this is the code for getting real

```
/*
 * SPDX-FileCopyrightText: 2022-2023 Espressif Systems (Shanghai) CO LTD
 *
 * SPDX-License-Identifier: Apache-2.0
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "soc/soc_caps.h"
#include "esp_log.h"
#include "esp_adc/adc_oneshot.h"
#include "esp_adc/adc_cali.h"
#include "esp_adc/adc_cali_scheme.h"

// IMPORTANT!!!!!!!!!!! YOU HAVE TO USE GPIO32 for this to work
const static char *TAG = "TEMPERATURE_SENSOR";

/*---------------------------------------------------------------
        ADC General Macros
---------------------------------------------------------------*/
//ADC1 Channels
#if CONFIG_IDF_TARGET_ESP32
#define EXAMPLE_ADC1_CHAN0          ADC_CHANNEL_4
#define EXAMPLE_ADC1_CHAN1          ADC_CHANNEL_5
#else
#define EXAMPLE_ADC1_CHAN0          ADC_CHANNEL_2
#define EXAMPLE_ADC1_CHAN1          ADC_CHANNEL_3
#endif

#if (SOC_ADC_PERIPH_NUM >= 2) && !CONFIG_IDF_TARGET_ESP32C3
/**
 * On ESP32C3, ADC2 is no longer supported, due to its HW limitation.
 * Search for errata on espressif website for more details.
 */
#define EXAMPLE_USE_ADC2            1
#endif

#if EXAMPLE_USE_ADC2
//ADC2 Channels
#if CONFIG_IDF_TARGET_ESP32
#define EXAMPLE_ADC2_CHAN0          ADC_CHANNEL_0
#else
#define EXAMPLE_ADC2_CHAN0          ADC_CHANNEL_0
#endif
#endif  //#if EXAMPLE_USE_ADC2

#define EXAMPLE_ADC_ATTEN           ADC_ATTEN_DB_12
```

```
 // LMT86 Temperature Sensor Constants
 // From LMT86 datasheet Section 8.3.1
 #define LMT86_SLOPE_MV_PER_C        -10.9f  // mV/°C (negative slope)
 #define LMT86_INTERCEPT_MV          2103.0f // mV at 0°C
 #define ADC_BITWIDTH_12             4096    // 2^12 for 12-bit ADC
 #define ADC_VREF_MV                 3300    // 3.3V reference in mV
(typical for 12dB attenuation)

 // Expected voltage ranges for LMT86:
 // At 25°C: ~1830 mV
 // At 0°C:  ~2103 mV
 // At 50°C: ~1830 mV

 static int adc_raw[2][10];
 static int voltage[2][10];
 static bool example_adc_calibration_init(adc_unit_t unit, adc_channel_t
channel, adc_atten_t atten, adc_cali_handle_t *out_handle);
 static void example_adc_calibration_deinit(adc_cali_handle_t handle);

 /**
  * Perform a single ADC sample
  *
  * @param adc_handle The ADC handle to use
  * @param channel The ADC channel to sample
  * @param result Pointer to store the raw result
  * @param cali_handle Calibration handle (NULL if no calibration)
  * @param voltage Pointer to store calibrated voltage (NULL if not needed)
  *
  * @return ESP_OK on success, error code otherwise
  */
 static esp_err_t sample(adc_oneshot_unit_handle_t adc_handle,
     adc_channel_t channel,
     int *result,
     adc_cali_handle_t cali_handle,
     int *voltage)
 {
     esp_err_t ret;

     // Read the raw value
     ret = adc_oneshot_read(adc_handle, channel, result);
     if (ret != ESP_OK) {
         return ret;
     }

     // Convert to voltage if calibration handle is provided and voltage
pointer is not NULL
     if (cali_handle != NULL && voltage != NULL) {
         ret = adc_cali_raw_to_voltage(cali_handle, *result, voltage);
     }

     return ret;
 }

 /**
```

```
  * Convert raw ADC value to temperature in Celsius
  *
  * This function implements the LMT86 temperature conversion formula:
  * - Raw ADC -> Voltage: voltage_mV = (raw_value * VREF_mV) /
ADC_MAX_VALUE
  * - Voltage -> Temperature: temp_C = (voltage_mV - INTERCEPT_mV) /
SLOPE_mV_per_C
  *
  * Combined formula for maximum precision:
  * temp_C = ((raw_value * VREF_mV / ADC_MAX_VALUE) - INTERCEPT_mV) /
SLOPE_mV_per_C
  *
  * @param raw_adc_value Raw ADC reading (0 to 4095 for 12-bit)
  * @return Temperature in degrees Celsius as int8_t
  */
 int8_t convert(int raw_adc_value)
 {
     // Convert raw ADC to voltage in mV
     // Using float for intermediate calculation to maintain precision
     float voltage_mv = ((float)raw_adc_value * ADC_VREF_MV) /
ADC_BITWIDTH_12;

     // Convert voltage to temperature using LMT86 formula
     // T(°C) = (V_out - V_0°C) / Slope
     // Where: V_0°C = 2103mV, Slope = -10.9mV/°C
     float temperature_c = (voltage_mv - LMT86_INTERCEPT_MV) /
LMT86_SLOPE_MV_PER_C;

     // Round to nearest integer and cast to int8_t
     int8_t result = (int8_t)(temperature_c >= 0 ? temperature_c + 0.5f :
temperature_c - 0.5f);

     // Debug output with diagnostic information
     ESP_LOGI(TAG, "ADC Raw: %d, Voltage: %.1f mV, Temperature: %.1f°C ->
%d°C",
              raw_adc_value, voltage_mv, temperature_c, result);

     // Diagnostic warnings
     if (voltage_mv < 1000) {
         ESP_LOGW(TAG, "⚠ Voltage too low (%.1f mV) - Check sensor
connections!", voltage_mv);
         ESP_LOGW(TAG, "Expected ~1830mV at 25°C, ~2103mV at 0°C");
     }
     if (temperature_c < -40 || temperature_c > 125) {
         ESP_LOGW(TAG, "⚠ Temperature out of expected range (%.1f°C)",
temperature_c);
     }

     return result;
 }

 /**
  * Alternative convert function using calibrated voltage input
  * Use this version if you have calibrated voltage values from the ADC
```

```
calibration
  *
  * @param voltage_mv Calibrated voltage in millivolts
  * @return Temperature in degrees Celsius as int8_t
  */
 int8_t convert_from_voltage(int voltage_mv)
 {
     // Convert voltage to temperature using LMT86 formula
     float temperature_c = ((float)voltage_mv - LMT86_INTERCEPT_MV) /
LMT86_SLOPE_MV_PER_C;

     // Round to nearest integer and cast to int8_t
     int8_t result = (int8_t)(temperature_c >= 0 ? temperature_c + 0.5f :
temperature_c - 0.5f);

     ESP_LOGI(TAG, "Voltage: %d mV, Temperature: %.1f°C -> %d°C",
              voltage_mv, temperature_c, result);

     // Diagnostic warnings
     if (voltage_mv < 1000) {
         ESP_LOGW(TAG, "⚠ Calibrated voltage too low (%d mV) - Check
sensor connections!", voltage_mv);
     }

     return result;
 }

 void app_main(void)
 {
     //-------------ADC1 Init---------------//
     adc_oneshot_unit_handle_t adc1_handle;
     adc_oneshot_unit_init_cfg_t init_config1 = {
         .unit_id = ADC_UNIT_1,
     };
     ESP_ERROR_CHECK(adc_oneshot_new_unit(&init_config1, &adc1_handle));

     //-------------ADC1 Config---------------//
     adc_oneshot_chan_cfg_t config = {
         .atten = EXAMPLE_ADC_ATTEN,
         .bitwidth = ADC_BITWIDTH_DEFAULT,
     };
     ESP_ERROR_CHECK(adc_oneshot_config_channel(adc1_handle,
EXAMPLE_ADC1_CHAN0, &config));
     ESP_ERROR_CHECK(adc_oneshot_config_channel(adc1_handle,
EXAMPLE_ADC1_CHAN1, &config));

     //-------------ADC1 Calibration Init---------------//
     adc_cali_handle_t adc1_cali_chan0_handle = NULL;
     adc_cali_handle_t adc1_cali_chan1_handle = NULL;
     bool do_calibration1_chan0 = example_adc_calibration_init(ADC_UNIT_1,
EXAMPLE_ADC1_CHAN0, EXAMPLE_ADC_ATTEN, &adc1_cali_chan0_handle);
     bool do_calibration1_chan1 = example_adc_calibration_init(ADC_UNIT_1,
EXAMPLE_ADC1_CHAN1, EXAMPLE_ADC_ATTEN, &adc1_cali_chan1_handle);
```

```
 #if EXAMPLE_USE_ADC2
     //-------------ADC2 Init---------------//
     adc_oneshot_unit_handle_t adc2_handle;
     adc_oneshot_unit_init_cfg_t init_config2 = {
         .unit_id = ADC_UNIT_2,
         .ulp_mode = ADC_ULP_MODE_DISABLE,
     };
     ESP_ERROR_CHECK(adc_oneshot_new_unit(&init_config2, &adc2_handle));

     //-------------ADC2 Calibration Init---------------//
     adc_cali_handle_t adc2_cali_handle = NULL;
     bool do_calibration2 = example_adc_calibration_init(ADC_UNIT_2,
EXAMPLE_ADC2_CHAN0, EXAMPLE_ADC_ATTEN, &adc2_cali_handle);

     //-------------ADC2 Config---------------//
     ESP_ERROR_CHECK(adc_oneshot_config_channel(adc2_handle,
EXAMPLE_ADC2_CHAN0, &config));
 #endif  //#if EXAMPLE_USE_ADC2

     ESP_LOGI(TAG, "Starting temperature measurement...");
     ESP_LOGI(TAG, "Touch the LMT86 sensor to see temperature changes!");

     while (1) {
         // Sample ADC1 Channel 0 (Temperature Sensor)
         ESP_ERROR_CHECK(sample(adc1_handle, EXAMPLE_ADC1_CHAN0,
&adc_raw[0][0],
                                do_calibration1_chan0 ?
adc1_cali_chan0_handle : NULL,
                                &voltage[0][0]));

         // Convert raw ADC to temperature
         int8_t temperature_raw = convert(adc_raw[0][0]);

         ESP_LOGI(TAG, "=== TEMPERATURE SENSOR ===");
         ESP_LOGI(TAG, "ADC%d Channel[%d] Raw Data: %d", ADC_UNIT_1 + 1,
EXAMPLE_ADC1_CHAN0, adc_raw[0][0]);

         if (do_calibration1_chan0) {
             ESP_LOGI(TAG, "ADC%d Channel[%d] Cali Voltage: %d mV",
ADC_UNIT_1 + 1, EXAMPLE_ADC1_CHAN0, voltage[0][0]);
             // Also convert using calibrated voltage for comparison
             int8_t temperature_cal = convert_from_voltage(voltage[0][0]);
             ESP_LOGI(TAG, "Temperature (calibrated): %d°C",
temperature_cal);
         }

         ESP_LOGI(TAG, "Temperature (raw conversion): %d°C",
temperature_raw);
         ESP_LOGI(TAG, "========================");

         vTaskDelay(pdMS_TO_TICKS(2000)); // 2 second delay for readability

         // Optional: Sample other channels for comparison
         /*
```

```
        // Sample ADC1 Channel 1
        ESP_ERROR_CHECK(sample(adc1_handle, EXAMPLE_ADC1_CHAN1,
&adc_raw[0][1],
                               do_calibration1_chan1 ?
adc1_cali_chan1_handle : NULL,
                               &voltage[0][1]));
        ESP_LOGI(TAG, "ADC%d Channel[%d] Raw Data: %d", ADC_UNIT_1 + 1,
EXAMPLE_ADC1_CHAN1, adc_raw[0][1]);
        if (do_calibration1_chan1) {
            ESP_LOGI(TAG, "ADC%d Channel[%d] Cali Voltage: %d mV",
ADC_UNIT_1 + 1, EXAMPLE_ADC1_CHAN1, voltage[0][1]);
        }
        vTaskDelay(pdMS_TO_TICKS(1000));
        */

 #if EXAMPLE_USE_ADC2
        // Sample ADC2 Channel 0
        ESP_ERROR_CHECK(sample(adc2_handle, EXAMPLE_ADC2_CHAN0,
&adc_raw[1][0],
                               do_calibration2 ? adc2_cali_handle : NULL,
                               &voltage[1][0]));
        ESP_LOGI(TAG, "ADC%d Channel[%d] Raw Data: %d", ADC_UNIT_2 + 1,
EXAMPLE_ADC2_CHAN0, adc_raw[1][0]);
        if (do_calibration2) {
            ESP_LOGI(TAG, "ADC%d Channel[%d] Cali Voltage: %d mV",
ADC_UNIT_2 + 1, EXAMPLE_ADC2_CHAN0, voltage[1][0]);
        }
        vTaskDelay(pdMS_TO_TICKS(1000));
 #endif  //#if EXAMPLE_USE_ADC2
    }

    //Tear Down
    ESP_ERROR_CHECK(adc_oneshot_del_unit(adc1_handle));
    if (do_calibration1_chan0) {
        example_adc_calibration_deinit(adc1_cali_chan0_handle);
    }
    if (do_calibration1_chan1) {
        example_adc_calibration_deinit(adc1_cali_chan1_handle);
    }

 #if EXAMPLE_USE_ADC2
    ESP_ERROR_CHECK(adc_oneshot_del_unit(adc2_handle));
    if (do_calibration2) {
        example_adc_calibration_deinit(adc2_cali_handle);
    }
 #endif //#if EXAMPLE_USE_ADC2
 }

 /*---------------------------------------------------------------
        ADC Calibration
 ---------------------------------------------------------------*/
 static bool example_adc_calibration_init(adc_unit_t unit, adc_channel_t
channel, adc_atten_t atten, adc_cali_handle_t *out_handle)
 {
    adc_cali_handle_t handle = NULL;
    esp_err_t ret = ESP_FAIL;
    bool calibrated = false;

#if ADC_CALI_SCHEME_CURVE_FITTING_SUPPORTED
    if (!calibrated) {
        ESP_LOGI(TAG, "calibration scheme version is %s", "Curve Fitting");
        adc_cali_curve_fitting_config_t cali_config = {
            .unit_id = unit,
            .chan = channel,
            .atten = atten,
            .bitwidth = ADC_BITWIDTH_DEFAULT,
        };
        ret = adc_cali_create_scheme_curve_fitting(&cali_config, &handle);
        if (ret == ESP_OK) {
            calibrated = true;
        }
    }
#endif

#if ADC_CALI_SCHEME_LINE_FITTING_SUPPORTED
    if (!calibrated) {
        ESP_LOGI(TAG, "calibration scheme version is %s", "Line Fitting");
        adc_cali_line_fitting_config_t cali_config = {
            .unit_id = unit,
            .atten = atten,
            .bitwidth = ADC_BITWIDTH_DEFAULT,
        };
        ret = adc_cali_create_scheme_line_fitting(&cali_config, &handle);
        if (ret == ESP_OK) {
            calibrated = true;
        }
    }
#endif

    *out_handle = handle;
    if (ret == ESP_OK) {
        ESP_LOGI(TAG, "Calibration Success");
    } else if (ret == ESP_ERR_NOT_SUPPORTED || !calibrated) {
        ESP_LOGW(TAG, "eFuse not burnt, skip software calibration");
    } else {
        ESP_LOGE(TAG, "Invalid arg or no memory");
    }

    return calibrated;
}

static void example_adc_calibration_deinit(adc_cali_handle_t handle)
{
#if ADC_CALI_SCHEME_CURVE_FITTING_SUPPORTED
    ESP_LOGI(TAG, "deregister %s calibration scheme", "Curve Fitting");
    ESP_ERROR_CHECK(adc_cali_delete_scheme_curve_fitting(handle));
```

```
#elif ADC_CALI_SCHEME_LINE_FITTING_SUPPORTED
    ESP_LOGI(TAG, "deregister %s calibration scheme", "Line Fitting");
    ESP_ERROR_CHECK(adc_cali_delete_scheme_line_fitting(handle));
#endif
}
```

This ESP-IDF code is designed to read temperature from an **LMT86 analog temperature sensor** connected to an **ESP32 microcontroller**. It leverages the ESP32's **Analog-to-Digital Converter (ADC)** to get voltage readings from the sensor and then converts those voltage values into actual temperature in Celsius using a specific formula provided by the sensor's datasheet. It also incorporates **ADC calibration** for improved accuracy.

Core Functionality

The program continuously:

1. **Initializes the ADC:** Sets up the necessary ADC unit and channels.
2. **Calibrates the ADC:** Attempts to use Espressif's built-in ADC calibration schemes to make voltage readings more precise.
3. **Reads LMT86 Sensor:** Samples the analog voltage output from an LMT86 temperature sensor connected to a designated ADC pin (specifically, **GPIO32** is highlighted as important for this code to work, which corresponds to `ADC_CHANNEL_4` on ADC1 for ESP32).
4. **Converts Voltage to Temperature:** Applies the LMT86 sensor's characteristic formula to transform the measured voltage into a temperature reading in degrees Celsius.
5. **Prints Results:** Outputs both the raw ADC readings, calibrated voltages (if available), and the calculated temperatures to the serial monitor.
6. **Provides Diagnostics:** Includes warnings if the measured voltage or calculated temperature falls outside expected ranges, which can help in troubleshooting connections or sensor issues.

Code Breakdown

1. Includes and Configuration Macros

This section brings in all the necessary libraries and defines key parameters:

- **Standard C Libraries:** For input/output (`stdio.h`), general utilities (`stdlib.h`), and string manipulation (`string.h`).
- **FreeRTOS Headers:** For task management and delays (`freertos/FreeRTOS.h`, `freertos/task.h`).
- **ESP-IDF Specific Headers:** For SoC capabilities (`soc/soc_caps.h`), logging (`esp_log.h`), and the core ADC functionality (`esp_adc/adc_oneshot.h`, `esp_adc/adc_cali.h`, `esp_adc/adc_cali_scheme.h`).
- **`TAG`**: Used for identifying log messages from this specific application.
- **ADC Channels (`EXAMPLE_ADC1_CHAN0`, etc.):** These macros define which specific GPIO pins are configured as analog input channels. On an ESP32, `ADC_CHANNEL_4` maps to **GPIO32**, which is highlighted as crucial for this example.

- **ADC Attenuation (`EXAMPLE_ADC_ATTEN`):** Set to `ADC_ATTEN_DB_12`, which configures the ADC to measure voltages up to approximately 3.9V. This range is suitable for the LMT86 sensor, which typically outputs voltages within this range.
- **LMT86 Temperature Sensor Constants:** These are critical for the temperature conversion:
  - `LMT86_SLOPE_MV_PER_C`: The sensor's characteristic output change per degree Celsius (e.g., -10.9 mV/°C). This is from the LMT86 datasheet.
  - `LMT86_INTERCEPT_MV`: The sensor's output voltage at 0°C (e.g., 2103.0 mV). Also from the datasheet.
  - `ADC_BITWIDTH_12`: The maximum raw value for a 12-bit ADC (4096, which is 2^12).
  - `ADC_VREF_MV`: The analog reference voltage used by the ADC (typically 3300 mV or 3.3V, especially with the 12dB attenuation).

2. `sample` Function (ADC Reading Helper)

This is a utility function used to perform a single ADC reading.

- It first performs a **raw ADC conversion** using `adc_oneshot_read()`, returning an integer value.
- If ADC calibration is successfully set up (via `cali_handle`), it then uses `adc_cali_raw_to_voltage()` to convert that raw reading into a more precise **voltage value in millivolts (mV)**. This accounts for manufacturing variations in the ESP32's ADC.
- It returns an error code if the reading fails.

3. `convert` Function (Raw ADC to Temperature)

This is a core part of the application, taking the raw ADC reading and converting it to temperature.

- **Raw ADC to Voltage Conversion:** It first converts the raw ADC value into a voltage in millivolts. This is done using the formula: $$V_{mV} = \frac{\text{Raw ADC Value} \times \text{ADC Vref (mV)}}{\text{ADC Max Value (e.g., 4096)}}$$
- **Voltage to Temperature Conversion (LMT86 Formula):** It then applies the specific formula from the LMT86 datasheet: $$T_{\text{°C}} = \frac{V_{\text{out (mV)}} - V_{\text{0°C (mV)}}}{\text{Slope (mV/°C)}}$$ Where $V_{\text{0°C}}$ is `LMT86_INTERCEPT_MV` and Slope is `LMT86_SLOPE_MV_PER_C`.
- **Rounding and Casting:** The floating-point temperature is rounded to the nearest integer and stored as an `int8_t`.
- **Diagnostic Logging:** It prints detailed information, including the raw ADC value, calculated voltage, and temperature. It also includes helpful **warning messages** if the calculated voltage or temperature falls outside typical operating ranges, which can aid in debugging sensor connections or unexpected environmental conditions.

4. `convert_from_voltage` Function (Calibrated Voltage to Temperature - Alternative)

This is an alternative conversion function. If you have already obtained a calibrated voltage from the ADC (meaning `example_adc_calibration_init` was successful), you can pass that directly to this function. It skips the raw-to-voltage conversion step and directly applies the LMT86 formula to the provided `voltage_mv`. It's generally preferred for higher accuracy if calibration is available.

5. `app_main` Function (Application Entry Point)

This is where the ESP32 program begins execution.

- **ADC Initialization:**
  - It initializes **ADC Unit 1**.
  - It configures the ADC channel designated for the temperature sensor (`EXAMPLE_ADC1_CHAN0`, which is GPIO32). It also configures another channel (`EXAMPLE_ADC1_CHAN1`), though it's commented out in the main loop, suggesting it's for potential future use or debugging.
- **ADC Calibration Setup:** It calls `example_adc_calibration_init()` to set up calibration for the ADC channel connected to the temperature sensor. This process attempts to create a calibration handle that ensures more accurate voltage readings.
- **ADC2 Initialization (Conditional):** Similar to the previous ADC example, there's a section to optionally initialize and configure **ADC Unit 2** if your ESP32 board supports it. This part is commented out in the main loop's sampling section, meaning it won't actively read from ADC2 unless uncommented.
- **Main Measurement Loop (`while (1)`):** The core of the application runs indefinitely here.
  - **Temperature Sensor Sampling:** It calls the `sample` function to read data from `EXAMPLE_ADC1_CHAN0` (GPIO32), getting both the raw ADC value and the calibrated voltage (if calibration is active).
  - **Temperature Conversion & Logging:**
    - It first converts the `adc_raw` value to temperature using the `convert` function and logs it.
    - If ADC calibration was successful, it also logs the `calibrated voltage` and then uses `convert_from_voltage` to get a temperature reading based on the *calibrated* voltage. This allows you to compare the results from raw and calibrated conversions.
  - **Delay:** A 2-second delay is introduced using `vTaskDelay` before the next reading, making the serial output easier to follow.
- **Resource Teardown (Unreachable Code):** Like many embedded applications with infinite loops, the code includes functions to de-initialize ADC units and calibration handles. In this specific program, these lines are placed after the `while(1)` loop, meaning they will not be executed under normal operation. They are present for good practice and if the application were to terminate or transition.

6. ADC Calibration Functions (`example_adc_calibration_init` and `example_adc_calibration_deinit`)

These are helper functions, very similar to those found in standard ESP-IDF ADC examples:

- **`example_adc_calibration_init`:** Attempts to create an ADC calibration handle. It prioritizes the more accurate **Curve Fitting** scheme (which relies on factory-burnt eFuse data). If that's not available, it tries **Line Fitting**. It logs success or warnings if calibration data isn't found.
- **`example_adc_calibration_deinit`:** Releases the resources used by the calibration handle.

How to Use This Code

1. **Hardware Connection:**
   - Connect an **LMT86 analog temperature sensor** to **GPIO32** on your ESP32 board.
   - Ensure the LMT86 sensor is correctly wired (Power, Ground, and Output to GPIO32).
   - **Important:** The comment explicitly states, "YOU HAVE TO USE GPIO32 for this to work" – this is crucial, as the code directly uses `ADC_CHANNEL_4` which maps to GPIO32 on many ESP32

development boards.

2. **Build and Flash:** Compile and flash this code onto your ESP32 board using the ESP-IDF toolchain.
3. **Serial Monitor:** Open a serial monitor (at 115200 baud) connected to your ESP32.

You will see continuous readings of the temperature, showing the raw ADC value, the calibrated voltage (if available), and the calculated temperature in Celsius, updated every two seconds. Touching the LMT86 sensor (e.g., with your finger) should cause the temperature readings to change.