Power Amplifier Group No.1

National Taiwan University
Department of Engineering Science and Ocean Engineering
Jing-Hua Chang、Zhi-Cheng Chen、Bo-Wen Pan

A. Design Methods

- Power supply stability
- Low-pass Filter
- FIR Null Filter
- Differential output

B. Power supply stability

The main problem encountered first was the continuous beating sound "Dadadadadada" which occurs regularly once the amplifier start working. Then it is observed that as the voltage of battery supply to the driver board starts to drop, this phenomenon worsened. Moreover, this doesn't bother other teams for the reason that their speaker wasn't loud enough to hear this distinct noise.

Therefore, let's attribute the beating sound to the instability of the battery supplier. Instead, we made a customized USB power supplier for the driver board. Then the beating sound had gone so small that people can barely hear it when they put ears near the loudspeaker.

C. Low-pass Filter

Use MATLAB to create a low-pass filter, copy the coefficients for implementation on STM32CubeIDE.

Below demonstrate a convolution between digital data and filter coefficients using for loop.

Under `/* USER CODE BEGIN PV */` assign

```
float coefficients_fir[FIR_LENGTH] = {coefficients get from MATLAB};
uint32_t buffer_fir[FIR_LENGTH];
uint32_t counter_fir;
volatile uint32_t output_fir;
float out_fir;
uint32_t signal;
uint32_t index_fir;
```

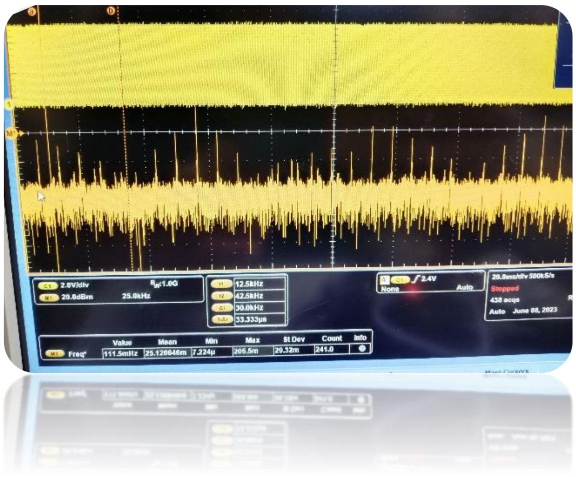
In `HAL_TIM_PWM_PulseFinishedCallback()` write：

```
//low-pass Filter
out_fir = 0;
buffer_fir[counter_fir] = adc;

for(int i = 0; i < FIR_LENGTH; i++)
{
     index_fir = counter_fir - i;
     if(index_fir < 0)
     {
          index_fir = FIR_LENGTH + counter_fir - i;
     }
     out_fir += coefficients_fir[i] * buffer_fir[index_fir];
     TIM3->CCR4 += coefficients_fir[i] * buffer_fir[index_fir];
}
counter_fir++;
if(counter_fir == FIR_LENGTH)
{
     counter_fir = 0;
}
TIM3->CCR4 = output_fir;
```

Additional noise are generated after the filter is implemented. This design is discarded.

D. FIR Null Filter

Observing the Fourier transform of the output signals on oscilloscope. There exist a 7.5k Hz and a 15k Hz evident signals other than our 70k Hz sampling frequency, possibly the source of the output noise. Thus, removing them might reduce the white noise.



C1 2.0V/div BW:1.0G
M1 20.0dBm 25.0kHz

t1	12.5kHz
t2	42.5kHz
Δt	30.0kHz
1/Δt	33.333µs

A C1 ∫ 2.4V None Auto

20.0ms/div 500kS/s
Stopped
438 acqs
Auto June 08, 2023

	Value	Mean	Min	Max	St Dev	Count	Info
M1 Freq'	111.5mHz	25.126846m	7.224µ	205.5m	29.32m	241.0	●

Under *Project explorer* –> *Core* –> *Src* create *Filter.h*：

```
#ifndef INC_FILTER_H_
#define INC_FILTER_H_

#include <math.h>
#include <stdint.h>

typedef struct Filter{

    /* Sample time (s) */
    float sampleTime_s;

    /*Filter I/O (x[0]/y[y] = current I/O sample)*/
    float x[3];
    float y[3];

    /*x[n]/y[n] coefficients*/
    float a[3];
    float b[3];

} Filter;
```

```
void Filter_Init(Filter *filt, float sampleRate_Hz);
void Filter_SetParameters(Filter *filt, float centerFrequency_Hz, float
bandwidth_Hz, float boostCut);
float Filter_Update(Filter *filt, float input_latest);

#endif /* INC_FILTER_H_ */
```

Under *Project explorer* -> *Core* -> *Inc* create *Filter.c*：

```
#include "Filter.h"

void Filter_Init(Filter *filt, float sampleRate_Hz)
{

    /*Compute sample time*/
    filt->sampleTime_s = 1.0f / sampleRate_Hz;

    /*Clear filter momory*/
    for(uint32_t n = 0; n < 3; n++)
    {
        filt->x[n] = 0.0f;
        filt->x[n] = 0.0f;
    }

    /*Calculate 'default' filter coefficients (all-pass)*/
    Filter_SetParameters(filt, 1.0f, 0.0f, 1.0f);
}

void Filter_SetParameters(Filter *filt, float centerFrequency_Hz, float
bandwidth_Hz, float boostCut)
{

    /*Convert Hz to rad/s (rps), pre-warp cut-off frequency (bilinear
    transform), multiply by sampling time (wcd * T =
    2/T*tan(2*pi*fc*T/2) * T = 2*tan(pi*fc*T)*/
    float wcdT = 2.0f * tanf(M_PI * centerFrequency_Hz * filt-
    >sampleTime_s);
    /*Compute quality factor (Q = fc/fbw)*/
    float Q = centerFrequency_Hz / bandwidth_Hz;

    /*Compute filter coefficients*/
    filt->a[0] = 4.0f + 2.0f * (boostCut / Q) * wcdT + (wcdT * wcdT);
    filt->a[1] = 2.0f * (wcdT * wcdT) - 8.0f;
    filt->a[2] = 4.0f - 2.0f * (boostCut / Q) * wcdT + (wcdT * wcdT);

    filt->b[0] = 1.0f / (4.0f + 2.0f / Q * wcdT + (wcdT * wcdT));
    /*1 / b0*/
    filt->b[1] = -(2.0f * (wcdT * wcdT) - 8.0f);
    /*-a0 for feedback*/
    filt->a[2] = -(4.0f - 2.0f * (boostCut / Q) * wcdT + (wcdT *
wcdT));  /*-a0 for feedback*/

}

float Filter_Update(Filter *filt, float input_latest)
{

    /*Shift samples*/
    filt->x[2] = filt->x[1];
    filt->x[1] = filt->x[0];
    filt->x[0] = input_latest;

    filt->y[2] = filt->y[1];
    filt->y[1] = filt->y[0];
    filt->y[0] = ((filt->a[0]*filt->x[0] + filt->a[1]*filt->x[1] +
    filt->a[2]*filt->x[2]) + (filt->b[1]*filt->y[1] + filt->b[2]*filt-
    >y[2])) * filt->b[0];

    /*Return current output sample*/
    return (filt->y[0]);
}
```

Under `/* USER CODE BEGIN Includes */` include：

```
#include "Filter.h"
```

```c
#include <stdlib.h>
```

Under `/* USER CODE BEGIN PD */` define：

```c
#define SAMPLE_RATE_HZ 44100.0f
#define UINT16_TO_FLOAT 0.00001525878f
#define FLOAT_TO_UINT16 32768.0f
#define AUDIO_BUFFER_SIZE 256
#define outputVolume 0.00005f
```

Under `/* USER CODE BEGIN PV */` assign：

```c
Filter filt, filt2;
static volatile uint16_t *bufferCHin;
static uint16_t *adcOut[AUDIO_BUFFER_SIZE];
static volatile uint16_t *bufferCHout = &adcOut[0];
static float filterDividing3, filterDividing4;
static float filtered3, filtered4;
static uint16_t blackout3, blueout4;
```

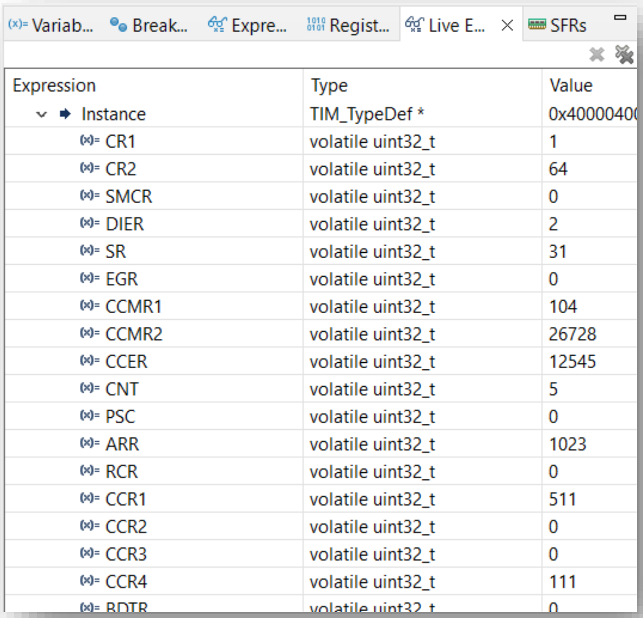
Under `/* USER CODE BEGIN 0 */` write a function：

```c
void Process(uint16_t *adcIn)
{
    //Filter
    //Duty cycle (AUDIO_BUFFER_SIZE/2 - 1)
    for(uint8_t n = 0; n < AUDIO_BUFFER_SIZE / 2 - 1; n += 2)
    {
        adcIn[AUDIO_BUFFER_SIZE/2] = 1024 - adcIn[0];
        bufferCHin = adcIn[AUDIO_BUFFER_SIZE/2];

        /*Divide the adc into 2 channel,
        normalize the signal amplitude*/
        filterDividing3 = UINT16_TO_FLOAT * ((float) bufferCHin[n]);

        /*Filtered compare not filtered*/
        //Filters initialization
        Filter_Init(&filt, SAMPLE_RATE_HZ);
        Filter_SetParameters(&filt, 7500.0f, 500.0f, 0.5f);
        /*Filtered signals*/
        filtered3 = Filter_Update(&filt, filterDividing3);

        /*normalize the signal amplitude*/
        blackout3 = (uint16_t) (FLOAT_TO_UINT16 * outputVolume *
        filtered3);
        bufferCHout[n] = blackout3;

        adcOut[AUDIO_BUFFER_SIZE/2] = adcOut[0];
        TIM3->CCR4 = adcOut[AUDIO_BUFFER_SIZE/2];
    }
}
```

In `HAL_TIM_PWM_PulseFinishedCallback()` write：

```
uint32_t adc = HAL_ADC_GetValue(&hadc4) >> 2;
if(htim == &htim3)
{
    TIM3->CCR3 = adc;
}
uint16_t adcIn[AUDIO_BUFFER_SIZE];
adcIn[0] = HAL_ADC_GetValue(&hadc4) >> 2;
Process(adcIn);
```

No sound or music is recognized at the output. However, there are numbers in the CCR4 and outputs is showed on the oscilloscope. In conclusion, the algorithm works fine but need more adjustment to recover the original signals. The “Live Expression” window in debug mode shows that signals successfully pass through the FIR null filter, then go through a low-pass filter, reach the



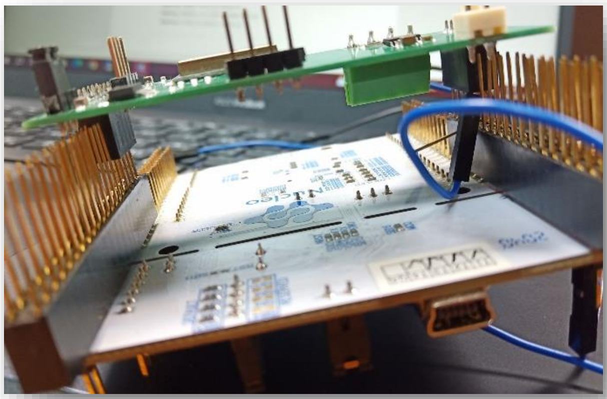
Expression	Type	Value
Instance	TIM_TypeDef *	0x4000040
CR1	volatile uint32_t	1
CR2	volatile uint32_t	64
SMCR	volatile uint32_t	0
DIER	volatile uint32_t	2
SR	volatile uint32_t	31
EGR	volatile uint32_t	0
CCMR1	volatile uint32_t	104
CCMR2	volatile uint32_t	26728
CCER	volatile uint32_t	12545
CNT	volatile uint32_t	5
PSC	volatile uint32_t	0
ARR	volatile uint32_t	1023
RCR	volatile uint32_t	0
CCR1	volatile uint32_t	511
CCR2	volatile uint32_t	0
CCR3	volatile uint32_t	0
CCR4	volatile uint32_t	111
BDTR	volatile uint32_t	0

final location in CCR4.

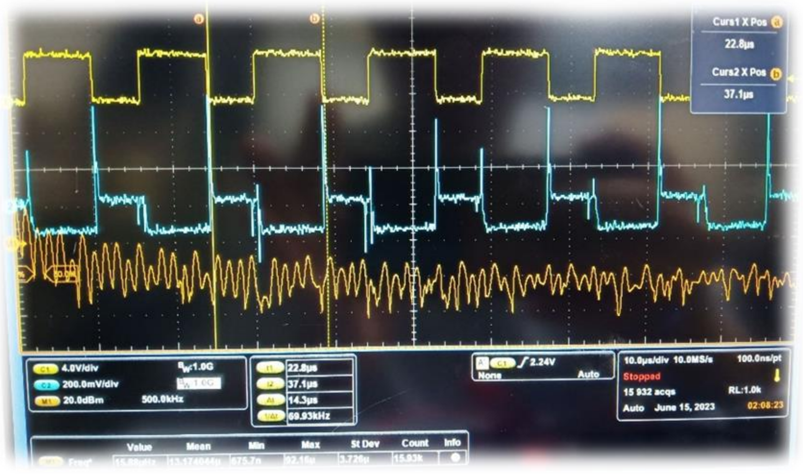
E. Differential output

At first, we tried using both channel 3 and channel 4 in TIM3 to compare the difference between original output and the filtered output. But it is found that channel 3 and channel 4 effect upon each other. When assigning nothing the CCR4 register, the output of channel 4 still shows small PWM signals on the oscilloscope once the channel 3 is activated. Taking a look at the schematic of the Arm architecture, we reckon that it might be the crosstalk between PC8 and PC9 that causes the unexpected result.

Thus, for the purpose of implementing differential output method from to channels to cancel the main noises, the approach is assigning TIM3_CH4 to PB1. Then the crosstalk is gone. Then of course there will be some adjustment on our physical design connection between mother board and child board.



Set the polarity of CH3 and CH4 to high and low, so that the signal magnitude in CH4 is the exact opposite to that in CH3. Noise is well reduced after the differential method applied. However, the coil movement require a reference ground. To apply differential method to the loudspeaker, one side of the input must connect to the ground.



F. Summary

The performance of a loudspeaker depend mainly on the speaker itself. If the speaker is broke or bad, there is nothing we can try in digital processing.

The battery stability and the connections between wires are responsible for unwanted noises. The filter is best for equalizer but might not go well on noise canceling.

“Live Expression” and “Serial Wire Viewer” are good tools to analyze our implementation. Oscilloscope is also great for debugging.