# TITAN Core External Haptics Trigger

This document shows how to trigger haptic effects on TITAN Core from common external inputs using the Vector Haptics library. The included examples cover a push button, a rotary encoder, and a capacitive touch input.

These examples provide a simple starting point for building interactive haptic prototypes with TITAN Core. By combining external inputs with the Vector Haptics library, you can quickly create button-driven, touch-driven, or user-adjustable haptic behaviors for your own projects.

The same structure can be extended to other sensors, switches, and control interfaces depending on your application.

# Requirements

## Software

- Arduino IDE
    - Legacy **v1.8.x** is recommended
- ESP32 board package for Arduino
- Vector Haptics library files

## Hardware

- TITAN Core
- USB cable for programming and power
- Haptic actuator connected to the configured output channel
- External input hardware depending on the example:
    - push button
    - capacitive touch pad or conductive touch surface
    - rotary encoder with push switch

# Setting up Arduino IDE

## 1. Install Arduino IDE

Install Arduino IDE and open it. (**Recommended version:** Arduino IDE v1.8.x)

## 2. Install ESP32 board support

Install the ESP32 board package by following the official Espressif setup process for Arduino.

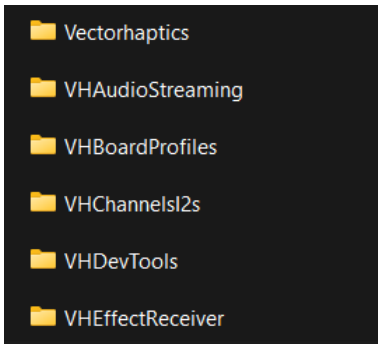
Once installed:

- Open **Tools → Board → Board Manager**
- Search for **ESP32**
- Install the ESP32 package
- Select **ESP32 Dev Module** as your active board
- Official Guide.
- When downloading the ESP32 Board manager you **MUST USE version 2.0.17**

## 3. Install the Vector Haptics libraries

Copy all Vector Haptics library folders into your local Arduino libraries folder:

...\Arduino\libraries




Then restart the Arduino IDE so the libraries are detected properly.

### 4. Confirm your setup

Before moving on, make sure:

- the board compiles successfully
- the correct COM port is selected
- TITAN Core is detected by your computer
- the required VH libraries appear under **Sketch → Include Library**

## Setting up PlatformIO

PlatformIO can also be used to build and upload TITAN Core example projects.

### 1. Install Visual Studio Code

Download and install Visual Studio Code.

### 2. Install PlatformIO IDE

In Visual Studio Code, install the PlatformIO IDE extension.

### 3. Create a new project

Create a new PlatformIO project with:

- Board: `esp32dev`
- Framework: `Arduino`

## 4. Add the Vector Haptics libraries

Copy the required Vector Haptics library folders into the project's `lib` folder.

**Example structure:**

your-project/

├── lib/

├── src/

│ └── main.cpp

├── platformio.ini

## 5. Configure `platformio.ini`

Use:

[env:esp32dev]

platform = espressif32

board = esp32dev

framework = arduino

monitor_speed = 115200

upload_speed = 921600

## 6. Add your code

Place your example code in `src/main.cpp`.

If using PlatformIO, comment out this line where shown in the examples:

```
C/C++
#include "ESP32Profiles.h"
```

## 7. Build and upload

Connect TITAN Core by USB, then build and upload the project through PlatformIO.

## 8. Confirm setup

Make sure:

- the project builds successfully
- the board uploads correctly
- the correct serial port is selected
- the Serial Monitor is set to `115200`

You can also add this note once near the first code example:

PlatformIO note: If using PlatformIO, comment out `#include "ESP32Profiles.h"` where indicated.

# General code structure

All examples in this workshop follow the same basic structure:

## Library includes

```
C/C++
#include "VectorHaptics.h"
#include "ESP32Profiles.h" // If using PlatformIO comment this line out
```

These include the Vector Haptics API and the ESP32 hardware profile used by TITAN Core.

## Object creation

```
C/C++
VectorHaptics vh;
ESP32Profile board;
VHChannels channels;
```

- vh is the main Vector Haptics controller
- board defines the ESP32 hardware profile
- channels assigns the haptic output channel and output pin

## Channel setup

```C/C++
channels.add(1, {25}, {"main"});
vh.init(&board, {&channels});
```

This creates one haptic output channel named "main" on GPIO 25.

If your hardware uses a different output pin or channel configuration, this is the section to modify.

- Channel IO25 → Left Channel
- Channel IO26 → Right Channel

# Push button example

This example plays a short haptic pulse when a push button is pressed.

## What this example shows

- reading a digital input
- basic button debouncing
- triggering a haptic effect from a button press

## Wiring notes

- One side of the button should connect to **GPIO 7**
- The other side should connect to **GND**
- INPUT_PULLUP is used, so the pin reads:
    - **HIGH** when idle
    - **LOW** when pressed

## Code

```C/C++
#include "VectorHaptics.h"
#include "ESP32Profiles.h" // If using PlatformIO comment this line out

VectorHaptics vh;
ESP32Profile board;
VHChannels channels;

const int BUTTON_PIN = 7;
long lastDebounceTime = 0;
const long debounceDelay = 50;

void setup() {
```

```
    Serial.begin(115200);

    pinMode(BUTTON_PIN, INPUT_PULLUP);

    channels.add(1, {25}, {"main"});
    vh.init(&board, {&channels});
}

void loop() {
    if (millis() - lastDebounceTime > debounceDelay) {
        if (digitalRead(BUTTON_PIN) == LOW) {
            vh.play(PULSE(20, 1.0, 0.5)); // change this to any effect
        }
        lastDebounceTime = millis();
    }
}
```

### How it works

The button pin is configured using INPUT_PULLUP, so no external pull-up resistor is needed. The code checks the button state every 50 ms to reduce repeated triggering from switch bounce. When the button is pressed, vh.play() sends a pulse effect to the haptic output.

### Effect used

PULSE(20, 1.0, 0.5)

This example uses a simple pulse effect as a starting point. You can replace it with any other supported effect once the basic trigger is working.

### Common use cases

- Button click feedback
- Confirmation feedback
- Simple event notifications

## Capacitive touch example

This example triggers a pulse when the user touches a capacitive touch input.

### What this example shows

- Using ESP32 capacitive touch sensing
- Triggering haptics without a mechanical button
- Threshold-based touch detection

### Wiring notes

- Connect a conductive touch pad to **GPIO 4**
- The touch area can be a metal pad, copper pad, or other conductive surface

- Sensitivity depends on pad size, grounding, cable length, and surrounding environment

## Code

```C/C++
#include "VectorHaptics.h"
#include "ESP32Profiles.h" // If using PlatformIO comment this line out

VectorHaptics vh;
ESP32Profile board;
VHChannels channels;

long lastDebounceTime = 0;
const int debounceDelay = 100;

const int TOUCH_PIN = 4;
const int threshold = 30;

void setup()
{
   Serial.begin(115200);

   channels.add(1, {25}, {"main"});
   vh.init(&board, {&channels});
}

void loop()
{
   if (millis() - lastDebounceTime > debounceDelay) {
       if (touchRead(TOUCH_PIN) < threshold) {
           vh.play(PULSE(20, 1.0, 0.5));
           lastDebounceTime = millis();
       }
   }
}
```

## How it works

touchRead(TOUCH_PIN) returns a capacitive measurement value. When a finger touches the pad, that value changes. If the reading falls below the selected threshold, the code triggers a pulse effect.

The debounce timer is used here to prevent repeated rapid triggering while the pad is still being touched.

## Threshold setting

const int threshold = 30;

This threshold may need to be adjusted depending on:

- touch pad size
- Wiring length
- Enclosure design
- Grounding
- Electrical noise

A good way to tune this is to print touchRead() values to the Serial Monitor for both touched and untouched states, then set a threshold with reasonable margin between them.

## Common use cases

- Touch surfaces
- Hidden or sealed UI inputs
- Non-mechanical trigger interfaces

# Rotary encoder example

This example uses a rotary encoder to change vibration intensity. Pressing the encoder button plays the effect using the current intensity value.

## What this example shows

- Reading a rotary encoder
- Increasing and decreasing haptic intensity in code
- Triggering a haptic effect with a separate button press

## Wiring notes

- Encoder channel A → **GPIO 7**
- Encoder channel B → **GPIO 8**
- Encoder push button → **GPIO 5**
- The push button should be wired to ground when using INPUT_PULLUP

## Code

```C/C++
#include "VectorHaptics.h"
#include "ESP32Profiles.h" // If using PlatformIO comment this line out

VectorHaptics vh;
ESP32Profile board;
VHChannels channels;

const int PIN_A = 7;
const int PIN_B = 8;
const int BUTTON_PIN = 5;
int intensity = 10;
```

```
int lastValueA;
int valueA;
long lastDebounceTime = 0;
const long debounceDelay = 50;

void setup()
{
   Serial.begin(115200);
   pinMode(PIN_A, INPUT_PULLUP);
   pinMode(PIN_B, INPUT_PULLUP);
   pinMode(BUTTON_PIN, INPUT_PULLUP);
   lastValueA = digitalRead(PIN_A);
   channels.add(1, {25}, {"main"});
   vh.init(&board, {&channels});
}
void loop()
{
   valueA = digitalRead(PIN_A);
   if (valueA != lastValueA)
   {
       if (digitalRead(PIN_B) != valueA)
       {
           if (intensity < 10)
               intensity++;
           Serial.println(intensity);
       }
       else
       {
           if (intensity > 0)
               intensity--;
           Serial.println(intensity);
       }
       vh.play(VIBRATE(20, float(intensity) / 10, 60, 0.5));
   }
   lastValueA = valueA;
   if (millis() - lastDebounceTime > debounceDelay) {
```

```
        if (digitalRead(BUTTON_PIN) == LOW) {
            vh.play(VIBRATE(20, float(intensity) / 10, 60, 0.5));
        }
        lastDebounceTime = millis();
    }
}
```

## How it works

The rotary encoder updates the intensity variable between 0 and 10. Turning in one direction increases intensity, while turning the other direction decreases it.

The current intensity value is also printed to the Serial Monitor, which is useful during testing and tuning.

## Effect used

VIBRATE(20, float(intensity) / 10, 60, 0.5)

In this example, the encoder value is scaled into a range from 0.0 to 1.0.

For example:

- 10 becomes 1.0
- 5 becomes 0.5
- 1 becomes 0.1

This makes the encoder a simple live haptic strength control.

## Common use cases

- Live haptic tuning
- User-adjustable vibration strength
- Demo interfaces for testing different output levels

### Note

This is a simple polling-based encoder example. For more precise or faster encoder reading, an interrupt-based approach or dedicated encoder library may be preferred.

## Customizing haptic effects

In all examples, the effect is defined by the vh.play(...) call.

Examples:

```
C/C++
vh.play(PULSE(20, 1.0, 0.5));
vh.play(VIBRATE(20, 0.8, 60, 0.5));
```

Once the trigger logic is working, you can begin changing the effect parameters to suit your application.

Common parameters to experiment with include:

- Duration
- Intensity
- Frequency
- Sharpness

The safest way to tune is to change one parameter at a time and test the result.

# Troubleshooting

## No haptic output

Check the following:

- TITAN Core is powered correctly
- the correct board is selected in Arduino IDE
- The correct COM port is selected
- The haptic actuator is connected properly
- The output pin in channels.add(...) matches your hardware
- The sketch compiled and uploaded successfully

## Button does not trigger

- Verify the button is wired to ground correctly
- Confirm the correct GPIO pin is being used
- Remember that INPUT_PULLUP means pressed = LOW
- Print the button state to Serial Monitor to confirm the input is changing

## Encoder behaves inconsistently

- Encoder pins may be floating or noisy
- Some mechanical encoders benefit from debouncing
- Direction may appear reversed depending on wiring

## Capacitive touch is too sensitive or not sensitive enough

- Adjust the threshold
- Shorten the connection to the touch pad
- Reduce nearby electrical noise
- Increase or decrease the pad size as needed