**Materials (for all activities)**

* 1× Arduino Uno (or Nano) + USB cable
* Breadboard + male–male jumper wires
* 1× **Tactile pushbutton**
* 1× **10 kΩ potentiometer** (linear)
* 1× **LED** (optional if you don’t want to use the built-in LED)
* 1× **220 Ω resistor** (only if you use an external LED)
* *(Optional but nice)* 0.1 µF capacitor across the button for extra hardware debouncing
* Laptop with **Arduino IDE** and **Arduino\_FreeRTOS** library installed:
  + **Sketch → Include Library → Manage Libraries…** → search **“Arduino\_FreeRTOS”** → Install

**Board assumptions**:

* Built-in LED on **D13** (Uno/Nano).
* External interrupt pin: **D2**.
* Analog input: **A0**.

**Wiring (summary)**

* **Potentiometer (A0)**:
  + Left pin → **5 V**
  + Right pin → **GND**
  + Middle (wiper) → **A0**
* **Button (D2 with internal pull-up)**:
  + One leg → **D2**
  + Other leg → **GND**
  + *(Optional)* 0.1 µF cap across button legs for hardware debounce
* **LED (optional external)**:
  + Arduino **D13** → **220 Ω** → **LED anode (+)**
  + LED cathode (–) → **GND**
  + (You can also just use the **built-in LED on D13**—no wiring needed.)

**Part 2 — Tasks (Blink + Print)**

**Goal**

Create **two concurrent tasks**: one blinks an LED; one prints text to Serial.

**Steps**

1. **Wire**: none needed (use built-in LED D13).
2. **Open IDE** → select the correct **Board** and **Port**.
3. **Paste** the code below → **Upload**.
4. Open **Serial Monitor** @ **9600 baud**; observe text while LED blinks.

**Code (full sketch)**

#include <Arduino\_FreeRTOS.h>

// Task 1: Blink LED every 500 ms

void TaskBlink(void \*pvParameters) {

pinMode(13, OUTPUT);

while (1) {

digitalWrite(13, HIGH);

vTaskDelay(500 / portTICK\_PERIOD\_MS);

digitalWrite(13, LOW);

vTaskDelay(500 / portTICK\_PERIOD\_MS);

}

}

// Task 2: Print to Serial every 1000 ms

void TaskPrint(void \*pvParameters) {

while (1) {

Serial.println("Hello from FreeRTOS Task!");

vTaskDelay(1000 / portTICK\_PERIOD\_MS);

}

}

void setup() {

Serial.begin(9600);

xTaskCreate(TaskBlink, "Blink", 128, NULL, 1, NULL);

xTaskCreate(TaskPrint, "Print", 128, NULL, 1, NULL);

}

void loop() {}

**Expected**: LED on D13 blinks; “Hello from FreeRTOS Task!” prints every second.

**Part 3 — Delays (vTaskDelayUntil for precise timing)**

**Goal**

Show **drift-free periodic** behavior using vTaskDelayUntil.

**Steps**

1. Use the same wiring as Part 2.
2. **Upload** the sketch below.
3. Observe a steady, precise 2 Hz blink (every 500 ms).

**Code (full sketch)**

#include <Arduino\_FreeRTOS.h>

void TaskBlinkPrecise(void \*pvParameters) {

pinMode(13, OUTPUT);

const TickType\_t period = 500 / portTICK\_PERIOD\_MS;

TickType\_t lastWake = xTaskGetTickCount();

while (1) {

digitalWrite(13, !digitalRead(13));

vTaskDelayUntil(&lastWake, period); // exact cadence

}

}

void setup() {

xTaskCreate(TaskBlinkPrecise, "BlinkPrecise", 128, NULL, 1, NULL);

}

void loop() {}

**Expected**: Very stable blink with no timing drift over time.

**Part 4 — Queues (Producer–Consumer)**

**Goal**

Use a **queue** for safe inter-task communication: one task reads the **potentiometer** and sends data; another receives and prints.

**Extra Wiring**

* **Pot** as described (5 V–A0–GND).

**Steps**

1. Wire the **pot** to A0.
2. **Upload** the sketch.
3. Open **Serial Monitor** @ **9600** and turn the pot; watch values change.

**Code (full sketch)**

#include <Arduino\_FreeRTOS.h>

#include <queue.h>

QueueHandle\_t sensorQueue;

void TaskSensor(void \*pvParameters) {

int value;

while (1) {

value = analogRead(A0); // 0..1023

xQueueSend(sensorQueue, &value, portMAX\_DELAY); // enqueue

vTaskDelay(200 / portTICK\_PERIOD\_MS); // 5 Hz

}

}

void TaskDisplay(void \*pvParameters) {

int received;

while (1) {

if (xQueueReceive(sensorQueue, &received, portMAX\_DELAY)) {

Serial.print("Potentiometer: ");

Serial.println(received);

}

}

}

void setup() {

Serial.begin(9600);

sensorQueue = xQueueCreate(5, sizeof(int));

xTaskCreate(TaskSensor, "Sensor", 128, NULL, 1, NULL);

xTaskCreate(TaskDisplay, "Display", 128, NULL, 1, NULL);

}

void loop() {}

**Expected**: Smooth numeric updates as you rotate the knob.

**Part 5 — Interrupts (Button ISR → Queue → Task)**

**Goal**

On **button press**, an **ISR** sends a message to a **queue**; a task receives it and **toggles** the LED.

**Extra Wiring**

* **Button** between **D2** and **GND** (we’ll use INPUT\_PULLUP).

**Steps**

1. Wire the **button**: D2 ↔ button ↔ GND.
2. **Upload** the sketch.
3. Press the button; LED toggles on each press.

**Code (full sketch)**

#include <Arduino\_FreeRTOS.h>

#include <queue.h>

// Make IRAM\_ATTR a no-op on non-ESP32 to keep code portable

#ifndef ARDUINO\_ARCH\_ESP32

  #define IRAM\_ATTR

#endif

// Cross-port "yield from ISR" helper:

// - AVR (Arduino\_FreeRTOS): call taskYIELD() only if a higher-priority task was woken

// - Others (e.g., ESP32): use the standard macro with the argument

#if defined(ARDUINO\_ARCH\_AVR)

  #define YIELD\_FROM\_ISR\_IF\_NEEDED(hptw) do { if ((hptw) == pdTRUE) { taskYIELD(); } } while (0)

#else

  #define YIELD\_FROM\_ISR\_IF\_NEEDED(hptw) portYIELD\_FROM\_ISR(hptw)

#endif

QueueHandle\_t buttonQueue;

void IRAM\_ATTR buttonISR() {

  const int signal = 1;

  BaseType\_t xHigherPriorityTaskWoken = pdFALSE;

  // Send from ISR; request a context switch if this unblocks a higher-priority task

  xQueueSendFromISR(buttonQueue, &signal, &xHigherPriorityTaskWoken);

  YIELD\_FROM\_ISR\_IF\_NEEDED(xHigherPriorityTaskWoken);

}

void TaskLED(void \*pvParameters) {

  pinMode(13, OUTPUT);

  int msg;

  for (;;) {

    if (xQueueReceive(buttonQueue, &msg, portMAX\_DELAY) == pdTRUE) {

      digitalWrite(13, !digitalRead(13));  // toggle LED on each button press

    }

  }

}

void setup() {

  pinMode(2, INPUT\_PULLUP);  // use internal pull-up; button to GND

  buttonQueue = xQueueCreate(5, sizeof(int));

  // Attach ISR on falling edge (button press)

  attachInterrupt(digitalPinToInterrupt(2), buttonISR, FALLING);

  // Create LED task (increase stack if you add more logic)

  xTaskCreate(TaskLED, "LED", 128, NULL, 2, NULL);

}

void loop() {

  // Empty: FreeRTOS takes over after setup()

}

**Expected**: Each press toggles LED state; no Serial needed.

**Part 6 — Mini-Project (Tasks + Delays + Queues + Interrupts)**

**Goal**

Combine everything:

* **TaskLED** blinks at 1 Hz; **button press** toggles it to 2 Hz and back.
* **TaskSensor** samples **A0** every 200 ms and sends to queue.
* **TaskDisplay** receives sensor values and prints them.

**Wiring (full)**

* **Potentiometer** to **A0** (5 V / GND / wiper→A0).
* **Button** to **D2** (D2↔button↔GND).
* **LED**: use **built-in D13** *(or external LED + 220 Ω to GND)*.

**Step-by-Step**

1. Build the circuit for **pot** and **button**.
2. Paste the **full sketch** below.
3. Upload → Open **Serial Monitor** @ 9600.
4. Watch potentiometer values streaming.
5. Press the button to toggle blink rate between **1 s** and **0.5 s**.

**Code (full sketch)**

#include <Arduino\_FreeRTOS.h>

#include <queue.h>

// Keep code portable: IRAM\_ATTR for ESP32; no-op on AVR

#ifndef ARDUINO\_ARCH\_ESP32

#define IRAM\_ATTR

#endif

// Cross-port "yield from ISR":

#if defined(ARDUINO\_ARCH\_AVR)

// Arduino\_FreeRTOS AVR: macro takes no args — call taskYIELD() if needed

#define YIELD\_FROM\_ISR\_IF\_NEEDED(hptw) do { if ((hptw) == pdTRUE) { taskYIELD(); } } while (0)

#else

// ESP32/others: standard macro with argument

#define YIELD\_FROM\_ISR\_IF\_NEEDED(hptw) portYIELD\_FROM\_ISR(hptw)

#endif

QueueHandle\_t sensorQueue;

QueueHandle\_t buttonQueue;

volatile int ledDelayMs = 1000; // default 1 Hz

// ---------- ISR: Button ----------

void IRAM\_ATTR buttonISR() {

const int signal = 1;

BaseType\_t xHigherPriorityTaskWoken = pdFALSE;

xQueueSendFromISR(buttonQueue, &signal, &xHigherPriorityTaskWoken);

// Request context switch if a higher-priority task was unblocked

YIELD\_FROM\_ISR\_IF\_NEEDED(xHigherPriorityTaskWoken);

}

// ---------- Task: LED Blinker ----------

void TaskLED(void \*pvParameters) {

pinMode(13, OUTPUT);

int msg;

for (;;) {

digitalWrite(13, HIGH);

vTaskDelay(ledDelayMs / portTICK\_PERIOD\_MS);

digitalWrite(13, LOW);

vTaskDelay(ledDelayMs / portTICK\_PERIOD\_MS);

// Non-blocking check for button message

if (xQueueReceive(buttonQueue, &msg, 0) == pdTRUE) {

ledDelayMs = (ledDelayMs == 1000) ? 500 : 1000; // toggle 1s <-> 0.5s

}

}

}

// ---------- Task: Sensor Producer ----------

void TaskSensor(void \*pvParameters) {

int value;

for (;;) {

value = analogRead(A0); // 0..1023

xQueueSend(sensorQueue, &value, portMAX\_DELAY); // enqueue

vTaskDelay(200 / portTICK\_PERIOD\_MS); // 5 Hz sample

}

}

// ---------- Task: Serial Consumer ----------

void TaskDisplay(void \*pvParameters) {

int received;

for (;;) {

if (xQueueReceive(sensorQueue, &received, portMAX\_DELAY) == pdTRUE) {

Serial.print("Potentiometer: ");

Serial.println(received);

}

}

}

void setup() {

Serial.begin(9600);

// Queues

sensorQueue = xQueueCreate(8, sizeof(int));

buttonQueue = xQueueCreate(4, sizeof(int));

// Button (to GND) with internal pull-up

pinMode(2, INPUT\_PULLUP);

attachInterrupt(digitalPinToInterrupt(2), buttonISR, FALLING);

// Tasks (note: AVR stack size is in words for Arduino\_FreeRTOS)

xTaskCreate(TaskLED, "LED", 128, NULL, 2, NULL);

xTaskCreate(TaskSensor, "Sensor", 128, NULL, 1, NULL);

xTaskCreate(TaskDisplay, "Display", 160, NULL, 1, NULL); // extra for Serial

}

void loop() {}

**Expected**:

* LED blinks; button press speeds it up (and toggles back).
* Serial Monitor shows live potentiometer values.
* All actions run smoothly in parallel.

**Troubleshooting & Notes**

* **Library not found**: install **Arduino\_FreeRTOS** from Library Manager.
* **Out of memory (Uno has 2 KB RAM)**:
  + Close Serial prints or reduce task stacks (e.g., 128 → 110).
  + Don’t create unnecessary global arrays.
* **Bouncy button**: add a **0.1 µF** cap across button legs or debounce in the LED task (ignore new button signals for 50–100 ms after one is received).
* **Serial gibberish**: set **9600 baud** in Serial Monitor.
* **External LED not lighting**: check **polarity** and the **220 Ω** resistor.

**ESP32 Users (optional path)**

* ESP32 already includes FreeRTOS; you **don’t** need Arduino\_FreeRTOS.h.
* Replace includes with:
* #include <Arduino.h>
* #include "freertos/FreeRTOS.h"
* #include "freertos/task.h"
* #include "freertos/queue.h"
* Keep IRAM\_ATTR as-is (it’s valid on ESP32).
* Use 3.3 V logic, and wire the **pot** to **3.3 V** (not 5 V).
* ESP32 pins that support interrupts: most GPIOs (avoid strapping pins).
* Update LED pin to a valid GPIO (e.g., **2** with onboard LED on many dev boards).