COMPENG 3DY4 Project Report

Group 41-Thursday
Hao Wu (wuh113@mcmaster.ca)
Lifeng Mei (meil4@mcmaster.ca)
Yiming Chen (cheny466@mcmaster.ca)
Ruiyi Deng (dengr6@mcmaster.ca)

1. Introduction

This project aims to implement a real-time software-defined radio system and give us an understanding of the implementation of a complex computing system in real life. The project requires us to develop and run the C++ code on Raspberry Pi, which can work in a real-time environment like a radio receiver.

2. Project Overviews

Software-Defined Radio (SDR)

Software-defined radio is a communication system that receives FM audio signals in real time.

Block processing with states

In this project, we read, generate and write in blocks to be more efficient. State saving methods are also applied to prevent the loss of information.

Frequency Modulation including Mono, Stereo and RDS

One FM channel has a bandwidth of 200 kHz. The mono sub-channel is from 0 to 15 kHz; the stereo sub-channel is from 23 to 53 kHz; there is a 19 kHz stereo pilot tone between the mono, stereo subchannel and the RDS subchannel is from 54 to 60 kHz.

Downsamplers and Resamplers

The downsampler methods are used in both RF Front-end and mono and stereo audio. The signal is downsampled to achieve the desired frequency range, and the resampler method might be used

depending on the signal processing mode. Our project combines downsampling and resampling methods with LPF to be more efficient.

Finite-Impulse Response Filter

The FIR filter generates the filter coefficients and passes them to the low pass filter or bandpass filter. We change the value of the number of taps, cutoff frequency, and sampling rate to implement different coefficients depending on the requirement.

FM Demodulator

For the demodulator, we use the arctan version to implement it. It is used in the RF front-end sub-block.

Phase-Locked-Loops (PLLs)

PLLs are phase tracking devices used to produce a clean output. They are used in stereo sub-channel and RDS sub-channel. The 19 kHz pilot tone is synchronized to the extraction signal by the PLL.

Radio Data System (RDS)

RDS block will recover the subcarrier from the RDS channel, downconvert and resamples it. Then it will apply clock and data recovery and frame synchronization. Finally, it passes the extracted bits at the threading part.

3. Implementations details

Summary of previous Lab experiment

In lab 1, we implemented some basic algorithms like Fourier transformation, filters and state processing. In the second lab, we focused on implementing C++ and learnt some useful tools and websites for C++ development. Lab 3 is the extension of the previous two labs, and in this lab,

we implemented Python and C++ code with processing samples from a frequency-modulated (FM) channel to produce mono audio.

Implementation of RF Front-end & Mono audio

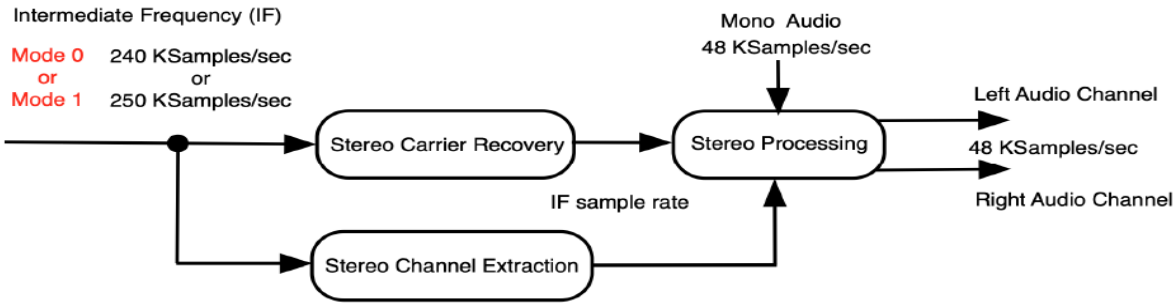
The downsample factor and the upsample factor are calculated by simply dividing the current Fs and the target Fs, the numerator and denominator will be the upsample factor and the downsample factor, respectively.

The implementation of the front-end and the mono channel is similar. We created a downsampled version of the low-pass filter (LPF). Function filter_ds is created to do the downsample convolution. We do not need to take every data from the input, instead we take the nth value of the input (n is the downsample factor) by changing indexes. In this case, a downsampled output y is obtained.

For modes 2 and 3, which required us to do the upsampling, we implemented a convolution function that has the structure of LPF but also combined with downsampler and upsampler at the same time. To do resampling, we have to increase the sampling rate and expand the size of the filter coefficient to a factor of the expander. We also need to create an extra local variable named phase, representing the offset. After expanding the size of coefficients, we are only using every nth value, which n is the factor of the expander. The index of h(coefficient set) is then set to be (phase+k*us). Since we are downsampling and upsampling at the same time. The index of the original signal(xb) is the same index from the downsampling-only function but divided by the factor of the expander. We subtract the phase value here before division to prevent any rounding error. In the end, for every value we get after filtering, we provide it with a gain of expander since when we expand the size of coefficients, the value decreases.

Implementation of Stereo audio

In the stereo sub-channel, it takes the input from the RF front-end, the FM demodulated signal at an intermediate frequency sample rate. It is passed to two different sub-blocks (see figure below). The outputs of the two sub-blocks are passed to the Stereo Processing.



In stereo carrier recovery, a bandpass filter with a cutoff frequency of 18.5 kHz to 19.5 kHz is applied to extract the stereo pilot tone centered around 19 kHz. function impulseresponseBPF is created (from the pseudocode) to obtain the impulse response coefficients of the bandpass filter. We can use it by varying its cutoff frequencies and the number of taps. Then, the signal is passed to the phase-lock loop (PLL), the PLL module is given in python, we refactor it in c++ and add states saving. In terms of state saving, in order to make the output of the PLL continuous, we need to save the last value for each variable in each block and reuse it as the first value and do the same process again in the next block.

At the same time, the stereo channel is extracted in the stereo channel extraction sub-block, it uses the same input signal as the stereo carrier recovery sub-block, the input signal is passed to a bandpass filter with cutoff frequency between 22 KHz and 54 KHz.

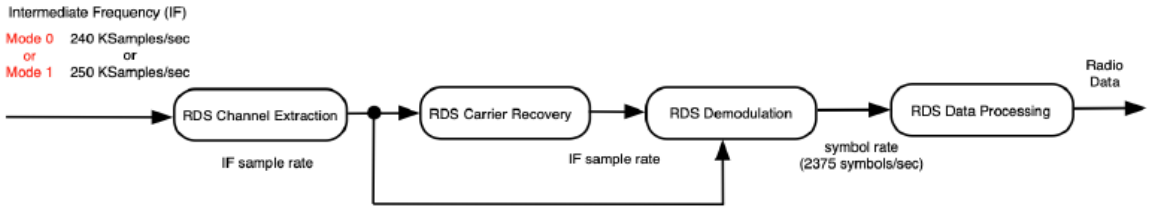
Then the recovered stereo carrier and the stereo channel are mixed in the mixer, pointwise multiplication is applied here. We have to apply a gain of 2 to recover the loss in the pointwise multiplication, then a down sample low pass filter with cutoff frequency of 16 KHz is created to

obtain the downsampled version of the mixed signal, now the sample rate is 48 KHz or 44.1 KHz(depends on different modes). A combiner is applied to combine the mono channel and the stereo channel, the delays introduced by the band-pass filters in stereo carrier recovery and stereo channel extraction must match the mono channel, so an allpass filter is used in this case, the mono channel is passed to the allpass filter to create a half period delay. In the allpass filter, the output holds the second-half period of the input signal from the previous block and the first half of the input signal from the current block, state saving is applied to store the second-half of the input.

The mono channel is the summation of the left and right channel, the stereo channel is the difference between the left and right channel. We add the two channels to obtain the left channel and subtract them to get the right channel.

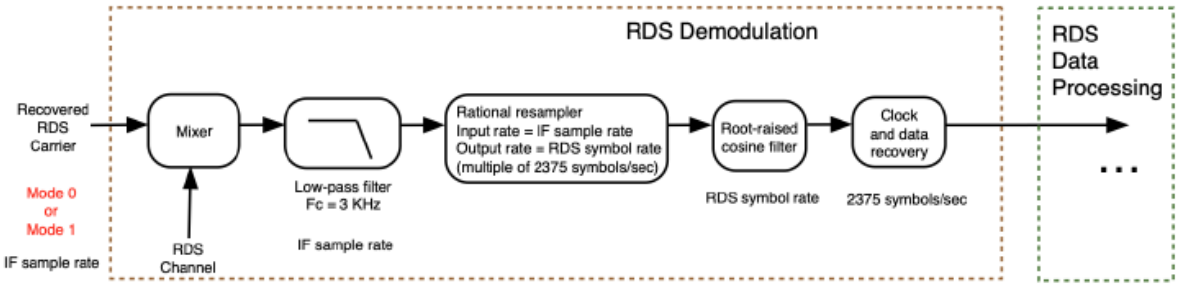
Implementation of RDS audio

Before starting the implementation of the RDS audio, the first thing to do is to modify the PLL function. Unlike the PLL function implemented in the stereo part, we will have to use the ncoOutQ for the quadrature values besides the in-phase values.



For RDS Channel Extraction, we used state processing with coefficients obtained from the bandpass filter with 54kHz - 60kHz. Then we start with the RDS carrier recovery process. The extracted channel will go into two branches. First, it will go through the squaring non-linearity

process which will multiply by itself. Next, it goes through a bandpass filter of 113.5kHz to 114.5kHz. pllQ function will then be applied. The other branch uses the all-pass filter to generate a phase shift to match the delay introduced by the bandpass filter. We finally obtain the recovery data by mixing the in-phase, quadrature, and phase-shift values by multiplication.



The next step is to demodulate the signal. The first step is to get it through a low-pass filter with a cutoff frequency of 3 kHz. This signal will then be resampled to yield a multiple of 2375. Our constraint is 22, and it will give us the frequency at 52.25 kHz. In our implementation, we used techniques similar to the us_ds implemented in the previous part to generate the resampled data. This data will pass through an anti-imaging filter. For the clock and data recovery part, it is important to know whether we will count the first L or H signal. Therefore we set a flag to determine the offset. Because it will never happen when three successive signals come together, we only need to compare the adjacent ones.

Finally, we are moving to the decoding part. From the bitstream obtained in the previous part, HL means 1 and LH means 0. We used the bool type value to represent the two results for we will have to use XOR for differential decoding next. As for block processing, the first block's prev_bit is determined with the first bit in the bitstream. And for the others, the first signal will be determined, referencing the last bit in the previous block.

Unfortunately, we did not finish the synchronization part due to the time issue, so our journey stopped here.

4. Analysis and measurements

For Mono mode 0 and 1, the block size is 1024*10*2 =20480 samples. Number of taps is 101 and audio decimation factor is 5, which means we need 20480/5*101=413696 multiplications.

For Mono mode 2, the block size is the same as above which is 20480 samples. The number of taps is 147*101 = 14847, as we need to upsample 101 by 147. The audio decimation factor is 10. This means we need 20480/800*14847 = 380083 multiplications.

For Mono mode 3, the block size is the same as above which is 20480 samples. The number of taps is 49*101 = 4949, as we need to upsample 101 by 49. The audio decimation factor is 10. This means we need 20480/320*4949 = 316736 multiplications.

For the stereo part, the block size we used in carrier recovery and channel extraction is 20480 and number of taps is 101, then we can calculate the number of multiplications for each is 2068480. Because there is no downsampling here, we need to experience 4 more times of multiplications here which means the decimation factor here is 1. And our runtime results match this trend.

For Stereo mode 0&1, the number of multiplications is over 10 times of mono mode 0&1. And the runtime can also match the trend of multiplication.

For Stereo mode 2, the number of multiplications is 4280178. And for Stereo mode 3, the number of multiplications is 4261479. To obtain one sample, we need 5*101 +1 multiplications and 5*1 + 3 accumulations to be done.

Parts	Functions	Runtimes (s)	Number of

			Multiplications
RF_Front_end	Filter & decimation	9.643*10^-3	206848
	Demodulation	9.587*10^-5	20480
Mono mode 0	Filter & decimation combine	5.612*10^-4	413696
Mono mode 1	Filter & decimation combine	5.886*10^-4	413696
Mono mode 2	Filter & decimation combine	4.727*10^-3	380083
Mono mode 3	Filter & decimation combine	4.448*10^-3	316736
Stereo mode 0	Stereo carrier recovery	2.761*10^-3	2068480
	PLL	2.135*10^-3	20480
	Extraction	2.754*10^-3	2068480
	Mixing	1.296*10^-5	20480
	LPF	5.036*10^-7	4096
	Combiner	3.844*10^-6	4096
Stereo mode 1	Mixing	1.148*10^-5	20480
	LPF	5.463*10^-7	4096
	Combiner	3.531*10^-6	4096
Stereo mode 2	Mixing	1.364*10^-5	98304
	LPF	2.565*10^-6	20480
	Combiner	2.586*10^-6	3954
Stereo mode 3	Mixing	1.136*10^-5	79872
	LPF	2.494*10^-6	20480
	Combiner	2.387*10^-6	3687

* For Stereo carrier recovery, PLL and Extraction, they are similar for stereo mode 0-3, so we only mention them for once.
** Since we didn't finish the RDS part, we are not able to measure the runtime of RDS, and that's why there is no RDS runtime in the above form.

5. Proposal for improvement

To improve the user experience, we can improve our audio quality by designing based on a more significant number of taps and avoid any underrun issues we encounter in our real-time testing. Also, as mentioned in the previous lecture, when we attenuate specific frequencies, a built-in pre-emphasis filter is used, making the audio quality worse than it should be. Therefore, we can develop a de-emphasis filter to neutralize the effect of the pre-emphasis filter and produce better audio quality.

We have several issues in real-time running and can be improved to prevent underrun error and runtime efficiency. For example, we can modify the value of the number of taps to see which one works in a real-time environment. We can further optimize our code by defining fewer variables and sharing variables if they have the same values. We can also check if some of the steps can be implemented simultaneously (in one for-loop specifically) to improve the performance further since the program will have less for-loop to run.

6. Project Activity

Week	Project progression	Contributors
2.14-2.20	Review project documents and translate lab3 code to C++	All Members
2.21-2.27	Hardware Setup	Ruiyi Deng
	Front-end implementation	Lifeng Mei
	Mono python implementation	Hao Wu
2.28-3.6	Front-end debugging	Lifeng Mei
	Mode 2&3 implementation	Hao Wu, Ruiyi Deng
	Mode 0&1 implementation	Lifeng Mei, Yiming Chen
3.7-3.13	Mono part implementation and debugging	All Members

3.14-3.20	Stereo python implementation	Yiming Chen
	Mono code debugging	All Members
3.21-3.27	Stereo C++ implementation	Yiming Chen, Lifeng Mei, Hao Wu
	Threading for Front-end, mono and stereo audio	Hao Wu
	Stereo code debugging	All Members
3.28-4.1	Mono and Stereo debugging and code optimization	All Members
	RDS implementation and debugging	Lifeng Mei, Yiming Chen
	Real-time testing	Ruiyi Deng

7. Conclusion

In conclusion, this project experience helps us gain valuable experience in industry-level real-time computing system implementation. All of our members have a much deeper understanding of the SDR system and other concepts in signal processing, including filtering, modulation and demodulation, PLLs etc. We also progress our coding ability in C++ during this valuable hands-on project opportunity. The project also teaches us how to cooperate when people have different schedules and different levels of understanding. It progressed very slowly at the beginning when there was limited information provided but smoothly transmitted as time went on after we had a basic idea about its structure and divided it into sub-groups to improve efficiency.

8. Reference

[1] N. Nicolici, “COE3DY4 Project, Real-time SDR for mono-stereo FM and RDS”.