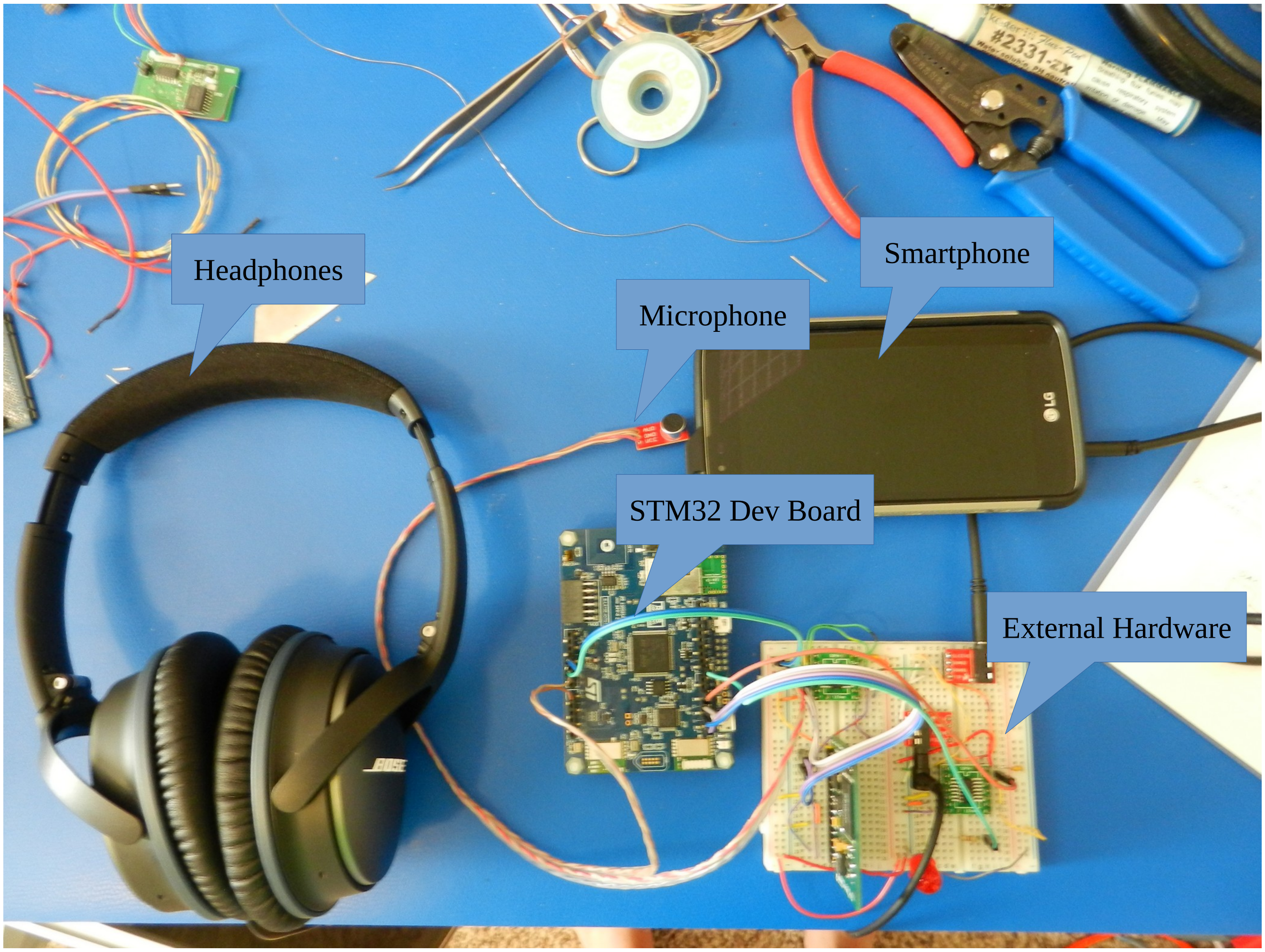
UCSD-Embedded-C-Final-Assignment
Audio Helper

5/25/2020

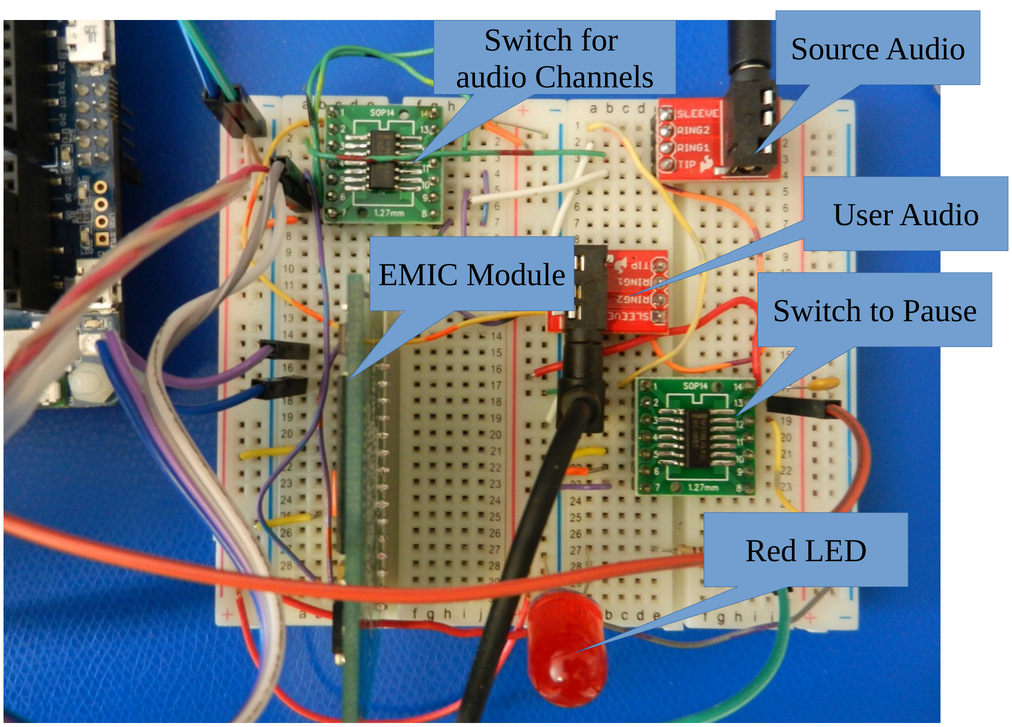
Introduction:
My final project is a idea that was really born from having to stay home and having my daughter staying home from school during this COVID-19 shelter in place order. My daughter loves to listen to her music with her earphones in and cut out us parents and my wife usually ends up have to 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.

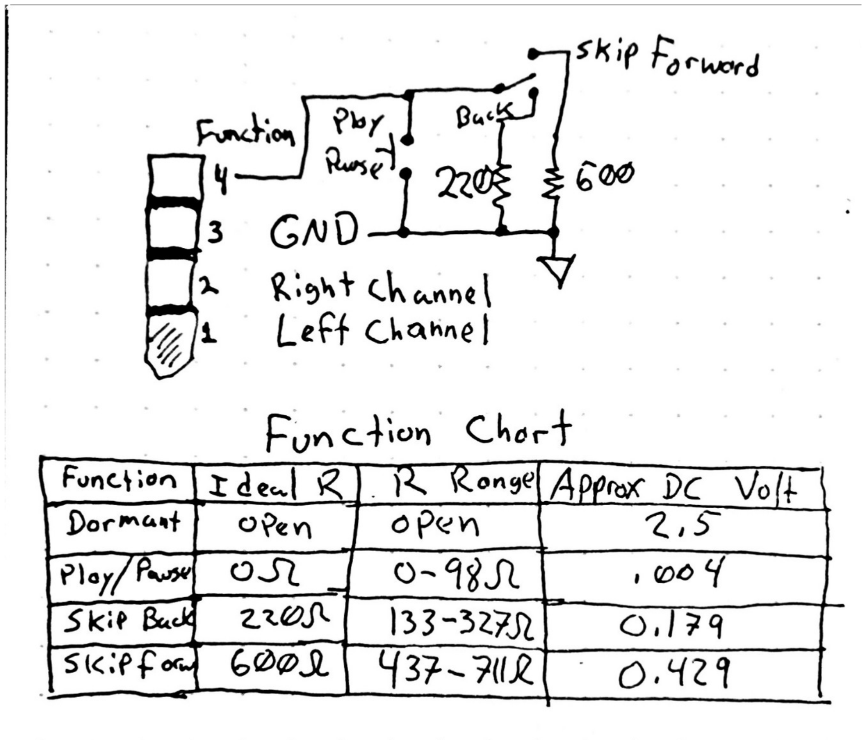
System Overview:
The system I made has the basic functions to find the average baseline “noise” of the surrounding environment, then while continually sampling the audio microphone, if a average sample is higher then the baseline the system enters a triggered mode. This is where it sets a time out for 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.



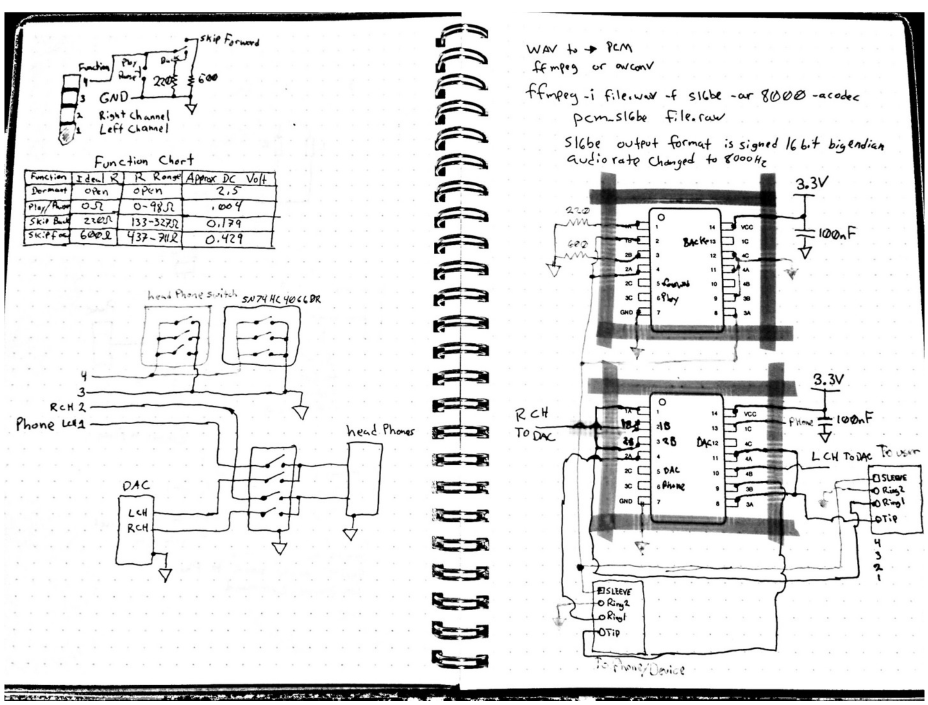
This shows the whole working system.





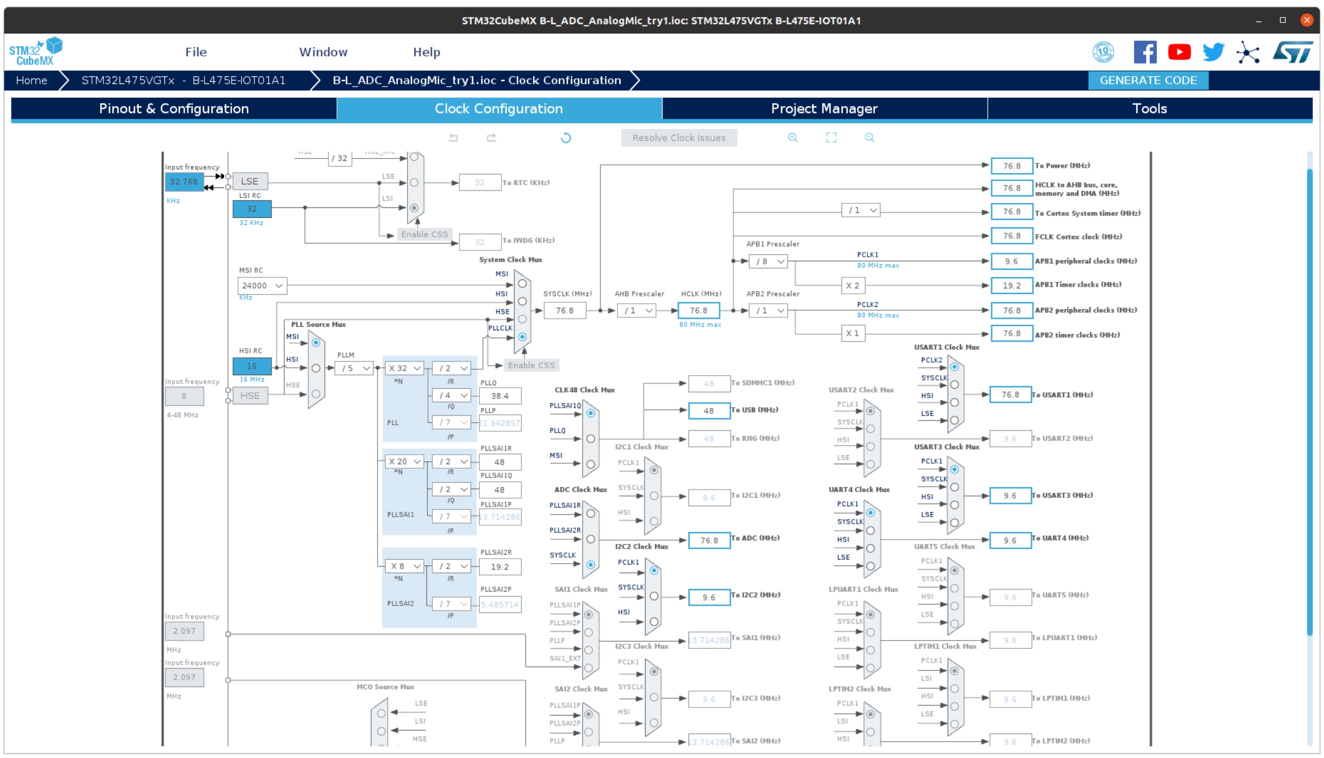
Function	Ideal R	R Range	Approx DC Volt
Dormant	open	open	2.5
Play/Pause	0Ω	0-98Ω	.004
Skip Back	220Ω	133-327Ω	0.179
Skip forw	600Ω	437-711Ω	0.429

This schematic shows how the pause command is sent to the smart phone or device. By bring the 4th 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.

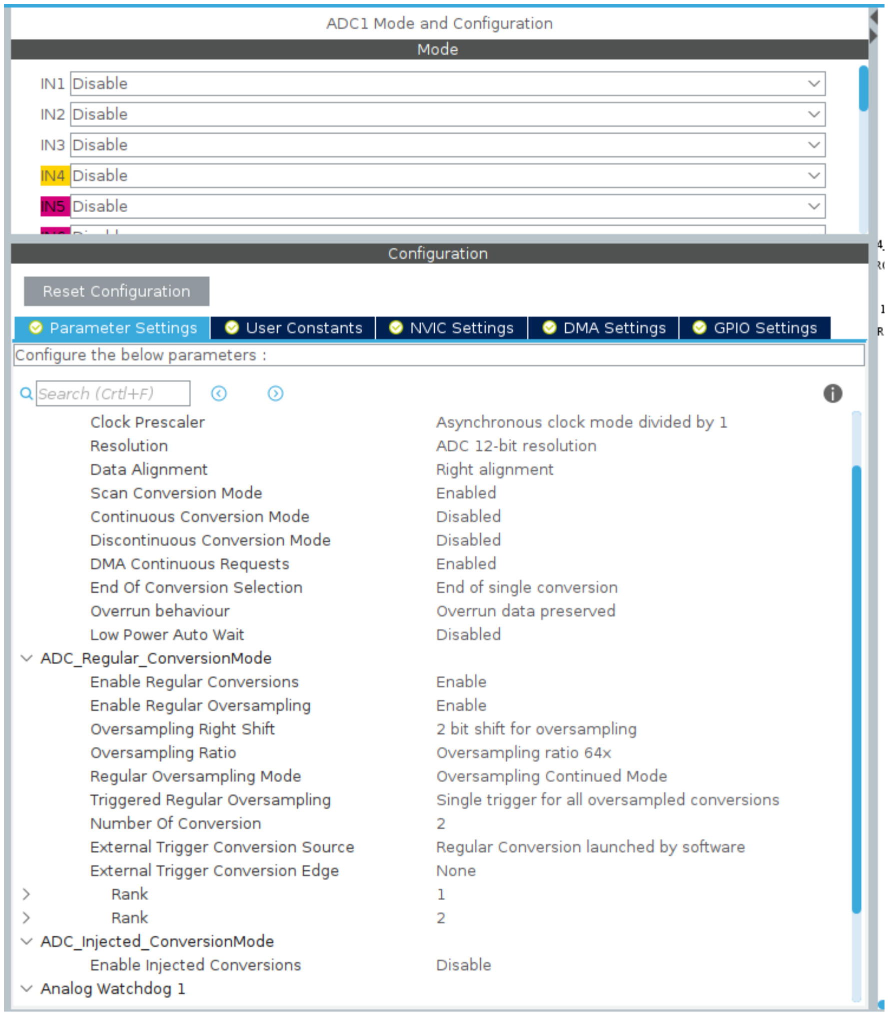


This shows the overall schematic of the audio switch over circuit and pause switch.

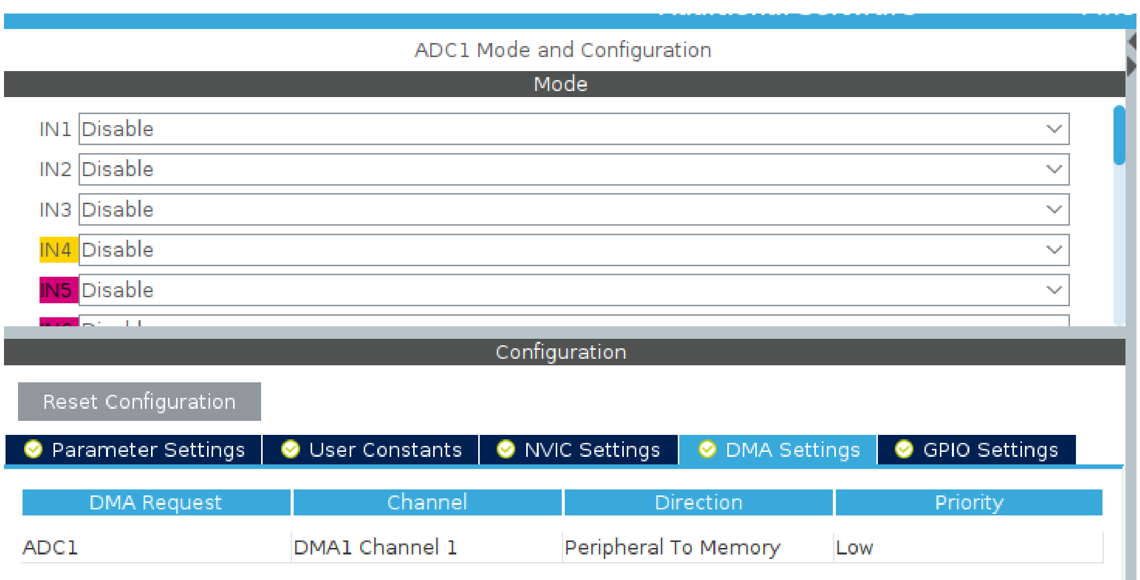
Embedded Code Configuration:
After doing some more research on the STM32 ADC and Audio Sampling I implemented the following system. I took advantage of a couple key components of the ADC system: the ADC can be clocked at a high rate, ADC can over sample, and the ADC has DMA support. I set up the ADC 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.



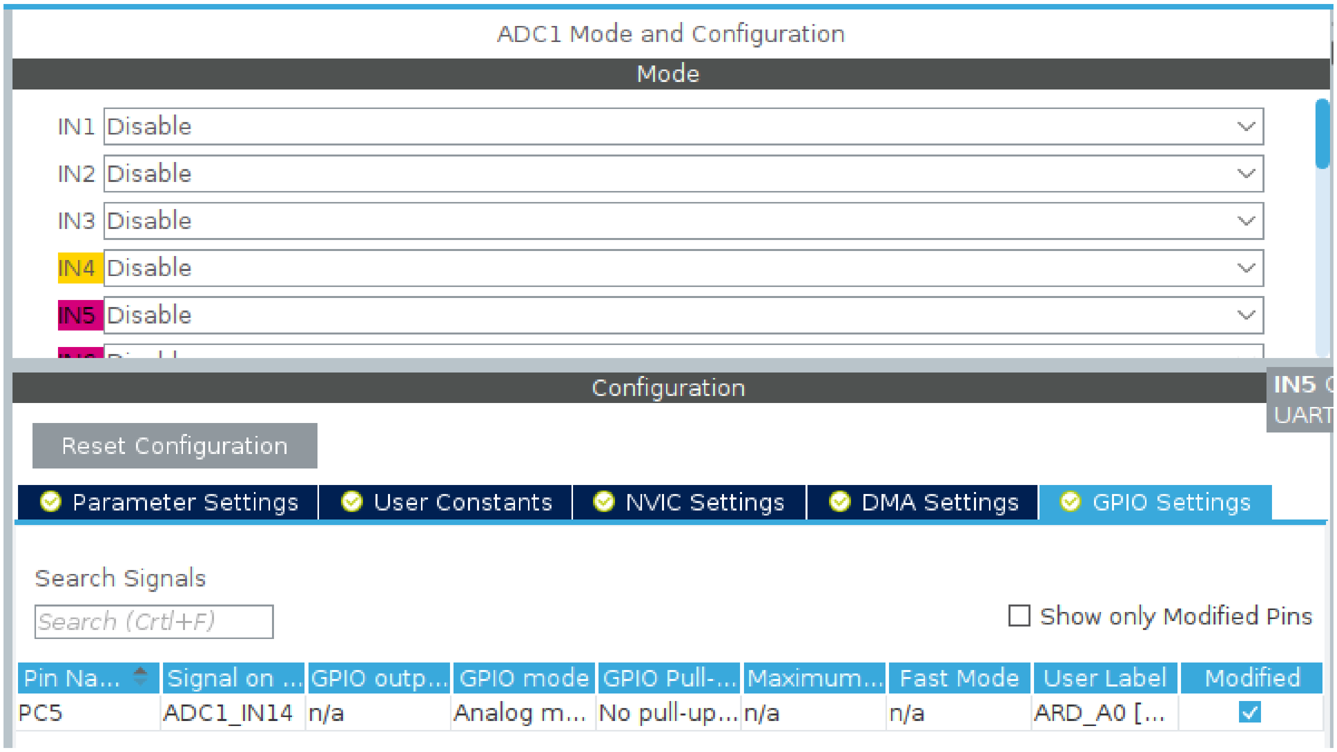
I used STM32CubeMX to configure all the Microcontroller Peripherals. Shown is how the clocks where configured to clock the ADC at 76.8MHz.



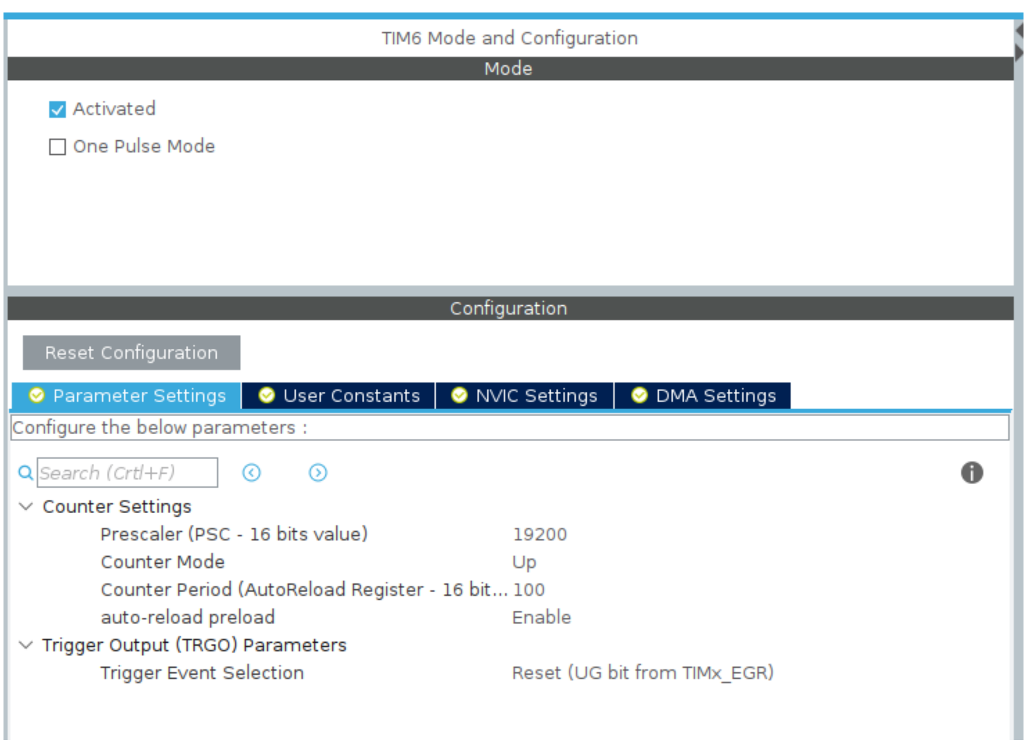
Shown is the ADC baseline configuration.



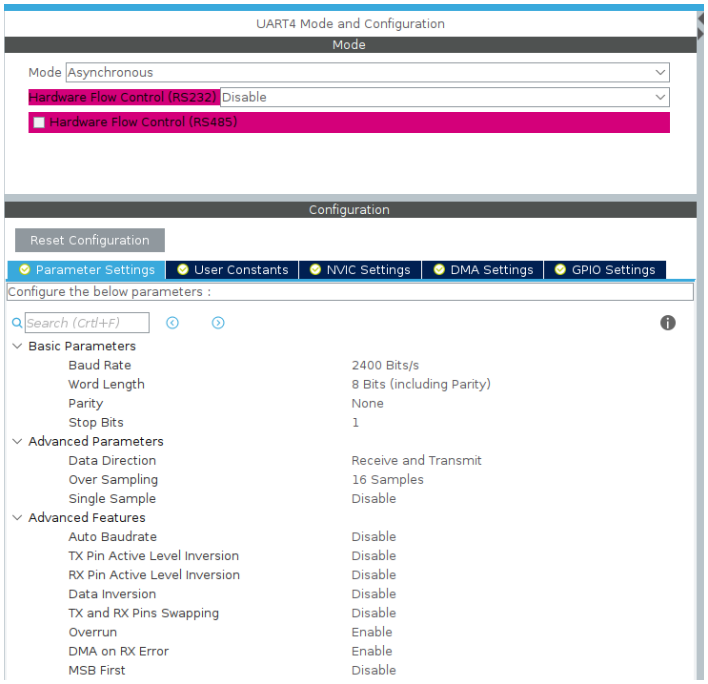
ADC DMA is configured. I used the circular buffer setting.



ADC input pin configuration.



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.



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:

```

/* USER CODE BEGIN PV */
uint16_t MyMicValues[256];
uint32_t AudioSampleAccum1 = 0, AudioSampleAccum2 = 0;
uint16_t AudioAv1, AudioAv2, AudioAvTotal;
uint16_t GlobalTriggerLevel;
uint16_t DescisionFLAG = 0;
uint16_t TriggerFLAG = 0;
uint16_t ButtonFLAG = 0;
uint16_t NewAudioSampleFLAG = 0;

uint32_t DescisionLoopCNT= 0;
uint32_t DescisionAccum = 0;
/* USER CODE END PV */

```
The global variables used in the call back functions.

```
    uint16_t TriggerLoopCount = 0;
    uint32_t TriggerAccum;
    uint16_t TriggerTotal;
    uint16_t TriggerLevel;


    char *msg = "say=Please listen for important message;";
    uint16_t len = strlen(msg);
    //char *backmsg[128];
```

Local variables used and the Text To Speech string used.

```
HAL_GPIO_WritePin(ARD_D3_GPIO_Port, ARD_D3_Pin, SET);
HAL_GPIO_WritePin(ARD_D4_GPIO_Port, ARD_D4_Pin, RESET);
HAL_GPIO_WritePin(ARD_D5_GPIO_Port, ARD_D5_Pin, RESET);


//HAL_TIM_Base_Start_IT(&htim6);

HAL_ADCEx_Calibration_Start(&hadc1, ADC_SINGLE_ENDED);
HAL_ADC_Start_DMA(&hadc1, (uint32_t*)MyMicValues, BUF_LEN);

//HAL_ADC_Start(&hadc1);

while (TriggerLoopCount <= 1024)
{
    //if (NewAudioSampleFLAG)
    //{
        TriggerAccum = TriggerAccum + AudioAvTotal;
        HAL_Delay(15);
        TriggerLoopCount++;
        HAL_GPIO_TogglePin(LED2_GPIO_Port, LED2_Pin);
    //}
    //NewAudioSampleFLAG = 0;
}


//__HAL_TIM_SET_AUTORELOAD(&htim6,2000);

TriggerTotal = TriggerAccum >> 10;
TriggerLevel = TriggerTotal + 540;
GlobalTriggerLevel = TriggerLevel;
HAL_TIM_Base_Stop_IT(&htim6);
TriggerAccum = 0;
/* USER CODE END 2 */
```

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)
  {
      if (AudioAvTotal > TriggerLevel)
      {
          //HAL_GPIO_TogglePin(LED2_GPIO_Port, LED2_Pin);
          HAL_TIM_Base_Start_IT(&htim6);
          while(!DescisionFLAG);
          //{
            //  if (NewAudioSampleFLAG)
              //{
                //  DescisionAccum = DescisionAccum + AudioAvTotal;
                  //DescisionLoopCNT++;
              //}
              //NewAudioSampleFLAG = 0;
          //}
          HAL_TIM_Base_Stop_IT(&htim6);
          if (TriggerFLAG)
          {
              //HAL_TIM_Base_Stop_IT(&htim6);
              HAL_GPIO_WritePin(ARD_D8_GPIO_Port, ARD_D8_Pin, SET); //red LED
              HAL_GPIO_WritePin(ARD_D5_GPIO_Port, ARD_D5_Pin, SET); //pause audio source
              HAL_Delay(400);
              HAL_GPIO_WritePin(ARD_D5_GPIO_Port, ARD_D5_Pin, RESET);

              HAL_GPIO_WritePin(ARD_D3_GPIO_Port, ARD_D3_Pin, RESET);
              HAL_GPIO_WritePin(ARD_D4_GPIO_Port, ARD_D4_Pin, SET);   //switch over audio

              for(uint16_t messageLoop =0; messageLoop < len; messageLoop++)
              {
                  HAL_UART_Transmit(&huart4,(uint8_t *)&msg[messageLoop], 1, 0);
                  //HAL_UART_Receive(&huart4,(uint8_t *)&backmsg,2,0);
                  HAL_Delay(5); //should find TX ready flag
              }
              //HAL_UART_Receive(&huart4,(uint8_t *)&backmsg,2,0);
              while(!ButtonFLAG);
              ButtonFLAG = 0;
              HAL_GPIO_WritePin(ARD_D3_GPIO_Port, ARD_D3_Pin, SET);
              HAL_GPIO_WritePin(ARD_D4_GPIO_Port, ARD_D4_Pin, RESET);   //switch over audio

              HAL_GPIO_WritePin(ARD_D5_GPIO_Port, ARD_D5_Pin, SET); //un-pause audio source
              HAL_Delay(400);
              HAL_GPIO_WritePin(ARD_D5_GPIO_Port, ARD_D5_Pin, RESET);

              HAL_GPIO_WritePin(ARD_D8_GPIO_Port, ARD_D8_Pin, RESET);//RED LED
              TriggerFLAG = 0;
              DescisionFLAG = 0;
          }
          else
          {
              // HAL_TIM_Base_Stop_IT(&htim6);
               HAL_GPIO_WritePin(ARD_D8_GPIO_Port, ARD_D8_Pin, RESET);
               //TriggerFLAG = 0;
               DescisionFLAG = 0;
          }
      }

      /* USER CODE END WHILE */
    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}
/*USER CODE BEGIN 4*/
```

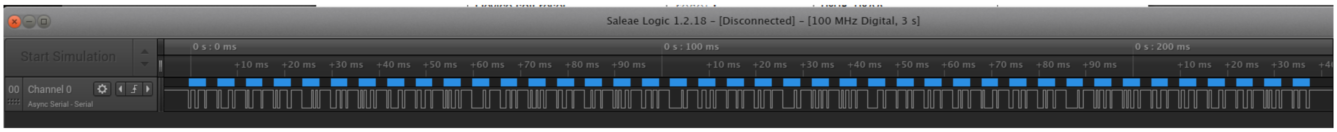
This is the main loop which handles the trigger timeout and then the main function of alerting the user and switch the audio.

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.

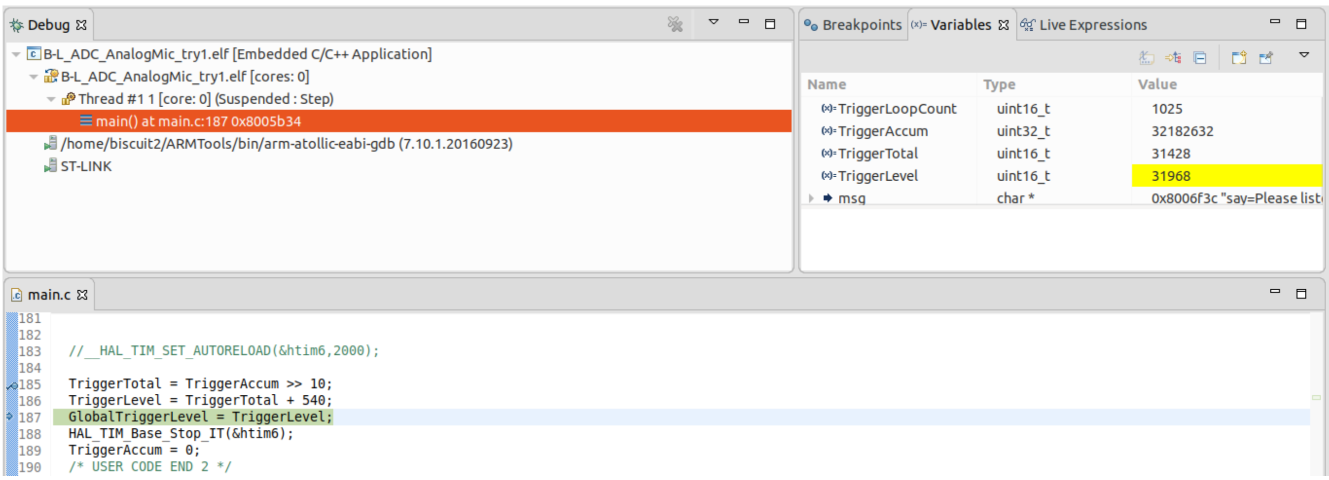
```
/*USER CODE BEGIN 4*/
void HAL_ADC_ConvHalfCpltCallback(ADC_HandleTypeDef *hadc)
{
    for(uint16_t AvCount = 0; AvCount <= HALF_BUFF_LEN; AvCount++)
    {
        AudioSampleAccum1 = AudioSampleAccum1 + MyMicValues[AvCount];
    }
    AudioAv1 = AudioSampleAccum1 >> 7;
    AudioSampleAccum1 = AudioAv1; //running audio average
}

void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef *hadc)
{
    for(uint16_t AvCount = HALF_BUFF_LEN; AvCount <= BUF_LEN; AvCount++)
    {
        AudioSampleAccum2 = AudioSampleAccum2 + MyMicValues[AvCount];
    }
    AudioAv2 = AudioSampleAccum2 >> 7;
    AudioAvTotal = (AudioAv1 + AudioAv2) >> 1;
    AudioSampleAccum2 = AudioAv2; //running audio average
    NewAudioSampleFLAG = 1;

}

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
    HAL_GPIO_TogglePin(LED2_GPIO_Port, LED2_Pin);
    //uint32_t temp;
    DescisionFLAG = 1;
    //temp = (DescisionAccum / DescisionLoopCNT);
    if (AudioAvTotal > GlobalTriggerLevel)
    {
        TriggerFLAG = 1;
    }
    DescisionAccum = 0;
    DescisionLoopCNT = 0;
}

void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
    UNUSED(GPIO_Pin);
    ButtonFLAG = 1;
}
/*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

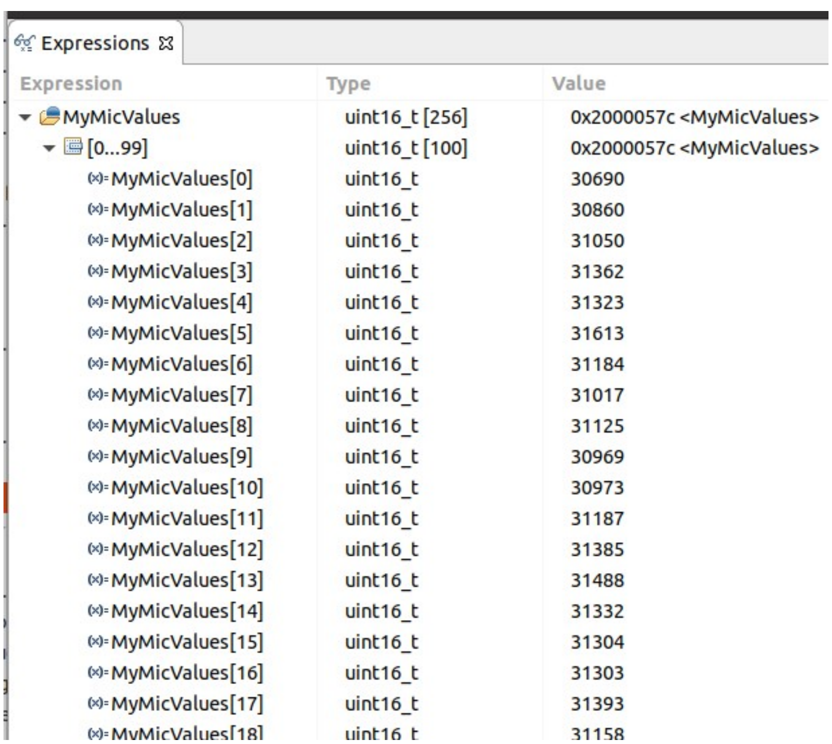
that new audio samples are always being written into the buffer and the audio sample averages are always being calculated.
The TIM call back is for the Timer 6 period elapsed which is when the period compare register is equal to the timer counter. This signals the end of the audio trigger timeout and 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.



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



Expressions

Expression	Type	Value
MyMicValues	uint16_t [256]	0x2000057c <MyMicValues>
[0...99]	uint16_t [100]	0x2000057c <MyMicValues>
MyMicValues[0]	uint16_t	30690
MyMicValues[1]	uint16_t	30860
MyMicValues[2]	uint16_t	31050
MyMicValues[3]	uint16_t	31362
MyMicValues[4]	uint16_t	31323
MyMicValues[5]	uint16_t	31613
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
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
MyMicValues[18]	uint16_t	31158

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

MyMicValues	uint16_t [256]	0x2000057c <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		

This is a Audio Trigger that has made it through the timeout and will be processed.

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.