

IoT Microphone

Overview:

In this lab you will learn how to interface with the microphone on the Educational BoosterPack MKII. Additionally, you will further solidify your understanding of accessing input values from the onboard analog-to-digital converter. In the final part of this lab, you publish microphone data to the cloud and implement an IoT ambient noise sensor.

Accessing Microphone Analog Input:

In the first section of this lab, you will need to access the analog input values of the EDUMKII microphone. This procedure is like the process outlined in Lab 8 part 2 where you collected analog data from the 3-axis accelerometer.

```
void ADC_init(){
    GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P4, GPIO_PIN3, GPIO_TERTIARY_MODULE_FUNCTION);
    ADC14_registerInterrupt(ADC14_IRQHandler);

    /* Initializing ADC (ADCOSC/64/8) */
    ADC14_enableModule();
    ADC14_initModule(ADC_CLOCKSOURCE_ADCOSC, ADC_PREDIVIDER_64, ADC_DIVIDER_8, 0);

    /* Configuring ADC Memory */
    ADC14_configureSingleSampleMode(ADC_MEM0, true);
    ADC14_configureConversionMemory(ADC_MEM0, ADC_VREFPOS_AVCC_VREFNEG_VSS, ADC_INPUT_A10, false);

    /* Enabling Interrupts */
    Interrupt_enableInterrupt(INT_ADC14);
    ADC14_enableInterrupt(ADC_INT0);
    Interrupt_enableMaster();

    /* Setting up the sample timer to automatically step through the sequence
    * convert.*/
    ADC14_enableSampleTimer(ADC_AUTOMATIC_ITERATION);
    ADC14_setResolution(ADC_10BIT);

    /* Triggering the start of the sample */
    ADC14_enableConversion();
    ADC14_toggleConversionTrigger();
}
```

Figure 1: ADC14 Initialization Function

The ADC14 initialization in this lab is much like the initialization for Lab 8, with a few minor differences. Here we need to access the P4_3 pin (A10 input) which corresponds to the microphone's pin. Additionally, we set the ADC to single sample mode here because there is

only one analog input from the microphone. We are also using a resolution of 10 bits from our ADC in this lab.

Just like in Lab 8, our interrupt handler function is “ADC14_IRQHandler.” To successfully sample our ADC, reference the function we used in Lab 8. You will need to store this result in a volatile 16-bit integer. The ADC samples almost every millisecond, so in order to visualize our sample data better, you must create a smoothing filter. Create a 5 variable 16-bit integer array that is used to average the 5 most recent microphone readings. On each interrupt iteration, move every variable one position higher in the array and sample the most recent microphone value in the 0 index of the buffer. Lastly, display this ADC value onto the LCD screen of the EDUMKII. The screen shot below outlines the section you must modify in order to get this working.

```
void ADC14_IRQHandler (void) {  
  
    // Clear the interrupt flags  
    status = ADC14_getEnabledInterruptStatus();  
    ADC14_clearInterruptFlag(status);  
  
    /* ADC_MEM2 conversion completed */  
    if(status & ADC_INT0)  
    {  
  
        /*  
        * 1. Insert Buffer Iteration Logic Here  
        * 2. Sample Newest Microphone Value  
        * 3. Normalize Input with this line: MicIn[0] = abs(MicIn[0] - 512) + 512;  
        * 4. Average Buffer  
        * 5. Display Value on LCD Screen  
        */  
  
    }  
}
```

Figure 2: ADC14 Interrupt Handler Function

IoT Ambient Sound Sensor:

In the final part of this lab, you will create a mapping function that maps microphone intensity values to the EDUMKII onboard LED. Additionally, you will publish the mapped LED output values to an Adafruit IO feed. As data is sent to the feed, you should be able to see a graphical representation of the data in your feed view. The data bandwidth for Adafruit IO is 1 sample per 2 seconds. So, you will need to implement a counter that only samples microphone values approximately every 2 seconds.

Design Specifications

The first step in this lab is to map the values of our ADC14 to a range within 0 to 255 (where 255 is the maximum intensity for an LED). The logic for this functionality is described below.

$$Out = (In - Mic_{LOW}) * (Mic_{HIGH} - Mic_{LOW}) / (LED_{HIGH} - LED_{LOW})$$

In our scenario, the low and high values for the LED are 0 and 255, respectively. To get you started, use 512 and 600 for the microphone intensity values. Implement this log in the function below and feel free to play around with the microphone intensity values. You now should see the EDUMKII LED flicker with changing noise intensity.

In order to write to the LED on the EDUMKII, use the function “**analogWrite(39, outValue)**” (this reduces design complexity, otherwise you would need to implement another interrupt to produce a PWM signal). Also note, that a solid tone will reduce the flickering seen at the LED. Using music might be hard for debugging because there are often quiet sections of a song.

```
uint8_t my_map (uint16_t in){  
  
    uint8_t out;  
    /*  
     * Map ADC14 values to a range between 0 and 255  
     */  
  
    if (in < mic_low){  
        out = 0;  
    }  
    else if (in > mic_high){  
        out = 255;  
    }  
    else {  
        // Place Mapping Logic Here  
    }  
  
    return out;  
}
```

Figure 3: LED Mapping Function

Now we want to send our data to a virtual gauge indicator on the cloud. **Note that this part of the lab requires a firm understanding of how interrupts and the main loop function interact.** Within the “loop” function you must publish the current microphone ADC reading every two seconds to the out feed “virtual_led” on Adafruit IO. **Hint, this amounts to roughly 200 iterations through the interrupt handler.** Additionally, you cannot perform the interrupt

functions and cloud connection tasks simultaneously. You must create a variable that controls which task you must perform. This logic is visualized below.

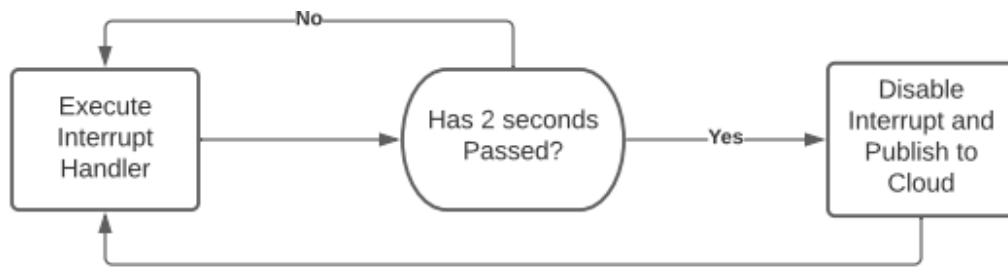


Figure 4: IoT Microphone Logic Diagram

```
void loop() {  
  
    if (counter >= 200){  
  
        // The Interrupt should be disabled!!!  
        // Connect to Adafruit IO  
        // Publish the most recent microphone reading to the cloud  
  
        counter = 0;  
  
    }  
    else{  
        // Go back to the interrupt handler function  
    }  
  
}
```

Figure 5: IOT Microphone Loop Outline

Once everything is working properly, link this new feed to a dashboard with a gauge block.

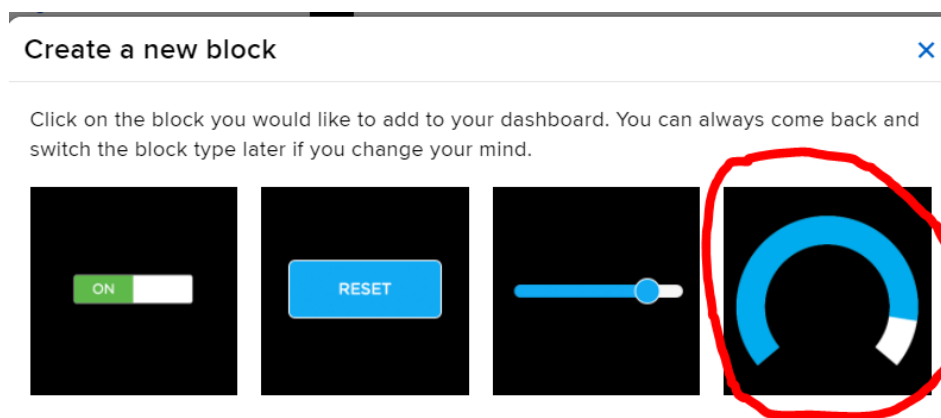



Figure 6: Gauge Indicator Block

Next, you will link this indicator to the “virtual_led” feed. Then set the limits of the gauge to 0 and 255. This ensures the gauge range aligns with the values from your output function.

Block settings✕

In this final step, you can give your block a title and see a preview of how it will look. Customize the look and feel of your block with the remaining settings. When you are ready, click the "Create Block" button to send it to your dashboard.

Block Title (optional)	Block Preview
<input type="text"/>	
Gauge Min Value	Gauge A gauge is a read only block type that shows a fixed range of values.
<input type="text" value="0"/>	
Gauge Max Value	Test Value
<input type="text" value="255"/>	<input type="text" value="45"/>
Gauge Width	
<input type="text" value="25px"/>	
Gauge Label	
<input type="text" value="Value"/>	
Low Warning Value	
<input type="text"/>	

Optional. If no low warning value is given,

Figure 5: Gauge Indicator Settings

After setup is complete, you should be able to see regular updates on Adafruit IO from your LaunchPad. You will only publish data every two seconds, but the data you publish to the cloud should be the most recent ADC14 readings. You should also be able to see real time feedback on EDUMKII LED.