ALS162 Time Signal SDR Receiver for GNU Radio

Henning Maier HENNINGM1R@GMAIL.COM

Abstract

A real-time SDR receiver of the French time signal ALS162 is studied, implemented and tested for GNU Radio. Inspired by the practical observations and the minor but yet appealing challenges encountered, this receiver provides another supportive learning exercise for practical signal processing with GNU Radio. Along with the transmitter and receiver design, the present paper addresses several commonly used signal processing concepts like signal synthesis, FIR and IIR filters, phase modulation, phase drift compensation, basic synchronization, symbol correlation, decision metrics and error detection that are applied in terms of GNU Radio.

1. Introduction

ALS162 is the official French time signal transmitted in the low frequency (LF) band on 162 kHz from Allouis, France. Before 2017, it was formerly known as *TéléDiffusion de France* (TDF) and broadcasted since 1977 alongside the AM radio *France Inter* audio transmission which was discontinued in 2017. The time transmitter still remains on air[1] and is synchronized by a highly precise cesium atomic clock. A survey on the history of the time signal and the Allouis transmitter station is provided in (ACHDR, 1999).

From a technical perspective, the official website of the operator (SYRTE, 2019) of the French legal reference time (SYRTE) provides two introducing papers (Gabry, 1980) and (Dubois, 1986) about the original TDF signal. Aside from that, there are hardly any further detailed official specifications on the ALS162 signal available, but just the very basic base-band phase modulated waveforms representing the binary symbols and the marker of a new minute. However, the real-life signal also contains several other uniquely recognizable additional waveforms that clearly belong to the intended signal. Those waveforms are not publicly available, but seem to be covered in the non-public standard NF C90-002 (AFNOR, 1988).

Otherwise, there are several technical and academic websites, e.g., discussing receiver properties of ALS162 in (EA4GMZ, 2017) and (Langley). Some publicly accessible receiver implementations are: A C-Sharp-based demodulator provided in (longview, 2022), a micro-controller-based implementation in (Quillevere, 2020), schematics of different receiver circuits in (DAtelec) and (lafibre.info, 2021), and a plug-in of the open source SDR project SDRAngel (srcejon and f4exb, 2021) for instance.

Interestingly, no fully capable ALS162/TDF implementation has been available for GNU Radio so far. Merely a single unanswered thread on TDF (Young, 2014) in 2014 is mentioned in the GNU Radio newsgroup. Accordingly, an implementation of ALS162 in GNU Radio covering the whole processing chain from transmitter, channel and receiver to decoder is discussed in the present paper.

2. Implementation in GNU Radio

The latest version of the GNU Radio implementations of the ALS162 transmitter, channel, receiver and decoder is provided in the related GitHub repository (henningM1r, 2023).

2.1. Transmitter Design

The transmitter flow contains the following main steps: encoding, signal synthesis, signal integration and modulation to the RF band.

2.1.1. Encoder

The encoding procedure of the current local time and date in France to the 60 binary time bits of ALS162 is provided in (Dubois, 1986) and other sources, so that its detailed discussion is omitted here for brevity. It is assumed that the encoder will continuously provide correct time bits. It is to note that the binary ALS162 time code is closely related to the German DCF77 time code pendant. The main difference of these two codes concerns the first 15 bits.

[1] The transmitter is usually offline each Tuesday morning for a few hours due to regular maintenance work.

Proceedings of the 13th GNU Radio Conference, Copyright 2023 by the author(s).

2.1.2. Signal Synthesis

The ALS162 phase signal conveys two composite codes: 1) The abovementioned binary time code and 2) a circularly repeated bit position code.

The triangle-shaped phase signal carries the previously encoded time bits within the first 8 time-slices of 25 ms each, covering a duration of $8 \cdot 25$ ms $= 200$ ms as depicted in Figure 1. The tick of each second is aligned to the first zero-traversal of a zero- or one symbol, i.e. precisely at 50 ms.

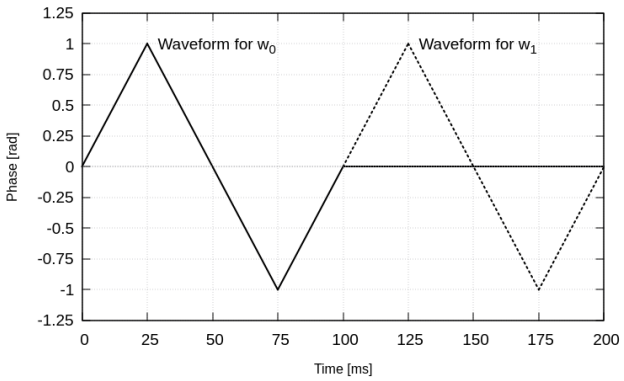

Figure 1. Triangle-shaped phase waveforms of time bits 0 (solid lines - 1 full wave) and 1 (dotted lines - 2 full waves)

From this point on the derivative - or slope - of these phase signals is used for the following description, since it appears easier to handle than the original phase signal itself. The derivative is represented by one of the following distinct signal levels $+1$, 0 and -1 for each time-slice. Hence, the above two time codewords for 0, 1 and the minute marker can be expressed as follows:

$$\begin{aligned} \mathbf{w}_0 &= (+1,-1,-1,+1,\ 0,\ 0,\ 0,\ 0), \\ \mathbf{w}_1 &= (+1,-1,-1,+1,+1,-1,-1,+1), \\ \mathbf{w}_{\min} &= (\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0). \end{aligned}$$

By using these three codewords and selectively muting the remaining 800 ms of each second, one could already construct a functioning ALS162 transmission scheme.

However, there are yet further waveform patterns transmitted during the remaining 32 time-slices for $32 \cdot 25$ ms $= 800$ ms complementing an entire second of 40 time-slices. A closer inspection reveals that there are simply 60 circularly recurring codewords transmitted in the course of a minute – one per each second. These codewords are used to locate the positions of the time signal within the minute. It is to note that these 60 bit position codewords had to be reverse-engineered[2] from the received signal. These bit positions are particularly useful for error correction and resynchronization in case some time bits are disturbed during reception. The waveforms of the position code include two further possible derivative signal levels: ± 2.

In order to assemble the ALS162 phase signal waveform, each time bit from the encoder is repeated 40 times to cover the 40 time-slices of a full second. It is applied as a control signal to switch between the three derivative signal waveforms for the zero symbol $\mathbf{w}_0$, the one symbol $\mathbf{w}_1$ and the muted minute break $\mathbf{w}_{\min}$, respectively. The Python-based Synchronized Waveform Selector block delivers the switched output signal continuously. Please note that the closely related GNU Radio Selector block is switched with the asynchronous messaging API instead.

Afterwards, the bit position codewords are added synchronously to the remaining 32 time-slices of the signal, so that each time bit is assigned to its position within the minute. The related flow of this signal synthesis procedure is shown in Figure 2.

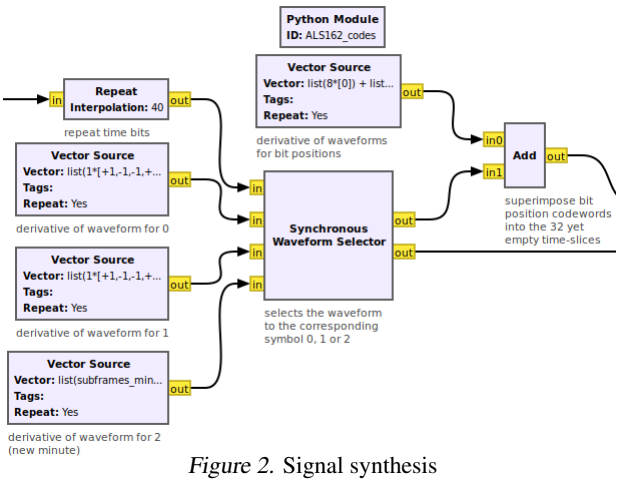

Figure 2. Signal synthesis

2.1.3. Signal Integrator

The composite derivative signal is then filtered by an ideal integrator. The integrator is realized by an IIR filter with the feed-forward coefficients $\mathbf{c}_{\mathrm{FF}} = (1, 0)$ and the feedback coefficients $\mathbf{c}_{\mathrm{FB}} = (0, 1)$. Its output signal is scaled by s/r with $s = 40$ time-slices and sample rate $r = 192$ kHz. The resulting phase angle is changing gradually between $+1$ and -1 radian in accordance to the desired real-valued base-band phase signal depicted in Figure 1.

2.1.4. RF Modulation

Next, the composite complex-valued signal is generated from the phase signal with a constant magnitude of 1 in the time domain. In the last step of the transmitter, the complex

[2] These bit position waveforms are obviously not encrypted. In (AFNOR, 1988), the publicly viewable table of contents even substantiates this with a chapter called "positions de la seconde".

base band signal is modulated to the RF band by shifting it with a complex cosine waveform to the carrier frequency of 162 kHz. Now it is transmitted over the wireless channel.

2.2. Wireless Channel Model

The channel response $Y(t)$ is represented by an additive white Gaussian noise (AWGN) channel with weak attenuation and occasional impulsive interference:

$$Y(t) = A(t) \cdot X(t) + I(t) + N(t).$$

The previous RF signal of the ALS162 transmitter is termed by $X(t)$. It is subject to several impairments. The multiplicative attenuation $A(t)$ imposes very slow Rayleigh distributed fading as observed from the real-life signal. Its coherence time is much longer than the symbol duration. Further attenuation effects from the long distance to the transmitter are not taken into account here. Deep fades occurred very rarely and only lasted a few minutes. The default power of the noise $N(t)$ is set to 0.25 to roughly match the noise power observed in the actual signal with the SDR hardware setup.

Impulsive interference $I(t)$ is occasionally observed in the practically received signals as shown in Figure 3.

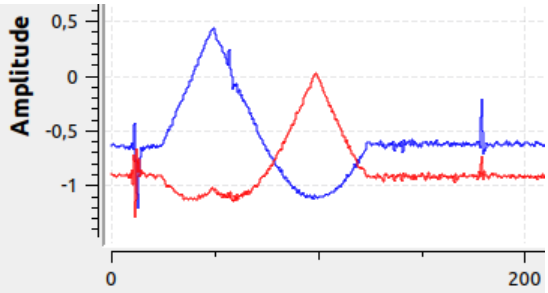

Figure 3. Exemplary impulsive interference as observed in the real-life complex base band signal - there are two stronger glitches at 10 ms and at 180 ms

Interference is only modeled by very basic properties to roughly match the observations and to induce sufficiently realistic occasional bit errors in the channel simulation. The interference waveforms seen in the base band time-domain are approximated with narrow sinc-functions that are independently scaled with a Rayleigh distributed amplitude factor for each occurrence. The relative sparsity of the interfering impulses is generated by a very dense ratio of sampled values from a uniform random source with a wide range of values. It is out of scope for this paper to investigate specific details about statistics and shapes of more realistic interference signals on ALS162.

2.3. Receiver Design

The receiver performs the following main steps: Demodulation, phase drift compensation, averaging signal differentiation, symbol level detection/quantization, synchronized time-sliced symbol detection, symbol correlation and decoding with error detection and error correction.

2.3.1. Demodulation

The received complex RF signal is firstly filtered by a Frequency XLating FIR Filter block with cut-off frequency 1 kHz and transition bandwidth 250 Hz in order to be shifted down to the complex base band. The resulting signal is further down-sampled with a decimation factor of 12. This decimation does not cause any harmful aliasing effects but rather reduces the computational effort for the receiver.

2.3.2. Phase Drift Compensation

Phase modulated signaling inherently implies a phase drift effect at the receiver. If no phase compensation were applied, discontinuous phase swaps would regularly be observed at $\pm\pi$ within very few minutes already. These phase swaps would severely disrupt the signal detection process.

As a countermeasure, a weighted long-term moving average value of the signal is computed over 15000 samples and subtracted from the phase signal at a frequency of 24 kHz to compensate this phase drift. The flow for estimating the average phase drift value is shown in Figure 4.

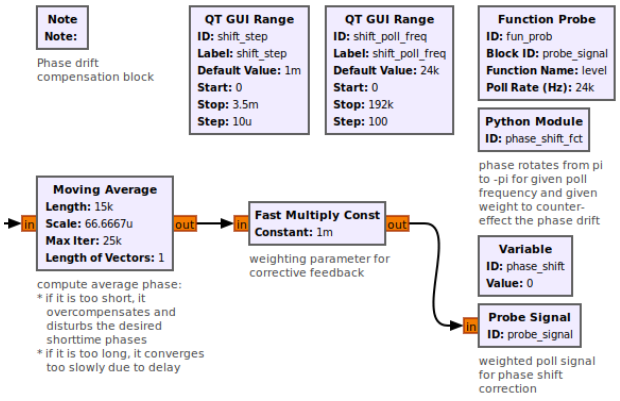

Figure 4. Phase drift compensation - average drift estimation

The Python module *phase_shift_fct* repetitively adjusts the weighted corrective phase. The flow for subtracting the corrective average phase from the phase of the complex baseband signal is shown in Figure 5.

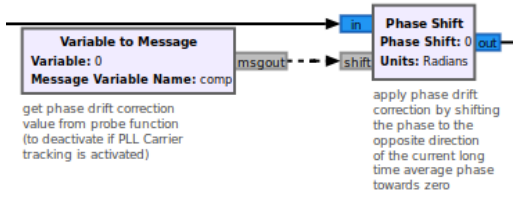

Figure 5. Phase drift compensation - subtracting average phase

This naïve approach is already sufficient for achieving phase stability given the weighting parameters are chosen adequately. No elaborate phase-locked-loop (PLL) is necessary here. Nonetheless, it is alternatively possible to apply the PLL Carrier Tracking block with a very low loop bandwidth of approximately $800 \cdot 10^{-9}$ Hz. But the PLL might easily impose some minor distortions to the signal. Note that even identical phase compensation parameters (for both alternatives) behave differently on various machines - but these parameters can be adapted manually.

The phase drift compensation converges quite fast after a few seconds already. The performance of this compensation is acceptable for our use case. No detrimental distortion is observed on the compensated signal. Afterwards the required real-valued phase signal is extracted from the complex base band signal for further processing by an averaging differentiator.

2.3.3. Averaging Differentiator

To reverse the effect of the ideal integrator presented in paragraph 2.1.3, a (non-ideal) differentiator is used to restore the derivative symbol levels for each time-slice. The FIR filter coefficients $\mathbf{c}_{\text{diff}}$ of the differentiator are:

$$\mathbf{c}_{\text{diff}} = (\underbrace{1, \cdots, 1}_{N_1}, \underbrace{-1, \cdots, -1}_{N_1})$$

A useful choice is $N_1 = 55$ to obtain distinguishable and fairly smooth signal levels. The filtered phase signal resembles a scaled derivative phase signal with five distinct and roughly equally-spaced symbol levels. An ideal differentiator with coefficients $\mathbf{c}_{\text{diff}} = (1, -1)$ would not expose the desired slope sufficiently enough from one sample to the next as it would be hardly stronger than noise.

Two short moving average filters are applied before and after differentiation to further smooth out some peaks caused by noise and interference. The non-infinite gradient of the differentiator and the moving average filters impose slight delays. The differentiator is also quite sensitive to stronger interference. The flow is shown in Figure 6.

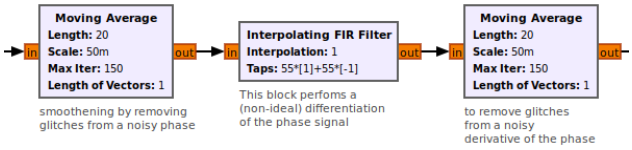

Figure 6. Averaging differentiator

2.3.4. Symbol Level Detection/Quantization

The resulting derivative phase signal is now amplified with a manually adjustable gain. The gain must be set so that the noisy symbol levels are observed at approximately $4000, 2000, 0, -2000$ and -4000. Four decision thresholds are located in between at $+3000$, $+1000$, -1000 and -3000, respectively. The threshold blocks and the logical gates shown in Figure 7 are used to map the input signal levels to one of the five different symbols $+2, +1, 0, -1, -2$. The logical gates ensure that only one output is active while the other four outputs are simultaneously muted.

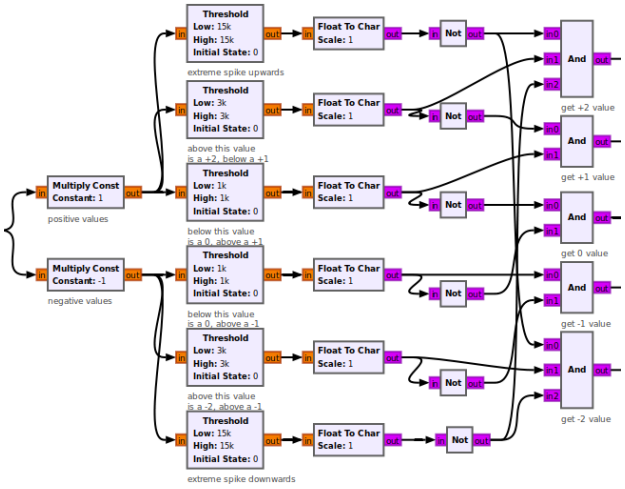

Figure 7. Symbol level detection

2.3.5. Time-Sliced Symbol Detector Block

A Python-based implementation of the symbol detector block is fed with all five output signals from the previous symbol level detection flow. To detect a symbol for each 25 ms time-slice, the detector separately counts the number of continuous and uninterrupted non-zero samples for each current signal level. After the required number of samples either fulfills the time-slice duration of 25 ms or otherwise if the level signal returns to zero, it assigns the current symbol to the corresponding time-slice. At that instant, all sample counters are reset to zero. Such a reset also serves as a basic synchronization of the time-slices. If the number of samples is yet slightly too low, it can still be accepted within a certain tolerance threshold. The decision tolerances are also built-in to ignore briefly occurring zero values while a steeper signal slope of ± 2 from ∓ 1 to ± 1 will pass through zero for instance. The time-sliced symbol detector and its five inputs are shown in Figure 8.

After successful detection, the symbol detector inserts the symbols for the current time-slice into a finite message queue. This is done in order to avoid symbol loss caused by the asynchronous GNU Radio messaging API. As soon as the queue is full, it will send out all its stored elements. This queuing actually entails slight delays within the order of a few milliseconds depending on the configured length of the queue. For presentation and debugging purposes, the symbol detector block also inserts message tags for each detected symbol level into a signal stream that is syn-

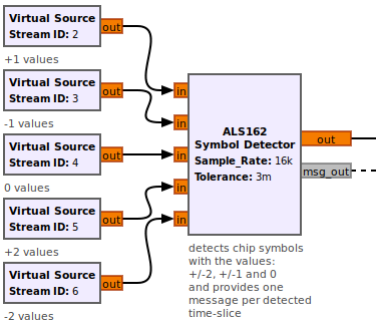

Figure 8. Symbol detector

chronous to the phase signal as depicted in 9. The slight delays of the tags result from the delays imposed by the non-ideal differentiator and the moving average filters.

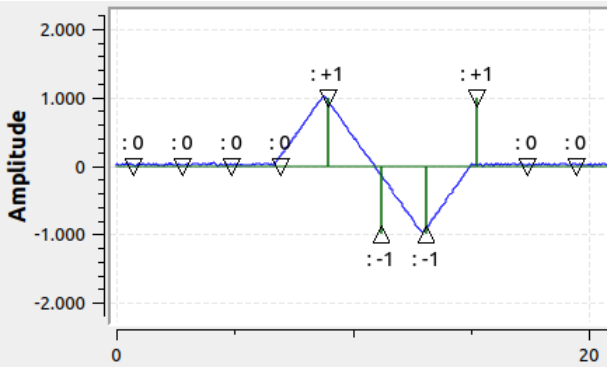

Figure 9. Example of a signal with detected derivative signal levels per time-slice – in this case it is a part of the codeword $\mathbf{w}_0$

2.3.6. Symbol Correlation Block

Next, we need to map the previously produced message stream of symbols per time-slice to the actual codewords of both the time code and the position code, respectively. A Python-based implementation of the symbol correlation block analyzes a current sliding window of at least 79 symbols, i.e. for almost 2 full seconds. It compares the sliding window with all 60 bit position codewords of length 32 until the closest codeword and its relative offset within the window is found. The Euclidean distance $d_{i,j}$ between two codewords $\mathbf{w}_i$ and $\mathbf{w}_j$ is used as a metric for non-binary signals to decide on the detection of a codeword at the presence of errors. As soon as an adequate bit position has been detected with a threshold value of $d_{i,j} \leq d_{\min} = 3$, a single human-readable symbol between 0 and 60 is stored for the position. The detection of the bit position is completed.

The codebook carrying all 60 codewords for the above comparison is circularly rotated by using the latest received bit position symbol. This way, the number of iterations per codeword is significantly reduced for each successive sliding window. In the error-free case, this detection process will only have to perform one iteration instead of up to 60 for the next second.

Furthermore, the provided offset of the first symbol within the sliding window of the bit position is also used to locate the yet undetected time symbol. The time symbol covers 8 time-slices just before the position symbol. In analogy to the detection of the bit positions, the time symbol is compared to the three codewords $\mathbf{w}_0$, $\mathbf{w}_1$ and $\mathbf{w}_{\min}$. At this point, the detected time codeword is also stored.

Then, the sliding window is truncated, so that the two detected codewords are removed from the front of our symbol stream and ready for successive codeword detections. The symbol correlation block outputs the stored pair of the time symbol and the position via the ZMQ PUSH Message Sink for further processing by the external decoder.

An evaluation of the pair-wise Euclidean distances between all 60 bit position codewords is depicted in Figure 10. The plot shows that the bit position codewords $\mathbf{w}_i$ and $\mathbf{w}_j$ for $i \neq j$ and $i, j \in \{00, \cdots, 59\}$ have a minimal Euclidean distance $d_{i,j}$ of at least $d_{\min} \approx 3.46$ and a maximal distance of $d_{\max} \approx 10.59$.

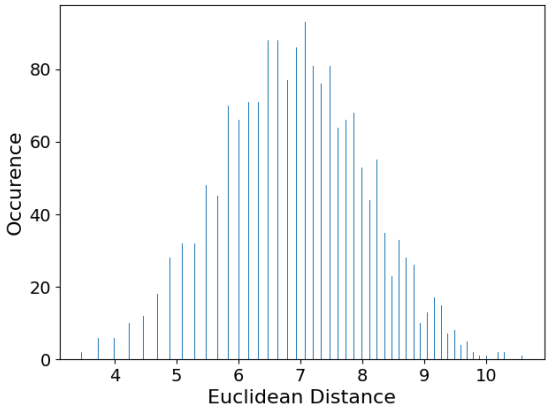

Figure 10. Pair-wise distances of all 60 position codewords

The Euclidean distance between the two time codewords $\mathbf{w}_0$ and $\mathbf{w}_1$ yields 2. The fairly smooth shape of distances in the plot also reveals that a well-structured code is used.

2.3.7. ALS162 Decoder

The last element in the receiving chain is the ALS162 decoder. The decoder is an independent Python-based script listening to transmitted ZMQ-messages. It synchronizes to the current bit position and gathers the time symbols until the next minute marker is received. Then it decodes the symbols in accordance to the binary ALS162 code (Dubois, 1986) and provides a time-and-date report. It is able to include the knowledge of error symbols and their positions so that it can still decode the remaining clean symbols of an erroneous stream of 60 bits. Since some bit errors can be located exactly within the stream by using the bit positions, parity bits are exploited to correct single errors.

Upon starting the decoder, the very first received symbols

are usually lost, due to the yet incomplete phase compensation and a few missing time-slices. This last step completes the overall receiver flow.

3. Practical SDR Receiver Setup

The simulation performance of the transmitter, channel and receiver is sufficient to work on a single machine. It is useful to selectively test receiver and decoder in a controlled environment and should work well in both Linux and Windows operating systems on contemporary computers.

3.1. Hardware Setup

For the practical evaluation of the receiver implementation, an SDR with low frequency reception capability within roughly 1 kHz to 1 MHz is required to observe and sample the original RF signal. Moreover, a loop antenna was used. In the test setup, the loop antenna was mounted indoor at a window and mechanically fixed to avoid movement. The sample-rate of the SDR is 192 kHz. The reception quality within less than 1000 km distance to Allouis was sufficient to obtain a fully clear signal most of the time. Only some deep fades were observed in rare occasions. However, impulsive interference signals of unknown origin occurred more often and disrupted single bits. The decoder ran reliably for several hours without any interruptions.

4. Extensions

Quite a few interesting extensions could be explored, e.g.:

- Resilience to errors by exploiting information from previous minutes to replace predictably wrong values
- Further exploiting the received Hamming distance information in the time code for error correction
- Testing other SDRs and (multiple) antennas
- Providing decoded time to external clocks
- Further improving synchronization to reference time
- Simulation and handling of leap seconds

5. Conclusions

A working GNU Radio implementation for an SDR receiver of the ALS162 time signal was presented. It serves the basic function to decode the real-life ALS162 time signal from the transmitter in Allouis within the range of approximately 1000 km covering France and even parts of neighboring countries. Several observed detrimental effects of the channel, e.g. from phase shifting, fading and impulsive interference have been included. Please note that that this receiver design does not claim to be theoretically optimal in terms of precise synchronization, SNR and resilience to interference.

References

ACHDR. Émetteurs radiodiffusion et télévision - Allouis, 1999. URL https://tvignaud.pagesperso-orange.fr/am/allouis/fr-allouis.htm.

AFNOR. NF C90-002. Technical report, October 1988.

DAtelec. Signaux Horaires. URL http://www.datelec.fr/signaux_horaires/p0.htm.

Dubois, B. L'émetteur francais de fréquence étalon et de signaux de temps codé. *Bulletin BNM*, (63-64):120–128, January 1986.

EA4GMZ. La señal horaria TDF en 162 kHz, January 2017. URL https://www.radiotecnia.es/la-senal-horaria-tdf-en-162-khz/.

Gabry, A. Diffusion de l'heure par codage de la phase d'un émetteur de radiodiffusion à modulation d'amplitude. *L'onde électrique*, 60(10):51–54, 1980.

henningM1r. gr_ALS162_Receiver, April 2023. URL https://github.com/henningM1r/gr_ALS162_Receiver.

lafibre.info. Construisez un Récepteur de Fréquence Étalon et de Signaux Horaires sur France Inter 162 khz, June 2021. URL https://lafibre.info/images/sat/200103_signal_horaire_france_inter.pdf.

Langley, R. Time and Standard Frequency Station TDF (France). URL https://www.eecis.udel.edu/~mills/ntp/tdf.html.

longview. TDF_Test, April 2022. URL https://github.com/longview/TDF_Test.

Quillevere, H. Récepteur Horaire France Inter, February 2020. URL https://www.rvq.fr/tech/fi.php.

srcejon and f4exb. SDRAngel, June 2021. URL https://github.com/f4exb/sdrangel/pull/934.

SYRTE. French legal time. URL https://heurelegalefrancaise.fr/.

SYRTE. Mise à disposition du temps légal par le signal ALS162, May 2019. URL https://syrte.obspm.fr/spip/services/ref-temps/article/mise-a-disposition-du-temps-legal-par-le-signal-als162.

Young, I. Decoding TDF's Phase Modulated Time Signal, May 2014. URL https://lists.gnu.org/archive/html/discuss-gnuradio/2014-05/msg00513.html.