Embedded Signal Processing System - ENE4069

Final Project

2018111902정상훈

## 문제 1. 출력 데이터에서 MATLAB에서 들을 수 있는 소리에 대한 두 음악을 분리합니다.

텍스트, 스크린샷, 소프트웨어, 운영 체제이(가) 표시된 사진

자동 생성된 설명

그림 1 Matlab 코드

‘twoChannelMusic’는 한 음악은 저역대에 위치해 있고, 한 음악은 고역대에 위치해 있습니다. 한 파일에 두 음악이 존재하지만 고역대에 있는 음악은 사람 귀에 잘 들리지 않기 때문에 저역대에 있는 음악만 들리게 됩니다.

이 두 음악을 분리하기 위해서 low pass filter를 이용하여 저역대에 있는 음악만 필터링하고, high pass filter를 이용하여 고역대에 있는 음악을 필터링한 후 modulation을 이용하여 저역대로 shift하였습니다. Low pass filter는 pi/4에서, high pass filter는 3\*pi/4에서 cut-off하였고, 101개의 length를 통해 필터링을 거쳤습니다.

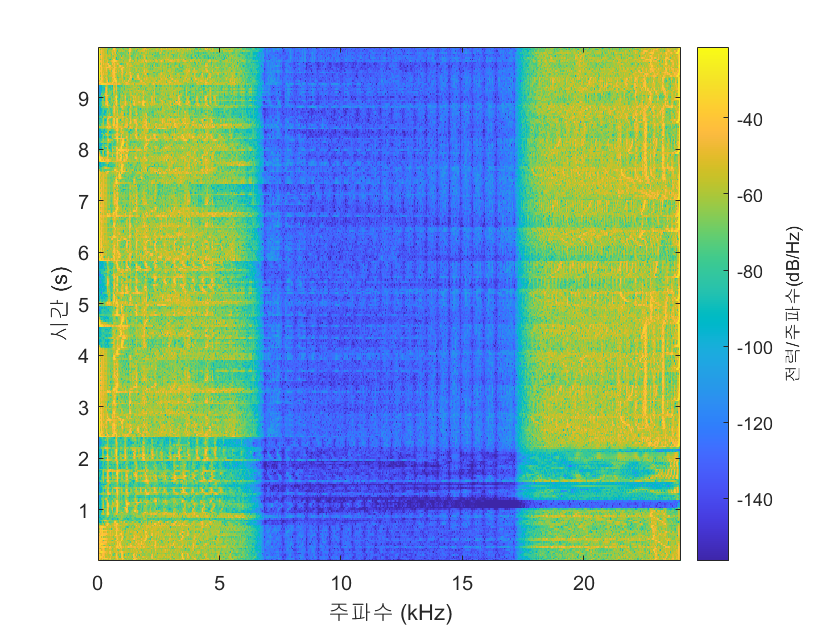


그림 2 output spectrogram

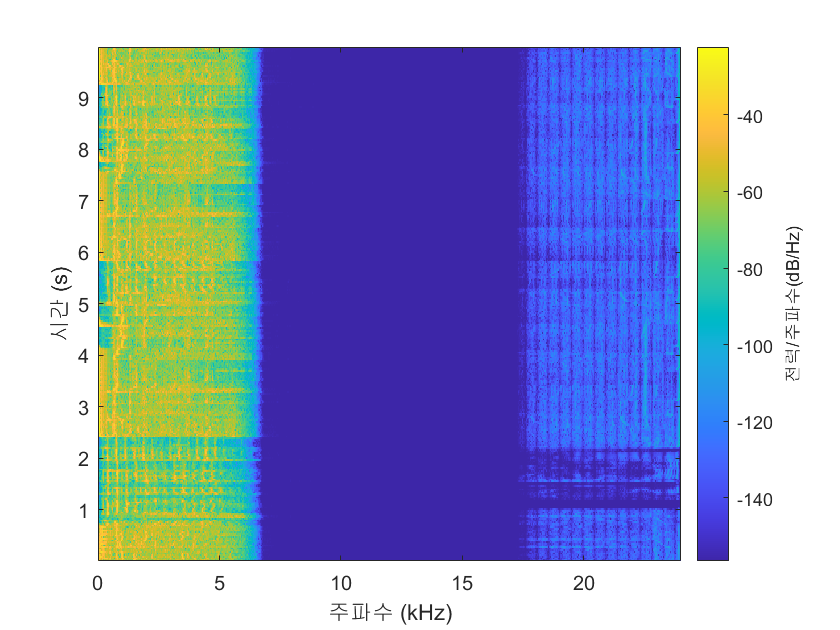
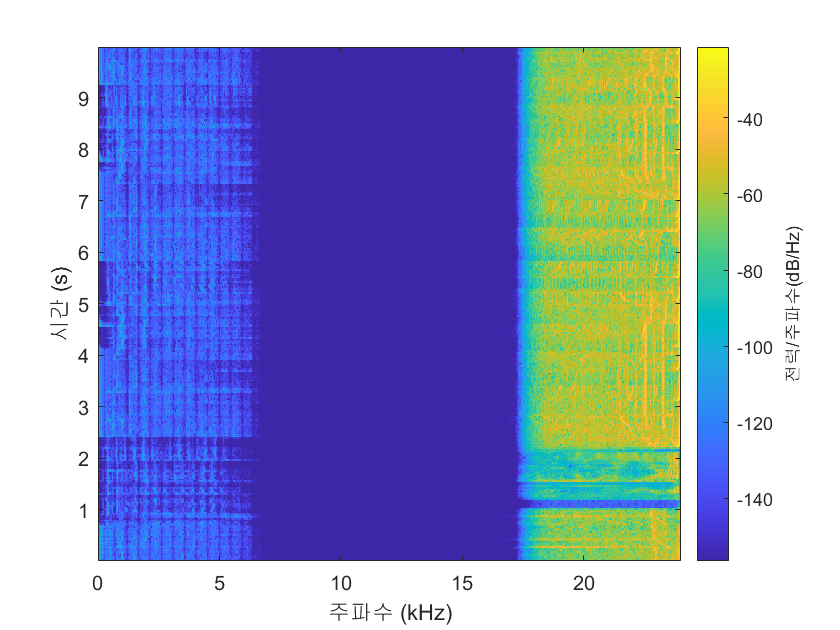


그림 3 DDUDU spectrogram



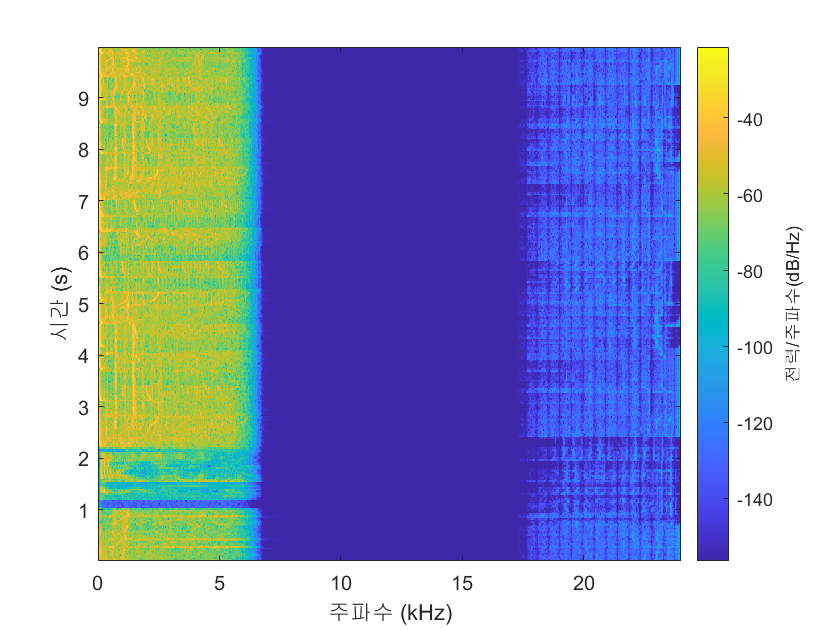
그림 4 output2 spectrogram

그림 5 FIRE spectrogram

Output : 두 음악이 합쳐져 있는 출력 spectrogram

DDUDU : Output 출력을 LPF한 출력 spectrogram

Output2 : Output 출력을 HPF한 출력 spectrogram

FIRE : Output2 출력을 shift한 출력 spectrogram

## 문제 2. ARM 프로세서를 사용하여 두 음악을 왼쪽 오른쪽 각각의 채널로 분리합니다.

텍스트, 라인, 스크린샷, 직사각형이(가) 표시된 사진

자동 생성된 설명

### CMSIS FIR 기본 코드를 작성

실습 동영상을 통해 설계한 CMSIS FIR 코드를 기반으로 코드를 작성하였습니다.

### 매트랩을 이용한 LPF 계수 추출

매트랩을 이용하여 low pass filter의 계수를 추출하였습니다. Length는 101로 하였고 format(long)을 이용하여 소수점 15자리 까지 추출하여 보다 깔끔한 필터링을 유도하였습니다.

### 매트랩을 이용한 HPF 계수 추출

HPF 계수 또한, 매트랩을 이용하여 추출하였습니다. Length는 101로 하였고 format(long)을 이용하여 소수점 15자리 까지 추출하여 보다 깔끔한 필터링을 유도하였습니다.

### CMSIS FIR 코드 추가 및 변경

1. Length 정의 변경

**#define** **FILTER\_LENGTH** 101

Filter\_length를 101로 설정했기 때문에 기존 코드에 있는 length 길이 정의를 101로 수정하였습니다.

1. 계수 값 변경 및 HPF 추가

**static** **float** LPF\_coeffs[FILTER\_LENGTH]

**static** **float** HPF\_coeffs[FILTER\_LENGTH]

기존에 있던 LPF 외에 추가적으로 HPF가 필요하기 때문에 추가하였습니다.

또한 매트랩에서 LPF, HPF 배열을 “ ‘ “를 이용하여 출력한 텍스트 값을 복사하여 계수값을 정의하였습니다.

1. Main 문 내의 함수의 매개변수 변경

**arm\_fir\_init\_f32**(&fir\_instance\_left, FILTER\_LENGTH, &LPF\_coeffs[0], &fir\_state\_left[0], FRAME\_LENGTH\_FLOAT); // Init. for left FIR filter

**arm\_fir\_init\_f32**(&fir\_instance\_right, FILTER\_LENGTH, &HPF\_coeffs[0], &fir\_state\_right[0], FRAME\_LENGTH\_FLOAT); // Init. for right FIR filter

Main 문 내의 함수의 매개변수 변경을 통해서 왼쪽, 오른쪽 필터 각각 적용합니다. 왼쪽은 LPF, 오른쪽은 HPF를 적용합니다.

1. While 문 내의 modulation 구문 추가

//modulation

**for** (**int** i = 0 ; i < FRAME\_LENGTH\_FLOAT ; i++) {

right\_buf\_out[i] = **pow**(-1, i)\*right\_buf\_out[i];

}

Modulation 구문을 통해 HPF한 출력버퍼를 쉬프트시켜 저역대로 끌어옵니다.

### Build 및 Run 수행

완성된 코드를 build하고 run하여 ‘twoChannelMusic\_46875Hz.mp3’ 을 재생시켜 테스트합니다.

### 최종 전체 코드

1. **#include** "main.h"
2. **#include** "usb\_host.h"
3. **#define** **ARM\_MATH\_CM4** // Define Coretex M4 architecture
4. **#include** "arm\_math.h" // Include advanced math header
5. **I2C\_HandleTypeDef** hi2c1;
6. **I2S\_HandleTypeDef** hi2s2;
7. **DMA\_HandleTypeDef** hdma\_i2s2\_ext\_rx;
8. **DMA\_HandleTypeDef** hdma\_spi2\_tx;
9. **SPI\_HandleTypeDef** hspi1;
10. **void** **SystemClock\_Config**(**void**);
11. **static** **void** **MX\_GPIO\_Init**(**void**);
12. **static** **void** **MX\_DMA\_Init**(**void**);
13. **static** **void** **MX\_I2C1\_Init**(**void**);
14. **static** **void** **MX\_SPI1\_Init**(**void**);
15. **static** **void** **MX\_I2S2\_Init**(**void**);
16. **void** **MX\_USB\_HOST\_Process**(**void**);
17. **#define** **FILTER\_LENGTH** 101 // Filter length
18. **#define** **FRAME\_LENGTH\_FLOAT** 512 // Processing frame length
19. **#define** **FRAME\_LENGTH\_U16** 2048 // integer frame length ( one data = 2 integers \* 2 ) left & right
20. **static** **float** LPF\_coeffs[FILTER\_LENGTH] = { // LPF coefficients
21. 0.000509052345141,
22. 0.000371467877138,
23. -0.000000000000000,
24. -0.000421930966378,
25. -0.000653272863360,
26. -0.000512505037478,
27. 0.000000000000000,
28. 0.000647601620055,
29. 0.001037034522740,
30. 0.000831895989141,
31. -0.000000000000000,
32. -0.001070425301757,
33. -0.001713651304070,
34. -0.001368728488021,
35. 0.000000000000000,
36. 0.001733043896939,
37. 0.002746633796917,
38. 0.002170588290959,
39. -0.000000000000000,
40. -0.002689952801760,
41. -0.004219324720115,
42. -0.003301670957015,
43. 0.000000000000000,
44. 0.004019046406006,
45. 0.006254446687866,
46. 0.004859383930577,
47. -0.000000000000000,
48. -0.005845867111595,
49. -0.009055859070559,
50. -0.007010511370258,
51. 0.000000000000000,
52. 0.008398982088119,
53. 0.013006613956334,
54. 0.010078810760424,
55. -0.000000000000000,
56. -0.012154177342397,
57. -0.018935270880269,
58. -0.014794326344006,
59. 0.000000000000000,
60. 0.018292964523664,
61. 0.029020686598050,
62. 0.023206828997117,
63. -0.000000000000000,
64. -0.030731760980923,
65. -0.051313387567824,
66. -0.043981293968470,
67. 0.000000000000000,
68. 0.074379480369694,
69. 0.158501841074448,
70. 0.224767270549089,
71. 0.249880485591675,
72. 0.224767270549089,
73. 0.158501841074448,
74. 0.074379480369694,
75. 0.000000000000000,
76. -0.043981293968470,
77. -0.051313387567824,
78. -0.030731760980923,
79. -0.000000000000000,
80. 0.023206828997117,
81. 0.029020686598050,
82. 0.018292964523664,
83. 0.000000000000000,
84. -0.014794326344006,
85. -0.018935270880269,
86. -0.012154177342397,
87. -0.000000000000000,
88. 0.010078810760424,
89. 0.013006613956334,
90. 0.008398982088119,
91. 0.000000000000000,
92. -0.007010511370258,
93. -0.009055859070559,
94. -0.005845867111595,
95. -0.000000000000000,
96. 0.004859383930577,
97. 0.006254446687866,
98. 0.004019046406006,
99. 0.000000000000000,
100. -0.003301670957015,
101. -0.004219324720115,
102. -0.002689952801760,
103. -0.000000000000000,
104. 0.002170588290959,
105. 0.002746633796917,
106. 0.001733043896939,
107. 0.000000000000000,
108. -0.001368728488021,
109. -0.001713651304070,
110. -0.001070425301757,
111. -0.000000000000000,
112. 0.000831895989141,
113. 0.001037034522740,
114. 0.000647601620055,
115. 0.000000000000000,
116. -0.000512505037478,
117. -0.000653272863360,
118. -0.000421930966378,
119. -0.000000000000000,
120. 0.000371467877138,
121. 0.000509052345141
122. };
123. **static** **float** HPF\_coeffs[FILTER\_LENGTH] = { // HPF coefficients
124. 0.000509052345141,
125. -0.000371467877138,
126. -0.000000000000000,
127. 0.000421930966378,
128. -0.000653272863360,
129. 0.000512505037478,
130. -0.000000000000000,
131. -0.000647601620055,
132. 0.001037034522740,
133. -0.000831895989141,
134. 0.000000000000000,
135. 0.001070425301757,
136. -0.001713651304070,
137. 0.001368728488021,
138. -0.000000000000000,
139. -0.001733043896939,
140. 0.002746633796917,
141. -0.002170588290959,
142. -0.000000000000000,
143. 0.002689952801760,
144. -0.004219324720115,
145. 0.003301670957015,
146. -0.000000000000000,
147. -0.004019046406006,
148. 0.006254446687867,
149. -0.004859383930577,
150. -0.000000000000000,
151. 0.005845867111595,
152. -0.009055859070559,
153. 0.007010511370258,
154. -0.000000000000000,
155. -0.008398982088119,
156. 0.013006613956334,
157. -0.010078810760424,
158. -0.000000000000000,
159. 0.012154177342397,
160. -0.018935270880269,
161. 0.014794326344006,
162. -0.000000000000000,
163. -0.018292964523664,
164. 0.029020686598050,
165. -0.023206828997117,
166. -0.000000000000000,
167. 0.030731760980923,
168. -0.051313387567825,
169. 0.043981293968471,
170. -0.000000000000000,
171. -0.074379480369694,
172. 0.158501841074448,
173. -0.224767270549090,
174. 0.249880485591676,
175. -0.224767270549090,
176. 0.158501841074448,
177. -0.074379480369694,
178. -0.000000000000000,
179. 0.043981293968471,
180. -0.051313387567825,
181. 0.030731760980923,
182. -0.000000000000000,
183. -0.023206828997117,
184. 0.029020686598050,
185. -0.018292964523664,
186. -0.000000000000000,
187. 0.014794326344006,
188. -0.018935270880269,
189. 0.012154177342397,
190. -0.000000000000000,
191. -0.010078810760424,
192. 0.013006613956334,
193. -0.008398982088119,
194. -0.000000000000000,
195. 0.007010511370258,
196. -0.009055859070559,
197. 0.005845867111595,
198. -0.000000000000000,
199. -0.004859383930577,
200. 0.006254446687867,
201. -0.004019046406006,
202. -0.000000000000000,
203. 0.003301670957015,
204. -0.004219324720115,
205. 0.002689952801760,
206. -0.000000000000000,
207. -0.002170588290959,
208. 0.002746633796917,
209. -0.001733043896939,
210. -0.000000000000000,
211. 0.001368728488021,
212. -0.001713651304070,
213. 0.001070425301757,
214. 0.000000000000000,
215. -0.000831895989141,
216. 0.001037034522740,
217. -0.000647601620055,
218. -0.000000000000000,
219. 0.000512505037478,
220. -0.000653272863360,
221. 0.000421930966378,
222. -0.000000000000000,
223. -0.000371467877138,
224. 0.000509052345141
225. };
226. **arm\_fir\_instance\_f32** fir\_instance\_left, fir\_instance\_right; // Filter instance
227. //FIR state size = processing frame length + filter length - 1
228. **float** fir\_state\_left [FRAME\_LENGTH\_FLOAT + FILTER\_LENGTH - 1]; // Left filter
229. **float** fir\_state\_right [FRAME\_LENGTH\_FLOAT + FILTER\_LENGTH - 1]; // Right filter
230. **uint16\_t** rxBuf[FRAME\_LENGTH\_U16\*2]; // ADC buffer for left & right (Two buffers for half and full complete callback)
231. **uint16\_t** txBuf[FRAME\_LENGTH\_U16\*2]; // DAC buffer for left & right (Two buffers for half and full complete callback)
232. **float** left\_buf\_in [FRAME\_LENGTH\_FLOAT]; // Filter input buffer for left
233. **float** right\_buf\_in [FRAME\_LENGTH\_FLOAT]; // Filter input buffer for right
234. **float** left\_buf\_out [FRAME\_LENGTH\_FLOAT]; // Filter output buffer for left
235. **float** right\_buf\_out [FRAME\_LENGTH\_FLOAT]; // Filter output buffer for right
236. **uint8\_t** state = 0; // ADC/DAC state for half or full complete callback
237. **int** **main**(**void**)
238. {
239. **int** offset\_read\_ptr; // Read pointer offset
240. **int** buf\_ptr;
241. **HAL\_Init**();
242. **SystemClock\_Config**();
243. **MX\_GPIO\_Init**();
244. **MX\_DMA\_Init**();
245. **MX\_I2C1\_Init**();
246. **MX\_SPI1\_Init**();
247. **MX\_USB\_HOST\_Init**();
248. **MX\_I2S2\_Init**();
249. **arm\_fir\_init\_f32**(&fir\_instance\_left, FILTER\_LENGTH, &LPF\_coeffs[0], &fir\_state\_left[0], FRAME\_LENGTH\_FLOAT); // Init. for left FIR filter
250. **arm\_fir\_init\_f32**(&fir\_instance\_right, FILTER\_LENGTH, &HPF\_coeffs[0], &fir\_state\_right[0], FRAME\_LENGTH\_FLOAT); // Init. for right FIR filter
251. **HAL\_I2SEx\_TransmitReceive\_DMA**(&hi2s2, txBuf, rxBuf,FRAME\_LENGTH\_U16); // DMA setting (CAREFUL in DMA size!!!)
252. **while** (1)
253. {
254. **MX\_USB\_HOST\_Process**();
255. **if** (state != 0) { // If ADC/DAC is not ready, do nothing
256. **if** (state == 1) { // In half complete call back
257. offset\_read\_ptr = 0;
258. }
259. **else** **if** (state == 2) { // In full complete call back
260. offset\_read\_ptr = FRAME\_LENGTH\_U16;
261. }
262. buf\_ptr = 0;
263. //Bring data from ADC buffer to floating buffer (2\*16bit integer -> one floating data)
264. **for** (**int** i=offset\_read\_ptr; i<offset\_read\_ptr+FRAME\_LENGTH\_U16; i=i+4) {
265. left\_buf\_in[buf\_ptr] = (**float**) ((**int**) (rxBuf[i]<<16)|rxBuf[i+1]);
266. right\_buf\_in[buf\_ptr] = (**float**) ((**int**) (rxBuf[i+2]<<16)|rxBuf[i+3]);
267. buf\_ptr++;
268. }
269. //DO FIR
270. **arm\_fir\_f32** (&fir\_instance\_left, &left\_buf\_in[0], &left\_buf\_out[0],FRAME\_LENGTH\_FLOAT);
271. **arm\_fir\_f32** (&fir\_instance\_right, &right\_buf\_in[0], &right\_buf\_out[0],FRAME\_LENGTH\_FLOAT);
272. //modulation
273. **for** (**int** i = 0 ; i < FRAME\_LENGTH\_FLOAT ; i++) {
274. right\_buf\_out[i] = **pow**(-1, i)\*right\_buf\_out[i];
275. }
276. buf\_ptr = 0;
277. //Send data to DAC buffer to integer buffer (one floating data -> 2\*16bit integer)
278. **for** (**int** i=offset\_read\_ptr; i<offset\_read\_ptr+FRAME\_LENGTH\_U16; i=i+4) {
279. txBuf[i] = (((**int**)left\_buf\_out[buf\_ptr])>>16)&0xFFFF;
280. txBuf[i+1] = ((**int**)left\_buf\_out[buf\_ptr])&0xFFFF;
281. txBuf[i+2] = (((**int**)right\_buf\_out[buf\_ptr])>>16)&0xFFFF;
282. txBuf[i+3] = ((**int**)right\_buf\_out[buf\_ptr])&0xFFFF;
283. buf\_ptr++;
284. }
285. state = 0;
286. }
287. }
288. }
289. **void** **HAL\_I2SEx\_TxRxHalfCpltCallback**(**I2S\_HandleTypeDef** \*hi2s){
290. state = 1; // Half complete callback state
291. }
292. **void** **HAL\_I2SEx\_TxRxCpltCallback**(**I2S\_HandleTypeDef** \*hi2s){
293. state = 2; // Full complete callback state
294. }
295. **void** **SystemClock\_Config**(**void**)
296. {
297. **RCC\_OscInitTypeDef** RCC\_OscInitStruct = {0};
298. **RCC\_ClkInitTypeDef** RCC\_ClkInitStruct = {0};
299. /\*\* Configure the main internal regulator output voltage
300. \*/
301. \_\_HAL\_RCC\_PWR\_CLK\_ENABLE();
302. \_\_HAL\_PWR\_VOLTAGESCALING\_CONFIG(PWR\_REGULATOR\_VOLTAGE\_SCALE1);
303. /\*\* Initializes the RCC Oscillators according to the specified parameters
304. \* in the RCC\_OscInitTypeDef structure.
305. \*/
306. RCC\_OscInitStruct.OscillatorType = RCC\_OSCILLATORTYPE\_HSE;
307. RCC\_OscInitStruct.HSEState = RCC\_HSE\_ON;
308. RCC\_OscInitStruct.PLL.PLLState = RCC\_PLL\_ON;
309. RCC\_OscInitStruct.PLL.PLLSource = RCC\_PLLSOURCE\_HSE;
310. RCC\_OscInitStruct.PLL.PLLM = 8;
311. RCC\_OscInitStruct.PLL.PLLN = 336;
312. RCC\_OscInitStruct.PLL.PLLP = RCC\_PLLP\_DIV2;
313. RCC\_OscInitStruct.PLL.PLLQ = 7;
314. **if** (**HAL\_RCC\_OscConfig**(&RCC\_OscInitStruct) != *HAL\_OK*)
315. {
316. **Error\_Handler**();
317. }
318. /\*\* Initializes the CPU, AHB and APB buses clocks
319. \*/
320. RCC\_ClkInitStruct.ClockType = RCC\_CLOCKTYPE\_HCLK|RCC\_CLOCKTYPE\_SYSCLK
321. |RCC\_CLOCKTYPE\_PCLK1|RCC\_CLOCKTYPE\_PCLK2;
322. RCC\_ClkInitStruct.SYSCLKSource = RCC\_SYSCLKSOURCE\_PLLCLK;
323. RCC\_ClkInitStruct.AHBCLKDivider = RCC\_SYSCLK\_DIV1;
324. RCC\_ClkInitStruct.APB1CLKDivider = RCC\_HCLK\_DIV4;
325. RCC\_ClkInitStruct.APB2CLKDivider = RCC\_HCLK\_DIV2;
326. **if** (**HAL\_RCC\_ClockConfig**(&RCC\_ClkInitStruct, FLASH\_LATENCY\_5) != *HAL\_OK*)
327. {
328. **Error\_Handler**();
329. }
330. }
331. /\*\*
332. \* @brief I2C1 Initialization Function
333. \* @param None
334. \* @retval None
335. \*/
336. **static** **void** **MX\_I2C1\_Init**(**void**)
337. {
338. /\* USER CODE BEGIN I2C1\_Init 0 \*/
339. /\* USER CODE END I2C1\_Init 0 \*/
340. /\* USER CODE BEGIN I2C1\_Init 1 \*/
341. /\* USER CODE END I2C1\_Init 1 \*/
342. hi2c1.Instance = I2C1;
343. hi2c1.Init.ClockSpeed = 100000;
344. hi2c1.Init.DutyCycle = I2C\_DUTYCYCLE\_2;
345. hi2c1.Init.OwnAddress1 = 0;
346. hi2c1.Init.AddressingMode = I2C\_ADDRESSINGMODE\_7BIT;
347. hi2c1.Init.DualAddressMode = I2C\_DUALADDRESS\_DISABLE;
348. hi2c1.Init.OwnAddress2 = 0;
349. hi2c1.Init.GeneralCallMode = I2C\_GENERALCALL\_DISABLE;
350. hi2c1.Init.NoStretchMode = I2C\_NOSTRETCH\_DISABLE;
351. **if** (**HAL\_I2C\_Init**(&hi2c1) != *HAL\_OK*)
352. {
353. **Error\_Handler**();
354. }
355. /\* USER CODE BEGIN I2C1\_Init 2 \*/
356. /\* USER CODE END I2C1\_Init 2 \*/
357. }
358. /\*\*
359. \* @brief I2S2 Initialization Function
360. \* @param None
361. \* @retval None
362. \*/
363. **static** **void** **MX\_I2S2\_Init**(**void**)
364. {
365. /\* USER CODE BEGIN I2S2\_Init 0 \*/
366. /\* USER CODE END I2S2\_Init 0 \*/
367. /\* USER CODE BEGIN I2S2\_Init 1 \*/
368. /\* USER CODE END I2S2\_Init 1 \*/
369. hi2s2.Instance = SPI2;
370. hi2s2.Init.Mode = I2S\_MODE\_MASTER\_TX;
371. hi2s2.Init.Standard = I2S\_STANDARD\_PHILIPS;
372. hi2s2.Init.DataFormat = I2S\_DATAFORMAT\_24B;
373. hi2s2.Init.MCLKOutput = I2S\_MCLKOUTPUT\_ENABLE;
374. hi2s2.Init.AudioFreq = I2S\_AUDIOFREQ\_48K;
375. hi2s2.Init.CPOL = I2S\_CPOL\_LOW;
376. hi2s2.Init.ClockSource = I2S\_CLOCK\_PLL;
377. hi2s2.Init.FullDuplexMode = I2S\_FULLDUPLEXMODE\_ENABLE;
378. **if** (**HAL\_I2S\_Init**(&hi2s2) != *HAL\_OK*)
379. {
380. **Error\_Handler**();
381. }
382. /\* USER CODE BEGIN I2S2\_Init 2 \*/
383. /\* USER CODE END I2S2\_Init 2 \*/
384. }
385. /\*\*
386. \* @brief SPI1 Initialization Function
387. \* @param None
388. \* @retval None
389. \*/
390. **static** **void** **MX\_SPI1\_Init**(**void**)
391. {
392. /\* USER CODE BEGIN SPI1\_Init 0 \*/
393. /\* USER CODE END SPI1\_Init 0 \*/
394. /\* USER CODE BEGIN SPI1\_Init 1 \*/
395. /\* USER CODE END SPI1\_Init 1 \*/
396. /\* SPI1 parameter configuration\*/
397. hspi1.Instance = SPI1;
398. hspi1.Init.Mode = SPI\_MODE\_MASTER;
399. hspi1.Init.Direction = SPI\_DIRECTION\_2LINES;
400. hspi1.Init.DataSize = SPI\_DATASIZE\_8BIT;
401. hspi1.Init.CLKPolarity = SPI\_POLARITY\_LOW;
402. hspi1.Init.CLKPhase = SPI\_PHASE\_1EDGE;
403. hspi1.Init.NSS = SPI\_NSS\_SOFT;
404. hspi1.Init.BaudRatePrescaler = SPI\_BAUDRATEPRESCALER\_2;
405. hspi1.Init.FirstBit = SPI\_FIRSTBIT\_MSB;
406. hspi1.Init.TIMode = SPI\_TIMODE\_DISABLE;
407. hspi1.Init.CRCCalculation = SPI\_CRCCALCULATION\_DISABLE;
408. hspi1.Init.CRCPolynomial = 10;
409. **if** (**HAL\_SPI\_Init**(&hspi1) != *HAL\_OK*)
410. {
411. **Error\_Handler**();
412. }
413. /\* USER CODE BEGIN SPI1\_Init 2 \*/
414. /\* USER CODE END SPI1\_Init 2 \*/
415. }
416. /\*\*
417. \* Enable DMA controller clock
418. \*/
419. **static** **void** **MX\_DMA\_Init**(**void**)
420. {
421. /\* DMA controller clock enable \*/
422. \_\_HAL\_RCC\_DMA1\_CLK\_ENABLE();
423. /\* DMA interrupt init \*/
424. /\* DMA1\_Stream3\_IRQn interrupt configuration \*/
425. **HAL\_NVIC\_SetPriority**(*DMA1\_Stream3\_IRQn*, 0, 0);
426. **HAL\_NVIC\_EnableIRQ**(*DMA1\_Stream3\_IRQn*);
427. /\* DMA1\_Stream4\_IRQn interrupt configuration \*/
428. **HAL\_NVIC\_SetPriority**(*DMA1\_Stream4\_IRQn*, 0, 0);
429. **HAL\_NVIC\_EnableIRQ**(*DMA1\_Stream4\_IRQn*);
430. }
431. /\*\*
432. \* @brief GPIO Initialization Function
433. \* @param None
434. \* @retval None
435. \*/
436. **static** **void** **MX\_GPIO\_Init**(**void**)
437. {
438. **GPIO\_InitTypeDef** GPIO\_InitStruct = {0};
439. /\* USER CODE BEGIN MX\_GPIO\_Init\_1 \*/
440. /\* USER CODE END MX\_GPIO\_Init\_1 \*/
441. /\* GPIO Ports Clock Enable \*/
442. \_\_HAL\_RCC\_GPIOE\_CLK\_ENABLE();
443. \_\_HAL\_RCC\_GPIOC\_CLK\_ENABLE();
444. \_\_HAL\_RCC\_GPIOH\_CLK\_ENABLE();
445. \_\_HAL\_RCC\_GPIOA\_CLK\_ENABLE();
446. \_\_HAL\_RCC\_GPIOB\_CLK\_ENABLE();
447. \_\_HAL\_RCC\_GPIOD\_CLK\_ENABLE();
448. /\*Configure GPIO pin Output Level \*/
449. **HAL\_GPIO\_WritePin**(CS\_I2C\_SPI\_GPIO\_Port, CS\_I2C\_SPI\_Pin, *GPIO\_PIN\_RESET*);
450. /\*Configure GPIO pin Output Level \*/
451. **HAL\_GPIO\_WritePin**(OTG\_FS\_PowerSwitchOn\_GPIO\_Port, OTG\_FS\_PowerSwitchOn\_Pin, *GPIO\_PIN\_SET*);
452. /\*Configure GPIO pin Output Level \*/
453. **HAL\_GPIO\_WritePin**(GPIOD, LD4\_Pin|LD3\_Pin|LD5\_Pin|LD6\_Pin
454. |Audio\_RST\_Pin, *GPIO\_PIN\_RESET*);
455. /\*Configure GPIO pin : CS\_I2C\_SPI\_Pin \*/
456. GPIO\_InitStruct.Pin = CS\_I2C\_SPI\_Pin;
457. GPIO\_InitStruct.Mode = GPIO\_MODE\_OUTPUT\_PP;
458. GPIO\_InitStruct.Pull = GPIO\_NOPULL;
459. GPIO\_InitStruct.Speed = GPIO\_SPEED\_FREQ\_LOW;
460. **HAL\_GPIO\_Init**(CS\_I2C\_SPI\_GPIO\_Port, &GPIO\_InitStruct);
461. /\*Configure GPIO pin : OTG\_FS\_PowerSwitchOn\_Pin \*/
462. GPIO\_InitStruct.Pin = OTG\_FS\_PowerSwitchOn\_Pin;
463. GPIO\_InitStruct.Mode = GPIO\_MODE\_OUTPUT\_PP;
464. GPIO\_InitStruct.Pull = GPIO\_NOPULL;
465. GPIO\_InitStruct.Speed = GPIO\_SPEED\_FREQ\_LOW;
466. **HAL\_GPIO\_Init**(OTG\_FS\_PowerSwitchOn\_GPIO\_Port, &GPIO\_InitStruct);
467. /\*Configure GPIO pin : B1\_Pin \*/
468. GPIO\_InitStruct.Pin = B1\_Pin;
469. GPIO\_InitStruct.Mode = GPIO\_MODE\_EVT\_RISING;
470. GPIO\_InitStruct.Pull = GPIO\_NOPULL;
471. **HAL\_GPIO\_Init**(B1\_GPIO\_Port, &GPIO\_InitStruct);
472. /\*Configure GPIO pin : I2S3\_WS\_Pin \*/
473. GPIO\_InitStruct.Pin = I2S3\_WS\_Pin;
474. GPIO\_InitStruct.Mode = GPIO\_MODE\_AF\_PP;
475. GPIO\_InitStruct.Pull = GPIO\_NOPULL;
476. GPIO\_InitStruct.Speed = GPIO\_SPEED\_FREQ\_LOW;
477. GPIO\_InitStruct.Alternate = GPIO\_AF6\_SPI3;
478. **HAL\_GPIO\_Init**(I2S3\_WS\_GPIO\_Port, &GPIO\_InitStruct);
479. /\*Configure GPIO pin : BOOT1\_Pin \*/
480. GPIO\_InitStruct.Pin = BOOT1\_Pin;
481. GPIO\_InitStruct.Mode = GPIO\_MODE\_INPUT;
482. GPIO\_InitStruct.Pull = GPIO\_NOPULL;
483. **HAL\_GPIO\_Init**(BOOT1\_GPIO\_Port, &GPIO\_InitStruct);
484. /\*Configure GPIO pins : LD4\_Pin LD3\_Pin LD5\_Pin LD6\_Pin
485. Audio\_RST\_Pin \*/
486. GPIO\_InitStruct.Pin = LD4\_Pin|LD3\_Pin|LD5\_Pin|LD6\_Pin
487. |Audio\_RST\_Pin;
488. GPIO\_InitStruct.Mode = GPIO\_MODE\_OUTPUT\_PP;
489. GPIO\_InitStruct.Pull = GPIO\_NOPULL;
490. GPIO\_InitStruct.Speed = GPIO\_SPEED\_FREQ\_LOW;
491. **HAL\_GPIO\_Init**(GPIOD, &GPIO\_InitStruct);
492. /\*Configure GPIO pins : I2S3\_MCK\_Pin I2S3\_SCK\_Pin I2S3\_SD\_Pin \*/
493. GPIO\_InitStruct.Pin = I2S3\_MCK\_Pin|I2S3\_SCK\_Pin|I2S3\_SD\_Pin;
494. GPIO\_InitStruct.Mode = GPIO\_MODE\_AF\_PP;
495. GPIO\_InitStruct.Pull = GPIO\_NOPULL;
496. GPIO\_InitStruct.Speed = GPIO\_SPEED\_FREQ\_LOW;
497. GPIO\_InitStruct.Alternate = GPIO\_AF6\_SPI3;
498. **HAL\_GPIO\_Init**(GPIOC, &GPIO\_InitStruct);
499. /\*Configure GPIO pin : OTG\_FS\_OverCurrent\_Pin \*/
500. GPIO\_InitStruct.Pin = OTG\_FS\_OverCurrent\_Pin;
501. GPIO\_InitStruct.Mode = GPIO\_MODE\_INPUT;
502. GPIO\_InitStruct.Pull = GPIO\_NOPULL;
503. **HAL\_GPIO\_Init**(OTG\_FS\_OverCurrent\_GPIO\_Port, &GPIO\_InitStruct);
504. /\*Configure GPIO pin : MEMS\_INT2\_Pin \*/
505. GPIO\_InitStruct.Pin = MEMS\_INT2\_Pin;
506. GPIO\_InitStruct.Mode = GPIO\_MODE\_EVT\_RISING;
507. GPIO\_InitStruct.Pull = GPIO\_NOPULL;
508. **HAL\_GPIO\_Init**(MEMS\_INT2\_GPIO\_Port, &GPIO\_InitStruct);
509. /\* USER CODE BEGIN MX\_GPIO\_Init\_2 \*/
510. /\* USER CODE END MX\_GPIO\_Init\_2 \*/
511. }
512. /\* USER CODE BEGIN 4 \*/
513. /\* USER CODE END 4 \*/
514. /\*\*
515. \* @brief This function is executed in case of error occurrence.
516. \* @retval None
517. \*/
518. **void** **Error\_Handler**(**void**)
519. {
520. /\* USER CODE BEGIN Error\_Handler\_Debug \*/
521. /\* User can add his own implementation to report the HAL error return state \*/
522. **\_\_disable\_irq**();
523. **while** (1)
524. {
525. }
526. /\* USER CODE END Error\_Handler\_Debug \*/
527. }
528. **#ifdef** USE\_FULL\_ASSERT
529. /\*\*
530. \* @brief Reports the name of the source file and the source line number
531. \* where the assert\_param error has occurred.
532. \* @param file: pointer to the source file name
533. \* @param line: assert\_param error line source number
534. \* @retval None
535. \*/
536. **void** assert\_failed(uint8\_t \*file, uint32\_t line)
537. {
538. /\* USER CODE BEGIN 6 \*/
539. /\* User can add his own implementation to report the file name and line number,
540. ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) \*/
541. /\* USER CODE END 6 \*/
542. }
543. **#endif** /\* USE\_FULL\_ASSERT \*/