OPENSATCOM- IQ MODULATOR

SC Robotics - Libre Space Foundation

Table of contents

Table of figures

About the project

This project and exploration is a sub-activity within the OpenSatCom activity of Libre Space Foundation and ESA. Within the context of OpenSatCom various technology explorations are executed with the goal to advance open hardware and open software SATCOM ecosystem and create steppingstones for more projects and explorations. The goals of this specific project are:

- Investigate the usage of off-the-shelf SoCs (with embedded DACs) to generate digital radiofrequency signal, given an IQ bit stream.
- Target the use case of spread-spectrum SATCOM applications
- Evaluate the results of the experimental design

The target project architecture (the main one that will be used to validate it) is explained in the following diagram:

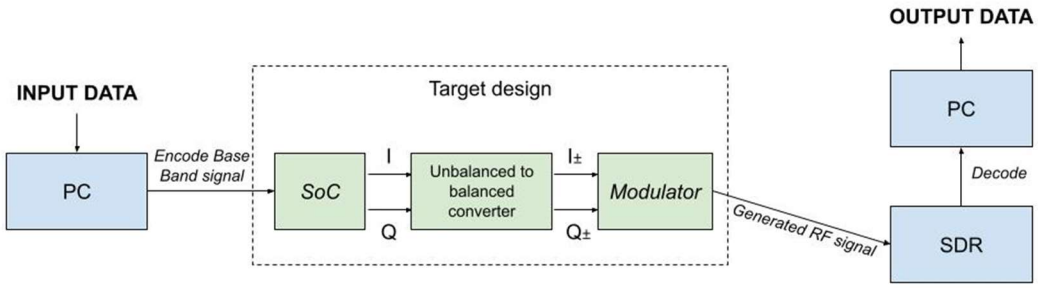

Figure 1. Design architecture diagram

Please note how the signal (and samples) generation falls out of the scope of the present project.

All the materials (including this report) have been released under Open-Source licenses. Next you can find the source files as well as the links to them:

- Firmware: https://gitlab.com/scrobotics/opensatcom/lsf-opensatcom-fw / GNU General Public License v3.0
- Hardware: https://gitlab.com/scrobotics/opensatcom/lsf-opensatcom-hw / CERN Open Hardware License v2

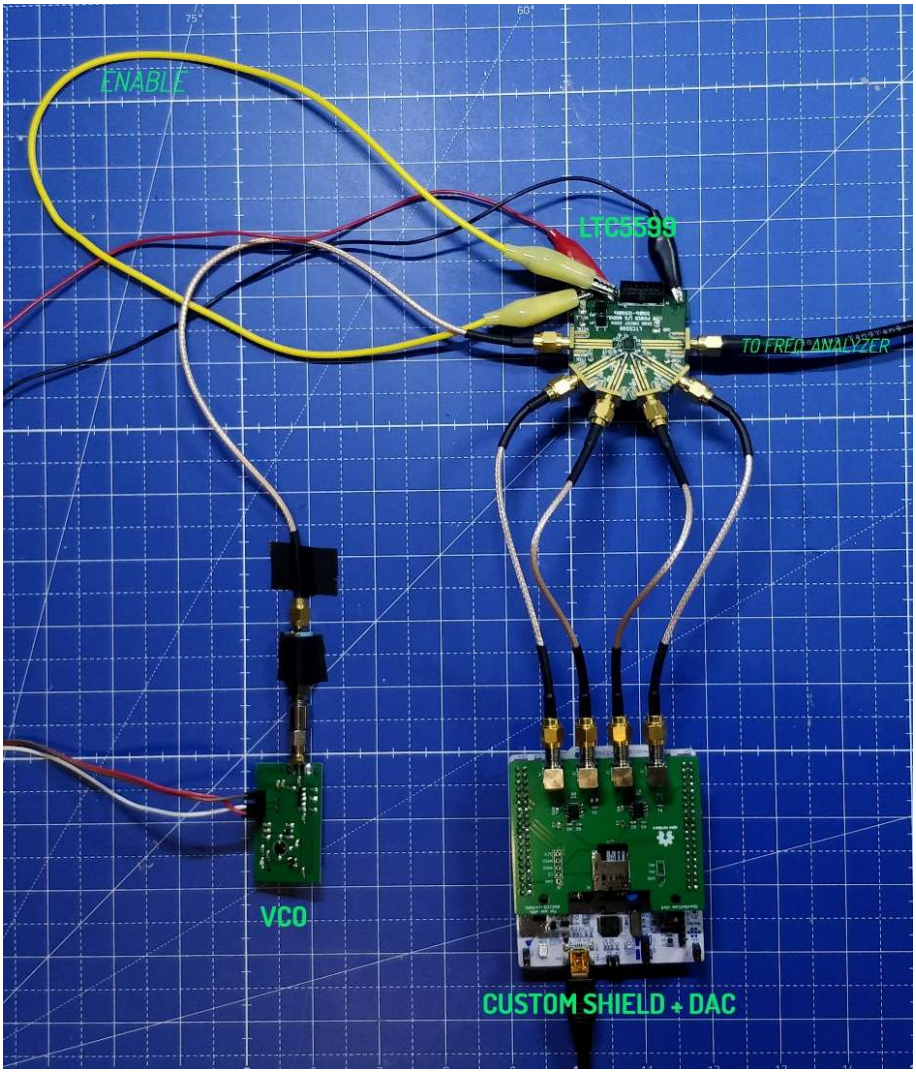

Figure 2. Test components

Firmware

For the firmware SDK we've decided to use STM32Cube on top of PlatformIO. The reasoning for the former was that some of the features we would need were quite specific and we feared other frameworks would not expose them like STM32Cube does (it's from the same manufacturer).

Testing DACs

The first step was to make sure the two DAC channels can be used at the same time. In the STM32 this is usually code *dual DAC channel*. As can be seen in the oscilloscope screenshot below (the blue line represents one channel, the yellow one another channel). Please bear in mind that we've used DAC blocking mode for these tests.

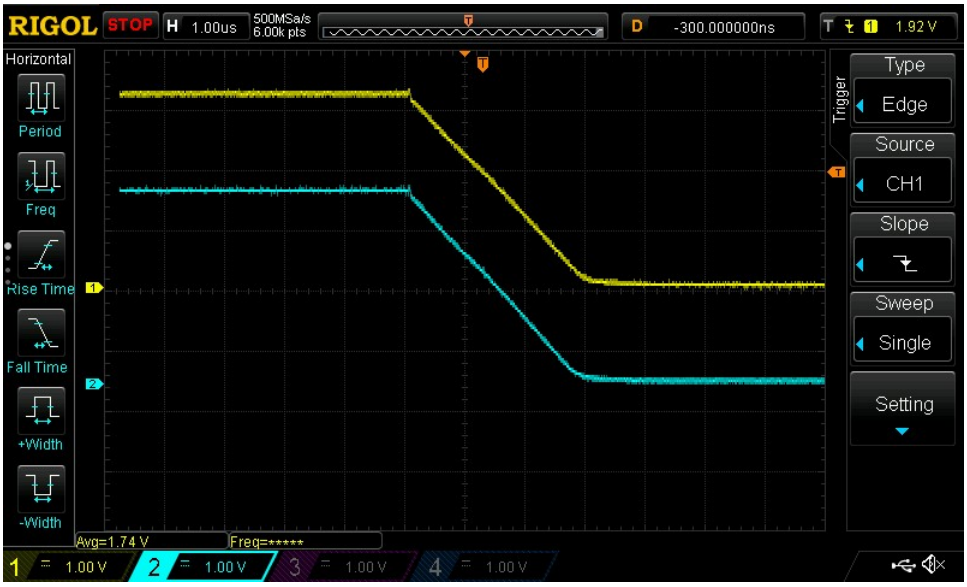

Figure 3. DAC output in dual channel mode (zoom in)

Quoting the Family Reference Manual (RM0351):

> *In dual DAC channel mode, conversions could be done independently or simultaneously when both channels are grouped together for synchronous update operations. - RM0352, 19.1*

We also wanted to read the signal from an SD card (micro SD card in this case). Again, the screenshot below represents the IQ signals of a sinewave in the DAC output. The dark purple is the sum of the two signals which should be constant (plus/minus the quantization errors).

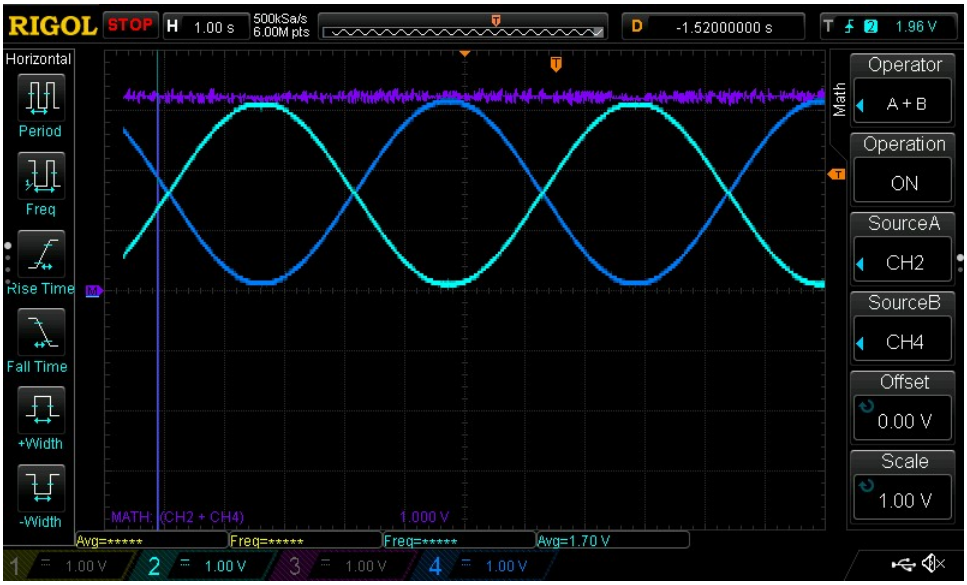

Figure 4. Sinewave in DAC output (dual channel mode)

The blocking mode was doing what was expected. However, we detected two issues:

1. The DAC were not able to work in the full range (0-255 in case 8 bits were being used). When using 12 bits we saw a similar behavior.
2. In some points we saw some signal distortion. After some tests we saw this error happened when reading certain samples from the SC card. Our assumption is that when changing blocks (due to the flash nature of the memory) the SPI message took longer than we expected.

DACs in DMA mode

To overcome the first issue, we had to reduce the signal amplitude and add some offset to it so that it never reaches the detected limits. Another way to say it is that the signal needs to be compressed in order to take advantage of the full range of the DAC.

In the figure below, we've captured a ramp signal (in terms of digital values, a linear signal that goes from 0 to 255) in the output of the two channels. In this picture the errors in the two ends of the ramp (upper and lower limits) can be easily seen (especially in the dark blue, which corresponds to the second DAC channel).

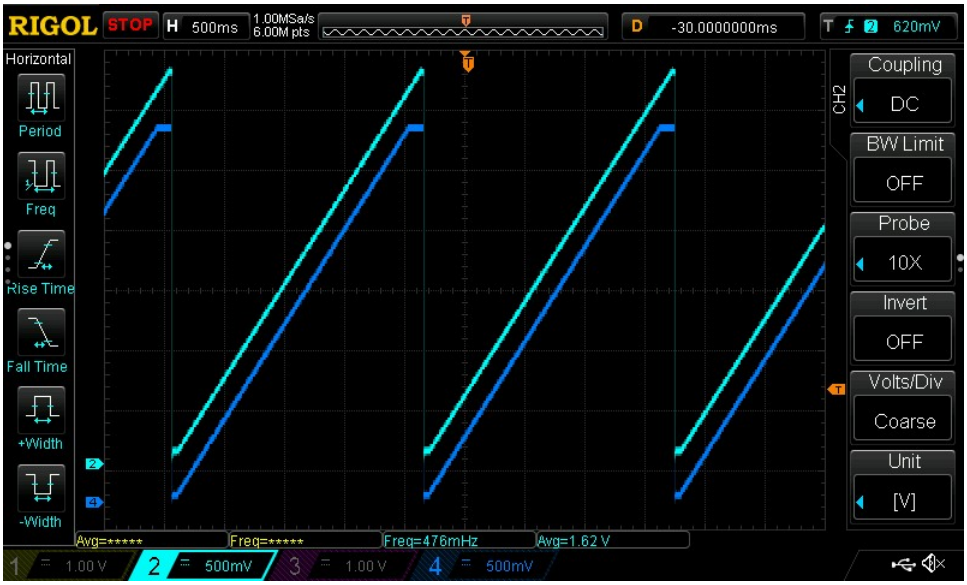

Figure 5. Signal ramp in dual channel mode through DMA

The second issue can be fixed by first reading from the SD card to RAM and then use DMA mode (from RAM to the DAC peripheral) to use a sampling rate as fast as possible. This is of course not the ideal solution, but it's the fastest one for a proof-of-concept like this one. The main disadvantage of this approach is that the RAM size limits the number of samples that can be stored and transmitted. In our case, we are using the STM32L476RG which has 128 kB of SRAM. We could also use the STM32L49x series, with 320 kB of RAM. The latter is very similar and only minor changes should be necessary.

To make sure the DAC dual mode was still working through DMA, we captured some zoomed in shots representing how the two channels change at the same time:

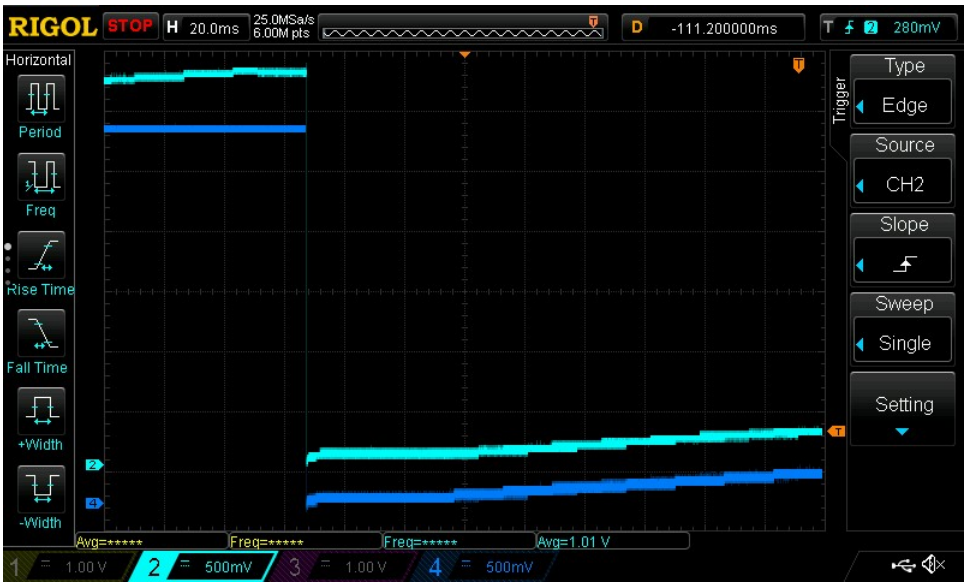

Figure 6. Dual channel mode through DMA (zoom in)

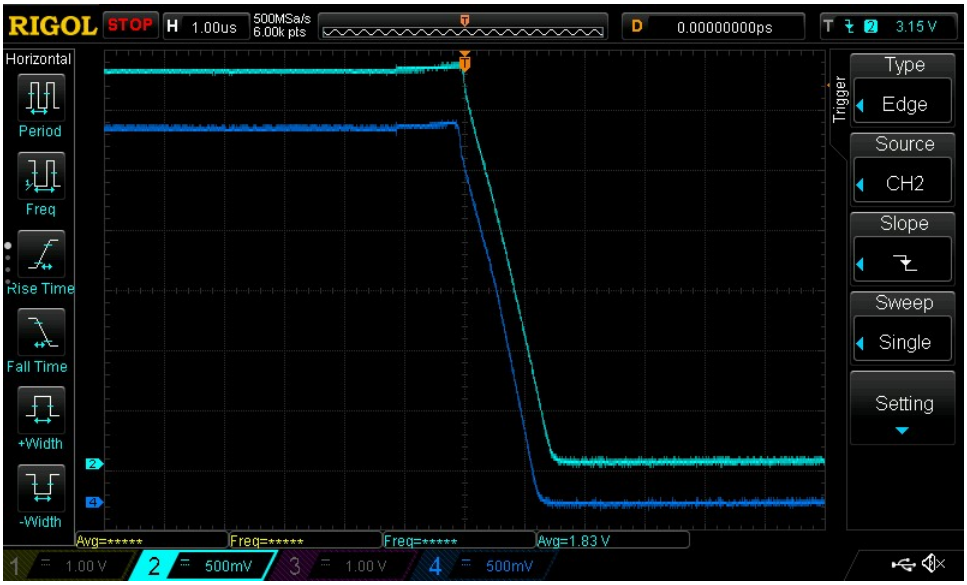

Figure 7. Dual channel mode through DMA

To sync signal the sample rate with the DAC (i.e., to output the signal samples at the frequency defined by the sample rate) we used a feature described in the reference manual as "DAC trigger selection":

> *If the TENx control bit is set, the conversion can then be triggered by an external event (timer counter, external interrupt line). - RM0352, 19.4.6*

What we have had to do is configure one timer (T4 in our case) and initialize it so that every tick a new sample is put in the output of the DAC. By adjusting this timer configuration we can adjust the sampling rate of the signal. In theory, with this approach the maximum sampling rate of 10 Msps (see Table 2 of AN4566) could be reached (although the AN mentions that "Values reported [...] have been measured on the bench, when bus is not used by any other system: in real applications it is necessary to have some margin.").

As we've seen, this 10 Msps limit is a theoretical number, but we have not managed to achieve this value. In our tests we've just been able to set each channel of the DAC to work at roughly 1.3 Msps. This is far less than the 10 Msps advertised in the AN4566 however there are at least two possible causes:

- The official application note also mentions the maximum bus speed being 80 MHz and, in our case, the same DAC bus (APB1) is being used for other peripherals like the timer 4 (TIM4).
- We are, at least, using 2 channels of 8 bits each, so in our case 5 Msps should be the maximum value instead of 10.

What we've noticed when trying sampling rates higher than 1.3 Msps is that the DAC continues working but some problems start to appear. In the figure below we can see some of them: for starters the two channels are not synchronized anymore. What we mean is that the two signals should be opposite signals (when ramp 1 goes up the other one goes down) and instead we see that big steps (from 0 to 255 and the other way around) in unexpected places. We also have to take into account the maximum slope rate. Notice how in the previous

picture it takes a while for the dark blue line to go from 0 to 255. In fact, 255 should correspond to 3.3 V and it doesn't even reach that value.

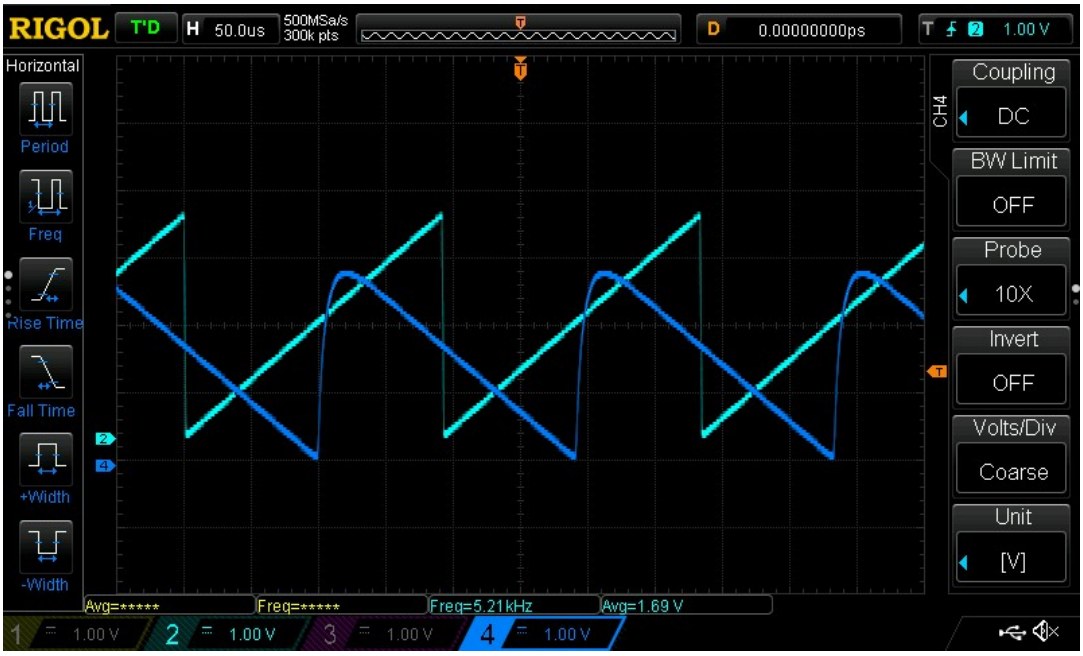

Figure 8. DAC failure with high sampling rates (ramp)

See how a similar behavior can be seen in the picture below, where the signal is a sine (I samples correspond to light blue, Q to dark blue). In this screenshot, the sinewaves have an offset of π/2.

Figure 9. DAC failuter with high sampling rates (sinewave)

Signal modulation

IMPORTANT: The bandwidth of the signals shown in screenshots of the frequency analyzer are not accurate (to be more precise, the deltas are much narrower than what can be seen in the pictures). That was cause by a bad configuration of the device but unfortunately, we didn't find out after the project was over.

Now that we can transmit a signal from the microprocessor to the output of the DAC, we want to attach the modulator (LTC5599) and verify the entire chain. To test the chosen modulator, we have to generate a signal for its local oscillator of 492.8 MHz (this is configurable through SPI, but we consider this out of the scope of this project). As we can see in the screenshot below that corresponds to a frequency analyzer, this signal generates a small delta.

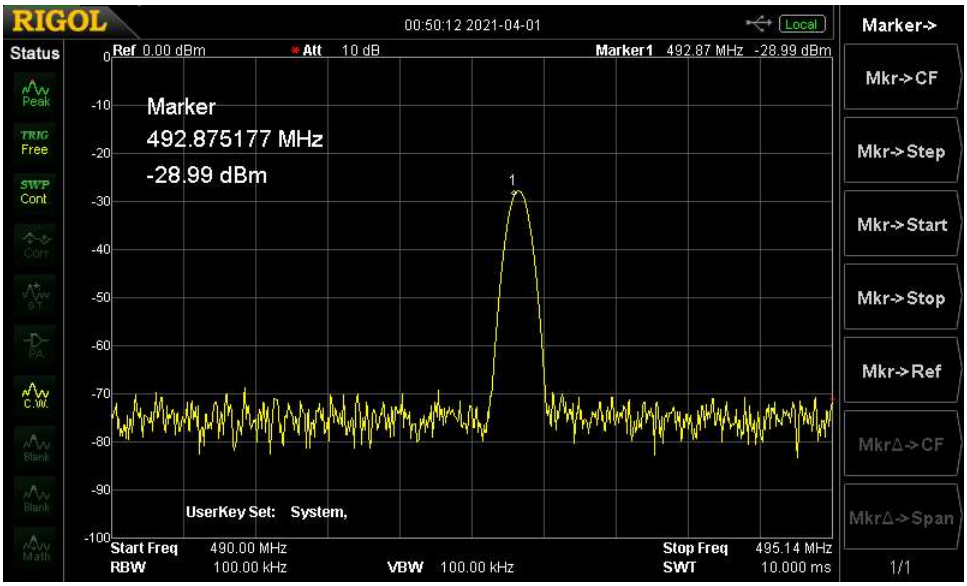

Figure 10. No signal (IF noise from the local oscillator)

As soon as we start transmitting a sine from the Nucleo board, we can see (picture below) how the peak is much stronger. **Since there are limits of samples per second the DAC can output, the higher the samples per cycle, the lower the frequency of the sinewave we can transmit**. In this first test, to maintain the sine shape (128 samples for I, 128 samples for I) we had to keep a low sine frequency (1kHz). As a result, we can barely notice the difference in the position between the previous peak (no signal) and the new one (1 kHz sinewave).

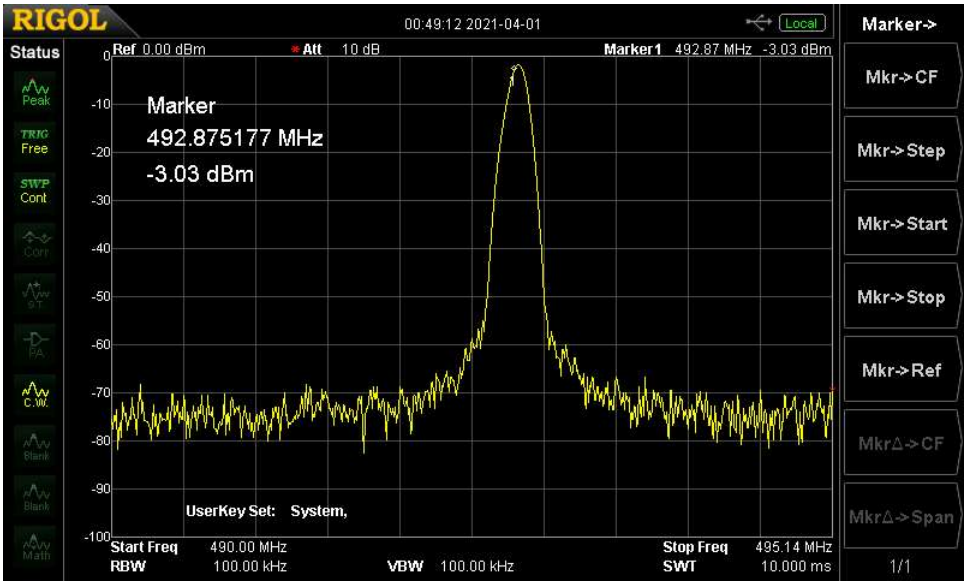

Figure 11. 1 kHz frequency spectrum quantized at 128 samples per cycle (128 for I and 128 for Q)

For the next test we wanted to generate a sine of a much higher sample rate. We've generated *a sinewave of around 53 kHz* however we had to reduce the number of samples of 8 per cycle (16 in total). This impacts in the shape of the delta, which now has much significant side-lobes.

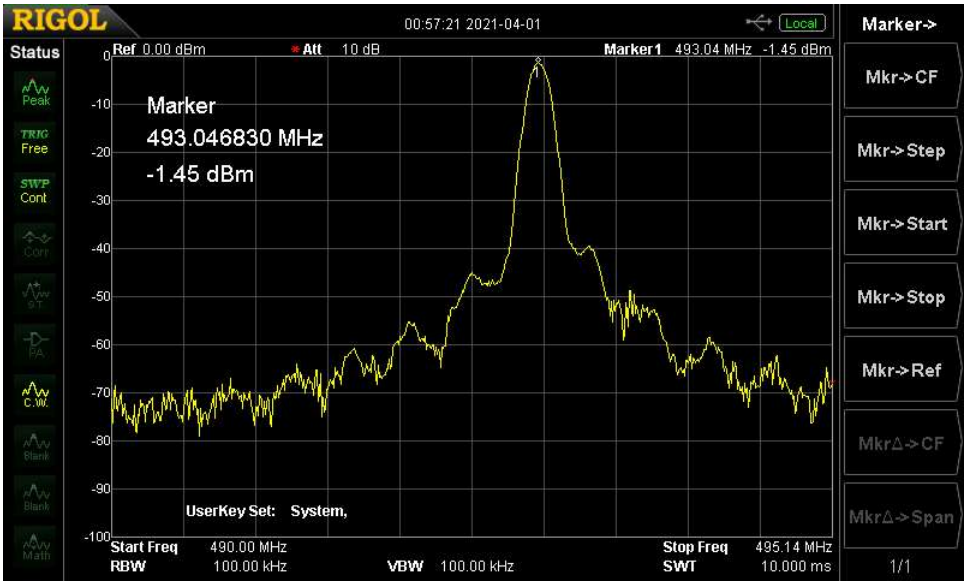

Figure 12. 53 kHz frequency spectrum quantized at 8 samples per cycle (8 for I and 8 for Q)

Just for the reference, we wanted to capture what the sinewave looks like. In the picture below we can clearly see the 8 (4 times Nyquist) samples per cycle of the sinewave which corresponds to the previous screenshot. If we want to reduce the quantization steps from this signal and convert it into a *softer* sinewave we would have to add analog filters in the output of the DAC (continue to the hardware section for more details). As we said previously, we consider this falls out of the scope of the present project.

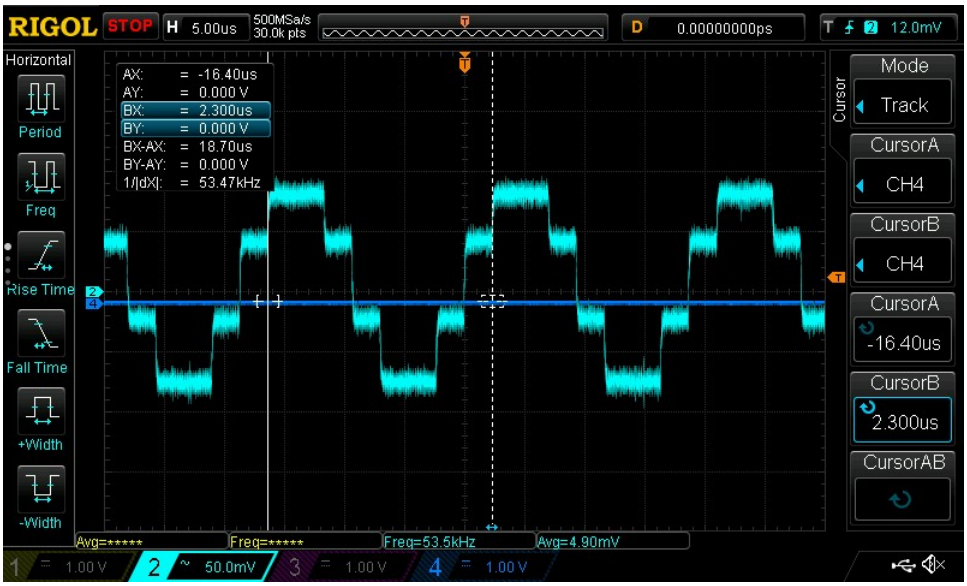

Figure 13. 53 kHz signal quantized at 8 samples per cycle

Hardware

The custom shield

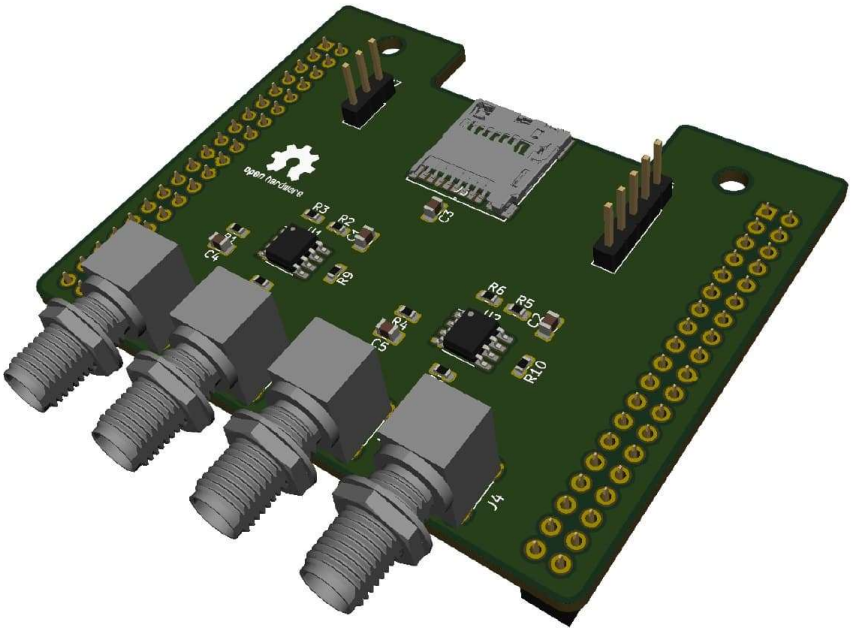

Figure 14. Render of the custom shield

First approach consisted in designing a shield for the NUCLEO-L476RG. We included all the elements needed for the development, including signal storing and unbalanced to balanced conversion. The hardware basically consists in two parts:

- Analog section – It's the part in charge to convert single-ended signals from DAQs to balanced signals needed to input the modulator. It has two identical circuits made up of Op-amps in follower configuration. One for the positive differential signal and one for the negative one. The common mode voltage is configurable with R2-R3 and R6-R7, each pair acting as a reference voltage for the circuit. In this case, we chose 1.4V because of the specs of the modulator used for the experimental test. Each branch of the circuit is terminated with SMA connector to facilitate the connection with the modulator. Op-amps are rail-to-rail ones to achieve the maximum voltage range, despite the fact of the modulator only need a few hundred of millivolts to work.

  Both pairs R1-C4 and R4-C5 can work as a simple analog low-pass RC filter if needed. They would soften the signal reducing quantization noise. Placing these filters in the unbalanced section reduces the complexity of the circuit and make the filters more controllable as they are referenced to GND.

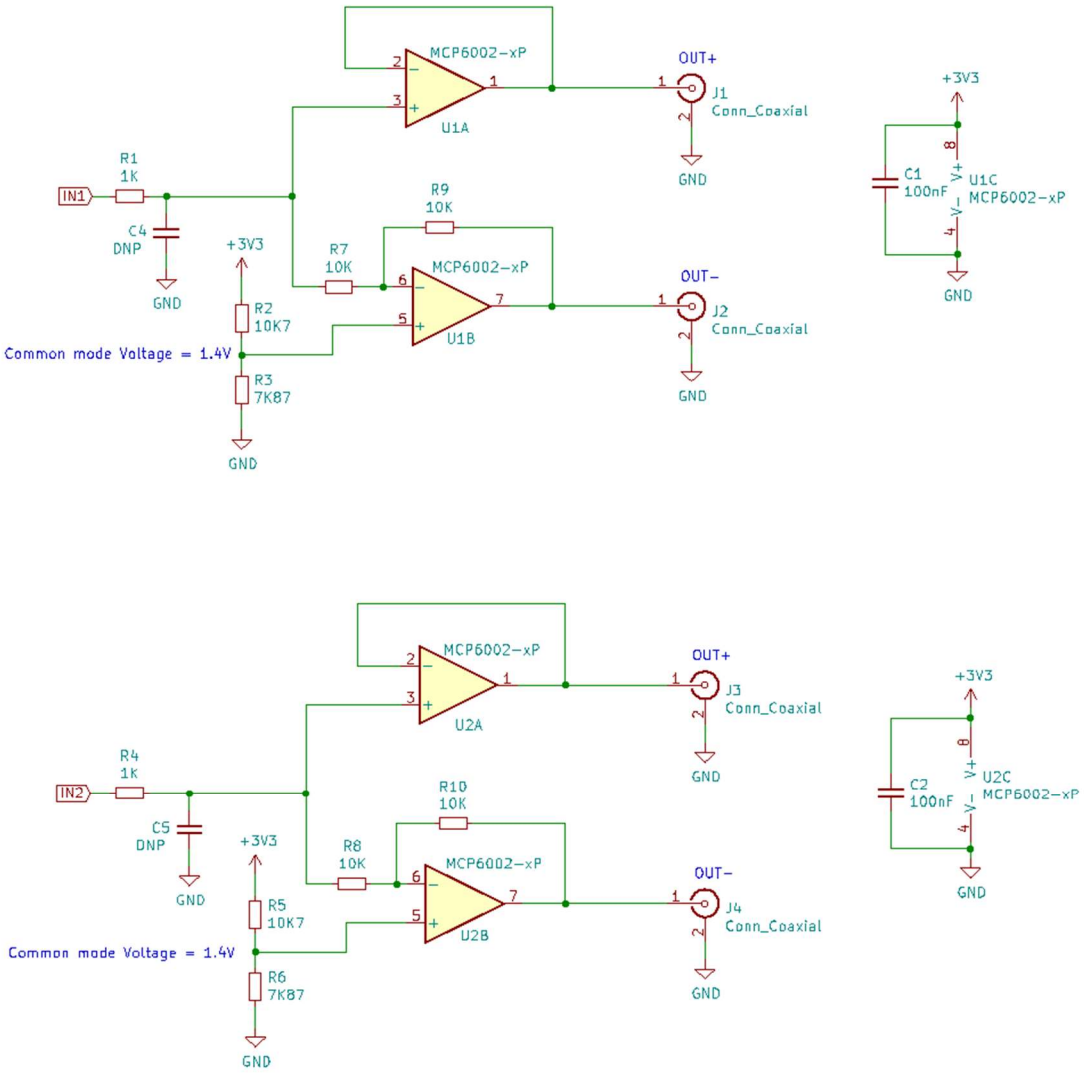

Figure 15. Unbalanced to balanced analog converter

- Micro SD card holder – Just a micro SD card holder connected with the SPI interface of the Nucleo ST board.

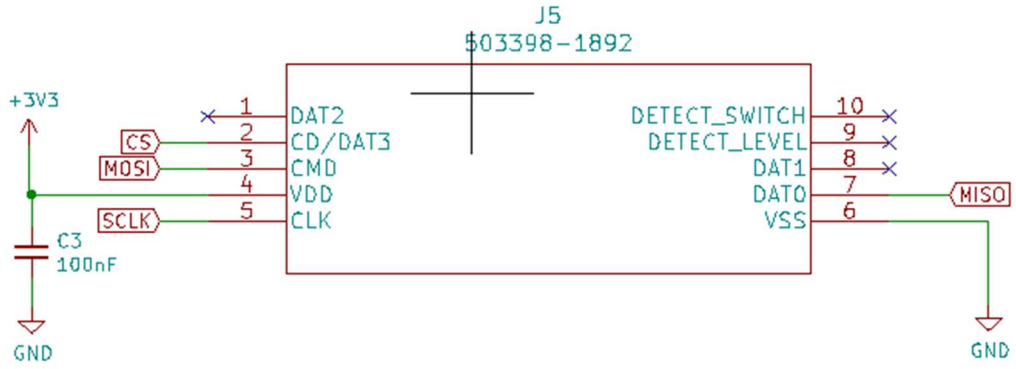

Figure 16. uSD Card interface

Other hardware blocks

The LTC5599 is a direct conversion I/Q modulator that enable direct modulation of differential baseband I and Q signals on an RF carrier. Its frequency range comes from 30MHz to 1300MHz and can be powered from 2.7V to 3.6V.

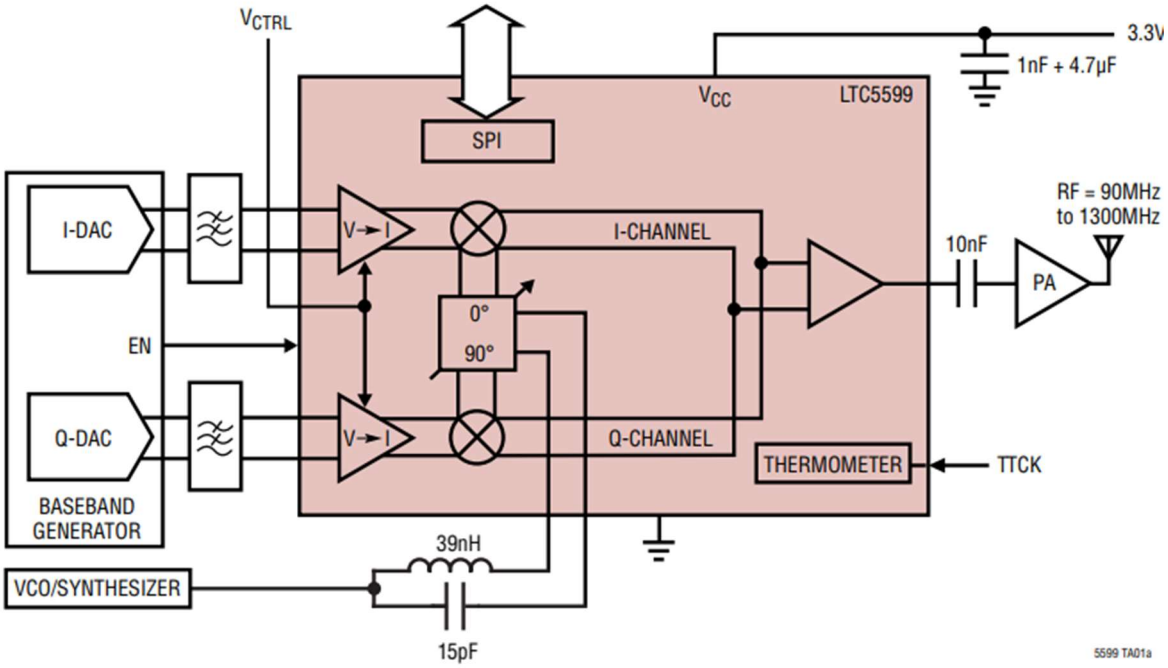

Figure 17. Modulator internal scheme

As we mentioned before, we relied on the LTC5599 from Analog Devices to modulate the signal. *By default,* this board needs a local oscillator of 492.8 MHz. This signal is generated with a VCO and tuned in with the assistance of a frequency analyzer. We also have to connect to it the I and Q signals generated in our shield. According to the Demo Manual, by default, it also needs a signal of 200 mVpp. Since the full range of the DAC is 0 to 3.3V, we have to either reduce the amplitude of the signal or add analog attenuation. We opted for the first option but we know reducing the amplitude of the signal means the DAC operates with a reduced number of bits.

MEASUREMENT SETUP

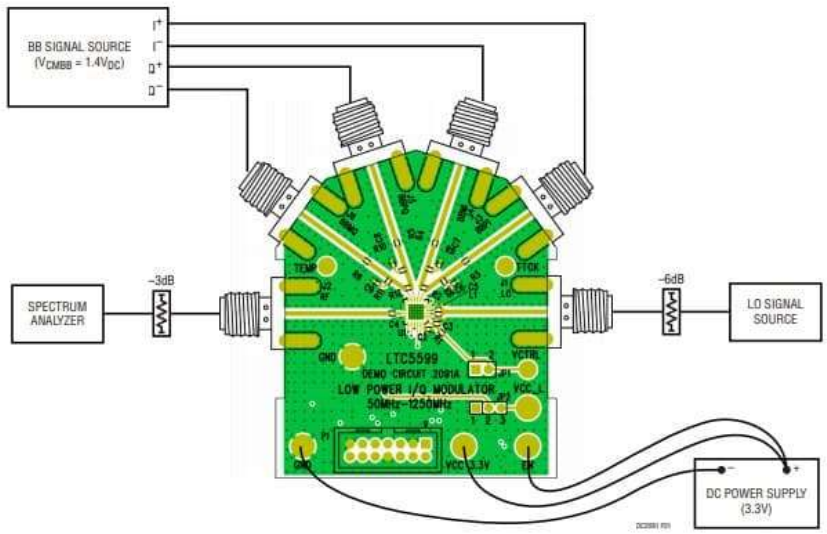

Figure 18. LTC5599 (modulator) connections (from datasheet)

We'd also like to emphasize the output power of this modulator since it may have an impact in the viability of the end application. In particular we've extracted from the electrical characteristics in the datasheet the following values:

SYMBOL	PARAMETER	CONDITIONS	MIN	TYP	MAX	UNITS
P_{OUT} (1)	Absolute Output Power	$1V_{P-P(DIFF)}$ CW Signal, I and Q		-3.5		dBm
P_{OUT} (2)	Absolute Output Power	$1V_{P-P(DIFF)}$ CW Signal, I and Q		-3.7		dBm
P_{OUT} (3)	Absolute Output Power	$1V_{P-P(DIFF)}$ CW Signal, I and Q		-4.9		DBm

1) f_{LO} = 150MHz, f_{RF1} = 147.9MHz, f_{RF2} = 148MHz, Register 0x00 = 0x62
2) f_{LO} = 500MHz, f_{RF1} = 497.9MHz, f_{RF2} = 498MHz, Register 0x00 = 0x2D
3) f_{LO} = 900MHz, f_{RF1} = 897.9MHz, f_{RF2} = 898MHz, Register 0x00 = 0x12

End-to-end validation

To wrap up this experiment, we generated -with the assistance of Libre Space Foundation signal specialists-, a series of flowgraphs and signals to better understand the results of this first trial. Each signal (or group signals if we consider I and Q two different signals) we had the following target values:

- Baudrate (symbols per second) = 9600
- Oversampling (samples per symbol): 8
- Samples per second (per channel): 76800
- Total samples per second: 153600

We previously found that 1.3 Msps per channel seemed to be the empirical limit. This means that the values we've just mentioned (namely baudrate and oversampling) should be small enough to be generated by this system but at the same time significant enough to validate the system in space applications. In other words, **if we wanted to use higher baudrates, as long as the the total sampling rate stays under 1.3 Msps, we should be able to generate a signal with a quality high enough to be decoded**.

Bear in mind that the number of bits per sample doesn't play a role here. However, as we'll shortly explain, our number of bits is limited so it doesn't make sense for us to use more bits per sample in this particular project and setup.

As we've just mentioned, the number of bits per sample is limited in our case and can only be as high as 3 bits (with 8-bits samples) to feed the modulator. This is due to the fact that the modulator requires just 200 mVpp and we have not considered in our initial design a base-band external attenuator to make use of. This fact really limits the performance of the modulator because of the high quantization noise and the low signal-to-noise (SNR) ratio.

Initially we intended to receive and decode a signal by making use of an SDR and GNU Radio. However, the limitations we've just mentioned, are too important to receive a valid signal. Instead, we focused on evaluating the frequency mask, which is not ideal but we see it as a starting point for an improved next experiment.

In our setup we've fed the local oscillator of the LTC5599 with an SDR (Lime SDR Mini in our case) and we used the same SDR to check the resulting signal in frequency. The flowgraph we used (apart from variables, GUI options and other constants) is what's shown in the image below:

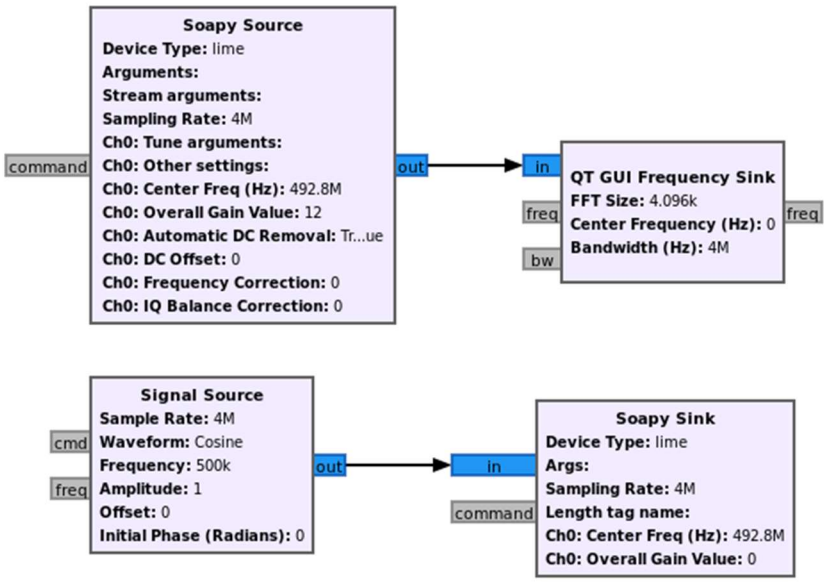

Figure 19. Signal for the local oscillator (sink) and for the receiver (source)

In this case, the signal we were testing with was a 2-FSK. The baseband bandwith of the signal was 20 kHz and so the two FSK peaks are 20 kHz apart. The resulting frequency mask can be seen in the image below:

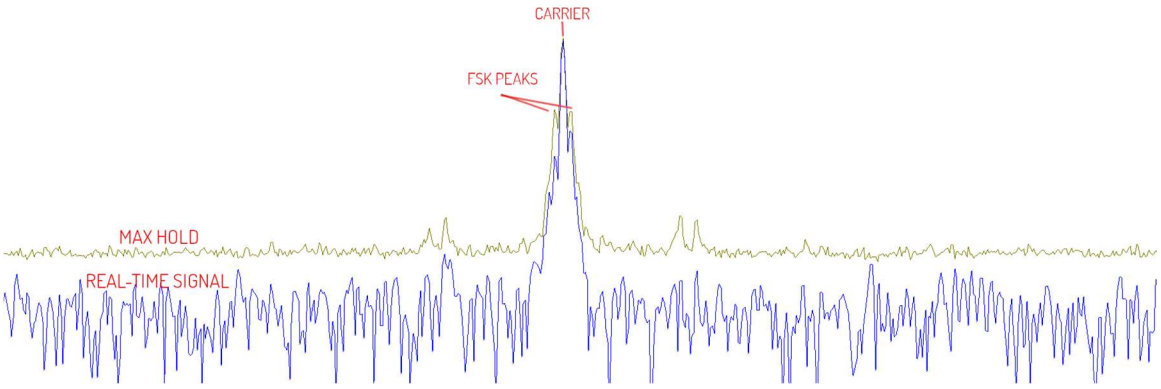

Figure 20. Frequency Mask of a FSK signal

Next steps

- ❑ Try other configurations of the modulator. Several parameters of the LTC5599 can be configured through SPI including the frequency of the local oscillator.
- ❑ Add analog filtering in the output of the DAC for softer signals. Since we had to limit the number of samples used due to the 200 mVpp limit in the modulator input signal, the quantization error was high and the SNR low. By adding some analog filtering in the DAC output we could easily improve the resulting signal and hopefully decode some frames. When doing so, special care should be taken in not to use RF-rated components since our signal is still in base-band.
- ❑ Adjusting the signal amplitude to 200 mVpp have been done manually beforehand. We could do that with analog attenuation in the custom PCB, or even test higher limits if the modulator allows that.
- ❑ Test the SDIO interface for the SD Card. According to the family reference, this interface uses a different bus other than the one SPI2 (see figure below), which we use for reading from the card. The reason we chose SPI over SDIO is because the commands are pretty much standardized and it could be shared more safely according with our chosen License. We could also have used SPI1 but the pins were not available. Overall, if reading from the SD Card is a project requirement, there are several things that could we do to improve it.

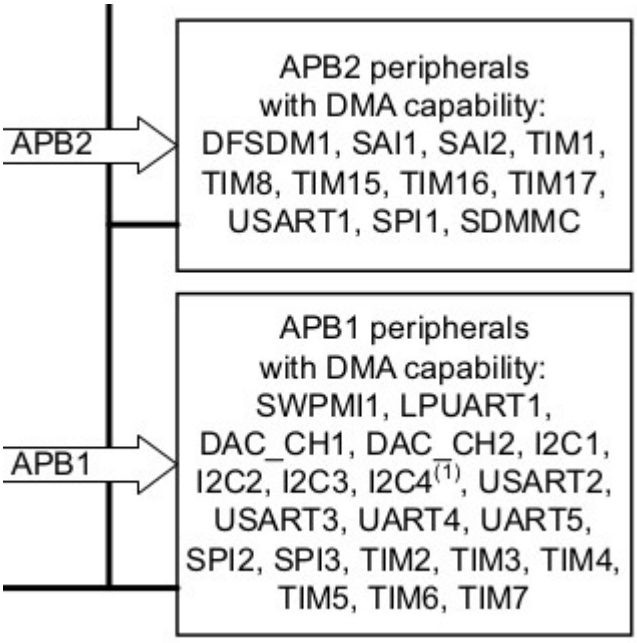

Figure 21. Screenshot of RM0351 corresponding to Figure 31 (DMA block diagram)

- ❑ Generate the signal in the SoC. In this project the signals were generated in a computer (for instance with GNU Radio), then stored in a micro SD card and finally read from the main microcontroller. If we would like to have an autonomous machine, we would have to generate (encode) the signals in *real time*. Depending on the complexity of the calculations (and the modulation) we might even need a DSP. With this same chip it should also be possible to send data to the FPU and then to the DAC, everything through DMA.
- ❑ Try other architectures based on frontends like the SX1276. This kind of frontends can usually generate BPSK or QPSK signals from a series of SPI/I2S messages (apart from LoRa).