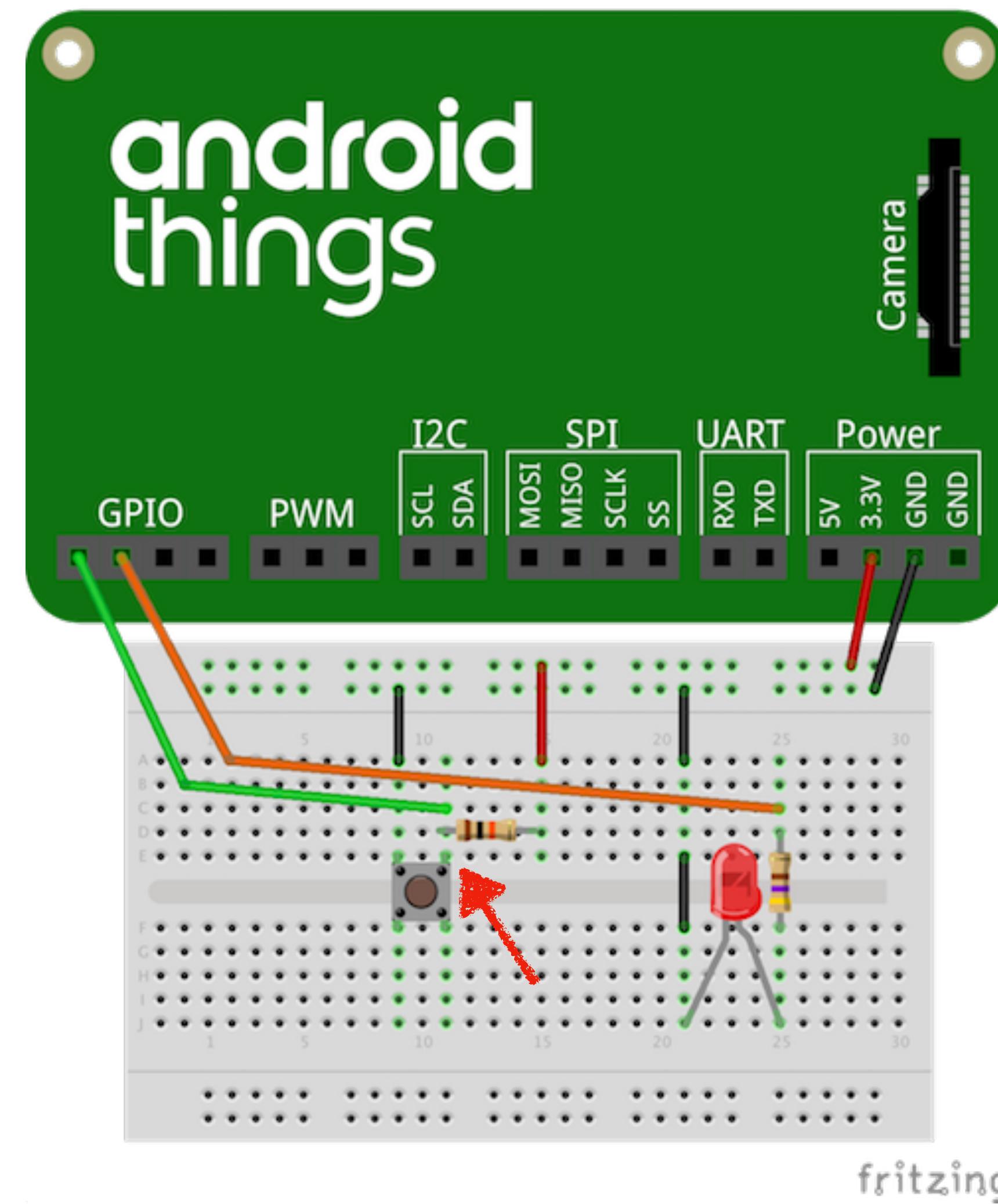
Create the Connections



fritzing

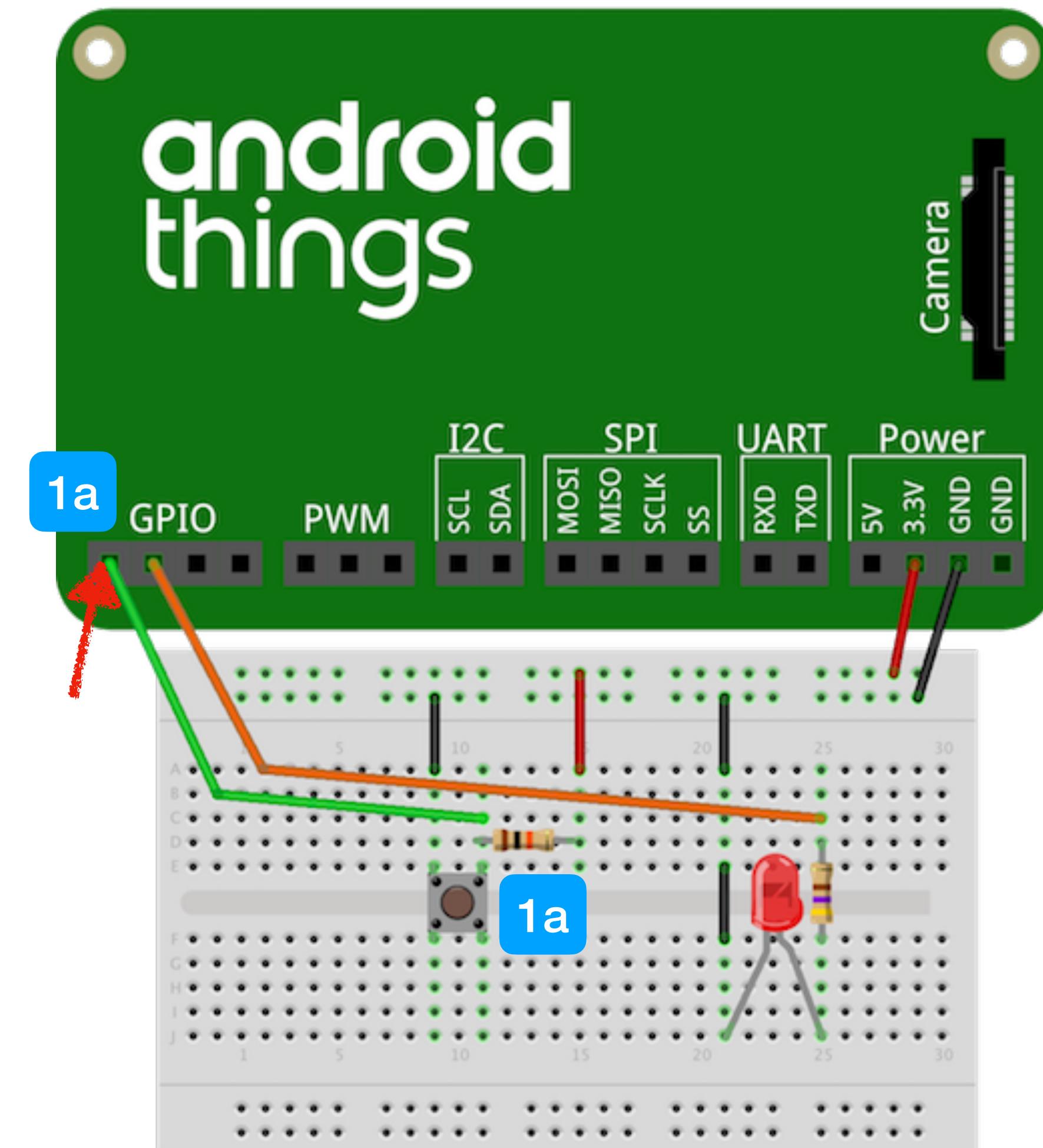
Create the Connections

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



Create the Connections

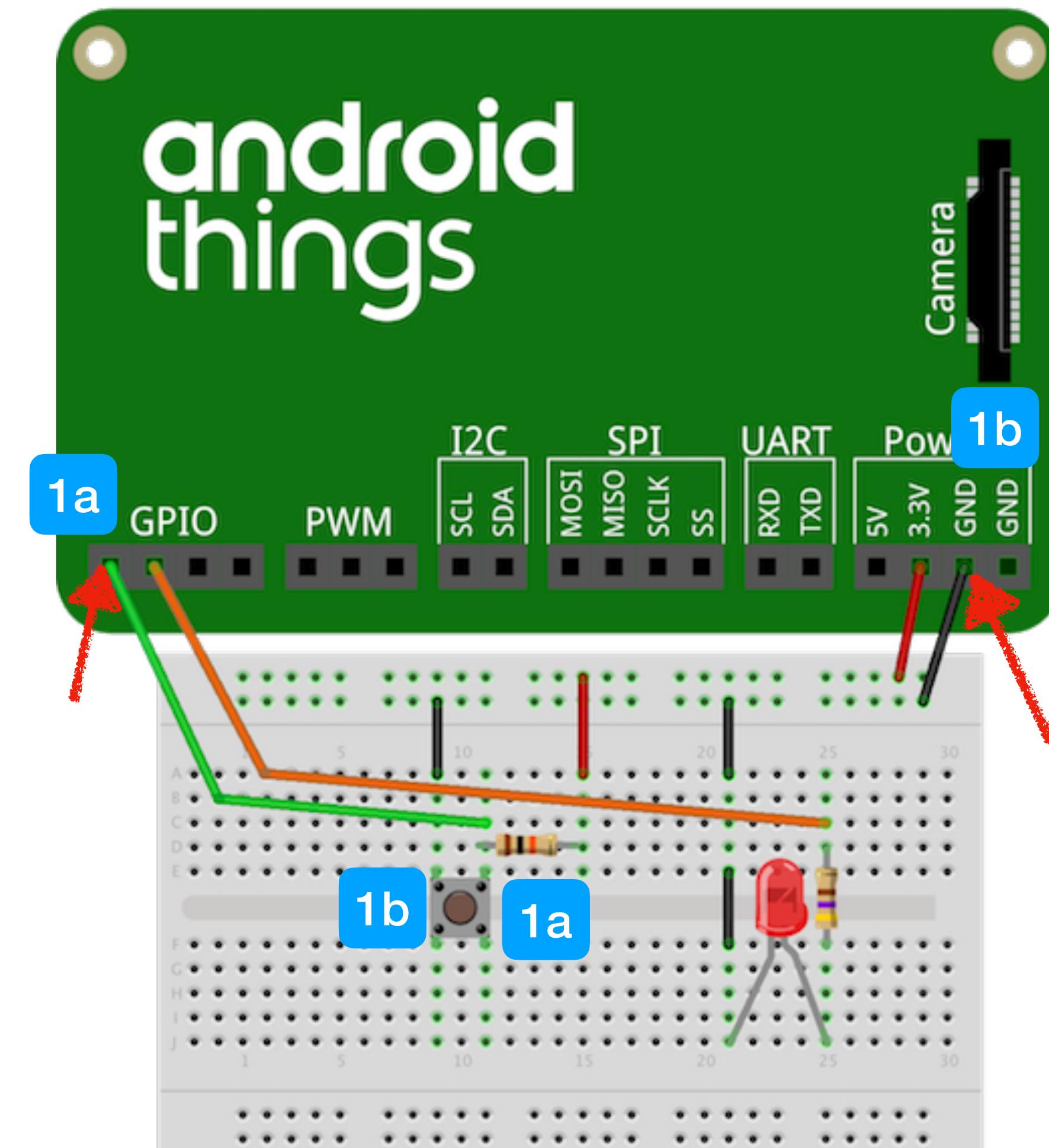
1. Connect one side of the button to the chosen GPIO input pin, and the other side to ground.



fritzing

Create the Connections

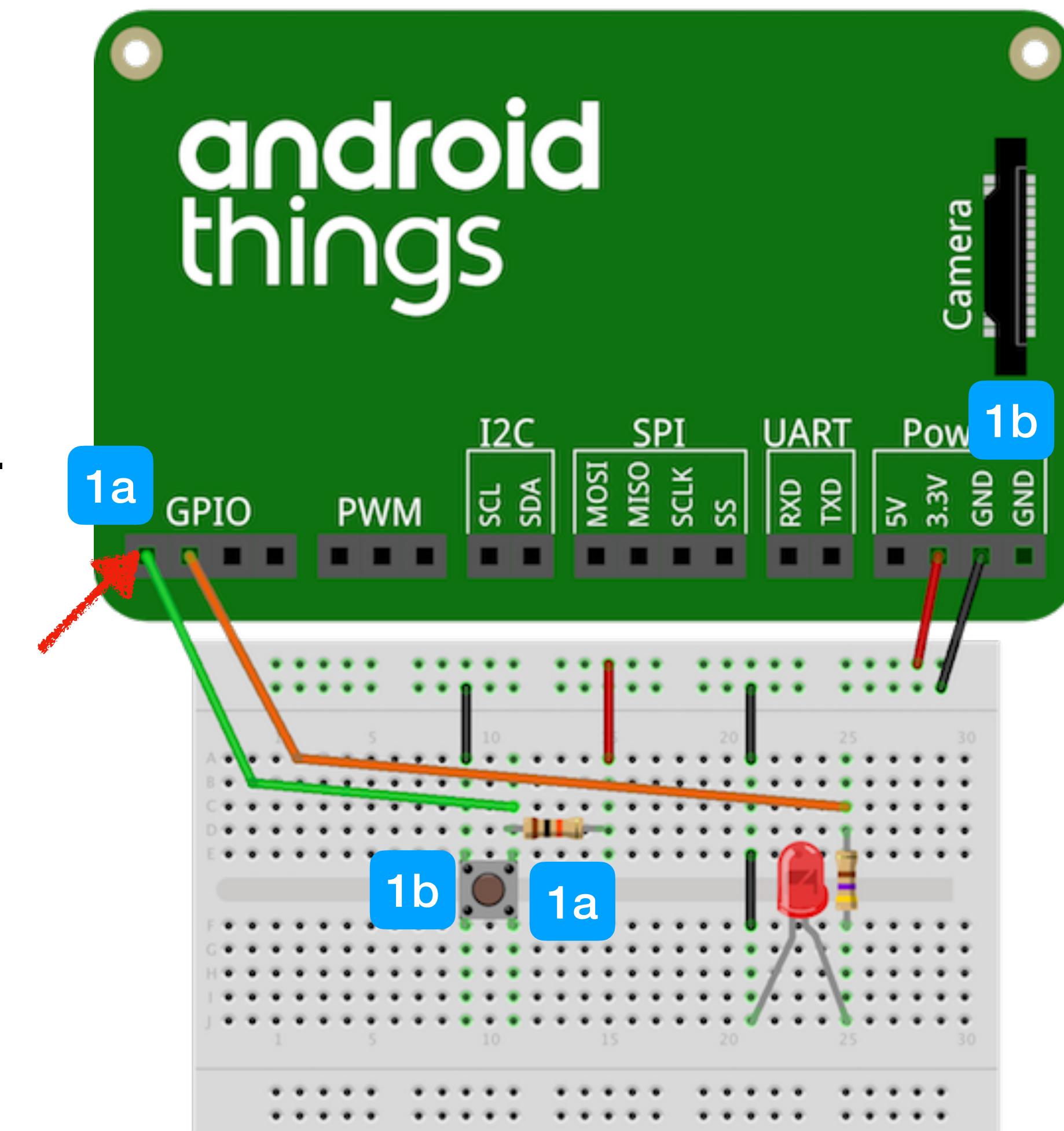
1. Connect one side of the button to the chosen GPIO input pin, and the other side to ground.



fritzing

Create the Connections

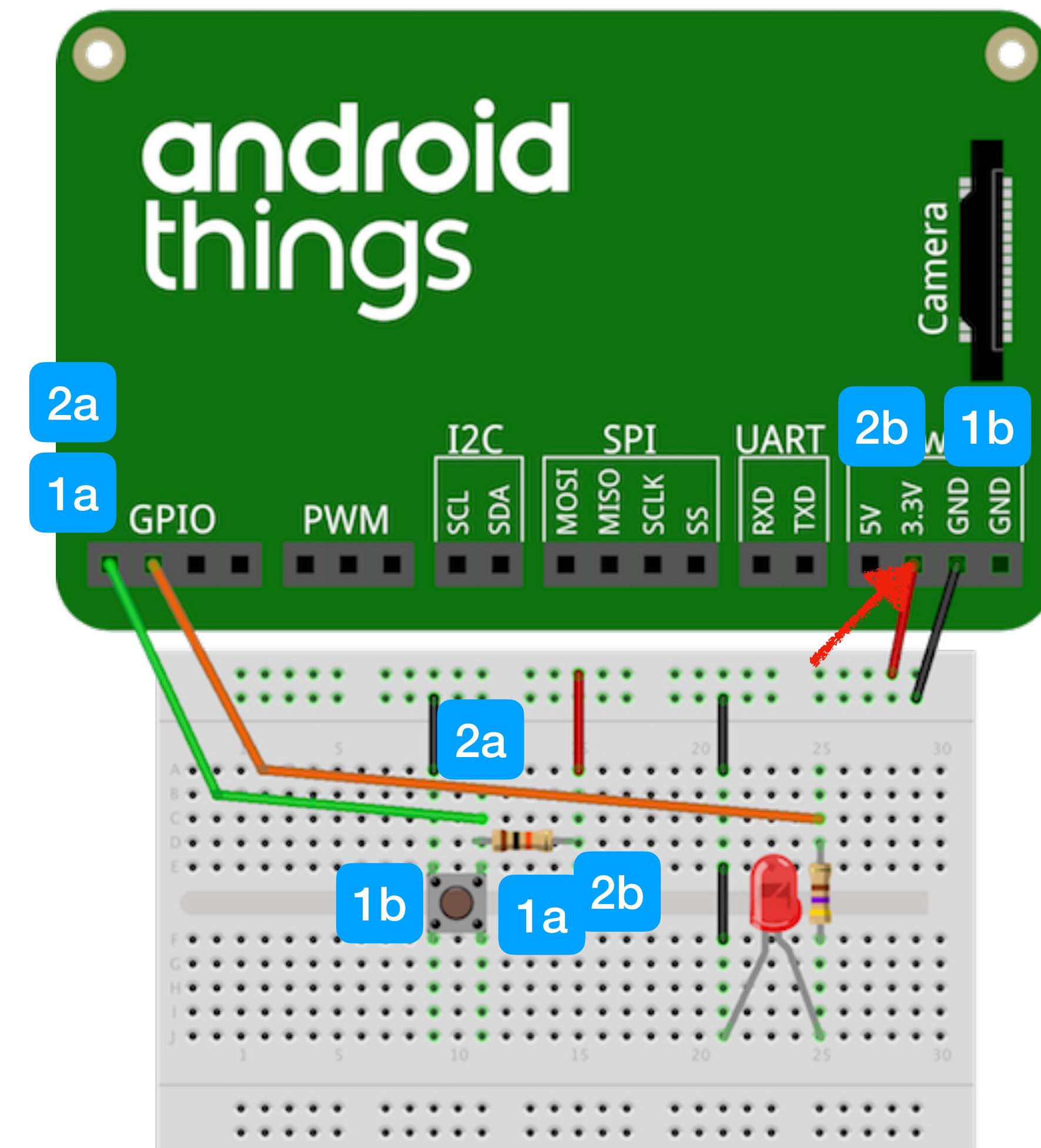
- 1 Connect one side of the button to the chosen GPIO input pin, and the other side to ground.
2. Connect the same GPIO input pin to +3.3V through a pull-up resistor.



fritzing

Create the Connections

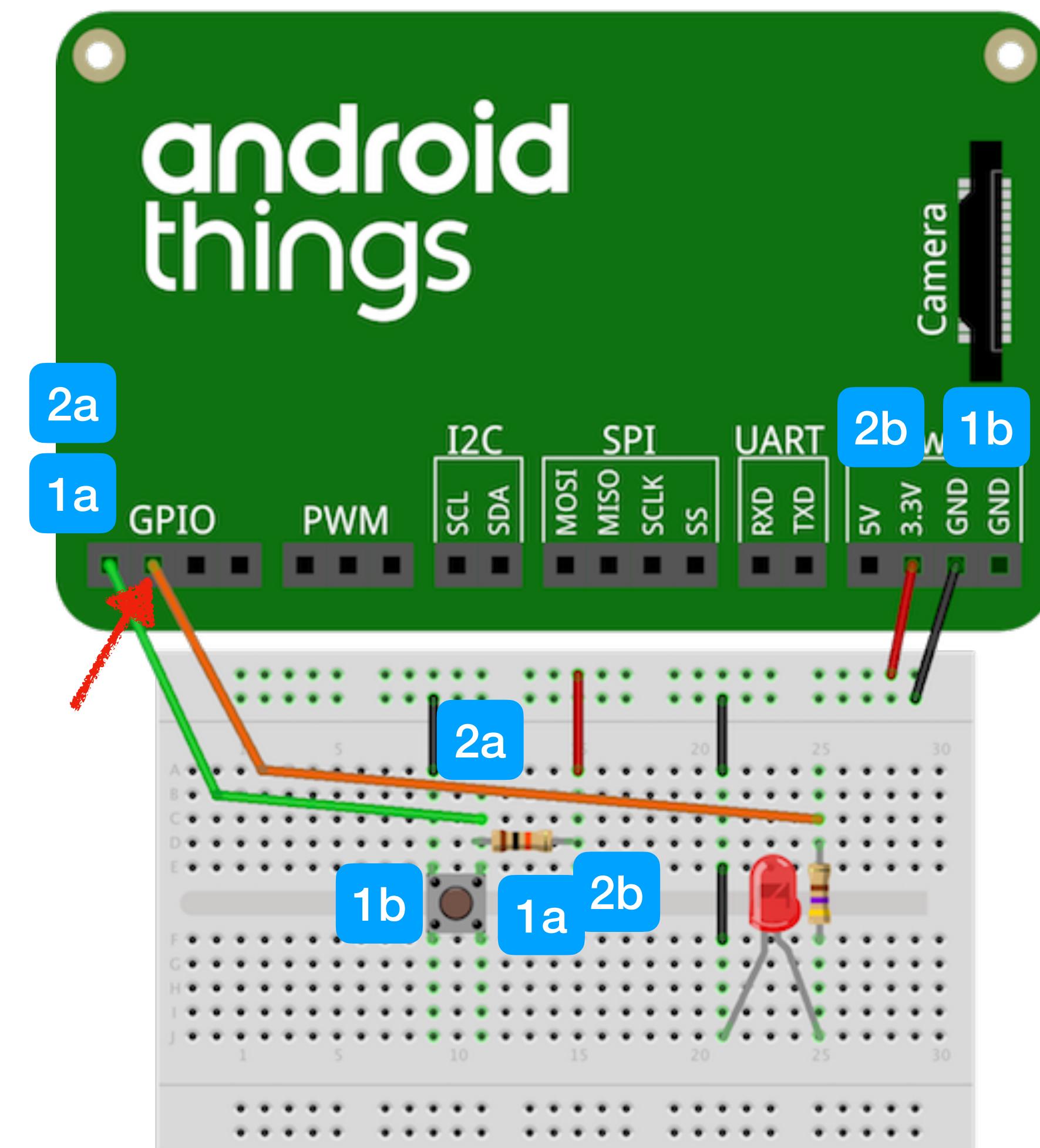
- 1 Connect one side of the button to the chosen GPIO input pin, and the other side to ground.
2. Connect the same GPIO input pin to +3.3V through a pull-up resistor.



fritzing

Create the Connections

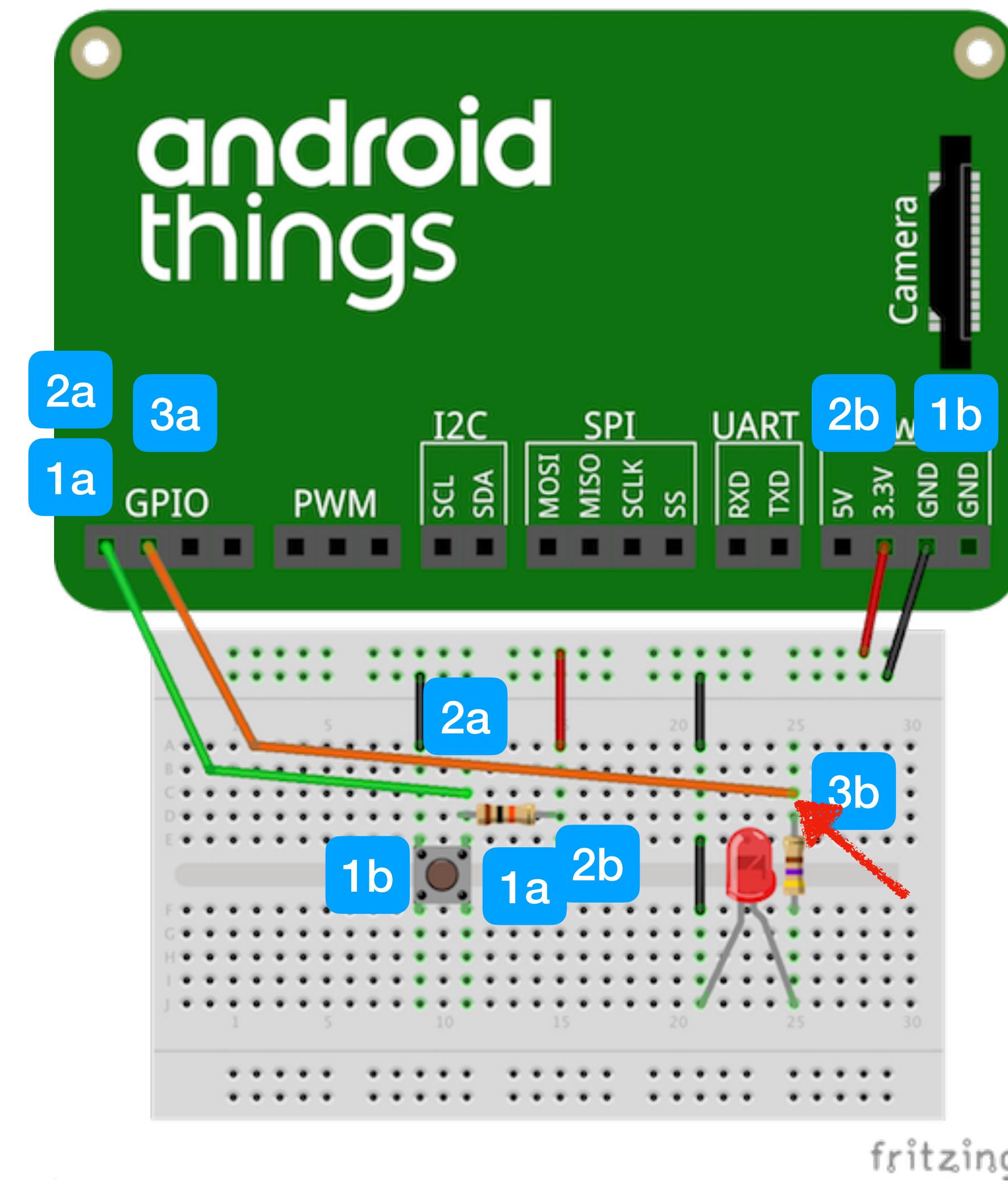
- 1 Connect one side of the button to the chosen GPIO input pin, and the other side to ground.
- 2 Connect the same GPIO input pin to +3.3V through a pull-up resistor.
3. Connect the chosen GPIO output pin to one side of a series resistor.



fritzing

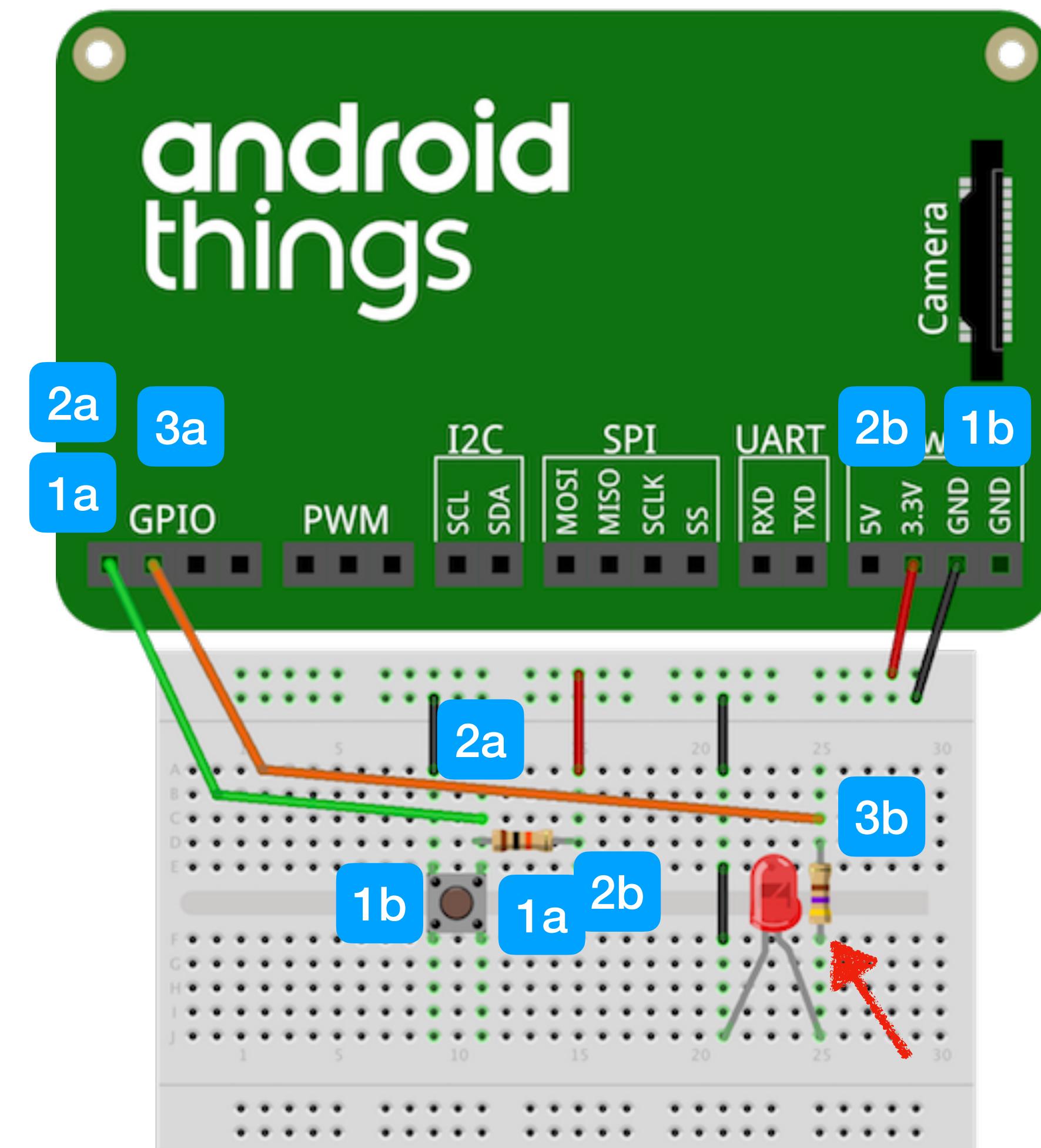
Create the Connections

- 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

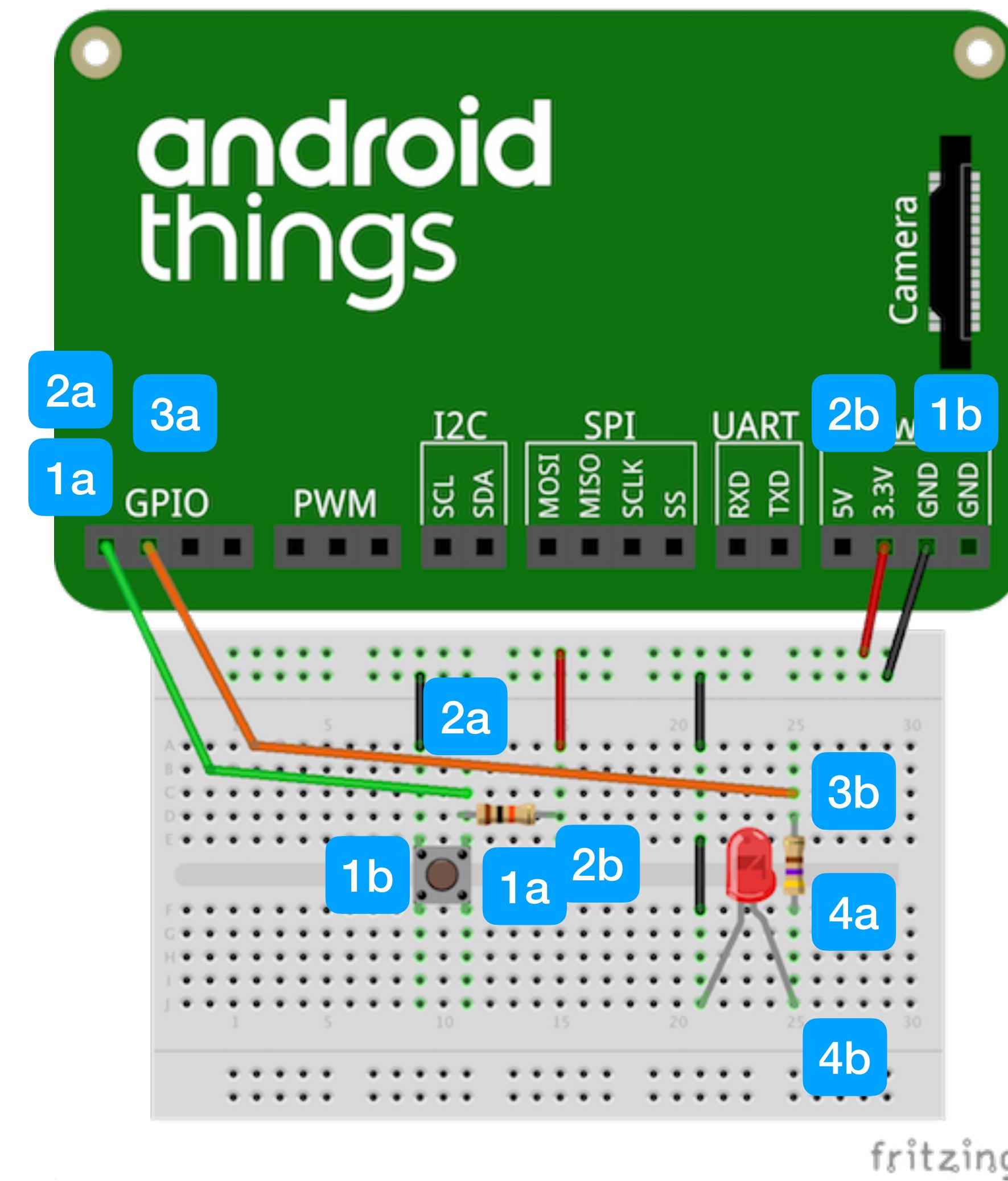
- 1 Connect one side of the button to the chosen GPIO input pin, and the other side to ground.
- 2 Connect the same GPIO input pin to +3.3V through a pull-up resistor.
- 3 Connect the chosen GPIO output pin to one side of a series resistor.
4. Connect the other side of the resistor to the anode side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



fritzing

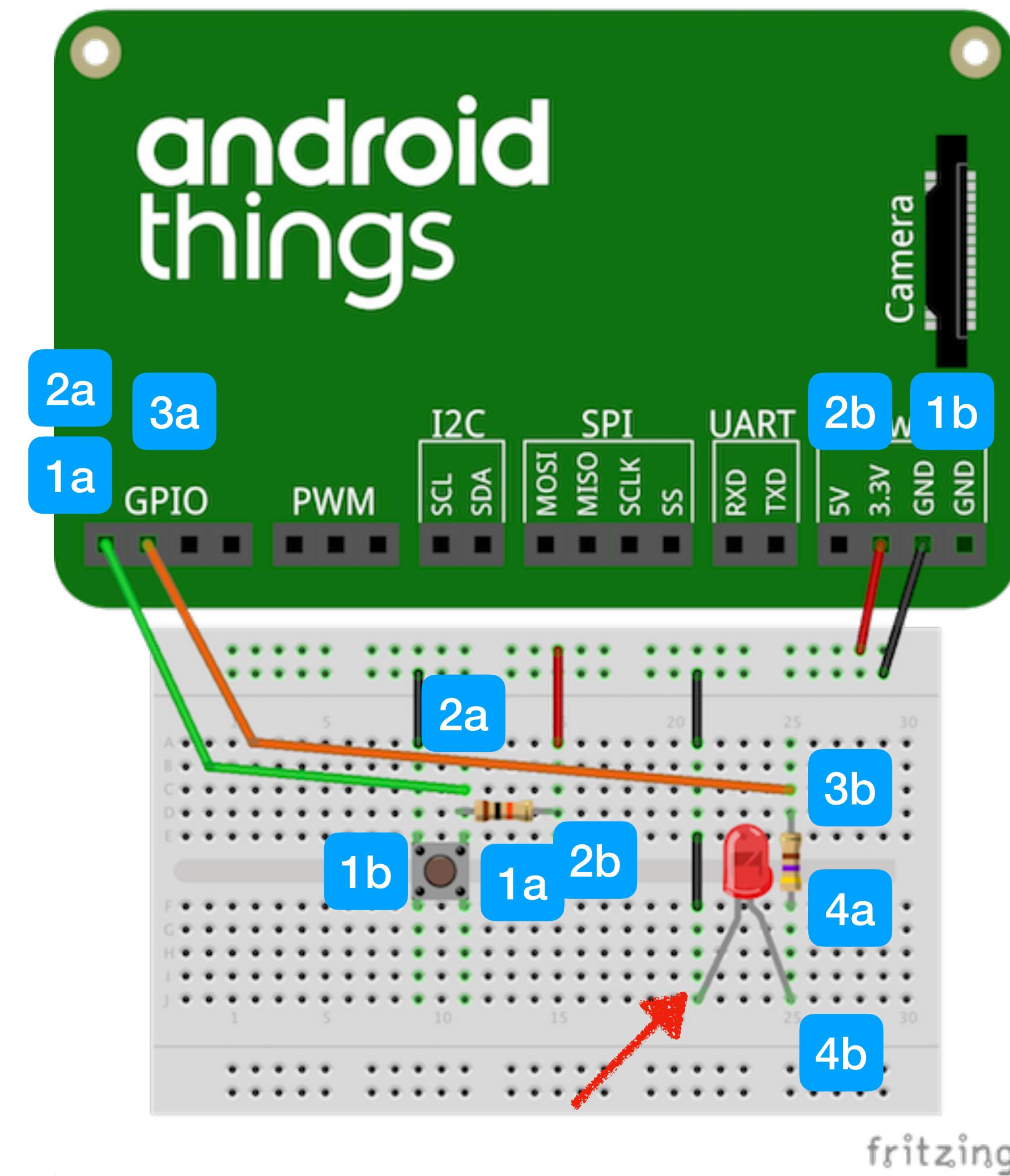
Create the Connections

- 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.



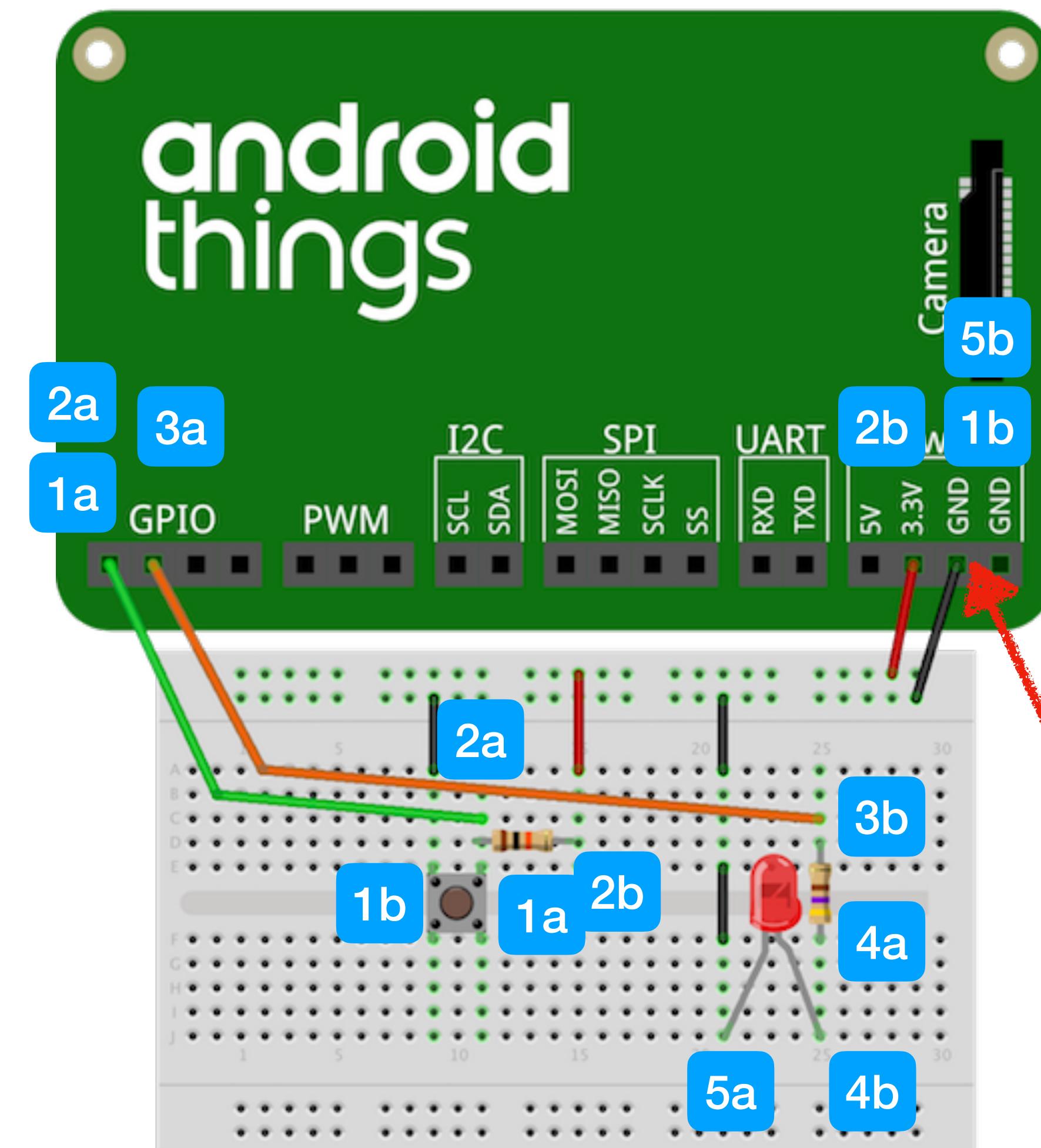
Create the Connections

- 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

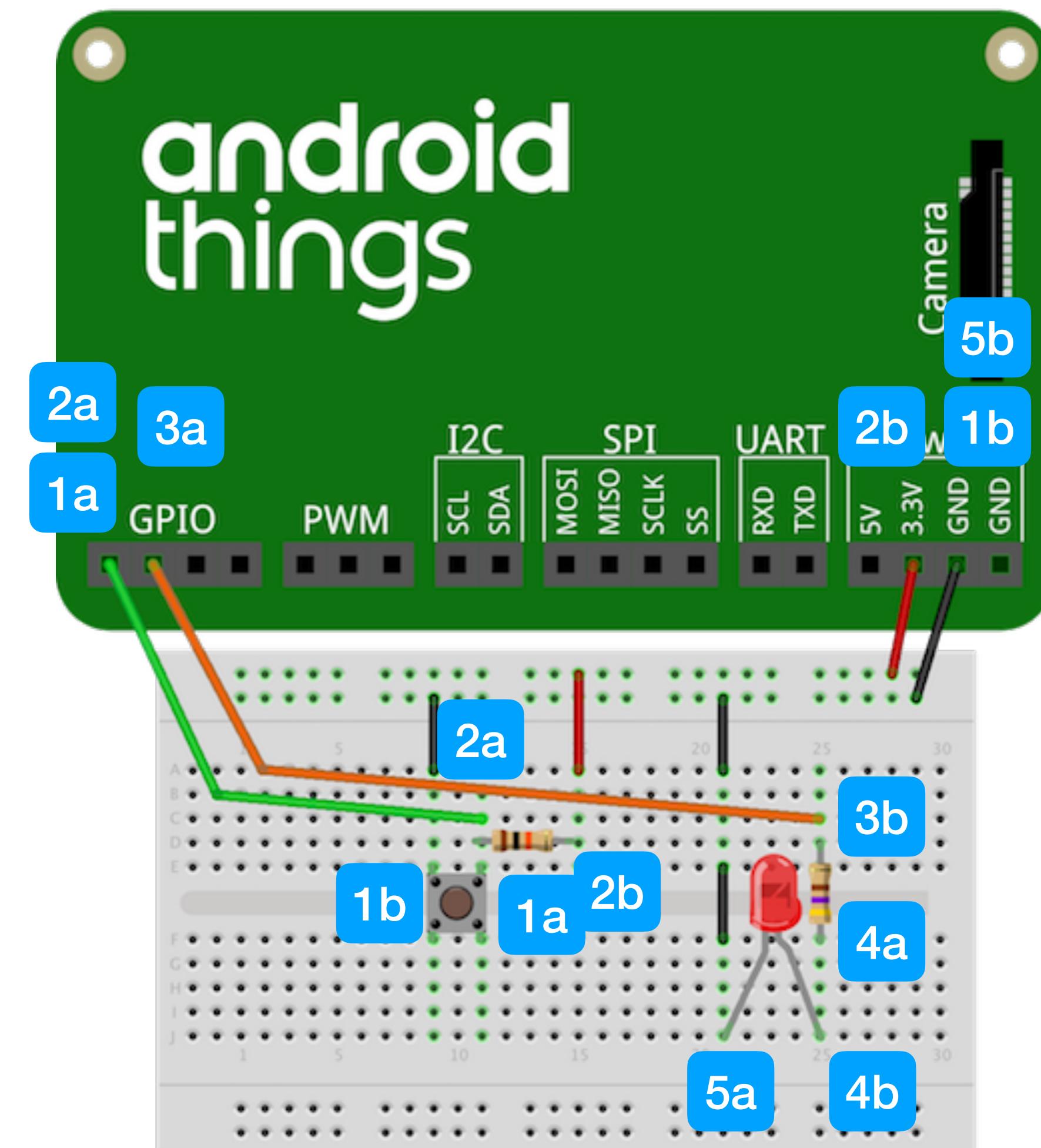
- 1 Connect one side of the button to the chosen GPIO input pin, and the other side to ground.
- 2 Connect the same GPIO input pin to +3.3V through a pull-up resistor.
- 3 Connect the chosen GPIO output pin to one side of a series resistor.
- 4 Connect the other side of the resistor to the anode side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



fritzing

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.



fritzing

NXP i.MX7D

J8				
3.3V	1	Orange	2	5V
I2C1 (SDA)	3	Cyan	4	5V
I2C1 (SCL)	5	Cyan	6	Ground
UART6 (RTS)	7	Magenta	8	UART6 (TXD)
Ground	9	Black	10	UART6 (RXD)
UART6 (CTS)	11	Magenta	12	PWM1
GPIO2_IO03	13	Green	14	Ground
GPIO1_IO10	15	Green	16	GPIO6_IO13
3.3V	17	Orange	18	GPIO6_IO12
SPI3 (MOSI)	19	Purple	20	Ground
SPI3 (MISO)	21	Purple	22	GPIO5_IO00
SPI3 (SCLK)	23	Purple	24	SPI3 (SS1)
Ground	25	Black	26	SPI3 (SS0)
I2C2 (SDA)	27	Cyan	28	I2C2 (SCL)
GPIO2_IO01	29	Green	30	Ground
GPIO2_IO02	31	Green	32	
PWM2	33	Brown	34	Ground
GPIO2_IO00	35	Green	36	GPIO2_IO07
GPIO2_IO05	37	Green	38	GPIO6_IO15
Ground	39	Black	40	GPIO6_IO14

NXP i.MX7D

J8				
3.3V	1	Orange	2	5V
I2C1 (SDA)	3	Cyan	4	5V
I2C1 (SCL)	5	Cyan	6	Ground
UART6 (RTS)	7	Magenta	8	UART6 (TXD)
Ground	9	Black	10	UART6 (RXD)
UART6 (CTS)	11	Magenta	12	PWM1
GPIO2_IO03	13	Green	14	Ground
GPIO1_IO10	15	Green	16	GPIO6_IO13
3.3V	17	Orange	18	GPIO6_IO12
SPI3 (MOSI)	19	Purple	20	Ground
SPI3 (MISO)	21	Purple	22	GPIO5_IO00
SPI3 (SCLK)	23	Purple	24	SPI3 (SS1)
Ground	25	Black	26	SPI3 (SS0)
I2C2 (SDA)	27	Cyan	28	I2C2 (SCL)
GPIO2_IO01	29	Green	30	Ground
GPIO2_IO02	31	Green	32	
PWM2	33	Brown	34	Ground
GPIO2_IO00	35	Green	36	GPIO2_IO07
GPIO2_IO05	37	Green	38	GPIO6_IO15
Ground	39	Black	40	GPIO6_IO14

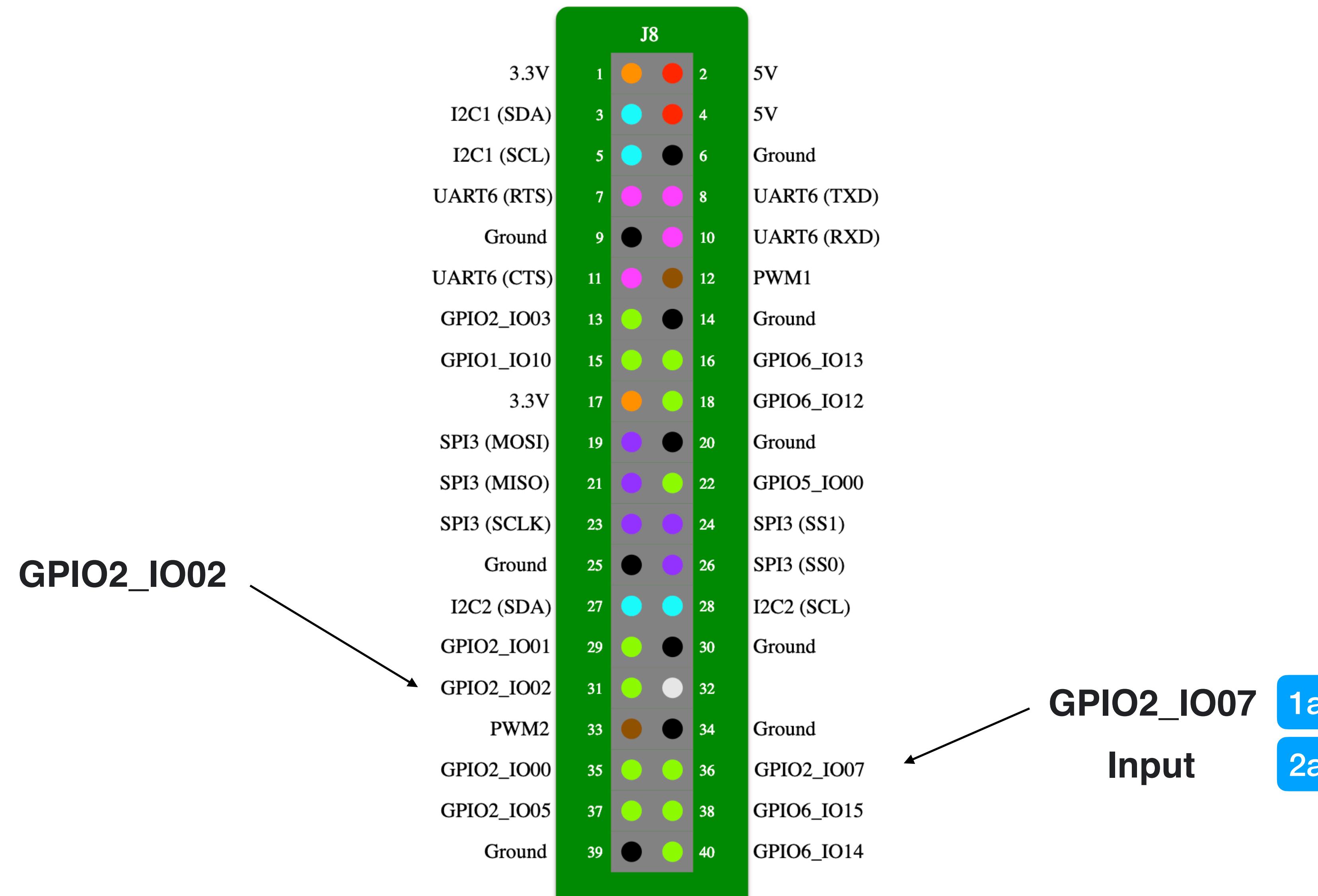
GPIO2_IO07

NXP i.MX7D

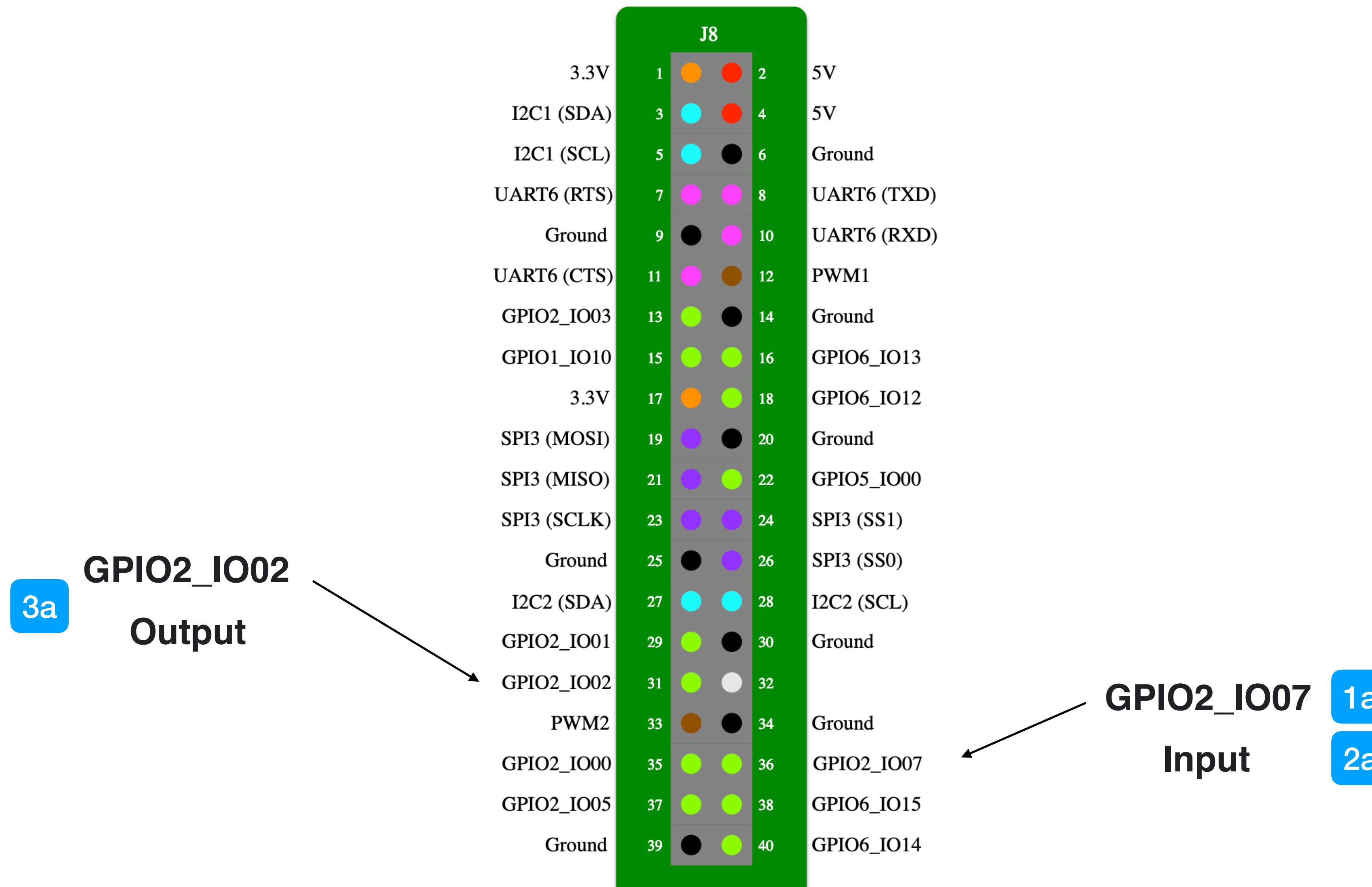
J8				
3.3V	1	Orange	2	5V
I2C1 (SDA)	3	Cyan	4	5V
I2C1 (SCL)	5	Cyan	6	Ground
UART6 (RTS)	7	Magenta	8	UART6 (TXD)
Ground	9	Black	10	UART6 (RXD)
UART6 (CTS)	11	Magenta	12	PWM1
GPIO2_IO03	13	Green	14	Ground
GPIO1_IO10	15	Green	16	GPIO6_IO13
3.3V	17	Orange	18	GPIO6_IO12
SPI3 (MOSI)	19	Purple	20	Ground
SPI3 (MISO)	21	Purple	22	GPIO5_IO00
SPI3 (SCLK)	23	Purple	24	SPI3 (SS1)
Ground	25	Black	26	SPI3 (SS0)
I2C2 (SDA)	27	Cyan	28	I2C2 (SCL)
GPIO2_IO01	29	Green	30	Ground
GPIO2_IO02	31	Green	32	
PWM2	33	Brown	34	Ground
GPIO2_IO00	35	Green	36	GPIO2_IO07
GPIO2_IO05	37	Green	38	GPIO6_IO15
Ground	39	Black	40	GPIO6_IO14

GPIO2_IO07
Input
1a
2a

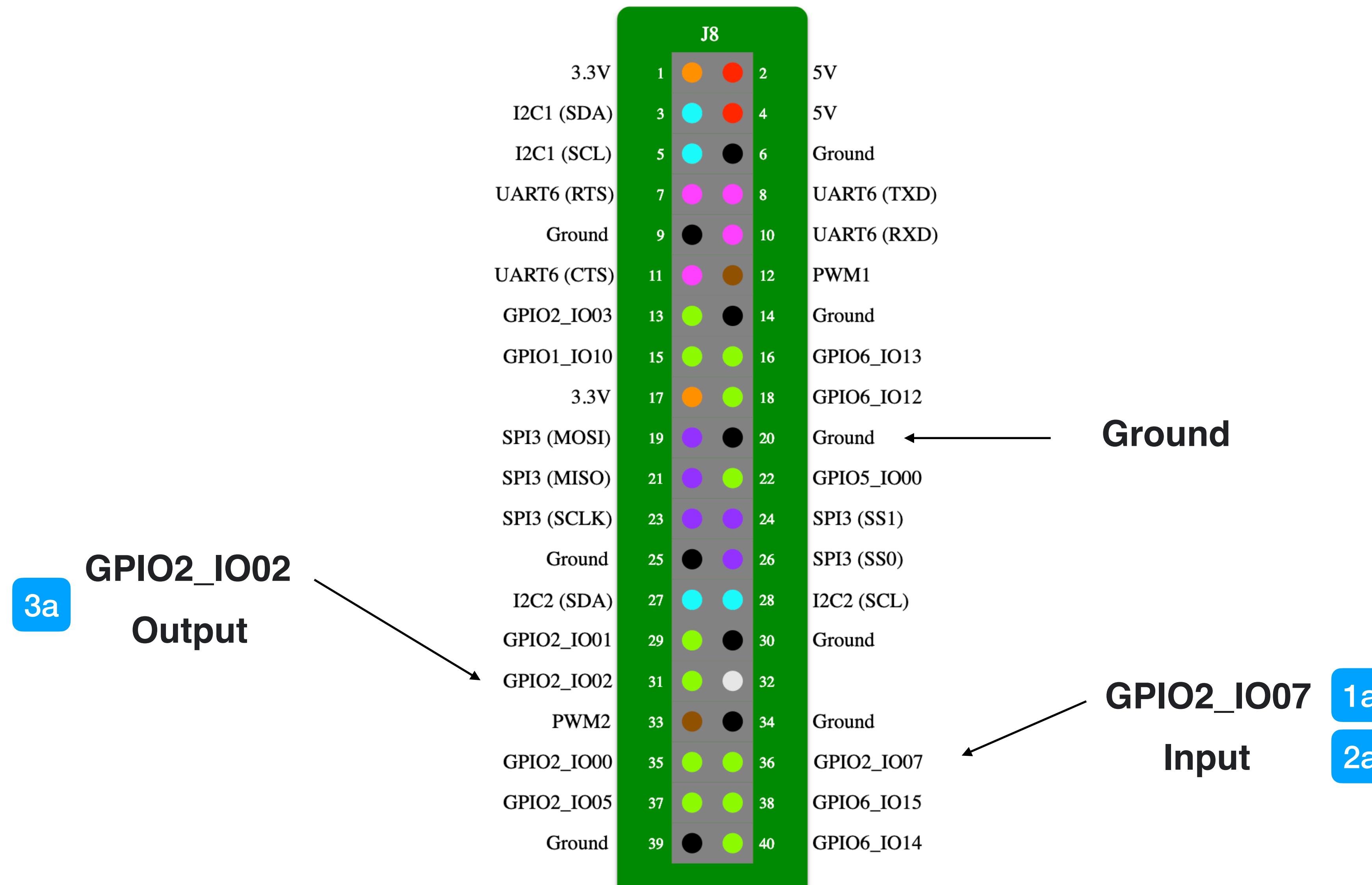
NXP i.MX7D



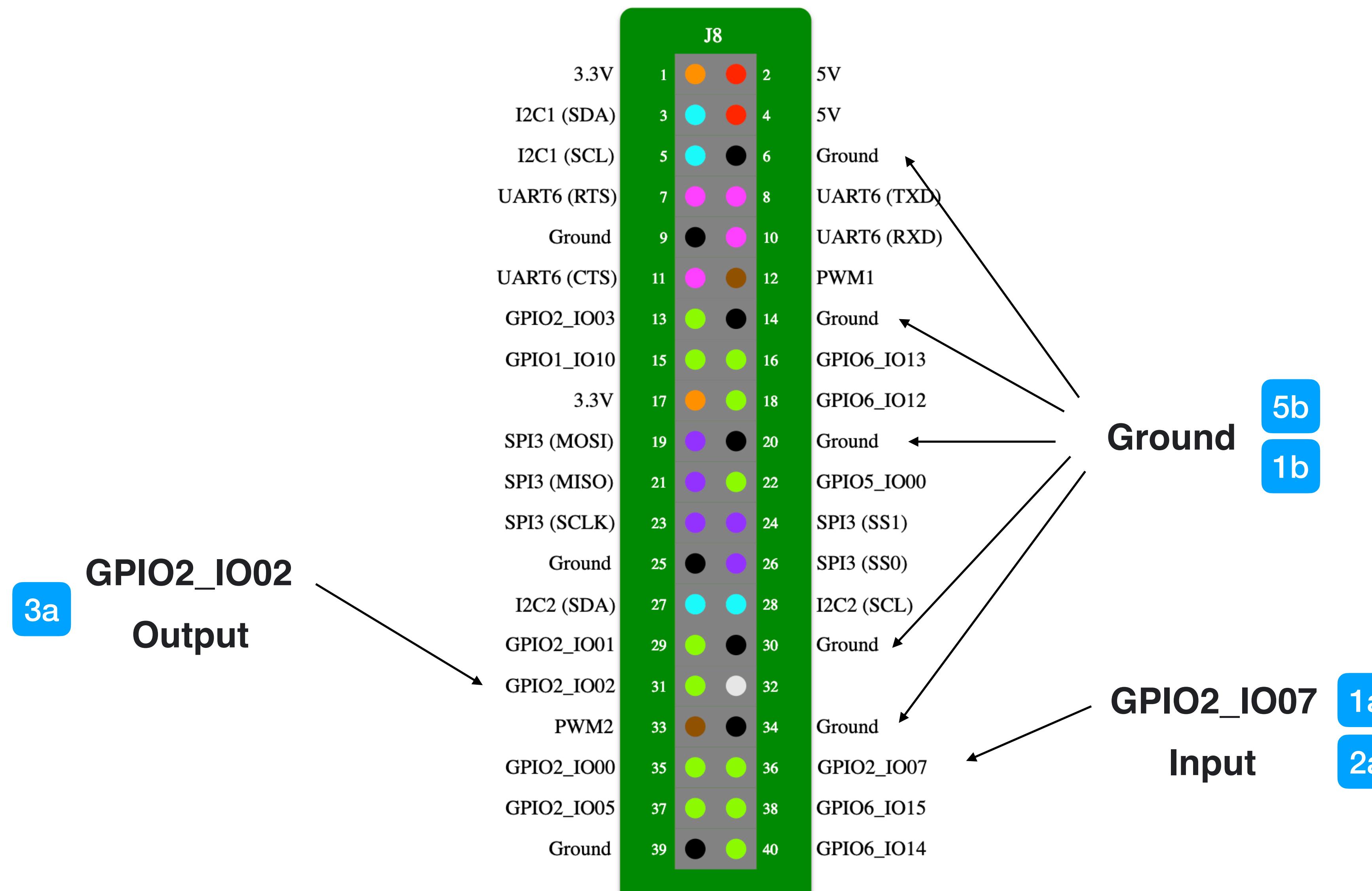
NXP i.MX7D



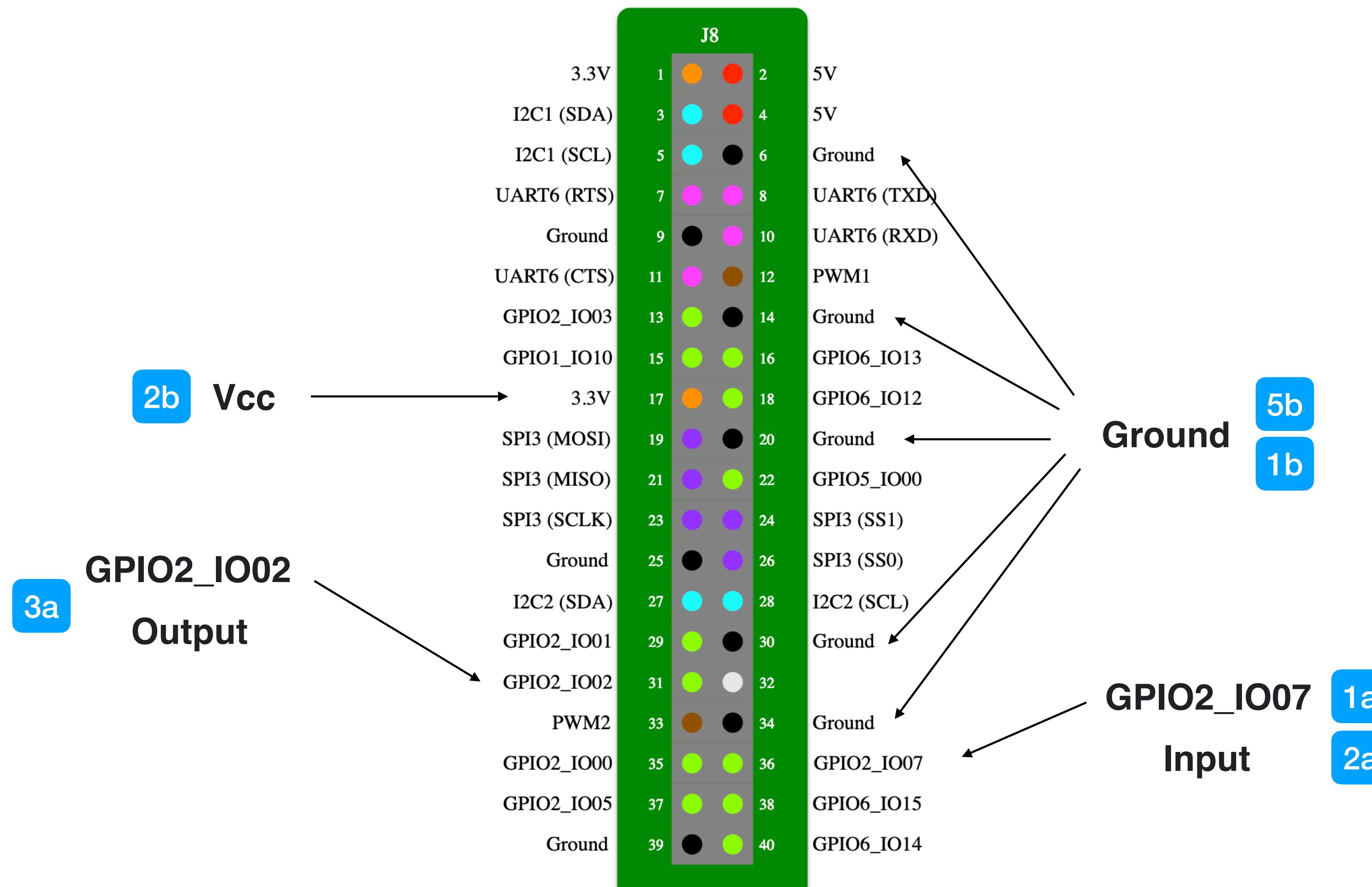
NXP i.MX7D



NXP i.MX7D



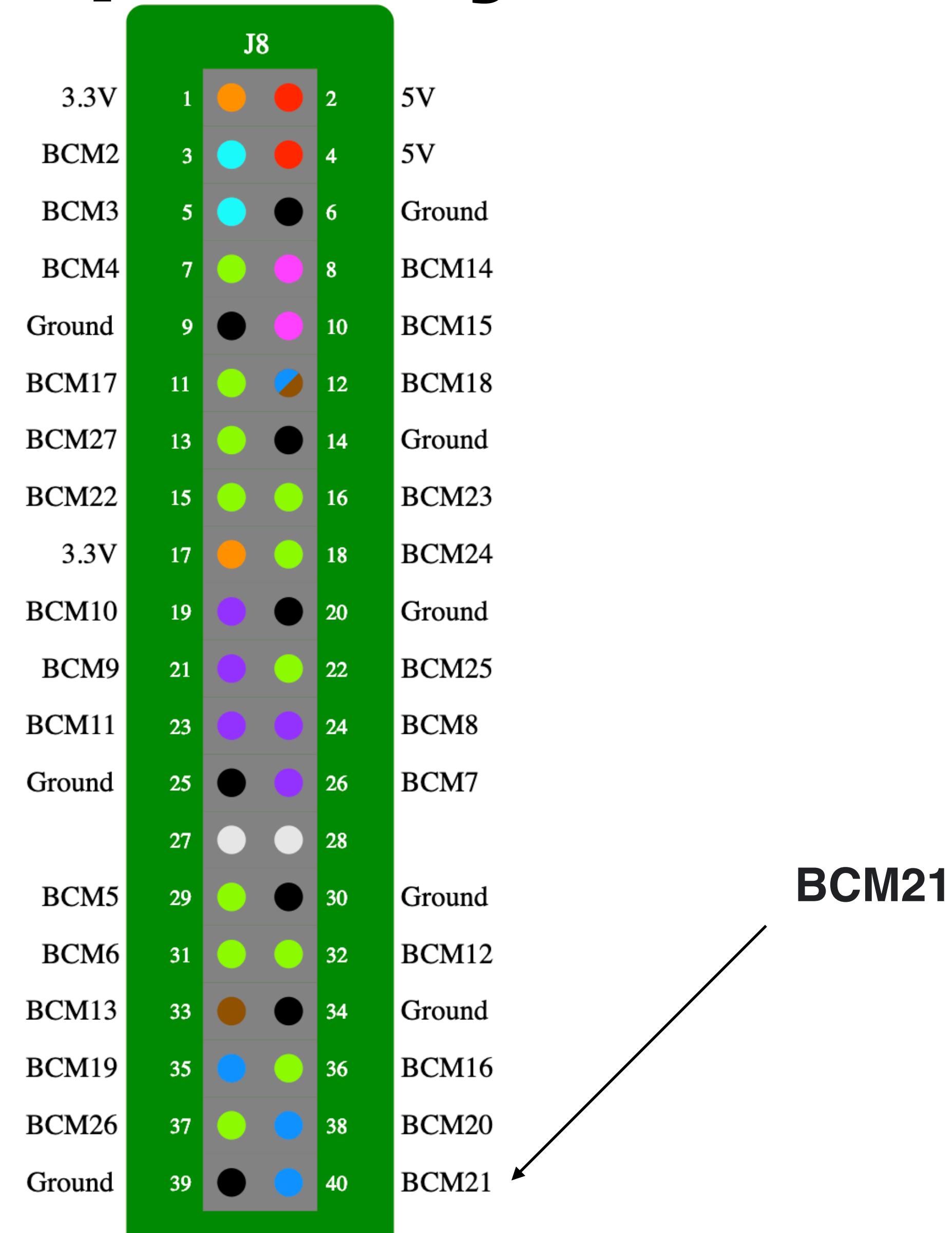
NXP i.MX7D



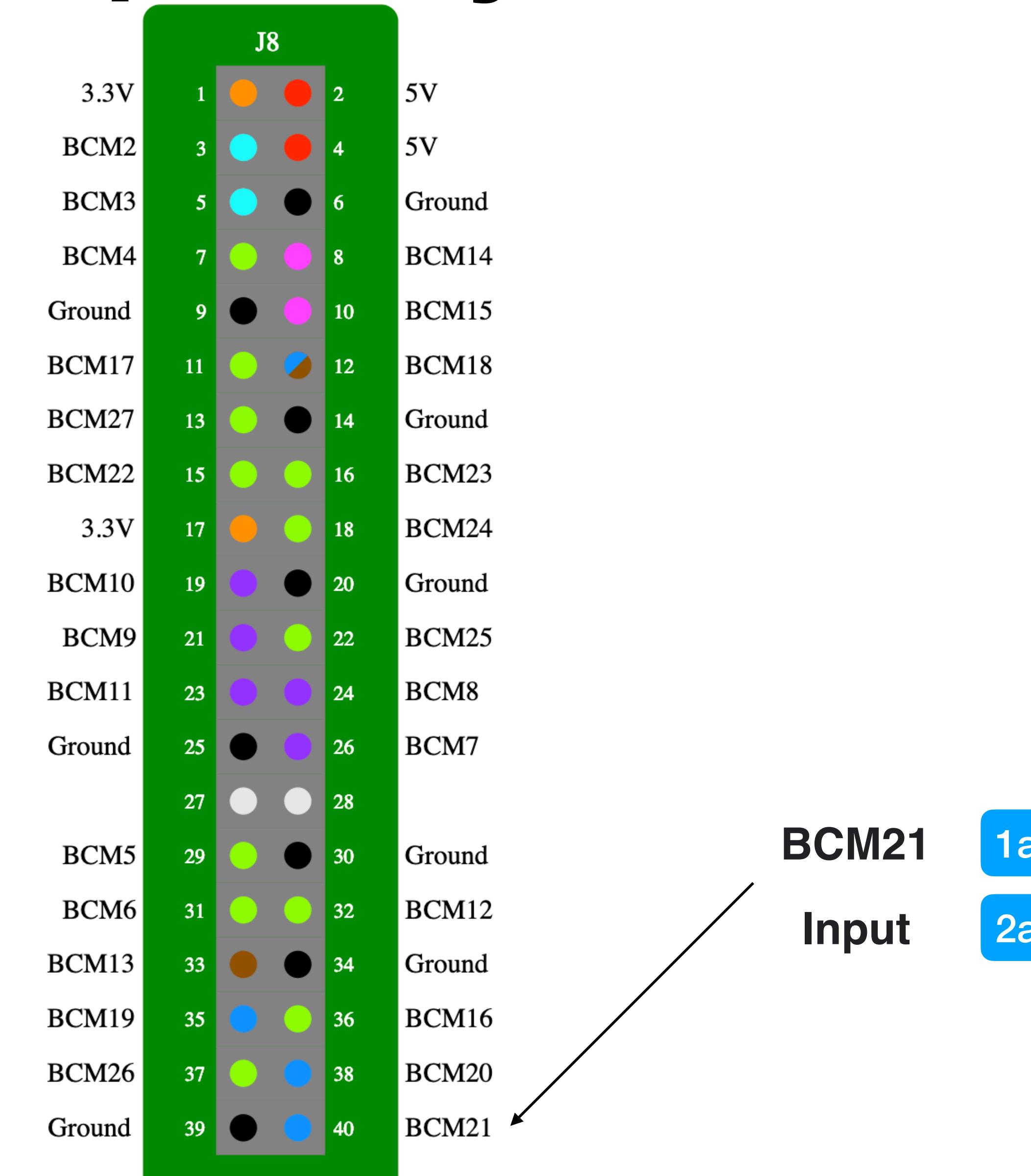
Raspberry Pi

J8			
3.3V	1	2	5V
BCM2	3	4	5V
BCM3	5	6	Ground
BCM4	7	8	BCM14
Ground	9	10	BCM15
BCM17	11	12	BCM18
BCM27	13	14	Ground
BCM22	15	16	BCM23
3.3V	17	18	BCM24
BCM10	19	20	Ground
BCM9	21	22	BCM25
BCM11	23	24	BCM8
Ground	25	26	BCM7
	27	28	
BCM5	29	30	Ground
BCM6	31	32	BCM12
BCM13	33	34	Ground
BCM19	35	36	BCM16
BCM26	37	38	BCM20
Ground	39	40	BCM21

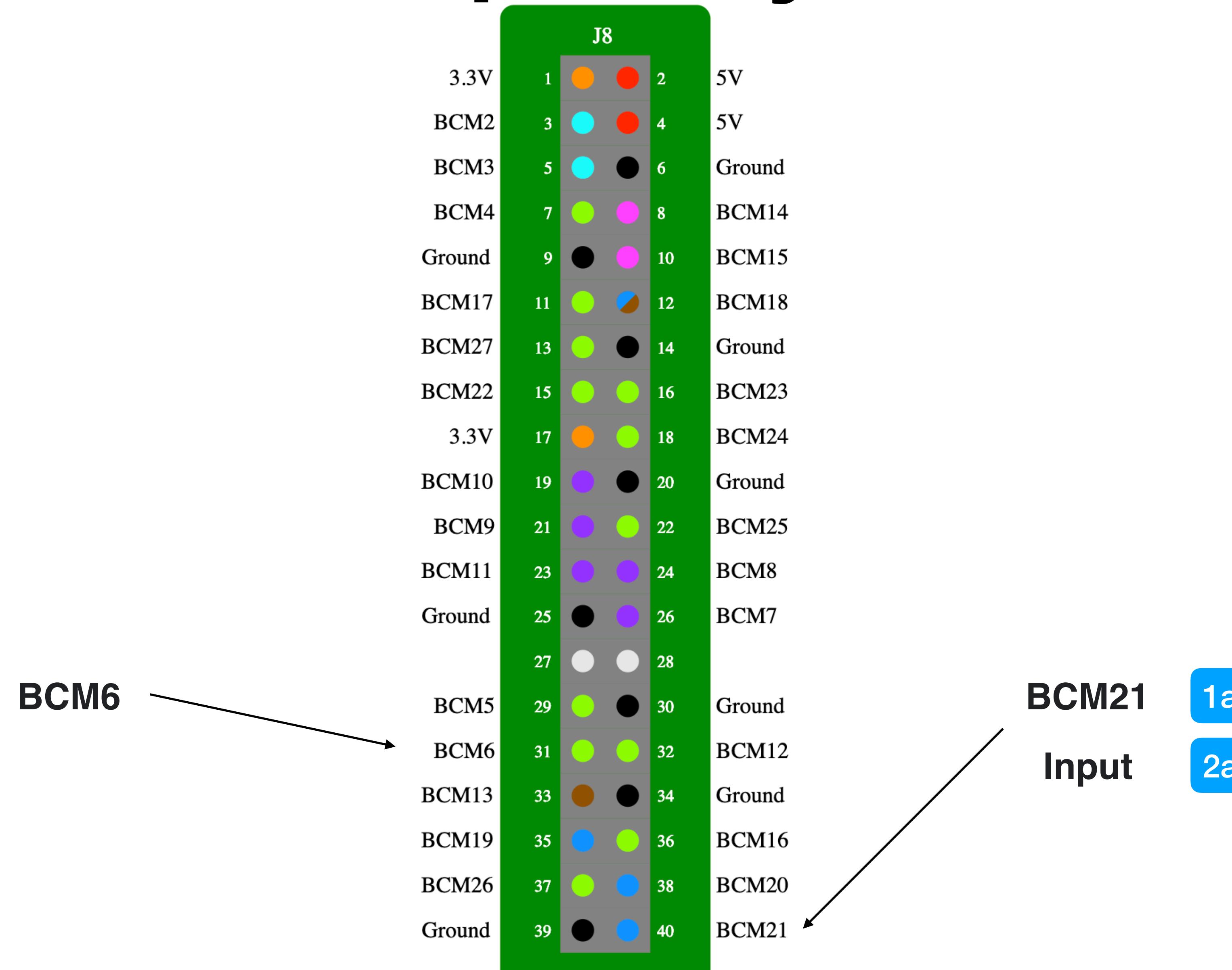
Raspberry Pi



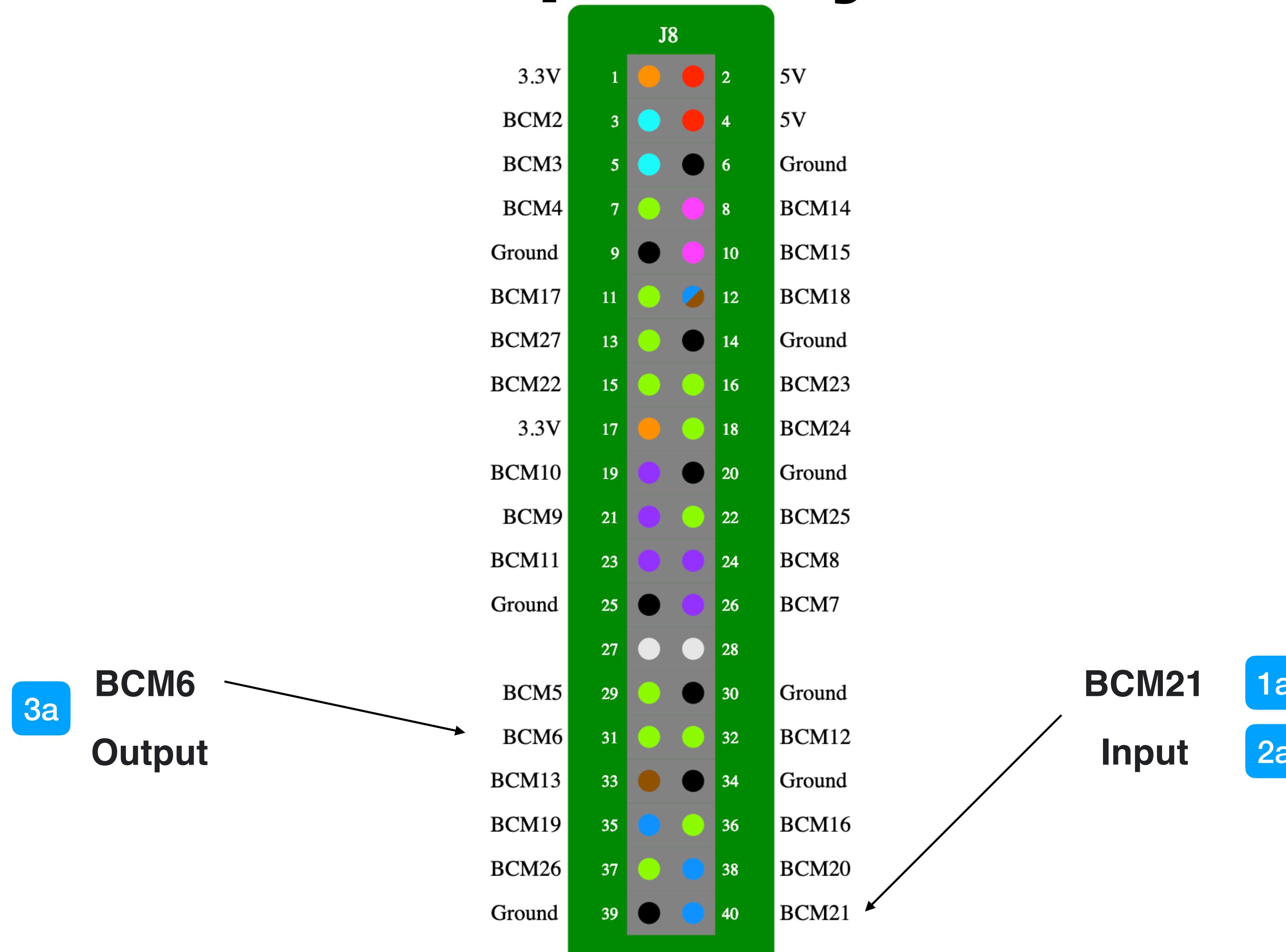
Raspberry Pi



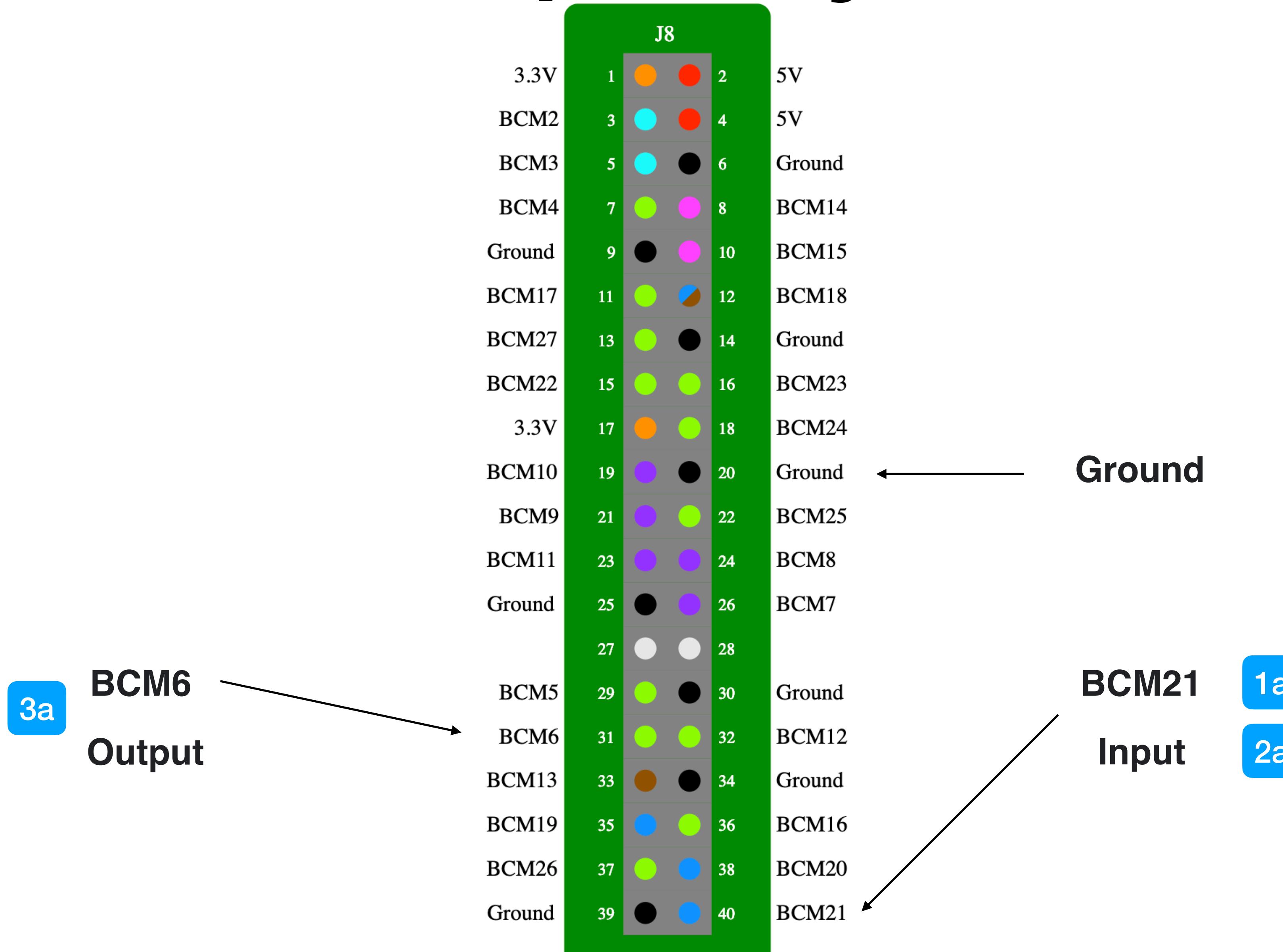
Raspberry Pi



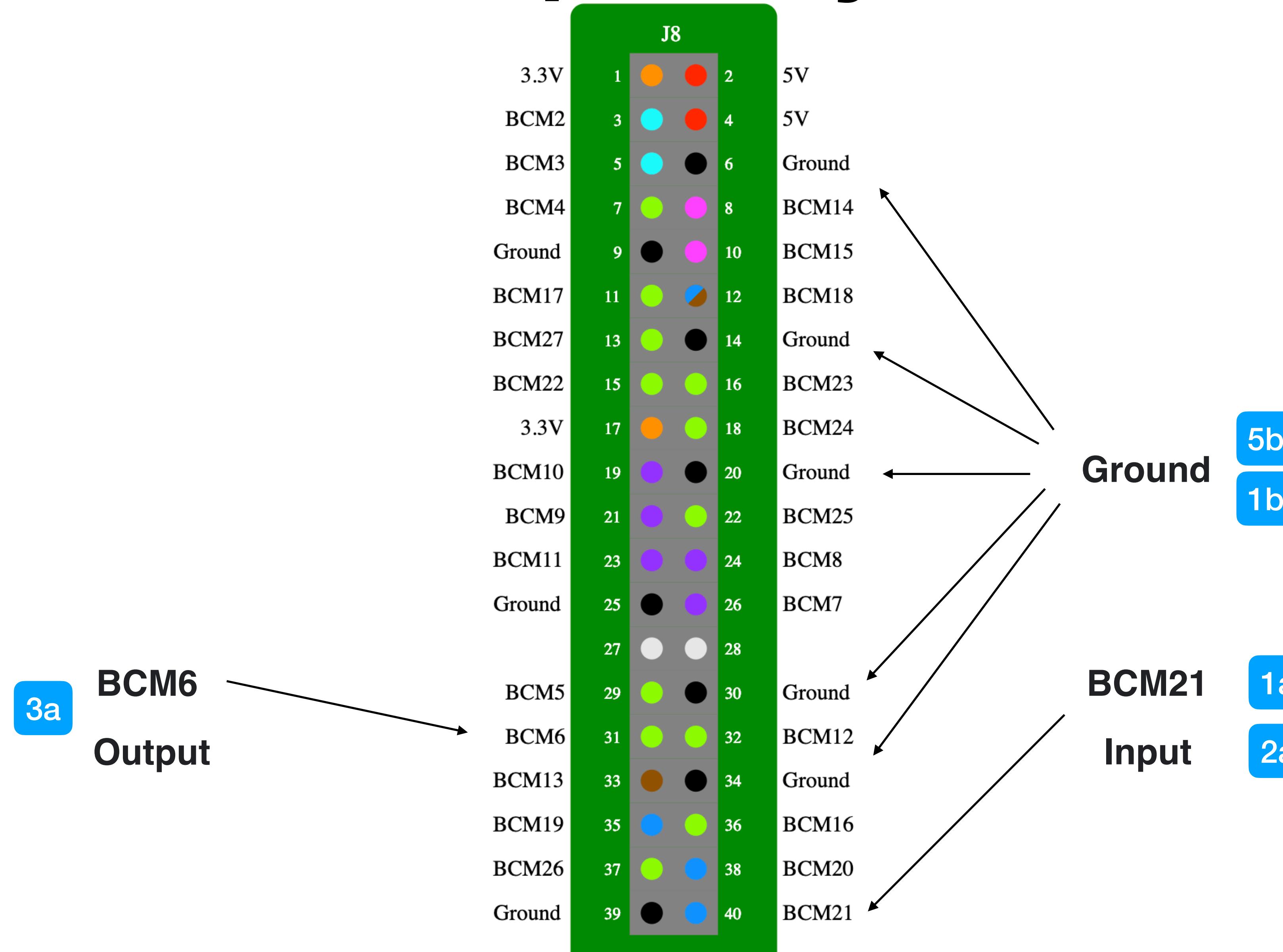
Raspberry Pi



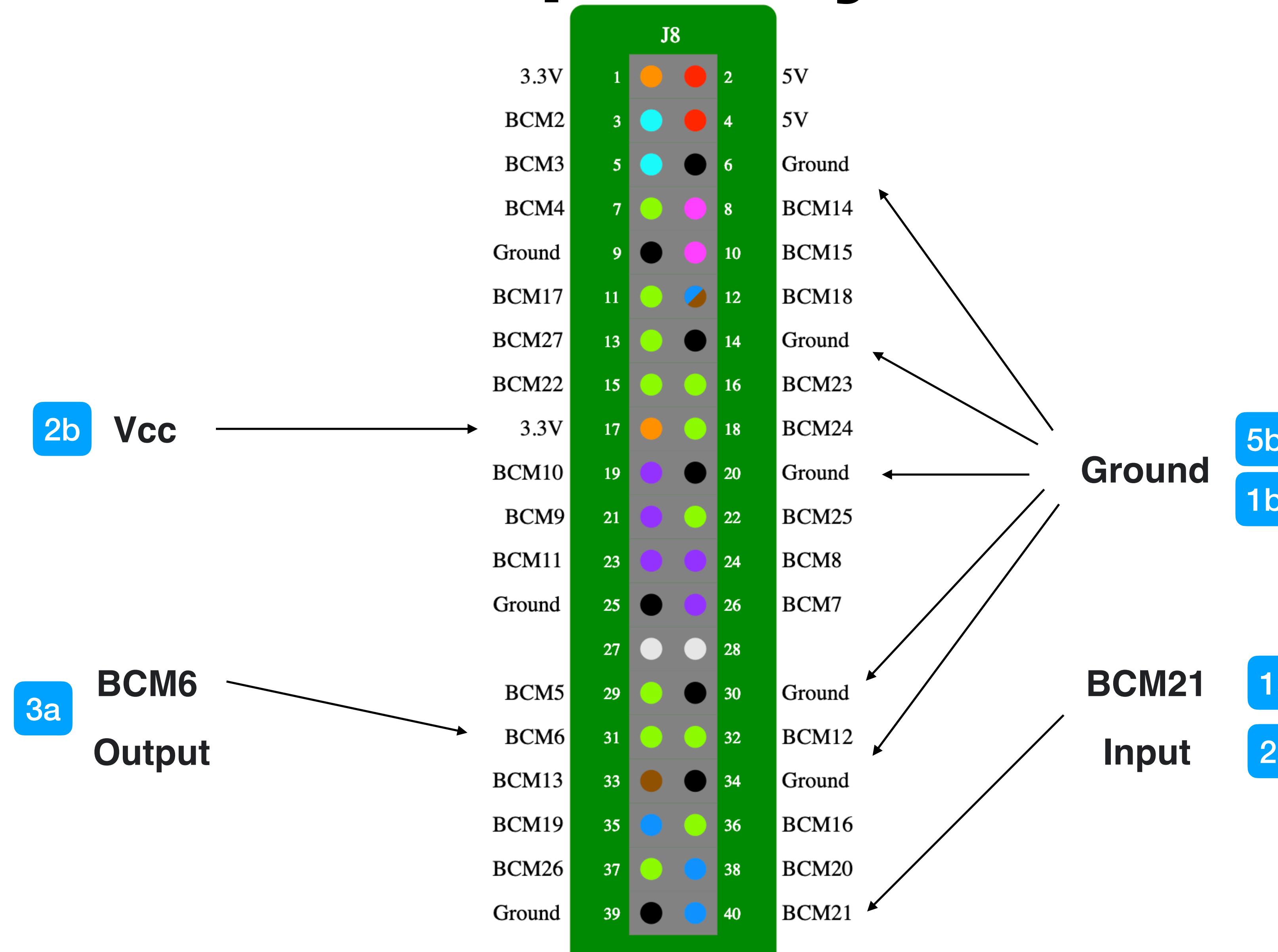
Raspberry Pi



Raspberry Pi



Raspberry Pi



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}")  
    }  
}
```

DEMO

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}")  
    }  
}
```

Handle button events

```
class ButtonActivity : Activity() {
    private companion object {
        const val BUTTON_PIN_NAME = "GPIO2_I007" // 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_I007" // 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_I007" // 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_I007" // 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_I007" // 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_I007" // 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_I002" // 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)
        }
    }
}
```

DEMO

```
// 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

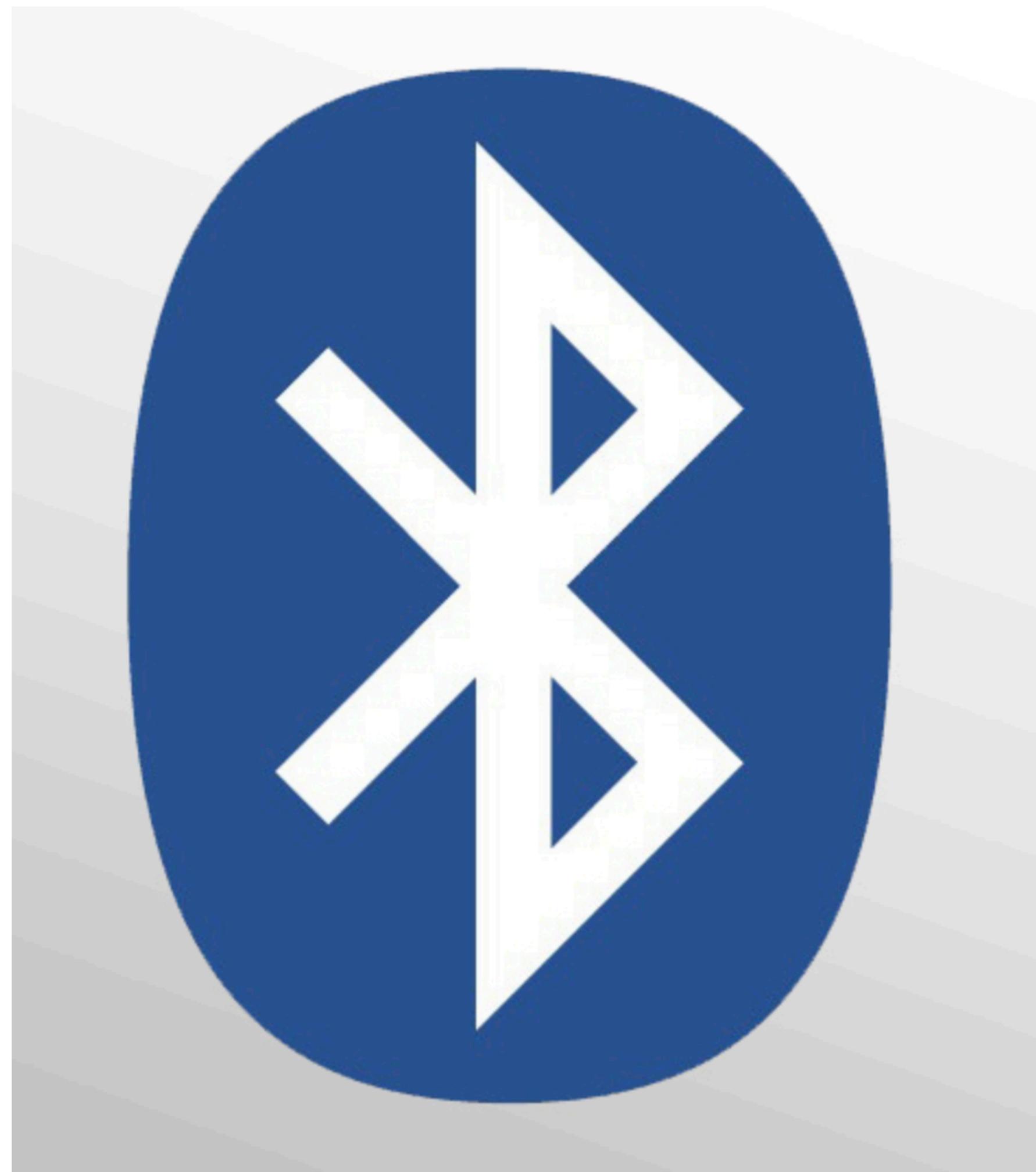


Image source: <http://blog.lenovo.com/>

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.

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_VIDEO_CONFERENCING

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_ANDLOUDSPEAKER
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_WEIGHTING
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

I/O capabilities

[Bluetooth with setIloCapability\(\)](#)

[Bluetooth Low Energy \(LE\) with setLeloCapability\(\)](#)

I/O capabilities

[Bluetooth with `setIoCapability\(\)`](#)

[Bluetooth Low Energy \(LE\) with `setLeloCapability\(\)`](#)

- **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.

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
```

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.

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)
}
```

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(blueoothPairingCallback)
        }
    }

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

    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)
        }
    }
}
```

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(blueoothDevice: BluetoothDevice) {
        bluetoothConnectionManager.connect(blueoothDevice, BluetoothProfile.A2DP_SINK)
```

```
        bluetoothConnectionCallback)
    }

    private fun connectToA2dp(blueoothDevice: BluetoothDevice) {
        blueoothConnectionManager.connect(blueoothDevice, BluetoothProfile.A2DP_SINK)
    }

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

        override fun onConnectionRequestCancelled(
            blueoothDevice: BluetoothDevice,
            requestCode: Int) {
            // Request cancelled
        }

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

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

DEMO

Connecting to a remote device

```
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)  
}
```

Lecture outcomes

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

