### UCSD-Embedded-C-Final-Assignment Audio Helper

# 5/25/2020

## Introduction:

My final project is a idea that was really born from having to stay home and having my daughter staying home from school during this COVID-19 shelter in place order. My daughter loves to listen to her music with her earphones in and cut out us parents and my wife usually ends up have to almost scream to get her attention. So that got me thinking that if I could make a device that "listens" for an event then it could pause music and even pipe in what is being said into the headphones.

In doing some research about using the microphones that are on the class development board turned out to be rather complex and there is no software support in the BSP package to use the microphones. So I would have to make a driver from the ground up and with the amount of time I had this was not a option. So I did the next best thing and used a analog microphone that will be sampled by the STM32 ADC1. I also did not have time to integrate an audio DAC that would pipe in the conversation audio.

## **System Overview:**

The system I made has the basic functions to find the average baseline "noise" of the surrounding environment, then while continually sampling the audio microphone, if a average sample is higher then the baseline the system enters a triggered mode. This is where it sets a time out for the audio event if the audio sample is still higher then the baseline at the end of the timeout then action is taken. The system action is to light the red LED, pause the source audio, switch the audio channels over to the message audio source, play message, wait for user to press the blue button, switch audio back to source, red LED off, and play source audio. The system then restates and goes back to looking for a trigger event again.

#### Hardware Setup:

The project was centered around the B-L475E-IOT01A1 STM32 Development board. I used several features of the STM32 Microcontroller which are: ADC1, UART4, Timer6, and GPIO (Arduino GPIO Header and the Blue Button). The external hardware I used was 2x 4 conductor headphone adapter boards, 2x SN74HC4066DR analog switch chips, Red LED, and EMIC Text to Speech module as my stand in DAC. The EMIC module will speak any ASCII text string you send it over a UART interface. This is what I used to also alert the user of someone trying to get there attention.



This shows the whole working system.



	y R 3 GNE Right	by Buck set 2205 ) Channel Channel	₹600 ▼	
	Function Chart			
Function	Ideal R	R Ronge	Approx DC Volt	
Dormant	OPen	open	2,5	
Play/Pause	NO	0-98N	,004	
Skip Buck	2205	133-3272	0.179	
SIC: Pform	6002	437-7112	0.429	

This schematic shows how the pause command is sent to the smart phone or device. By bring the 4<sup>th</sup> signal pin to GND which will pause the play of audio. I found that ~400mS was the right amount to pause and then play the audio on my phone. If the signal is low for longer the phone will skip to the next song which was a undesired command.



This shows the overall schematic of the audio switch over circuit and pause switch.

## **Embedded Code Configuration:**

After doing some more research on the STM32 ADC and Audio Sampling I implemented the following system. I took advantage of a couple key components of the ADC system: the ADC can be clocked at a high rate, ADC can over sample, and the ADC has DMA support. I set up the ADC to provide 48KHz audio samples. I referenced ST App note AN5012 page 17 to help with the ADC setup. I then used the DMA to create a circular audio buffer that is filled with out the use of the CPU. The microphone is sampled continually in the back ground with no CPU intervention. The CPU does have to calculate the averages of the audio at buffer half full and buffer full Interrupts.

The CPU just checks the average audio samples that come out and checks to see it the sample is higher then the trigger level. If so then runs the algorithm to alert the user.



I used STM32CubeMX to configure all the Microcontroller Peripherals. Shown is how the clocks where configured to clock the ADC at 76.8MHz.

ADC1 Mode a	and Configuration
	1ode
IN1 Disable	
IN2 Disable	
IN3 Disable	
IN4 Disable	~
IN5 Disable	~
Confi	guration 4
Reset Configuration	1
📀 Parameter Settings 🛛 😔 User Constants 🛛 😔 🕅	VIC Settings 🛛 😔 DMA Settings 🛛 😔 GPIO Settings 💦 🥫
Configure the below parameters :	
Q Search (Crtl+F) ③ ③	0
Clock Prescaler	v
Clock Prescaler Resolution	Asynchronous clock mode divided by 1 ADC 12-bit resolution
Data Alignment	Right alignment
Scan Conversion Mode	Enabled
Continuous Conversion Mode	Disabled
Discontinuous Conversion Mode	Disabled
DMA Continuous Requests	Enabled
End Of Conversion Selection	End of single conversion
Overrun behaviour	Overrun data preserved
Low Power Auto Wait	Disabled
V ADC_Regular_ConversionMode	
Enable Regular Conversions	Enable
Enable Regular Oversampling	Enable
Oversampling Right Shift	2 bit shift for oversampling
Oversampling Ratio	Oversampling ratio 64x
Regular Oversampling Mode	Oversampling Continued Mode
Triggered Regular Oversampling	Single trigger for all oversampled conversions
Number Of Conversion	2
External Trigger Conversion Source	Regular Conversion launched by software
External Trigger Conversion Edge	None
> Rank	1
> Rank	2
~ ADC_Injected_ConversionMode	
Enable Injected Conversions	Disable
✓ Analog Watchdog 1	U

Shown is the ADC baseline configuration.

	ADC1 N	Mode and Configurat	ion	
		Mode		
IN1 Disable				~
IN2 Disable				$\sim$
IN3 Disable				~
IN4 Disable				$\sim$
IN5 Disable				$\sim$
		Configuration		
Reset Configuration		conngulation		
🥺 Parameter Settings	📀 User Constants	🥝 NVIC Settings	🔗 DMA Settings	📀 GPIO Settings
DMA Request	Channel	Dir	ection	Priority
ADC1	DMA1 Channel 1	Peripheral T	o Memory Low	r

ADC DMA is configured. I used the circular buffer setting.

ADC1 Mode and Configuration		
Mode		
IN1 Disable	$\sim$	
IN2 Disable	$\sim$	
IN3 Disable	$\sim$	
IN4 Disable	$\sim$	
N5 Disable	$\sim$	
		_
Configuration		UART
Reset Configuration		
📀 Parameter Settings 🛛 📀 User Constants 🛛 📀 NVIC Settings 🛛 📀 DMA Settings 📄 😒 GPIO Settin	gs .	
Search Signals		D.
Search (Crtl+F) Show only Modif	led	PINS
	lodif	ied
PC5 ADC1_IN14 n/a Analog m No pull-up n/a n/a ARD_A0 [	$\checkmark$	

ADC input pin configuration.

TIM6 Mode and Configuration		
	Mode	
✓ Activated		
🗌 One Pulse Mode		
	Configuration	
Reset Configuration		
	🥺 NVIC Settings 🛛 😔 DMA Settings	
Configure the below parameters :		
Q Search (Crtl+F) ③		0
<ul> <li>Counter Settings</li> </ul>		
Prescaler (PSC - 16 bits value)	19200	
Counter Mode	Up	
Counter Period (AutoReload Register - 1		
auto-reload preload <ul> <li>Trigger Output (TRGO) Parameters</li> </ul>	Enable	
Trigger Event Selection	Reset (UG bit from TIMx_EGR)	

Timer 6 is used for the audio trigger timeout timer. The timer period was set for a second timeout. This was used to try and filter out load noise spikes that would give a false trigger. So if the trigger audio was still "load" at the end of the timeout then was considered a real trigger event. I did try other timeout period values as well. The timer prescaler is set to 1ms so the timer is incremented every millisecond.

	Mode and Configuration Mode	
Mode Asynchronous		×
Hardware Flow Control (RS232) Disable		
		Ý
Hardware Flow Control (RS485)		
	Configuration	
Reset Configuration		
	📀 NVIC Settings 🛛 📀 DMA Settings 🛛 📀 (	SPIO Settings
onfigure the below parameters :		
Search (Crtl+F)		0
Basic Parameters		
Baud Rate	2400 Bits/s	
Word Length	8 Bits (including Parity)	
Parity	None	
Stop Bits	1	
<ul> <li>Advanced Parameters</li> </ul>		
Data Direction	Receive and Transmit	
Over Sampling	16 Samples	
Single Sample	Disable	
<ul> <li>Advanced Features</li> </ul>		
Auto Baudrate	Disable	
TX Pin Active Level Inversion	Disable	
RX Pin Active Level Inversion	Disable	
	Disable	
Data Inversion	Disable	
Data Inversion TX and RX Pins Swapping		
	Enable	
TX and RX Pins Swapping		

UART4 is used to send the ASCII text strings to the EMIC Text To Speech Module The baud rate is set to a slow 2400 baud as the module is old.

## Embedded Code:

```
64
65 /* USER CODE BEGIN PV */
66 uint16_t MyMicValues[256];
67 uint32_t AudioSampleAccum1 = 0, AudioSampleAccum2 = 0;
68 uint16_t AudioAv1, AudioAv2, AudioAvTotal;
69 uint16_t GlobalTriggerLevel;
70 uint16_t DescisionFLAG = 0;
71 uint16_t TriggerFLAG = 0;
72 uint16_t ButtonFLAG = 0;
73 uint16_t NewAudioSampleFLAG = 0;
74
75 uint32_t DescisionLoopCNT= 0;
76 uint32_t DescisionAccum = 0;
77 /* USER CODE END PV */
78
```

The global variables used in the call back functions.

```
5110
         uint16 t TriggerLoopCount = 0;
111
         uint32 t TriggerAccum;
         uint16 t TriggerTotal;
112
113
         uint16 t TriggerLevel;
114
115
116
         char *msg = "say=Please listen for important message;";
         uint16 t len = strlen(msg);
117
         //char *backmsg[128];
118
119
```

Local variables used and the Text To Speech string used.

```
HAL GPIO WritePin(ARD D3 GPIO Port, ARD D3 Pin, SET);
158
159
       HAL GPIO WritePin(ARD D4 GPIO Port, ARD D4 Pin, RESET);
       HAL GPIO WritePin(ARD D5 GPIO Port, ARD D5 Pin, RESET);
160
161
162
163
       //HAL TIM Base Start IT(&htim6);
164
       HAL ADCEx Calibration Start(&hadc1, ADC SINGLE ENDED);
165
       HAL ADC Start DMA(&hadc1, (uint32 t*)MyMicValues, BUF LEN);
166
167
       //HAL ADC Start(&hadc1);
168
169
170
       while (TriggerLoopCount <= 1024)</pre>
171
      {
172
           //if (NewAudioSampleFLAG)
173
           111
               TriggerAccum = TriggerAccum + AudioAvTotal;
174
175
               HAL Delay(15);
               TriggerLoopCount++;
176
               HAL GPIO TogglePin(LED2 GPIO Port, LED2 Pin);
177
178
           11}
179
           //NewAudioSampleFLAG = 0;
       }
180
181
182
       // HAL TIM SET AUTORELOAD(&htim6,2000);
183
184
       TriggerTotal = TriggerAccum >> 10;
0185
186
       TriggerLevel = TriggerTotal + 540;
       GlobalTriggerLevel = TriggerLevel;
187
       HAL TIM Base Stop IT(&htim6);
188
189
       TriggerAccum = 0;
       /* USER CODE END 2 */
190
191
```

This is the start up code used. I calibrate the ADC and then start the ADC in DMA mode. This is the only call I need to make to start getting audio samples in the MyMicValues buffer.

The while loop is the code that is finding the average "noise" of the surrounding area. Then lastly is the code to take the average and set the trigger level that will be used.

```
while (1)
194
195
      {
           if (AudioAvTotal > TriggerLevel)
196
197
           {
               //HAL GPIO TogglePin(LED2 GPIO Port, LED2 Pin);
198
 199
              HAL TIM Base Start IT(&htim6);
 200
               while(!DescisionFLAG);
201
                 // if (NewAudioSampleFLAG)
202
 203
                  111
                    // DescisionAccum = DescisionAccum + AudioAvTotal;
204
                      //DescisionLoopCNT++;
205
                  117
206
                  //NewAudioSampleFLAG = 0;
207
               11}
 208
209
              HAL TIM Base Stop IT(&htim6);
              if (TriggerFLAG)
210
211
               {
212
                   //HAL TIM Base Stop IT(&htim6);
                  HAL GPIO WritePin(ARD D8_GPIO_Port, ARD_D8_Pin, SET); //red LED
213
                  HAL_GPIO_WritePin(ARD_D5_GPIO_Port, ARD_D5_Pin, SET); //pause audio source
214
215
                  HAL Delay(400);
216
                  HAL GPIO WritePin(ARD D5 GPIO Port, ARD D5 Pin, RESET);
217
                  HAL_GPI0_WritePin(ARD_D3_GPI0_Port, ARD_D3_Pin, RESET);
218
 219
                  HAL GPIO WritePin(ARD D4 GPIO Port, ARD D4 Pin, SET); //switch over audio
220
                  for(uint16 t messageLoop =0; messageLoop < len; messageLoop++)</pre>
222
                  {
223
                      HAL UART Transmit(&huart4,(uint8 t *)&msg[messageLoop], 1, 0);
                      //HAL UART Receive(&huart4,(uint8 t *)&backmsg,2,0);
224
                      HAL Delay(5); //should find TX ready flag
225
226
                  }
 227
                  //HAL_UART_Receive(&huart4,(uint8_t *)&backmsg,2,0);
228
                  while(!ButtonFLAG);
229
                  ButtonFLAG = 0;
                  HAL GPIO WritePin(ARD D3_GPIO_Port, ARD_D3_Pin, SET);
230
231
                  HAL GPIO WritePin(ARD D4 GPIO Port, ARD D4 Pin, RESET);
                                                                          //switch over audio
 232
233
                  HAL_GPIO WritePin(ARD D5 GPIO Port, ARD D5 Pin, SET); //un-pause audio source
                  HAL Delay(400):
234
 235
                  HAL GPIO WritePin(ARD D5 GPIO Port, ARD D5 Pin, RESET);
236
237
                  HAL GPIO WritePin(ARD D8 GPIO Port, ARD D8 Pin, RESET);//RED LED
238
                  TriggerFLAG = 0:
239
                  DescisionFLAG = 0;
              }
240
                   else
241
242
                   {
                       // HAL TIM Base Stop IT(&htim6);
243
244
                        HAL GPIO WritePin(ARD D8 GPIO Port, ARD D8 Pin, RESET);
245
                        //TriggerFLAG = 0;
246
                        DescisionFLAG = 0;
247
                   }
              }
248
249
250
              /* USER CODE END WHILE */
251
           /* USER CODE BEGIN 3 */
        }
252
253
        /* USER CODE END 3 */
254 }
255 /*USER CODE BEGIN 4*/
```

This is the main loop which handles the trigger timeout and then the main function of alerting the user and switch the audio. Once a valid trigger is found the code will turn ON the red LED, pause the audio, switch the audio to the Text To Speech module, Play the text message, then wait for the user to press the blue button, the audio this then switched back, and the source audio is set to play.

```
255 /*USER CODE BEGIN 4*/
256 void HAL ADC ConvHalfCpltCallback(ADC HandleTypeDef *hadc)
257 {
        for(uint16 t AvCount = 0; AvCount <= HALF BUFF LEN; AvCount++)</pre>
258
259
        {
            AudioSampleAccum1 = AudioSampleAccum1 + MyMicValues[AvCount];
260
        }
261
        AudioAv1 = AudioSampleAccum1 >> 7;
262
        AudioSampleAccum1 = AudioAv1; //running audio average
263
264 }
265
266 void HAL_ADC_ConvCpltCallback(ADC HandleTypeDef *hadc)
267 {
        for(uint16 t AvCount = HALF BUFF LEN; AvCount <= BUF LEN; AvCount++)</pre>
268
269
        {
            AudioSampleAccum2 = AudioSampleAccum2 + MyMicValues[AvCount];
270
        }
271
        AudioAv2 = AudioSampleAccum2 >> 7;
272
        AudioAvTotal = (AudioAv1 + AudioAv2) >> 1;
273
274
        AudioSampleAccum2 = AudioAv2; //running audio average
        NewAudioSampleFLAG = 1;
275
276
277 }
278
279 void HAL_TIM_PeriodElapsedCallback(TIM HandleTypeDef *htim)
280 {
281
        HAL GPIO TogglePin(LED2 GPIO Port, LED2 Pin);
        //uint32 t temp;
282
        DescisionFLAG = 1;
283
        //temp = (DescisionAccum / DescisionLoopCNT);
284
        if (AudioAvTotal > GlobalTriggerLevel)
285
286
        {
287
            TriggerFLAG = 1;
        }
288
        DescisionAccum = 0;
289
290
        DescisionLoopCNT = 0;
291 }
292
293⊖ void HAL GPIO EXTI Callback(uint16 t GPIO Pin)
294 {
        UNUSED(GPIO Pin);
295
        ButtonFLAG = 1;
296
297 }
298 /*USER CODE END 4*/
```

I used 4 call back functions in my program. The two ADC Interrupt functions are triggered when the ADC DMA has filled half the buffer and when the buffer is full. When the half is triggered the DMA is already filling the next half and when the full is triggered the DMA is filling the bottom half of the buffer again. This works so that new audio samples are always being written into the buffer and the audio sample averages are always being calculated.

The TIM call back is for the Timer 6 period elapsed which is when the period compare register is equal to the timer counter. This signals the end of the audio trigger timeout and sets a flag for the main loop to execute the audio switch over. The GPIO call back is for the blue button to signal the user is done listening to who or what need there attention and is ready to turn back to there music.



Capture of the UART4 serial data stream to the Text To Speech Module.

☆ Debug ⊠	×	⁰₀ Breakpoints (×)= Variab	oles 🛱 🙀 Live Expr	ressions	
<ul> <li>CB-L_ADC_AnalogMic_try1.elf [Embedded C/C++ Application]</li> <li></li></ul>	23)	Name ightarrow TriggerLoopCount $ ightarrow TriggerAccum  ightarrow TriggerTotal  ightarrow TriggerLevelightarrow msg$	Type uint16_t uint32_t uint16_t uint16_t char *	<ul> <li>人社</li> <li>Value</li> <li>1025</li> <li>32182632</li> <li>31428</li> <li>31968</li> <li>0x8006f3c</li> </ul>	
් main.c ස					
<pre>181 182 183 184 185 185 186 185 186 187 186 187 187 188 187 188 187 188 184 188 188 18 18 18 18 18 18 1 1 1 1</pre>					

Shows the debugger running and the "noise" trigger level has been found to be 31968.

Expression	Туре	Value
- 🥭 MyMicValues	uint16_t [256]	0x2000057c <mymicvalues></mymicvalues>
<b>▼</b> 🔚 [099]	uint16_t [100]	0x2000057c <mymicvalues></mymicvalues>
(×)= MyMicValues[0]	uint16_t	30690
(×)= MyMicValues[1]	uint16_t	30860
(≫= MyMicValues[2]	uint16_t	31050
(x)= MyMicValues[3]	uint16_t	31362
(×)= MyMicValues[4]	uint16_t	31323
(≫= MyMicValues[5]	uint16_t	31613
(X)= MyMicValues[6]	uint16_t	31184
(×)= MyMicValues[7]	uint16_t	31017
(≍)= MyMicValues[8]	uint16_t	31125
(×)= MyMicValues[9]	uint16_t	30969
(×)= MyMicValues[10]	uint16_t	30973
(×)= MyMicValues[11]	uint16_t	31187
(≫= MyMicValues[12]	uint16_t	31385
(x)= MyMicValues[13]	uint16_t	31488
⋈=MyMicValues[14]	uint16_t	31332
(×)= MyMicValues[15]	uint16_t	31304
(≈)= MyMicValues[16]	uint16_t	31303
⊮= MyMicValues[17]	uint16_t	31393
(X)= MyMicValues[18]	uint16 t	31158

This is the Audio samples buffer that is filled through DMA.

🕨 🥭 MyMicValues	uint16_t [256]	0x2000057c <mymicvalues></mymicvalues>
⊗=AudioAv1	uint16_t	31582
⊗=AudioAv2	uint16_t	41729
⊗= AudioAvTotal	uint16_t	36655
⊗= TriggerFLAG	uint16_t	1
🛤 DescisionFLAG	uint16_t	1
⋈=NewAudioSampleFLAG	uint16_t	1
⋈= DescisionAccum	uint32_t	0
⋈= DescisionLoopCNT	uint32_t	0
⇔= GlobalTriggerLevel	uint16_t	31968
🖶 Add new expression		

## **Problems and Future Steps:**

The Audio Trigger still has bugs and is hard to find a valid trigger sample because the audio sample buffer is being filled so fast and audio samples are complex. Audio noise in the environment also is causing false triggers but the timeout did help to filter out so spikes. I did not get to really test this system out but the audio pause and audio switching really works well and there is not noticeable sound degradation. I think a better trigger timer code would help to find valid audio triggers in the buffer along with better audio filtering.

I am going to keep working with this project and will look into more ways to process the audio samples to detect when the user needs to be interrupted and when the users does not. Maybe look into Audio Key Word Detection or some kind of Audio FFT Algorithm. I also would like to work with an I2S DAC to use instead of the Text To Speech Module.

#### **Conclusion and Lessons Learned:**

In conclusion I get a audio system to sample a analog microphone and then was able to process that data into an average audio level and make a decision based on that data to control a smart phone audio input. Even through the audio trigger is still has bugs and needs work I think this project was a success. I learned that audio systems are more complex then I originally thought but learned a lot about using embedded code and using the HAL drivers and STM32cubeMX. I hope to keeps this project moving forward and make a more usable system.