COE3DY4 Project
Real-time SDR for mono/stereo FM and RDS

Objective

The main objective of the project is to navigate a complex (industry-level) specification and understand (and rise to) the challenges that must be addressed for a real-time implementation of a computing system that operates in a form factor-constrained environment.

Preparation

- Revise the material from labs 1 to 3

Project Overview

The 3DY4 project leverages front-end radio-frequency (RF) hardware, like RF dongles [1] based on the Realtek RTL2832U chipset [2], and single-board computers, like Raspberry Pi 4 [3], for the implementation of a software-defined radio (SDR) system for real-time reception of frequency-modulated (FM) mono/stereo audio [4], as well as the reception of digital data sent through FM broadcast using the radio data system (RDS) protocol [5] [1]. Due to the affordability of RF hardware, a broad number of open source SDR projects have emerged over the past decade or so, like GNU Radio [8] or Gqrx [9], only to name a couple. The goal of the 3DY4 project is not to duplicate this type of open-source initiatives. Rather, the real-time SDR project in this course is used primarily as a vehicle for consolidating the knowledge acquired so far and to learn how to navigate from the first principles the complex interrelationships between seemingly disparate, yet practically related, topics from electrical and computer engineering.

Each FM channel occupies 200 KHz of the FM band and it is symmetric around its center frequency that can range from 88.1 MHz to 107.9 MHz in Canada (note, not all the FM channels are used in the same geographic region, hence it is common in some areas that two FM stations will broadcast 400 KHz, or even a higher multiple of 200 KHz, apart). When looking at the positive frequencies of an FM channel (0 to 100 KHz) there are three different sub-channels within each FM channel that are of interest to this project. The mono sub-channel is from 0 to 15 KHz; the stereo sub-channel is from 23 to 53 KHz and the RDS sub-channel is from 54 to 60 KHz. Above 60 KHz, there *might* be other sub-channels that use SCA subcarriers (where SCA stands for Subsidiary Communications Authorization), which allow FM stations to broadcast additional services. SCA sub-carriers are not standardized and the focus of the 3DY4 project will be on the mono audio, stereo audio and RBDS clearly labeled in Figure 1. Note, in between the mono and stereo sub-channels there is a 19 KHz stereo pilot tone used to synchronize the stereo sub-carrier to the second harmonic of this tone, i.e., 38 KHz. This mechanism was needed to broadcast stereo audio when the receivers were built with analog technologies. While the RDS sub-channel is expected to be centered around

[1]Since RDS was initially developed in Europe [6], the official name in the North America is Radio Broadcast Data System (RBDS) [7]. With the exception of some digital codes specific to geographic locations, the two standards are virtually identical and therefore the terms RDS and RBDS will be used interchangeably.

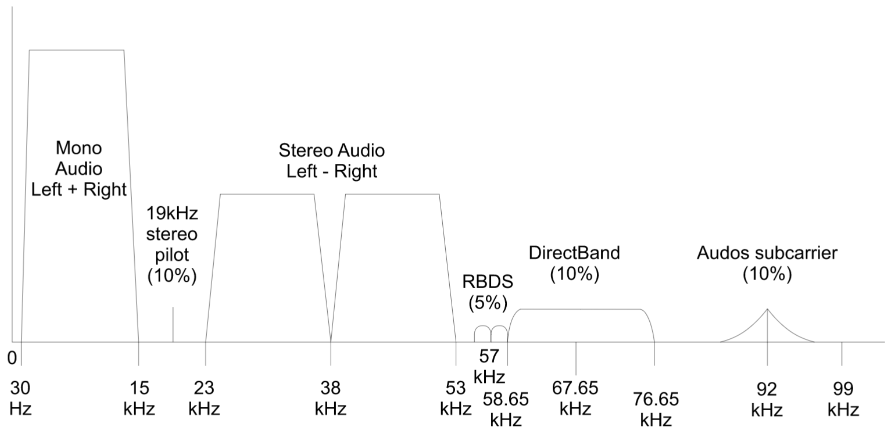


Figure 1: Spectrum of an FM channel [4]

the third harmonic of the pilot tone, i.e., 57 KHz, some FM stations do not necessarily lock the RDS sub-carrier to the pilot tone; hence the carrier recovery and downconversion for RDS will follow an approach that is different from the carrier recovery and downconversion for stereo audio; nonetheless, it is worth noting that the approach used for RDS is resemblant to the approach followed by many modern digital communication standards.

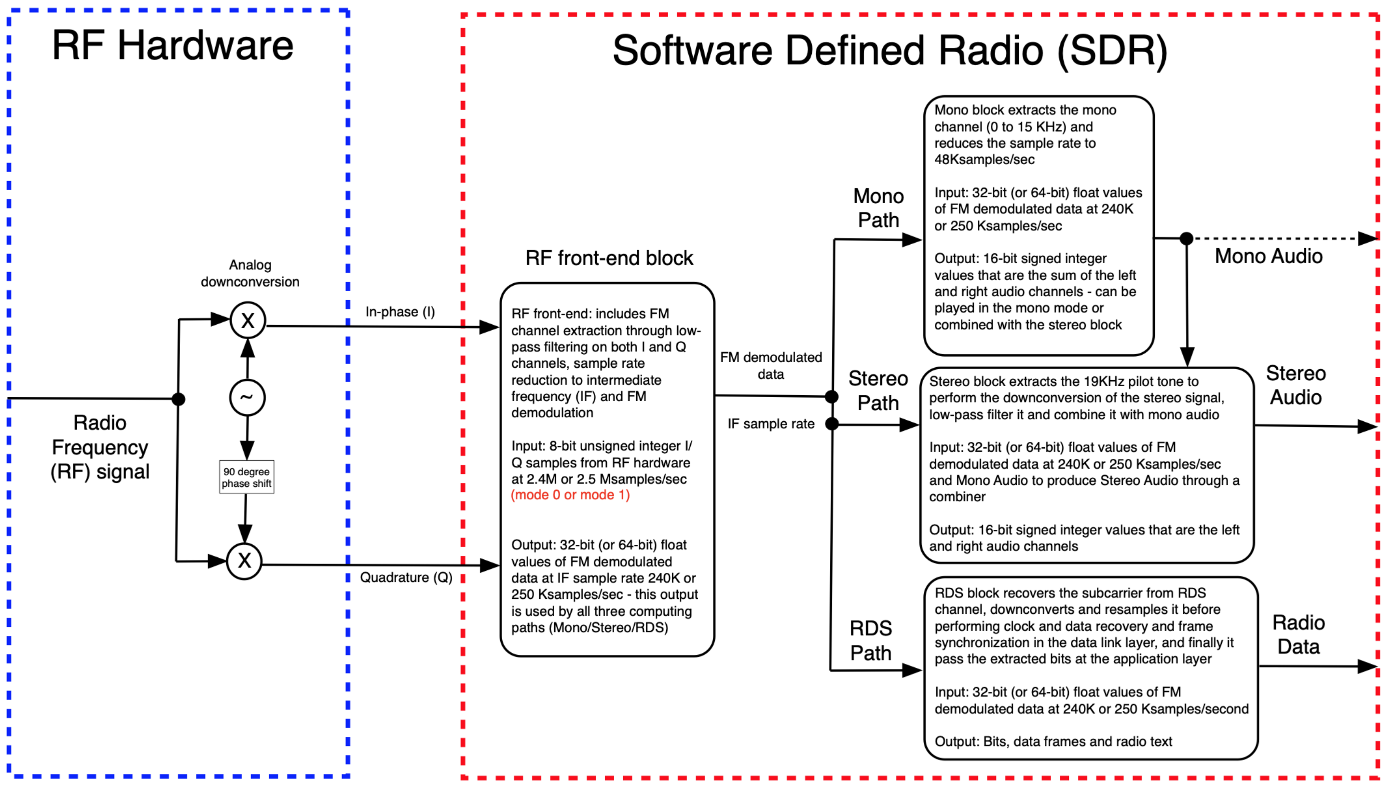
While the data that feeds the SDR system implemented in this project is driven by FM stations, in between the last lab, class discussion and this project document, all the essential FM-related information relevant to the project has been provided. For further information on FM technology, you are referred to academic textbooks on the topic or industry white papers, such as [10, 11, 12]. As a final note, because some terms can be implied from the context, we will refer to sub-channels of an FM channel also as channels, i.e., the mono channel (0 to 15 KHz), the stereo channel (23 to 53 KHz) and the RDS channel (54 to 60 KHz).

The big picture of the 3DY4 project is capture in Figure 2 and it is summarized below:

- the RF hardware is concerned with the acquisition of the RF signal through an antenna and translating it into the digital domain by producing an 8-bit sample for the in-phase (I) component and another 8-bit sample for the quadrature (Q) component of the RF signal (through a process of analog downconversion via mixing and filtering using a local oscillator). Since the software interface to the RF hardware has been developed by a third party [2], the key point that must be accounted for is that the I/Q data is transferred to the host in an interleaved format as I/Q 8-bit sample pairs, i.e., an 8-bit I sample followed by the corresponding 8-bit Q sample, followed by the next I/Q pair and so on.

- The RF front-end block from the SDR system extracts the FM channel through low-pass filtering followed by decimation to intermediate frequency (IF) and FM demodulation. It is conceptually the same as the RF front-end for the last lab, however for the project there will be two modes of operation: in **mode 0** the input sample rate is **2.4 MSamples/sec** and in *mode 1* the sample rate is *2.5 MSamples/sec*. Considering that the decimation scale factor in the RF front-end block is equal to ten, the FM demodulated signal at the IF sample rate will be 240 KSamples/sec for mode 0 and 250 KSamples/sec for mode 1. **By default, your SDR software should operate in mode 0**. *If a command line argument is provided and it is equal to 1 then your SDR software should operate in mode 1.*

- The mono block extracts the mono audio channel (0 to 15 KHz) and it reduces the sample rate to 48 KSamples/sec. The input is FM demodulated data that should be represented in



RF front-end: includes FM channel extraction through low-pass filtering on both I and Q channels, sample rate reduction to intermediate frequency (IF) and FM demodulation

Input: 8-bit unsigned integer I/Q samples from RF hardware at 2.4M or 2.5 Msamples/sec (mode 0 or mode 1)

Output: 32-bit (or 64-bit) float values of FM demodulated data at IF sample rate 240K or 250 Ksamples/sec - this output is used by all three computing paths (Mono/Stereo/RDS)

Mono block extracts the mono channel (0 to 15 KHz) and reduces the sample rate to 48Ksamples/sec

Input: 32-bit (or 64-bit) float values of FM demodulated data at 240K or 250 Ksamples/sec

Output: 16-bit signed integer values that are the sum of the left and right audio channels - can be played in the mono mode or combined with the stereo block

Stereo block extracts the 19KHz pilot tone to perform the downconversion of the stereo signal, low-pass filter it and combine it with mono audio

Input: 32-bit (or 64-bit) float values of FM demodulated data at 240K or 250 Ksamples/sec and Mono Audio to produce Stereo Audio through a combiner

Output: 16-bit signed integer values that are the left and right audio channels

RDS block recovers the subcarrier from RDS channel, downconverts and resamples it before performing clock and data recovery and frame synchronization in the data link layer, and finally it pass the extracted bits at the application layer

Input: 32-bit (or 64-bit) float values of FM demodulated data at 240K or 250 Ksamples/second

Output: Bits, data frames and radio text

Figure 2: Project Overview

floating point format with an IF sample rate at either 240 KSamples/sec or 250 Ksamples/sec depending on the mode of operation. The output should be in 16-bit signed integer format that can be transferred to an audio player. Note, if only mono audio is implemented the 16-bit values are the sum of the left and right audio channels. Since the intermediate data is in floating-point format, it is up to the SDR software developer to choose an internal scale factor for the output audio data in order to adequately use the dynamic range offered by 16-bit signed integer format.

- The stereo block extracts the 19KHz pilot tone through band-pass filtering to perform the downconversion of the stereo channel, which is subsequently filtered and combined with the mono audio data to produce the left and right audio channels. This can be achieved by averaging the sum and difference of the data from the mono and stereo channels because the mono channel contains the left **plus** right audio channels, whereas the stereo channel contains the the left **minus** right audio channels. The input and output formats are the same as for the mono path.

- The RDS block first recovers the subcarrier from the RDS channel, it downconverts it through a digital communication approach and it resamples it before performing clock and data recovery. Once the bitstream is generated, the frame synchronization is done in the data link layer before the extracted information words are passed to the RDS application layer. The input to this block is the same as for the mono and stereo paths, however the output is now in terms of bits and words of radio data.

The remainder of this document will provide the motivation and objectives of each processing block in the SDR receiver for FM mono/stereo/RDS and its most critical details. *The more specific details, such as the design methodology, including implementation choices, trade-offs, ...,* *will be discussed during classes and feedback on your unique approach (and thought process in general) will be provided during project meetings* [2]. Before proceeding with the description of each processing block for this project, below is a summary of the most important take-aways from the lab work:

- Filters are core building blocks in any digital signal processing (DSP) application and they can be implemented using the convolution of the filter's impulse response with the input sequence (discrete time sequence of digital samples).
- The impulse response coefficients can be derived using the *sinc* function, by relying on the Fourier transform relationship between the rectangular window, used to gate the pass band in the frequency domain, and the *sinc* function in the time domain.
- In practice, idealized (or brick-wall) filters cannot be implemented, hence finite-impulse response (FIR) filters that guarantee linear phase are commonly used; for a given sample rate, the number of FIR taps (input sample delay/coefficient pairs) will have an impact on both the stop band attenuation and the width of the transition band.
- The filter coefficients depend on the number of FIR taps, the cutoff frequency (or frequencies for pass-band) **and** the sample rate.
- For most practical applications, including SDRs, filtering cannot be performed in a single-pass over the entire sequence of input samples. The input sequence is processed in blocks because of the need to avoid both an excessive latency for data acquisition, as well as large memory requirements. This adds to the implementation challenge, nonetheless this is a common issue, which needs be tackled with the same type of mindset across a broad spectrum of real-time computing systems.
- A collection of digital filters and other types of signal processing blocks are connected together in a *signal-flow graph*, which is an abstract representation used to model DSP applications. Depending on the software language and its libraries, as well as the underlying hardware platform, the execution time for the same signal-flow graph with the same parameters can vary substantially. The signal-flow graphs should be modeled first in a high-level language. e.g., `Python` using its scientific libraries `SciPy/NumPy`, and only when the functional requirements have been met they are to be re-implemented in another language, e.g., `C/C++`, in order to facilitate real-time behavior.
- The methods used to check the correctness of an implementation while bringing-up a complex system can vary depending on the tasks to be performed and issues at hand; for example, when dealing with the interface to the physical world, there is no reference data in terms of exact values to be matched and therefore one has to rely on visual inspection of power spectra to pass judgment if the code has been implemented correctly; conversely, when re-factoring the `C++` code in order to make the implementation feasible for real-time execution, i.e., converting convolution from single-pass to block processing, bit equivalence is expected in order to guarantee that no implementation artifacts have been introduced during re-factoring.

As a final note, for the machine code compiled from `C++` to work in real-time, special considerations for optimizing the source code will be needed and some of the generic ones will be discussed during lectures and tutorials.

[2]For general details concerning Raspberry Pi, RTL-SDR, FM, RDS, ... that are not specific to this course you should refer to third-party literature.

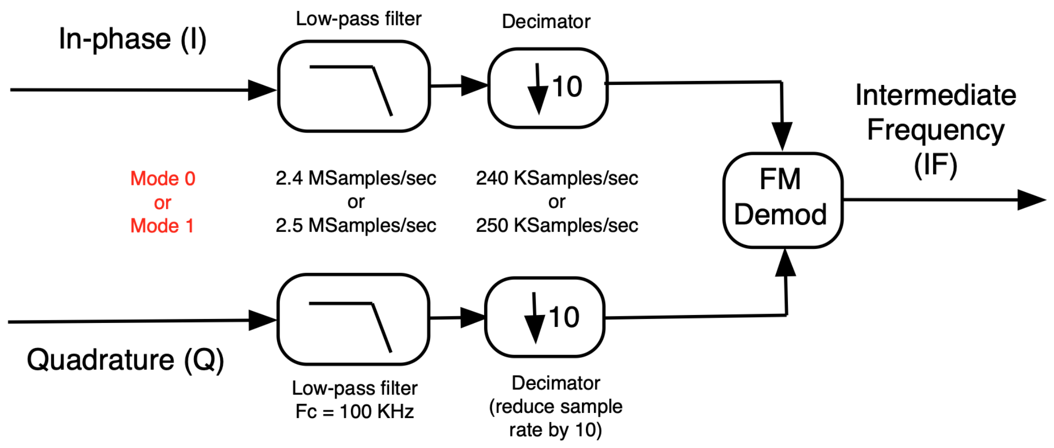

In-phase (I)
Low-pass filter
Decimator
10
Intermediate Frequency (IF)
Mode 0 or Mode 1
2.4 MSamples/sec or 2.5 MSamples/sec
240 KSamples/sec or 250 KSamples/sec
FM Demod
Quadrature (Q)
10
Low-pass filter Fc = 100 KHz
Decimator (reduce sample rate by 10)

Figure 3: RF Front-End Processing

RF Front-End Processing

The signal-flow graph for the RF front-end block is illustrated in Figure 3. Take note of the following points:

- Conceptually the signal flow graph is identical to the front-end signal-flow graph from the last lab, with one critical difference because now there are two modes of operation for the SDR system: in mode 0 the input sample rate, i.e., RF sample rate, is 2.4 MSamples/sec; in mode 1 the RF sample rate is 2.5 Msamples/sec. While this is inconsequential to the modeling of the RF front-end block (because the decimation scale factor is kept at 10 regardless of the mode of operation) it does have a ripple effect on the re-sampling to be done in the other processing blocks. It is worth noting nevertheless that, in any real-time implementation, any filter that is followed by a decimator $\downarrow$ should compute **only** the samples that will be kept **after** decimation.

- The I/Q samples that will be fed by the RTL-SDR driver [2] to your software will be represented as 8-bit unsigned integers (`unsigned char` in `C++`). If you are to reuse the software model for this block from the last lab, you will need to normalize these 8-bit unsigned integer values to the -1 to +1 range of real numbers. Note, the code from the labs assumes the `float` data type for real numbers in `C++`, which represents real values on 32-bits. While both the dynamic range and the precision offered by the `float` data type is normally sufficient, depending on your design approach you might consider using the `double` data type, which represents real numbers on 64-bits. In the context of this SDR application for this project the advantage of `double` over `float` is extra precision, which may or may not be needed because it depends on your unique parameters in the signal-flow graph and your project-specific implementation choices; however this would come at the expense of doubling the memory requirements for all the blocks represented as real numbers and, depending on your unique parameters and project-specific choices, there is a possibility of a small performance overhead, which in most cases is expected to be tolerable.

- As a side note, it is critical to emphasize that the only constraints on the data types are the 8-bit unsigned integer values for the inputs to your SDR software (I/Q samples) and 16-bit signed integer values for the outputs (audio samples) to be fed to an audio player. Note, the output of the RDS path is also clearly defined. However,the internal data representation is at the discretion of each project group and the above discussion concerning normalization to the -1 to +1 real range or `float` vs `double` is *a suggestion rather than a requirement.*

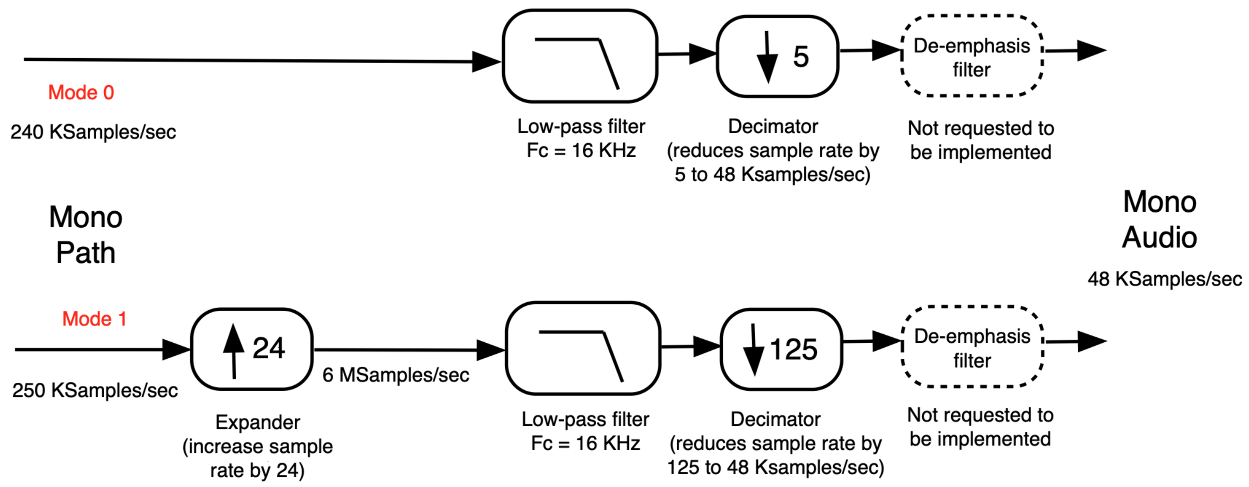


Figure 4: Mono Processing

Mono Processing

The signal-flow graph for mono processing is shown in Figure 4. Take note of the following points:

- Because the background noise in the very high frequency (VHF) range of electromagnetic waves can affect frequencies toward the higher range of the audio spectrum, the FM broadcasting stations pre-emphasize, i.e., boost, the higher frequency components during transmission. Consequently, receivers are commonly equipped with a de-emphasis filter in the final stage of FM audio decoding. This de-emphasis filter is **ignored** in this project because its absence is hardly noticeable from the user perception perspective and the available time during the project phase is deemed to be better utilized to develop the understanding for other important implementation techniques, like multi-threaded software, in greater depth.

- It is critical to notice that depending on the mode of operation there will be two different types of digital filters and sample rate conversions. In mode 0, the RF sample rate is 2.4 MSamples/sec, the IF sample rate (input to the mono processing) is 240 KSamples/sec and the audio sample rate (output from the mono processing) is 48 KSamples/sec. This is essentially the same signal-flow graph from the last lab for the mono channel extraction, filtering and downsampling to the audio sample rate.

- In mode 1, the RF sample rate is 2.5 MSamples/sec, the IF sample rate is 250 KSamples/sec and the audio sample rate is 48 KSamples/sec. The fundamental difference from mode 0 is that there is no integer scale factor that can be used to downsample the IF data to the audio data. Rather, a fractional resampler, which is an upsampler followed by a downsampler, is needed. Assuming an integer upscale factor of U for upsampling, the first step is to implement an expander $\uparrow$ where $U-1$ zeros are padded in between any two input samples. This will produce a sequence at the output of the expander whose sample rate is $U \times IF$, which is U times larger than the sample rate at the input of the resampler (IF is 250 KSamples/sec in mode 1). The decimator $\downarrow$ uses an integer downscale factor of D and it will remove every $D-1$ samples from the output sequence, which reduces the sample rate from $U \times IF$ to $\frac{U}{D} \times IF$. Since the expander $\uparrow$ needs to be followed by a low-pass filter for *anti-imaging* and the decimator $\downarrow$ needs to be preceded by a low-pass filter for *anti-aliasing*, the two low-pass filters are collapsed into a single filter **in between** the expander $\uparrow$ and the decimator $\downarrow$ with a maximum cutoff frequency equal to the minimum of $\{(\frac{U}{D} \times \frac{IF}{2}),\ \frac{IF}{2}\}$.

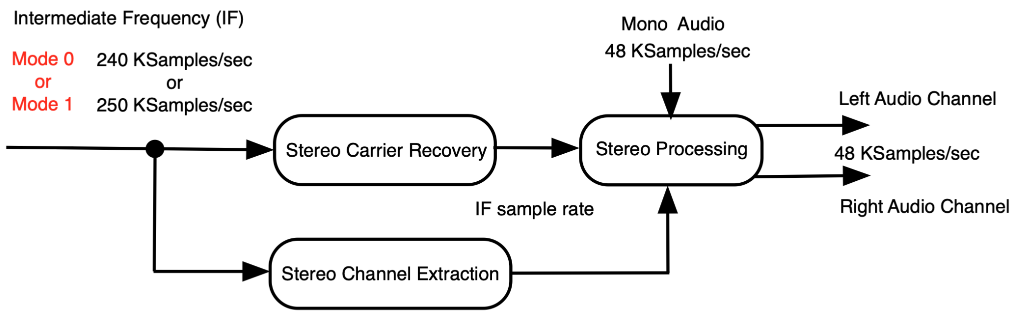


Figure 5: Stereo Processing Path

- To reduce the sample rate at which the low-pass filter operates in mode 1, U is chosen as the ratio between the output sample rate (48 Ksamples/sec) and the greatest common divisor between the input and output sample rates (250K and 48K respectively for mode 1). This results in U=24 for mode 1. Similarly, D=125 (ratio between the input sample rate and the greatest common divisor between the input and output sample rates). Simple math confirms that the maximum cutoff frequency for the low-pass filter between the expander $\uparrow$ and decimator $\downarrow$ for mode 1 is 24 KHz, however to extract only the mono channel we choose an even lower cutoff frequency at 16 KHz.
- What will make the implementation of this filter particularly challenging is that the sampling rate at its input is $U \times IF$, i.e., 24 $\times$ 250 KSamples/sec = 6 MSamples/sec. To reduce the transition band width and increase attenuation from the pass to the stop band, one would have to choose a very large number of filter taps. A straightforward implementation would be very computationally demanding, however, when leveraging some inherent properties of the input sequence to the filter, e.g., $U - 1$ zeros introduced by the expander $\uparrow$ in between any two non-zero samples, or canceling computations for the $D - 1$ output samples that are removed by the decimator $\downarrow$, one can speed up the execution time substantially. This will be one of the algorithmic optimization techniques to be learned while making the real-time implementation feasible.

Stereo Processing Path

The stereo signal carries the **difference** between the left and the right audio channels and for this reason its strength is commonly lower than the signal from the mono channel, which carries the *sum* between the two audio channels. The stereo signal is modulated onto a 38 KHz subcarrier, which is locked to the second harmonic of the 19 KHz pilot tone that is present in each FM channel. During transmission, after upconversion the sub-carrier is suppressed, i.e., the stereo signal is modulated using Double Side Band Suppressed Carrier (DSBSC). The bandwidth of the stereo signal is 15 KHz, with the two sidebands being symmetric with respect to the 38 KHz subcarrier.

The high-level overview of the stereo processing path is shown in Figure 5. The FM demodulated signal at the intermediate frequency (IF) sample rate (either 240 KSamples/sec or 250 KSamples/sec depending on the mode) is passed to two separate sub-blocks: stereo carrier recovery and stereo channel extraction. The outputs of these two sub-blocks are fed into another sub-block for stereo processing, which performs digital downconversion through mixing the recovered carrier with the stereo channel. The output of the mixer will subsequently be converted to the audio sample rate through the same line of reasoning followed in the mono processing path. Finally the stereo data will be combined with mono data to produce the left and right audio channels.

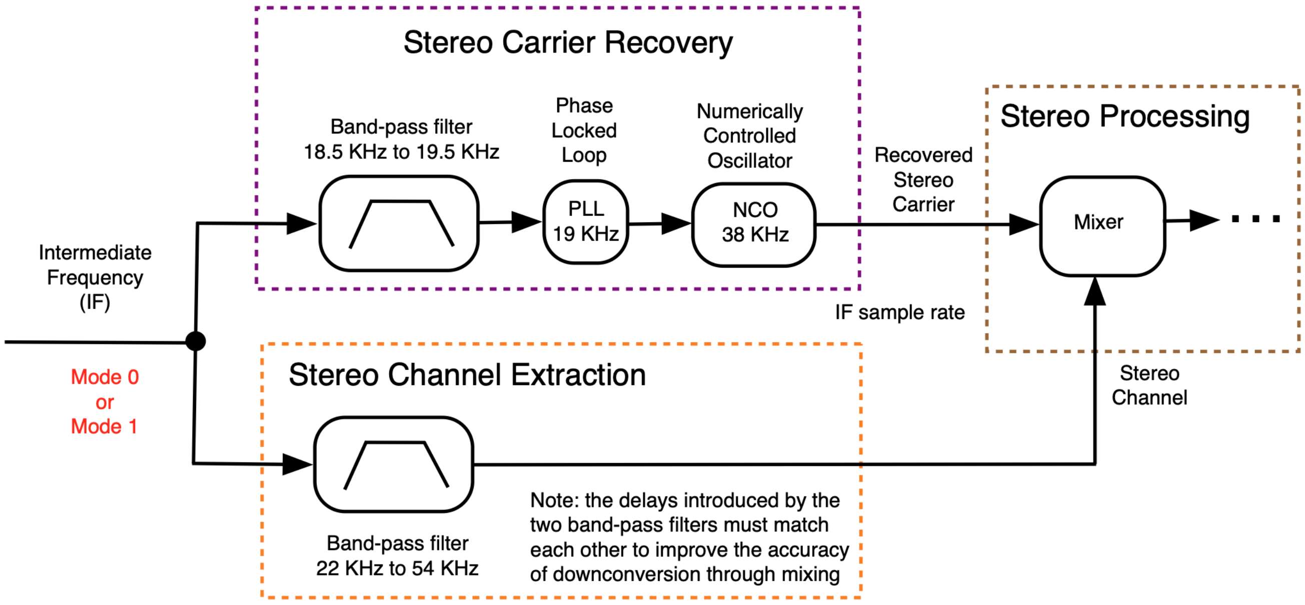

Stereo Carrier Recovery
Band-pass filter 18.5 KHz to 19.5 KHz
Phase Locked Loop
PLL 19 KHz
Numerically Controlled Oscillator
NCO 38 KHz
Recovered Stereo Carrier
Stereo Processing
Mixer
IF sample rate
Intermediate Frequency (IF)
Mode 0 or Mode 1
Stereo Channel Extraction
Stereo Channel
Band-pass filter 22 KHz to 54 KHz
Note: the delays introduced by the two band-pass filters must match each other to improve the accuracy of downconversion through mixing

Figure 6: Stereo Channel Extraction and Carrier Recovery

Stereo Channel Extraction and Carrier Recovery

The sub-blocks for the stereo carrier recovery and stereo channel extraction are shown in Figure 6. However, before explaining them, in order to better appreciate their purpose, it is important to elaborate using simplified formalism *why* their outputs need to be used together as inputs to the mixer from the stereo processing sub-block shown on the right hand side of the figure.

First, it is important to clarify that DSBSC is an amplitude modulation (AM) technique and the stereo signal is upconverted *within* the FM channel at 38 KHz using DSBSC (note, to avoid any confusion, subsequently all the baseband data from all the sub-channels from the FM channel are still frequency modulated before hitting the airwaves from the FM broadcaster). The basic idea of DSBSC modulation is to mix, i.e., multiply, the message signal with the carrier signal as follows: $A_m cos(2\pi f_m) \times cos(2\pi f_c)$, where A_m is the amplitude of the message, f_m is its frequency and f_c is the carrier frequency (for the sake of this discussion we ignore the amplitude of the carrier and phases of the two cosines). Using trigonometric identities, the above multiplication translates to the following DSBSC modulated signal: $\frac{A_m}{2}[cos(2\pi(f_c + f_m)) + cos(2\pi(f_c - f_m))]$. In the receiver, the same type of mixing is done for demodulation purposes, i.e., the carrier frequency is multiplied with the received signal, which has been DSBSC modulated. This mixing in the receiver produces $2\frac{A_m}{4}cos(2\pi f_m) + \frac{A_m}{4}[cos(2\pi(2f_c + f_m)) + cos(2\pi(2f_c - f_m))]$. It can be clearly seen that the first term is the original message (whose amplitude is scaled down by 2) and the second term has a much higher frequency than the first term, assuming f_c is larger than f_m. Based on the above, if we low-pass filter the output of the mixer from the right hand side of Figure 6, we will recover the original message (we can scale the gain of the low-pass filter to make-up for the loss in message strength due to mixing at the transmit side).

To perform the above-described downconversion accurately, first we have to extract the stereo channel, which can be done through a band pass filter with its start (or beginning frequency f_b) at 22 KHz and end frequency (f_e) at 54 KHz. This is the purpose of the stereo channel extraction sub-block from the bottom of Figure 6. Of equal importance is to mix the stereo channel at the receiver with a carrier frequency that is **synchronized** with the subcarrier frequency used by the

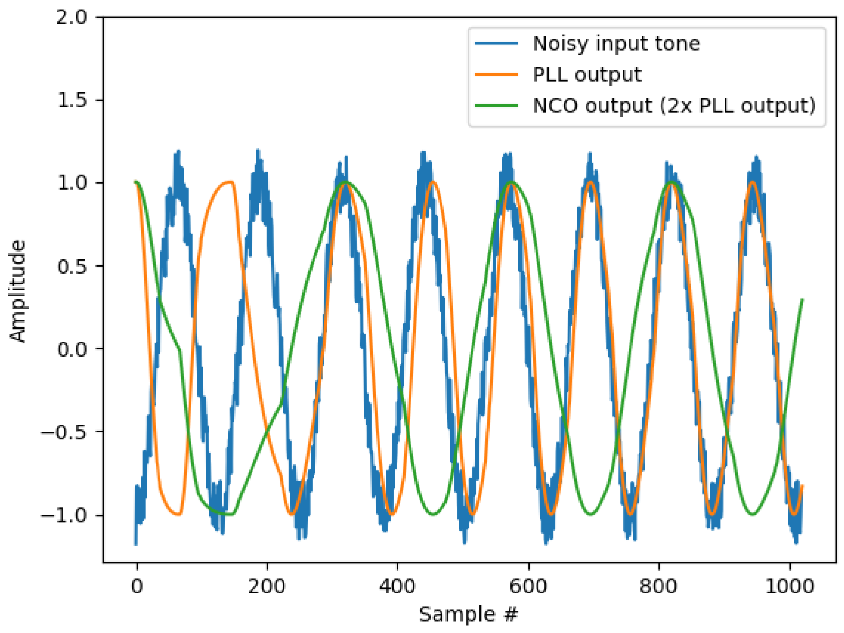


Figure 7: Phase Locked Loop (PLL) and Numerically Controlled Oscillator (NCO)

mixer at the transmitter. Considering that DSBSC suppresses the carrier to save transmit power, all the FM stations broadcast a 19 KHz stereo pilot tone whose second harmonic is used for mixing at the transmitter; it is important to note that this 19 KHz pilot tone is sent as a standalone signal in between the mono and stereo sub-channels within the FM channel. As it can be seen in the Stereo Carrier Recovery sub-block from Figure 6, we first extract the 19 KHz pilot tone through a band pass filter, then we synchronize to it using a phase locked loop (PLL) and finally we multiply the output of the PLL by a scale factor of 2 using a numerically controlled oscillator (NCO) in order to produce the recovered stereo carrier to be used for mixing.

It is important to clarify that PLLs are phase tracking devices that have traditionally been implemented using integrated circuits, nonetheless for frequency ranges of tens to hundreds of KHz, software PLLs can be implemented reliably these days even on embedded computing devices. The applications of PLLs in electrical and computer engineering are very broad, ranging from the control of electrical machines to clock distribution in microprocessors to keeping things orderly in wireless communications (or data transmission in general). Elaborating on the inner workings of a PLL is beyond the scope of this project document (note, the backbone code for a software PLL/NCO will be provided in `Python`). Nonetheless, it is worth mentioning at the intuitive level that an important goal of PLLs is to produce a *clean* output, e.g., filtered and strengthened, in lock to a noisy input. To articulate the importance PLLs in FM receivers, the example from Figure 7 illustrates how a PLL is capable of locking to the phase of a periodic yet noisy input signal and pass a “clean” copy to the NCO for frequency scaling. It is also worth noting that the waves from Figure 7 have been produced using a noisy input tone at 19 KHz oversampled at 2.4Msamples/sec; as it can be observed, the PLL can lock within less than 1 KSamples.

Both the stereo carrier recovery and stereo channel extraction rely on band pass filtering. There are two points worth making: (i) the number of filter taps for the two bandpass filters from Figure 6 should be matched to each other in order to guarantee the same delay from the FM demodulated data to both inputs of the mixer; (ii) the pseudocode for deriving the impulse response coefficients follows the structure and notation from the first lab, where pseudocode was given for deriving the impulse response coefficients for a low-pass filter.

```
Input: Filter parameters: $f_b$, $f_e$, $f_s$ and $N_{taps}$          ▷ $f_b$ is the beginning of the pass band
                                                                      ▷ $f_e$ is the end of the pass band
                                                                      ▷ $f_s$ is the sample rate
                                                                      ▷ $N_{taps}$ is the number of filter taps

Output: Coefficients of the impulse response for a band pass filter $h(i)$ with $i = 0 \dots N_{taps} - 1$
  $Norm_{center} \Leftarrow \frac{(f_e+f_b)/2}{f_s/2}$      ▷ define explicitly the normalized center frequency $Norm_{center}$
  $Norm_{pass} \Leftarrow \frac{f_e-f_b}{f_s/2}$          ▷ define explicitly the normalized pass band $Norm_{pass}$

  for $i \in [0, N_{taps} - 1]$ do
      if $i = (N_{taps} - 1)/2$ then
          $h(i) \Leftarrow Norm_{pass}$      ▷ avoid division by zero in $sinc$ for the center tap when $N_{taps}$ is odd
      else
          $h[i] \Leftarrow Norm_{pass} \frac{sin(\pi(Norm_{pass}/2)(i-(N_{taps}-1)/2))}{\pi(Norm_{pass}/2)(i-(N_{taps}-1)/2)}$
      end if
      $h(i) \Leftarrow h(i)cos(i\pi Norm_{center})$      ▷ apply a frequency shift by the center frequency
      $h(i) \Leftarrow h(i)sin^2(\frac{i\pi}{N_{taps}})$      ▷ apply the Hann window
  end for
```

Stereo Processing: Downconversion, Filtering and Combining

The final sub-block of the stereo path shown in Figure 8 performs downconversion through mixing, sample rate conversion and stereo combining. The principle of downconversion through mixing has been elaborated at the beginning of the previous subsection. From the computational standpoint mixing is a low-demand task because the two input streams are multiplied sample by sample (pointwise multiplication). The output of the mixer will, however, have a significantly higher computational demand because it follows exactly the same sample principles and computational path for digital filtering and sample rate conversion as for the mono channel. The stereo data at the audio sample rate (48 Ksamples/sec) will be combined with the mono audio in order to produce the left and right audio channels that can be subsequently fed to an audio player.

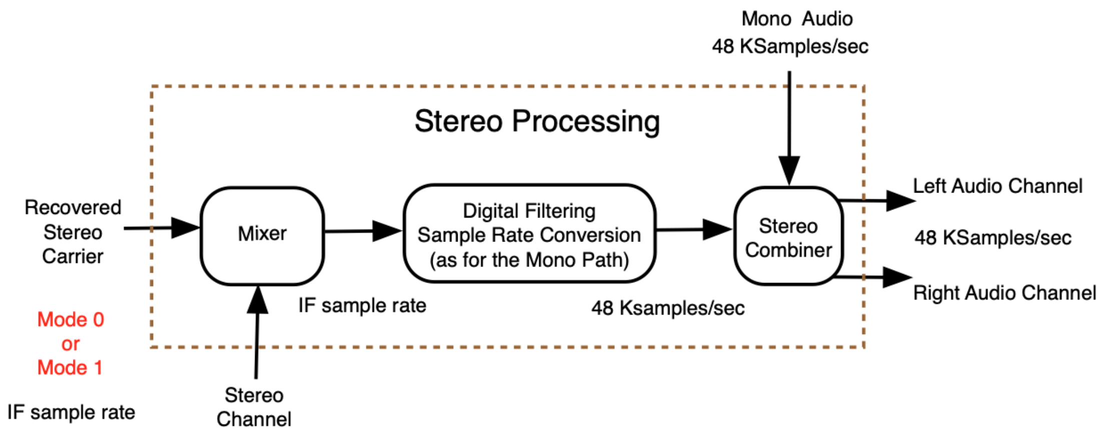


Figure 8: Stereo Downconversion, Filtering and Combining

RDS Processing Path

To be released on or before March 8.

References

[1] NESDR SMArt Series. https://www.nooelec.com/store/sdr/sdr-receivers/smart.html. Accessed: 2021-02.

[2] Osmocom RTL SDR. https://osmocom.org/projects/rtl-sdr/wiki/Rtl-sdr. Accessed: 2021-02.

[3] Raspberry Pi 4. https://www.raspberrypi.org/products/raspberry-pi-4-model-b/. Accessed: 2021-02.

[4] FM Broadcasting. https://en.wikipedia.org/wiki/FM_broadcasting. Accessed: 2021-02.

[5] Radio Data System. https://en.wikipedia.org/wiki/Radio_Data_System. Accessed: 2021-02.

[6] Specification of the Radio Data System (RDS) for VHF/FM Sound Broadcasting in the Frequency Range from 87,5 to 108,0 MHz. http://www.interactive-radio-system.com/docs/EN50067_RDS_Standard.pdf. Accessed: 2021-02.

[7] United States RBDS Standard Specification of the radio broadcast data system (RBDS). https://www.nrscstandards.org/standards-and-guidelines/documents/standards/nrsc-4-b.pdf. Accessed: 2021-02.

[8] GNU Radio. https://wiki.gnuradio.org/. Accessed: 2021-02.

[9] Gqrx SDR. https://gqrx.dk/. Accessed: 2021-02.

[10] FM Stereo/RDS Theory. http://rfmw.em.keysight.com/wireless/helpfiles/n7611b/Content/Main/FM_Broadcasting.htm. Accessed: 2021-02.

[11] Introduction to FM-Stereo-RDS Modulation. https://www.advantest.com/documents/11348/7898f05e-0a52-4e68-9221-3b8b75595436. Accessed: 2021-02.

[12] Frequency Modulation (FM). https://www.ni.com/en-ca/innovations/white-papers/06/frequency-modulation--fm-.html. Accessed: 2021-02.