3DY4 Project - Custom Settings for Group 62

For audio processing (both mono and stereo), the four modes of operation are defined in terms of: the front-end (input) sample rate (RF Fs) used to acquire the raw I/Q samples from the RF dongle; the intermediate frequency sample rate (IF Fs) used at the input/output of the FM demodulator - note, the three processing paths (mono/stereo/RDS) receive data from the FM demodulator at the IF Fs; and the audio (output) sample rate (Audio Fs) used to output the audio samples from your SDR software into an audio player. In the project spec for your group 62, the following values apply to RF Fs, IF Fs and Audio Fs in the four different modes of operation.

Settings for group 62	Mode 0	Mode 1	Mode 2	Mode 3
RF Fs (KSamples/sec)	2400	960	2400	1440
IF Fs (KSamples/sec)	240	120	240	360
Audio Fs (KSamples/sec)	48	40	44.1	44.1

As the above table implies, mode 0 has the same sample rate values as the ones used in lab 3 for audio processing. Your group 62 has a unique combination of settings for modes 1, 2, and 3. Hence, the values used by the descriptions for mode 1 from the project document need to be replaced with those in the above table for modes 1, 2 and 3.

In modes 0 and 2, the IF Fs equals 240 KSamples/sec at the input of the RDS path. Since the RDS path will not use audio Fs, modes 0 and 2 will be differentiated for the RDS path by changing the number of samples per symbol (SPS) to be used by the clock and data recovery. The project spec for your group 62 will have to support the following two custom values for SPS.

Settings for group 62	Mode 0	Mode 2
Samples per symbol (SPS)	34	21

Note that depending on the mode of operation, the RDS symbol rate (in samples/sec) at the output of the rational resampler from the RDS demodulator (see Figure 12 from the project document) will be the SPS value from the above table multiplied by 2,375 (symbols/sec).

Any packet of data transferred from one module in the signal flow graph to another should carry samples for no less than 22 ms and no more than 44 ms. The number of partial products accumulated for an output sample for any filter should be in the 75 to 125 range. This is the number of taps for most filters, except resamplers, where the number of taps is scaled by the expansion scale factor (the same scaling applies to the filter gain). Nonetheless, the number of non-zero partial products for the resampler still equals the above constraint.

Different types of use cases of your program and its interface to the `rtl_sdr` and `aplay` are given on the next couple of pages.

Interfacing your program to `rtl_sdr` and `aplay` through UNIX pipes

When going live, to enable UNIX pipes that stream data into your program from `rtl_sdr` and stream the output of your program into an audio player, such as `aplay`, you should run it as follows from the terminal:

`rtl_sdr` -f 107.1M -s 2400K - | ./`project` 0 m | `aplay` -f S16_LE -c 1 -r 48000

It is critical to note that for the Unix piping to work as above, it is assumed that your program (called `project` in the above example) prints any information for the user using `std::cerr` and **only** the audio samples are redirected to `stdout`. For the command line arguments for `rtl_sdr` and `aplay`, type the corresponding command in a Linux terminal. Your program should have two command line arguments: mode (0, 1, 2 or 3) and the most advanced processing path that is enabled for the current run of the program (*m* for mono, *s* for stereo and *r* for RDS). For example, to work in mode 3 with only the mono audio enabled, you should run it as follows:

`rtl_sdr` -f 107.1M -s 1440K - | ./`project` 3 m | `aplay` -f S16_LE -c 1 -r 44100

After the stereo path has been implemented, you can run it in mode 3 as follows:

`rtl_sdr` -f 107.1M -s 1440K - | ./`project` 3 s | `aplay` -f S16_LE -c 2 -r 44100

Note, in the above use, you will stream twice as much data from your program and, therefore, `aplay` needs to be aware that its input has two channels.

After implementing the RDS path, you should run it only in modes 0 or 2. For example, in mode 2, run it as follows:

`rtl_sdr` -f 107.1M -s 2400K - | ./`project` 2 r | `aplay` -f S16_LE -c 2 -r 44100

When the RDS path is enabled, all the audio **must** also be processed in the stereo mode; hence, your program must stream its output to `aplay` in two channels. Note, however, that the RDS processing path can be ignored when your software runs in modes 1 and 3; in other words, when operating in modes 1 and 3, and the second command line argument is *r*, you can print a message that confirms that RDS is not supported in the respective mode.

Emulating streaming during development

While developing your code, you can emulate the streaming of I/Q samples from `rtl_sdr` by redirecting the pre-recorded I/Q samples using the `cat` command in UNIX. For example, to stream the pre-recorded samples from *iq_samples.raw* to your program that runs in mode 1 for the mono path, do as follows:

`cat` *iq_samples.raw* | ./`project` 1 m | `aplay` -f S16_LE -c 1 -r 40000

Note that you must ensure that the I/Q samples have been acquired at the appropriate sample rate, e.g., 960 KSamples/sec for your project settings for mode 1. To record five seconds, do as follows when the RF dongle is plugged in:

`rtl_sdr` -f 107.1M -s 960K -n 4800000 - > *iq_samples.raw*

For modes 0 and 2, the I/Q samples from lab 3 can be reused because they have been acquired at 2.4 MSamples/sec, and they are sufficient to get you started with troubleshooting (even with the resampler that is needed on the mono path in mode 2). If you wish to reuse the I/Q samples from lab 3 for other modes at another sample rate, you can change their sample rate using the Python script `fmRateChange.py` from the `model` sub-folder.

Compilation settings

Regardless of the platform (virtual machine or physical hardware), you should have `-O3` enabled in `CFLAGS` in the `Makefile`. If you compile on Raspberry Pi, you can also enable `-mcpu=cortex-a72+crypto` in `CFLAGS`.

After the stereo path has been implemented, you will need to use threads to meet the real-time constraints on the Raspberry Pi (more details will be elaborated in class). To enable the use of the threads library in your source code, the `Makefile` from the `src` subfolder should have the `LDFLAGS` entry updated as follows:

```
LDFLAGS = -pthread
```

If you choose to use `cmake`, then `CMakeLists.txt` must be updated as follows for multithreading (place it in a line before `${PROJECT_LINK_LIBS}` is used):

```
set (PROJECT_LINK_LIBS pthread)
```