Configuring the INMP441 microphone on an ESP32 with AFE

medium.com/@pvginkel/configuring-the-inmp441-on-an-esp32-with-afe-658c1589de25

Pieter van Ginkel March 27, 2025

I’ve been having a hard time getting the INMP441 to work with ESP32. The INMP441 is a popular, cheap and very good I2S microphone. If you found this article, this all probably makes sense to you already, but for the rest, a little bit of background.

I2S is a standard to allow microcontrollers to communicate with audio components. It’s a digital protocol that’s used to talk with microphones and amplifiers. The ESP32 has hardware support for this protocol. There are different components on the market that implement the I2S protocol, and the INMP441 microphone is one of them.

I2S being a standard, you’d think it’d be easy to setup the ESP32 for use with the INMP441. However, I found that it wasn’t trivial. All examples I found were either outdated, of an older version of the ESP32 I2S framework, or just plainly didn’t work. I’ve also found examples where people did get it to work, but, had issues with the fidelity of the audio.

The issue is mostly the specific format the INMP441 uses to put data on the bus. The details can be found in this diagram:

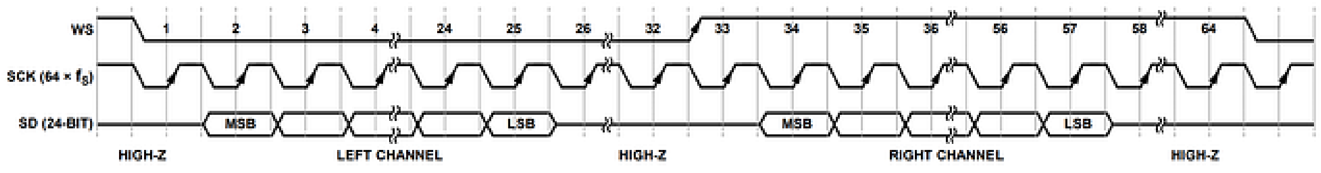

Figure 8. Stereo-Output I²S Format

This diagram comes from the official documentation with can be found here: https://invensense.tdk.com/wp-content/uploads/2015/02/INMP441.pdf.

This kind of looks like the Philips format in that it skips one bit of the slot, and I’ve seen multiple sources suggest using this format. However, it’s not. At least not according to the ESP32 code. The Philips format as defined by the ESP32 documentation actually extends the slot with one bit. It makes a 32 bit slot 33 bits wide.

Instead of depending on the `I2S_STD_PHILIPS_SLOT_DEFAULT_CONFIG` **macro, I’m setting up the INMP441 as follows:**

```
 rx_std_cfg = {    .clk_cfg = (SAMPLE_RATE),    .slot_cfg =        {
.data_bit_width = I2S_DATA_BIT_WIDTH_32BIT,            .slot_bit_width =
I2S_SLOT_BIT_WIDTH_32BIT,            .slot_mode = I2S_SLOT_MODE_MONO,
.slot_mask = I2S_STD_SLOT_LEFT,            .ws_width = ,            .ws_pol = ,
.bit_shift = ,            .left_align = ,            .big_endian = ,
.bit_order_lsb = ,        },    .gpio_cfg =        {            .mclk =
I2S_GPIO_UNUSED,            .bclk = ()SCK_PIN,            .ws = ()WS_PIN,
.dout = I2S_GPIO_UNUSED,            .din = ()DATA_PIN,            .invert_flags =
{                    .mclk_inv = ,                    .bclk_inv = ,
.ws_inv = ,                },        },};((chan, &rx_std_cfg));
```

This sets up a single microphone configured as the left channel. I'm setting the data width, slot width and ws width all to 31. I'm also disabling the bit shift. Basically I'm just capturing all data the INMP441 outputs verbatim. Btw this code also works if you want to capture stereo. Just set `slot_mode` to `I2S_SLOT_MODE_STEREO` and `slot_mask` to `I2S_STD_SLOT_BOTH`.

You have a few options then to capture samples. Assuming you've captured samples of type `int32_t`, you can sign extend the 24 bits of data as follows:

```
 raw_sample = ...; sample = (raw_sample >> ) << ;
```

This first shifts away the 7 LSB bits and then sign extends the 24 bits of data.

The sample code however also sets up AFE, or the ESP32 Audio Front-end. AFE is a framework that includes a few components that help build audio devices. I'm building an intercom and I want to use AFE to do noise suppression, acoustic echo cancellation and automatic gain control.

AFE only supports 16 bit samples. The simplest way to get the 16 sample out of the 32 bit sample is to take the upper 16 bits of the 24 bits sample. However, it's better to get a different range of bits. The range of the INMP441 microphone is quite large. The result is that the volume of the audio you get from it is quote low. If you implement a fixed gain when converting the 24 bit sample to 16 bits, you keep a lot more fidelity. This is implement as follows:

```
 raw_sample = ...; afe_sample = ()(raw_sample >> ( - GAIN_BITS))
```

This way of converting the 24 bit sample into 16 bits gets you a sample that's 8 times the volume of the sample that comes out of the INMP441 while keeping full 16 bit fidelity.

You can find the code sample at https://github.com/pvginkel/INMP441Test. It contains a fully working ESP32 app that captures audio and sends it to a Windows C# app using UDP to play it back. Forgive me for the hard coded settings in the `main.cpp` file as it's just a sample app to test out the INMP441.

As for the AFE setup, the code is a bit messy. This is mostly because AFE has quite a few settings and I've tried to make it easy to play around with them.

The code in the repo sets up Acoustic Echo Cancellation, Noise Suppression and Automatic Gain Control. Even though Acoustic Echo Cancellation is setup, it's not doing anything because the app in the repo doesn't have an amplifier configured. It's not playing audio. You would have to provide a reference audio channel to AFE. You'd replace the `0` in the following line of code in the app with samples you're sending to the amplifier:

```
feed_buffer[feed_buffer_offset++] = ;
```

And that's it. If you found this useful I wish you success and fun with your ESP32 project!