Digital signal processing: I2S in ESP32

Anthony LE CREN, KF4GOH
f4goh@orange.fr

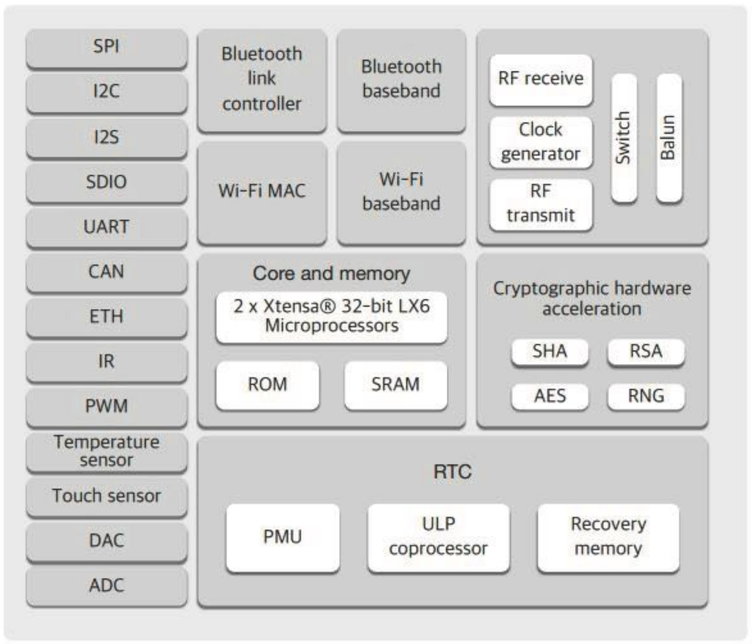
Abstract

How to decode and encode RTTY with an Espressif ESP32 and I2S (Inter-IC Sound)

1 Introduction

For several years I have been using an Arduino UNO (Atmega328p) to realize all kinds of applications around the radio. Most of the time, it's very easy to generate an FSK signal using the internal PWM of the processor added with an external low-pass filter. (see Standalone HAM modulation generator: TAPR N°37). On the other hand, it is much more complicated to decode an FSK-type audio signal with the same processor. Indeed, you have to apply signal processing algorithms and you realize that it is difficult to use an FFT algorithm in an Atmega328p while having real-time decoding. Robert Marshall (KI4MCW) explains how to decode an FSK signal with an Arduino Uno [1]. I was able to successfully test the algorithm written by Dennis Seguine in the realization of my APRS repeater with a DRA818 [2,3].

The manufacturer Espressif is known for the ESP8266, a processor that embeds WIFI connectivity. This integrated circuit has managed to federate a very large community of "makers", particularly around IOT connected objects. More powerful, the ESP32 integrates more memory and bluetooth connectivity.

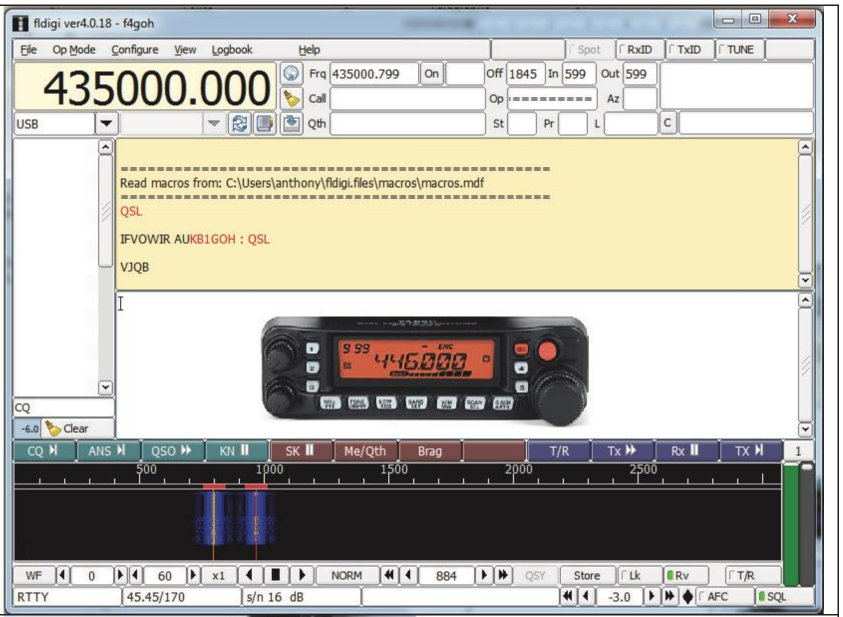
Main features of the ESP32: CPU: Xtensa dual-core operating at 240 MHz Memory: 520 KiB SRAM Wireless Connectivity : Wi-Fi: 802.11 b/g/n Bluetooth: v4.2 BR/EDR and Peripheral interfaces: 12-bit SAR ADC 18 channels 2 × 8-bit DACs 4 × SPI 2 × I²S interfaces 2 × I²C interfaces 3 × UART CAN bus 2.0 PWM outputs	

2 Brief presentation of the project

Step 1

A PC with Fldigi software is connected to a Yaesu ft7900 transceiver.

The user kb1goh sends a message in RTTY for example: **qsl**.



```
fldigi ver4.0.18 - f4goh
435000.000
=================================================
Read macros from: C:\Users\anthony\fldigi.files\macros\macros.mdf
=================================================
QSL
IFVOWIR AUKB1GOH : QSL
VJQB
```

Step 2

The RTTY frame is received by the DRA818 module and decoded by the ESP32.

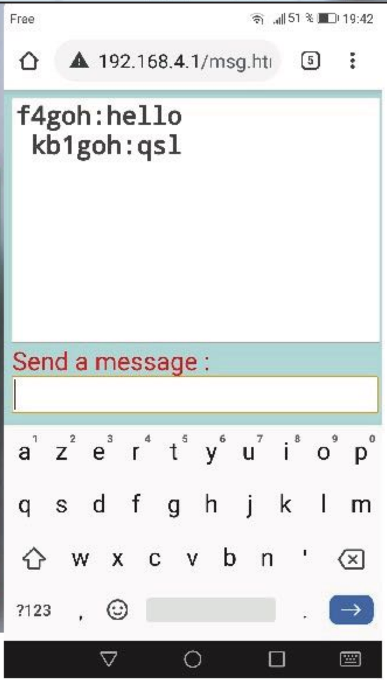
The esp32 is configured as a WiFi access point and web server.

The user F4GOH consults the RTTY messages on his phone using a WEB page.

There is no specific application to install and no Internet connexion.

communication is bi-directional.

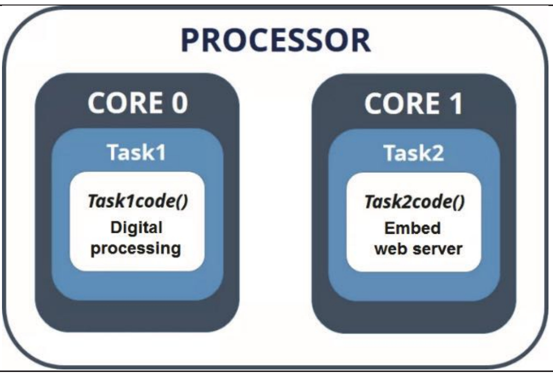
ESP32 Access point



```
f4goh:hello
 kb1goh:qsl
```

Send a message :

We notice that the processor has 2 cores. Core 0 handles digital processing and thus RTTY decoding while core 1 handles the main loop and the web server.



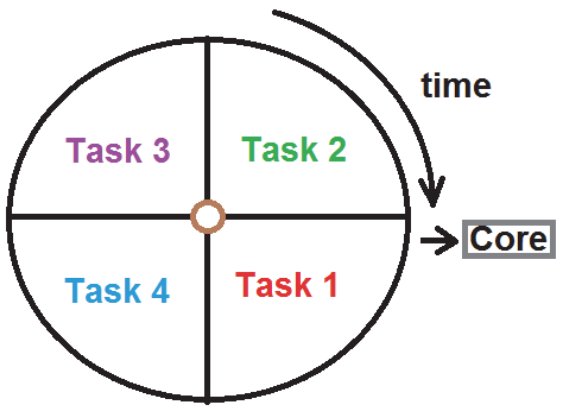
3 FreeRTOS

FreeRTOS is a portable, open source real-time operating system (RTOS) for microcontrollers. Created in 2003 by Richard Barry, it is today one of the most widely used RTOS in the real-time operating system market, including the ESP32

The advantage of using freeRTOS [4] is to be able to manage several tasks in parallel. The number of tasks executed simultaneously and their priority are limited only by ESP32.

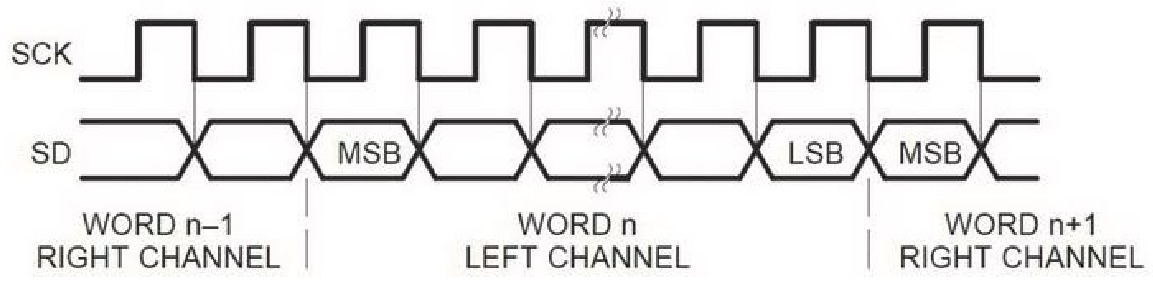
To assign specific parts of the code to a specific kernel, you need to create tasks. When you create a task, you can choose in which kernel it will run, as well as its priority. Priority values start at 0, where 0 is the lowest priority. The processor will run the tasks with the highest priority first.

Tasks are pieces of code that execute something. For example, it can be flashing a LED, making analog/digital acquisitions, measuring sensor readings, publishing these readings on a server, and so on.

Scheduling (the program that manages the different tasks is based on the Round-Robin model with priority management). The round-robin is a kind of turnstile where each process or task that is on the turnstile just passes in front of the processor, in turn and for a fixed time.	The drawing below shows 4 tasks running in a core: 

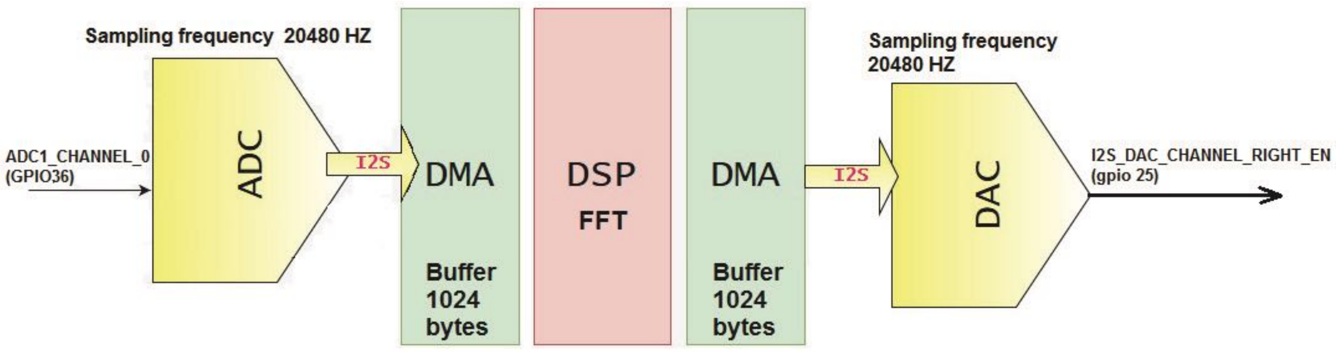
4 Inter-IC Sound

I²S (Inter-IC Sound), pronounced eye-squared-ess, is an electrical serial bus interface standard used for connecting digital audio devices together. It is used to communicate PCM audio data between integrated circuits ADC/DAC in an electronic device. The I²S bus separates clock and serial data signals, resulting in simpler receivers than those required for asynchronous communications systems that need to recover the clock from the data stream.
The chronogram below shows the clock and the data signals. In the case of stereo acquisition, the right and left channels are alternated.

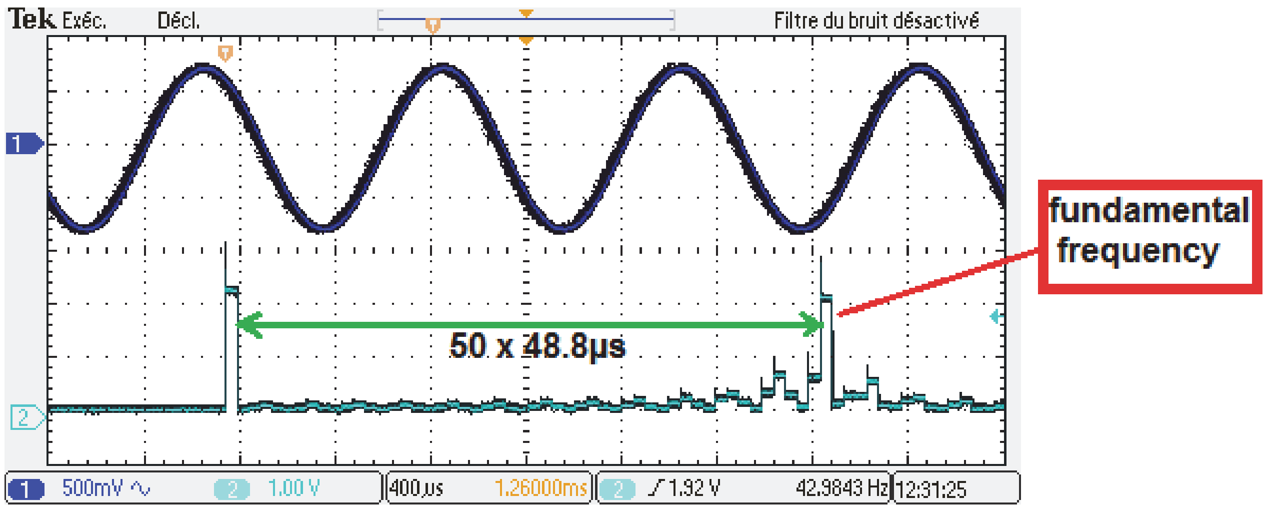


In an ESP32, the I2S bus uses DMA (Direct Memory Access) transfer. DMA is a process in which data flowing to and from a device is transferred directly by a suitable controller to the main memory of the machine, without any intervention by the microprocessor except to initiate and complete the transfer.

This allows the processor to have more time to perform signal processing calculations. This is a huge advantage over the Arduino UNO.


Sampling frequency 20480 HZ
ADC1_CHANNEL_0 (GPIO36)
ADC
I2S
DMA
Buffer 1024 bytes
DSP
FFT
DMA
Buffer 1024 bytes
I2S
Sampling frequency 20480 HZ
DAC
I2S_DAC_CHANNEL_RIGHT_EN (gpio 25)

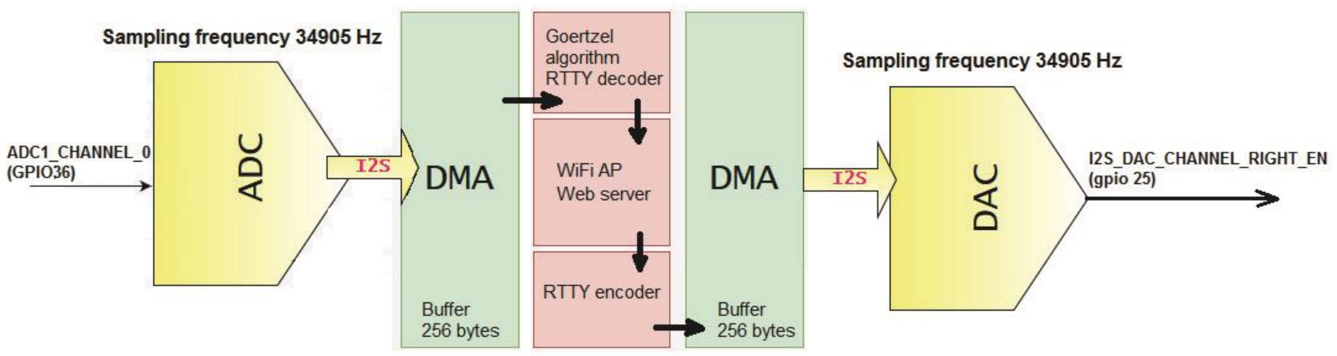
In order to test I2S, I injected with a function generator a 1000Hz sinusoidal signal on the input of the ADC0 (GPIO36: CH1). The processor then calculated the FFT of the signal. The result of the magnitude is then sent to the digital to analog converter DAC(CH2). Each step of 48.8µs (1/20480) corresponds to a 20Hz step (20480/1024) in the spectrum.



The time between the DC component and the fundamental is 50 x 48.8µs, so this corresponds to a frequency of 20 x 50 = 1000Hz.

5 RTTY decoding

An RTTY signal is composed of two frequencies (MARK and SPACE). There is no need to perform a full FFT. For this I used the Goertzel algorithm [5], which allows to detect the presence of a frequency in a sequence of samples. This is an efficient method to evaluate a particular term of the discrete Fourier transform; it requires only one multiplication and two additions per sample. The Goertzel algorithm is obviously used twice, one for the MARK frequency and the other for the SHIFT frequency.



The choice of sampling rate and buffer size is related to the RTTY baud rate (45.45 Bauds). I used the chunk method to detect a logical one or zero level corresponding to the RTTY transmission. If 3 consecutive chunks have the same frequency, I deduce the corresponding logical level of the RTTY bit. I can then calculate the sampling frequency:

Sampling frequency = RTTY speed x Number of chunks x buffer size.

34905 Hz = 45.45 x 3 x 256

At each change of state frequency, I count the number of consecutive previous chunks. The byte is composed of the start bit (always 0) followed by the five data bits and two stop bits (always 1).

0,0,1,->3	1,1,0,->3
1,1,1,1,1,1,1,1,1,1,1,0,->12 = 11+1 (previous)	0,0,1,->3
0,0,1,->3 = 2+1	1,1,1,1,1,0,->6
1,1,1,1,1,0,->6 = 5+1	Byte->D4 (11*01 010*0 char decoded 4)
Byte->DE (11*01 111*0 char decoded k)	0,0,1,->3
0,0,1,->3	1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,->21
1,1,0,->3	Byte->FE (char toggle to letters)
0,0,1,->3	0,0,0,0,0,1,->6
1,1,1,1,1,0,->6	1,1,0,->3
0,0,1,->3	0,0,1,->3
1,1,1,1,1,0,->6	1,1,1,1,1,1,1,1,1,1,1,0,->12
Byte->DA (11*01 101*0 char decoded f)	Byte->F4 (11*11 010*0 char decoded g)
0,1,shift!	0,0,0,0,0,0,0,0,0,0,0,1,->12
->3	1,1,1,1,1,1,1,1,1,1,1,0,->12
1,1,1,1,1,0,->6	Byte->F0 (char decoded o)
0,0,1,->3	0,0,0,0,0,0,0,0,1,->9
1,1,1,1,1,1,1,1,1,1,1,0,->12	1,1,0,->3
Byte->F6 (11*11 011*0 toggle to figures)	0,0,1,->3
0,0,0,0,0,1,->6	1,1,1,1,1,1,1,1,0,->9
1,1,0,->3	Byte->E8 (11*10 100*0 char decoded h)
0,0,1,->3	

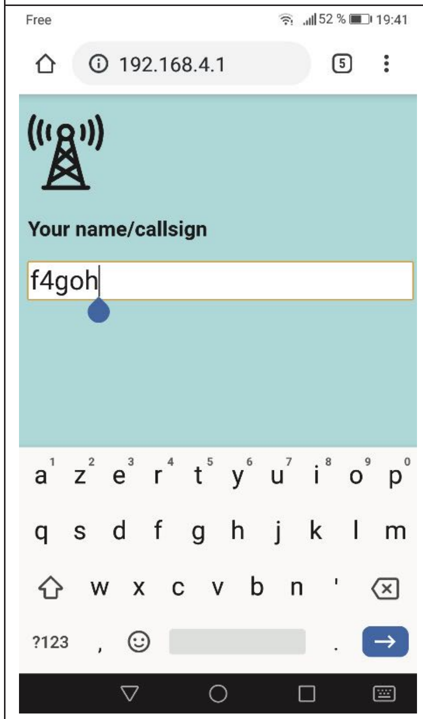
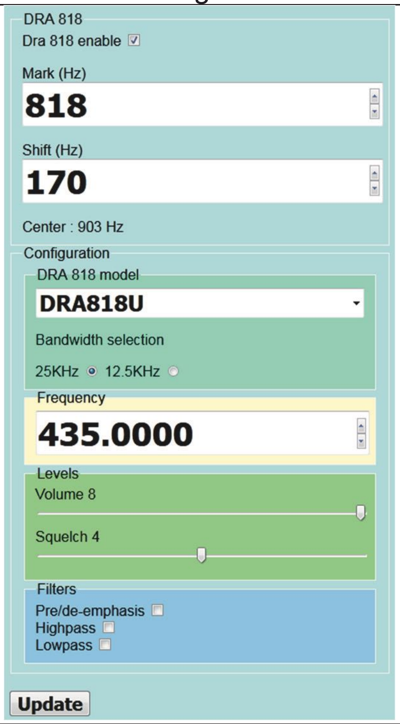
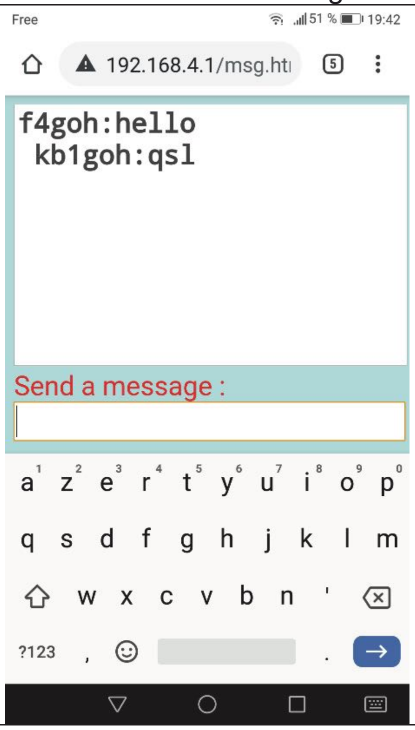
I preferred to use 2 stop bits rather than the traditional 1.5 stop bits. This to facilitate decoding during my first attempts. Nevertheless, I must detect the space character in order to synchronize the decoder and not to display random characters.

When sending an RTTY frame, I always start by sending a space character first. This is obviously useless when using 1.5 stop bits.

6 The web server

The WEB server is composed of four HTML and JavaScript pages. A home page allowing you to choose your callsign and an option to configure the DRA818 and the MARK / SHIFT frequency.

The HTML page that manages the sending and receiving of messages makes AJAX requests to the server to update the reception area. This reception area is refreshed every two seconds. Sent and received messages are also displayed on the mini OLED screen.

Main menu	DRA 818 configuration	Send and receive messages
		

Free 52 % 19:41

192.168.4.1

Your name/callsign

f4goh

DRA 818

Dra 818 enable ☑

Mark (Hz)

818

Shift (Hz)

170

Center : 903 Hz

Configuration

DRA 818 model

DRA818U

Bandwidth selection

25KHz ● 12.5KHz ○

Frequency

435.0000

Levels

Volume 8

Squelch 4

Filters

Pre/de-emphasis ☐

Highpass ☐

Lowpass ☐

Free 51 % 19:42

192.168.4.1/msg.htı

```
f4goh:hello
 kb1goh:qsl
```

Send a message :

7 Conclusion

Through these experimentations around the esp32 and the I2S bus, my main goal is achieved. It is possible to decode an FSK audio signal while managing a WEB page in the same processor. This reduces the hardware, but the time spent to develop the software increases. It would be interesting to switch to APRS decoding. I will end by pointing out a very interesting site, that of HA2NON which gives examples of decoding under PC using the JAVA language [6].

[1] https://sites.google.com/site/ki4mcw/Home/arduino-tnc

[2] https://hamprojects.wordpress.com/2015/07/01/afsk-dra818-aprs-tracker/

[3] https://github.com/f4goh/TNC

[4] https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/freertos.html

[5] https://en.wikipedia.org/wiki/Goertzel_algorithm

[6] http://dp.nonoo.hu/projects/ham-dsp-tutorial/