Lab 5: A/D and Intro to Digital Signal Processing

OBJECTIVES

- Learn how to use C (as an alternative to Assembly) in your programs.
- Learn how to use an analog-to-digital conversion (ADC, also known as A/D) system.
- Use the ADC on your XMEGA to sample an analog input, convert the binary value to decimal, and display the corresponding voltage on a console. (You are creating a simple voltage meter!)
- Learn how to use a digital-to-analog (DAC) system. Use the DAC on your XMEGA to output analog values to a speaker.
- Learn how to collect ADC samples to properly recreate audible sine waves.

REQUIRED MATERIALS

- uPAD 1.4 Board and Analog Backpack v1.2
- NAD/DAD (NI/Diligent Analog Discovery) kit
- XMEGA documents
  - doc8331: XMEGA AU Manual
  - doc8385: XMEGA A1U Manual
  - doc8032: XMEGA ADC
  - doc8075: Writing C-code for XMEGA
- Notes for A-to-D pertaining to uPAD
  - uPAD 1.4 documentation
  - Analog Backpack v1.2 schematic
- AC/DC Adapter (Barrel Jack Connector)

YOU WILL NOT BE ALLOWED INTO YOUR LAB SECTION WITHOUT THE REQUIRED PRE-LAB.

PRELAB REQUIREMENTS

You must adhere to the *Lab Rules and* Policies document for **every** lab.

NOTE: All software in this lab should be written in C. If you cannot get your programs working in C, you can write it in Assembly for partial credit.

NOTE: Although the C language has a multitude of built-in functions, you are **NOT** permitted to use any of them in EEL 3744C. For example, you are **NOT** allowed to use the `_delay_ms` or `_delay_us` functions. Also, do not use `sprintf`, `printf`, or any similar functions.

NOTE: Convert the 32 MHz clock configuration code that you previously used in Assembly to C. You **must** set the clock frequency to 32 MHz (as before), but now in C. Setting the FCPU is only used for Atmel's delay function which you are not allowed to use in our course. Also, either keep optimization at the default (which is –O1) or optimize for size (which is –OS). The reason for this is to assure that the setting the clock speed happens quickly enough. An alternative is to use in-line assembly for setting the clock speed as you have been doing previously.

PART A: USING AN ADC TO SENSE LIGHT

Up to now, you have been using the XMEGA exclusively with digital input/outputs. However, the world is not digital. You may want to connect sensors to a processor to measure velocity, temperature, sound, or light. Each of these devices typically generate analog quantities that need to be measured. An analog-to-digital converter (also known as an ADC, A/D, or A-to-D) allows us to measure analog values (e.g., 1.24 V, 0.37 V) and translate them into digital representations. If a microprocessor has and ADC (like our XMEGA) or can get data from an external ADC, the we can process this data digitally, i.e., we can manipulate this information to accomplish some necessary function like making poor singers sound good (Auto-Tune).

In this part of the lab, you will use XMEGA's ADC to detect light intensity using a CdS (Cadmium Sulfide) cell. CdS cells are a type of photoresistor that acts a variable resistor. The resistance of a CdS cell decreases as light intensity increases.

Properly install the Analog Backpack onto the uPAD. Read the Analog Backpack schematic sheet before you continue to determine the ports and pins that will be used with this device.

In this part of the lab you will be using the XMEGA's ADC to detect light conditions using a CdS cell. The CdS cell could be wired as shown in Figure 0, i.e., as a simple resistor divider circuit.

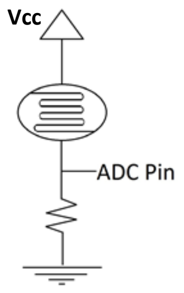


Figure 0: CdS cell in a resistor divider configuration

On the Analog Backpack, the CdS cell is instead wired in a Wheatstone bridge configuration as shown in Figure 1. (You can learn how a Wheatstone bridge circuit works at https://en.wikipedia.org/wiki/Wheatstone_bridge. A lab in EEL 3111C explores the Wheatstone bridge circuit.) The output voltage is measured between CdS+ and CdS-. (See the Analog Backpack schematic for pin outputs.) When the light on the CdS cell increases, the voltage increases; as the light decreases, the voltage decreases. In a balanced Wheatstone bridge, the resistance of the CdS cell would be approximately the same as that of R4, R3, and R1; this would result in a 0 V output. Because the resistance of the CdS in complete darkness is **NOT** the same as the other resistors in the bridge, the differential voltage between the pins will be approximately -

0.6 V. You can measure this with your multimeter or DAD/NAD (before creating your XMEGA program) to verify the values and also, later, to compare your XMEGA program to what you measure with these other devices. Measure the voltages from the J1 (PortA) header on the Analog Backpack. (See page 3 of the Analog Backpack schematic.)

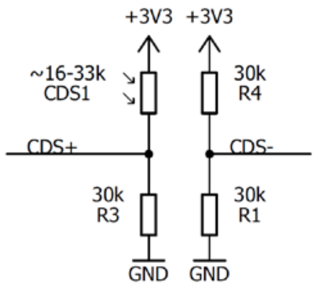


Figure 1: CdS cell configuration on Analog Backpack (on page 3 of Analog Backpack schematic).

Write a C program, `Lab5a.c`, that reads in the differential output of CDS+ and CDS-. Use the ADC on Port A for the pins of the CdS cell (as shown on page 1 of the Analog Backpack schematic).

Use a voltmeter to observe the changes in voltages across CDS+ and CDS- while you cover and uncover the CdS cell with your hand or another object. This will give you an idea of the values to be expected with your board. To get the maximum voltage shine a light (perhaps from your cell phone) directly at the CdS cell. To obtain the minimum voltage completely cover the CdS cell.

Carefully read Section 28 of doc8331 of the XMEGA manual and doc8032 to learn about how to configure and start a conversion of the ADC. You will also need to look at the Analog backpack schematic to see which Port/Pins are connected to the analog pins on the XMEGA to properly configure them as input pins. Below are the basic guidelines needed to set up the ADC:

1. Use the external reference of AREFB in your ADC register initializations. This will set your ADC voltage span to be between -2.5 V and +2.5 V.
2. Configure the ADC system for **signed 8-bit (or signed 12-bit, right-adjusted), differential mode, with a gain of 1. Use Channel 0 for this measurement.** You must set the direction of the ADC pins that you are as an input and enable the ADC module.
3. Find the equation ,$V_{ANALOG} = f(V_{DIGITAL})$, of a line in order to convert the digital value (between -128 and 127 for 8 bits or -2048 and 2047 for 12 bits) to an analog value (–2.5 V to +2.5 V). Put this equation in your lab report. For example, if the voltage is 1V, the digital representation should be about 51_{10} = 0x33 (for 8-bit) or 819_{10} = 0x333 (for 12-bit). If the voltage is 1.5V the digital representation should be about 77_{10} = 0x4D (for 8-bits) or 1228 0x4CC (for 12-bits).
4. Run your program Lab5a.c to detect the voltage across the bridge circuit. Plug this value into the equation determined in the previous step and compare the voltage measured by your voltmeter (or DAD/NAD). The values may differ by as much as 10%. In order to make comparisons, include breakpoints at the end of each analog conversion.

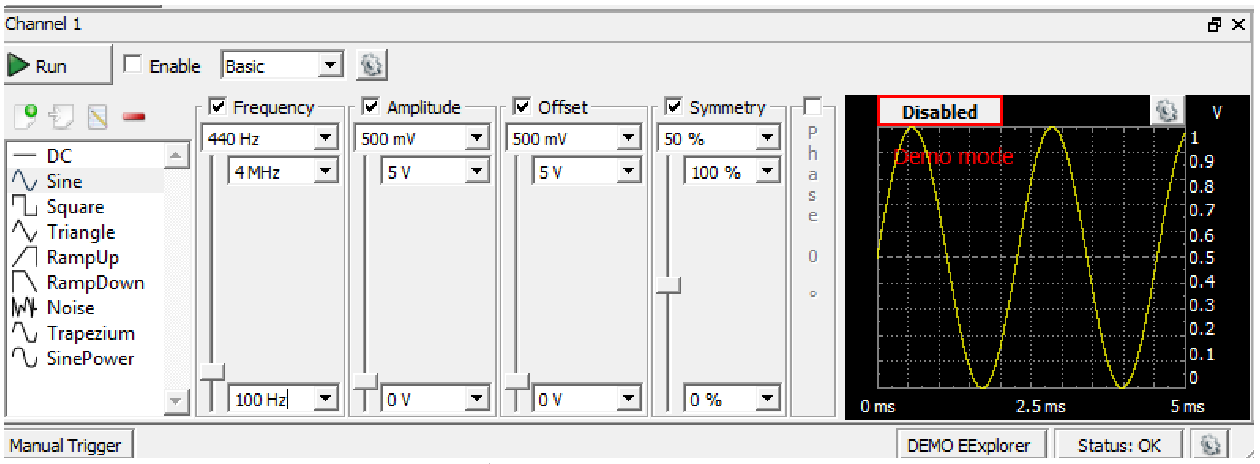
PART A – PRE-LAB QUESTIONS

1. Why do I suggest that you use 8-bit resolution over 12-bit resolution for this application? Why would you use 12-bit resolution instead of 8-bit resolution?

PART B: CREATING A VOLTMETER

1. In this part of the lab you will take the ADC value from Part A into a voltmeter. Add the program you did in Part A to this part, call this program **`Lab5b.c`**. **Setup the UART for 8 data bits, a baud rate of 1,000,000 Hz, no parity bit, and 1 stop bit**. If you have not already done so, translate the assembly program you did in Lab 4 to C. You will be outputting both the decimal and hexadecimal representations of the voltage from the CdS cell to your PC's screen using the Atmel Terminal continuously.
2. You must display the voltage of the ADC input pins as both a decimal number, e.g., 4.37 V, and as a hex number, e.g., 0x6FD in signed 12-bit or 0xDF in unsigned 8-bit. You can use either signed 12-bit or signed 8-bit. (See Figure 3-2 of doc8032 for the reason that I want you to use signed mode.) An example output for a signed 8-bit ADC with a range of –5 V to 5 V is **`+4.37 V (0x6F)`**, i.e., the format for the voltage output is the decimal voltage with three digits, two to the right of the decimal point and then the hex value in either 8 or 12 bits.
3. The ASCII characters of digits 0 through 9 are 0x30 through 0x39, i.e., just add 0x30 to the digit to find the ASCII representation of the digit. You will also need the hex values for the ASCII equivalents of the decimal point, a space, the letters "V" and "x," and both the left and right parenthesis.
4. If we assume that the input voltage calculated in part A was +3.14 V, the below algorithm describes how to send that value to the terminal, one character at a time. Note that using the type casting operation in C is very helpful for this algorithm. Type casting converts a value of one type to a value in another type. For example, if `I` is an integer equal to 3 and `F` is a floating point number, then `F` = (float) 3; will result in `F` = 3.0. Similarly, if `Pi` = 3.14159, then `I` = (int) `Pi`, with result in `I` = 3.

   First send the sign, either '+' or '–' out of the XMEGA's serial port, i.e., transmit it to your PC.



Frequency: 440 Hz (4 MHz, 100 Hz); Amplitude: 500 mV (5 V, 0 V); Offset: 500 mV (5 V, 0 V); Symmetry: 50 % (100 %, 0 %)

Figure 3: WaveGen Function

The below algorithm describes how you could output the digits of a decimal number. (Remember that you are **not** allowed to use library functions like `sprint` or `printf` in this course.)

- Pi = 3.14159…//variable holds original value
- Int1 = (int) Pi = 3 ➔ 3 is the first digit of Pi
- Transmit Int1 and then "."
- Pi2 = 10*(Pi - Int1) = 1.4159…
- Int2 = (int) Pi2 = 1 ➔ 1 is the second digit of Pi
- Transmit the Int2 digit
- Pi3 = 10*(Pi2 – Int2) = 4.159…
- Int3 = (int) Pi3 = 4 ➔ 4 is the third digit of Pi
- Transmit the Int2 digit, then a space, and then a 'V'

Then transmit another blank, a starting parenthesis, the three hex digits (for 12-bit ADC) or two hex digits (for 8-bit ADC) corresponding to the ADC value, and then an ending parenthesis. Another example output for signed 8-bit (–5 V to 5 V) ADC is **3.14 V (0x50).** You will also need to transmit a new line character at the end of each result.

PART C: DIGITAL TO ANALOG CONVERTER

In this part of the lab, you will set up one of the DACA channels to output a voltage of 0.7 V. Previously, you used the ADC system to read in an analog value to convert to a 12/8 bit signed/unsigned result now you'll utilize the DAC to convert a x-bit digital representation to an analog signal. For information on how to use the DAC, refer to section 29 of doc8331.

Specifications

For the portion of the lab (use filename **`Lab5c.c`**) to initialize the DAC properly to output a constant voltage of 0.7 V. You'll use the same reference voltage of AREFB.

Remove the Analog Back from your uPad to have access to Port A.

Refer to doc8385 to see the pinouts of the DAC channels to properly measure the voltage with your DAD/NAD boards. Include a screenshot of the 0.7 V measurement.

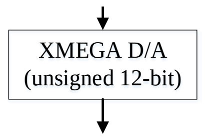


Figure 2: Sampling Flowchart

PART D: SAMPLING A SIGNAL

In this part of the lab you will be using your knowledge of both the ADC and DAC systems to approximate a 440 Hz sinusoidal wave (name the file **`Lab5d.c`**). You'll be using the WaveGen Function on "Waveforms" to output a 0.5 V amplitude sinusoid offset by 0.5 V to an available ADC pin on Port A (as described in Figure 2). At the same time, you'll be using the DAC to output the result using either DACA channel.

Using the DAD/NAD Board

1. In the Waveform's software, go to the "Wavegen" tab.
2. Switch the "Simple" menu to "Basic".
3. Click on Sine (as shown in Figure 3).

Specifications/Background Information

NOTE: The DAD/NAD Board samples digital and analog inputs and produces digital and analog outputs at a maximum rate of about 100 MS/s (100 million samples per second). This is significantly faster sampling than necessary to properly approximate coherent audible waveforms.

Digital signal processing is the use of digital processing to perform a wide variety of task to analyze and manipulate real-world signals in discrete form. DSP is used in filtering (e.g., low-pass filter, high-pass filter), audio processing and effects, speech recognition, image enhancement, etc. In this part of the lab, you will learn critical timing requirements in approximating an signal (a sine wave, in this case).

In general, when doing digital signal processing you'll need to collect enough analog samples to properly recreate the frequency of the signal. If you fall under a certain sampling rate, you will alias the signal. This means the approximated signal will be of different frequency from its parent signal. We will test how different sampling rates affect the approximation of the sinusoidal input. We are transforming a continuous waveform into a discrete form for our processor to understand and manipulate. To properly retain the information of the input signal we need enough data/samples for the processor to recreate it.

Remove the uPAD's Analog Backpack. Use the ADC system from Port A to measure the analog values from the WaveGen function at a rate of 1 kHz. At the same time, output these digital versions of the sampled analog values to the DACA. You can configure the ADC system however you like (i.e., single ended or differential). If you continue to use the previous set-up, you'll need to format the data properly in order to fit the 12-bit unsigned DAC. Use the DAD/NAD's oscilloscope function to view the DAC' s output.

Take a screen shot of the DAC's output using your DAD/NAD board's oscilloscope tools. (Remember that all screen shots should be included in your lab report.)

Keep in mind, in real-time signal processing, maintaining a precise and accurate sampling rate is especially important in cases like this one, where the sampling rate is not much more than double the sinusoidal frequency. The design must take instruction execution time into account. Each instruction takes some time to run, and the more instructions the smaller the window to meet your real-time guarantee. Most digital signal processors run clocks and dedicated hardware (pipelining, memory, etc.) as fast as possible to account for this added time.

You should notice that the signal with a 1 kHz sample rate seemed distorted and displayed, i.e., not very much like the original sinusoid. In theory since our sampling frequency is greater or equal to two times the original sinusoid's frequency (Shannon Sampling Theorem) we can properly recreate the shape of the sinusoid.

Shannon/Nyquist Sampling Theorem

(f_S = sampling rate, f = highest frequency content of signal)

$$f_S > 2f$$

If we take more samples per period, the shape would obviously be more accurate. Since we are sampling at a rate of 1 kHz, we are only producing slightly more than 2 samples per period of the 440 Hz sinusoid. It is possible to approximate your original sinusoid from the collected samples, but most DACs perform a zeroth-order hold. This means that if you write a value to the DAC, the DAC will hold that output voltage until you write the next value. This will give your output a "stair-step" appearance in comparison to the original signal. Further complicating matters, software like Waveforms doesn't have enough information about what is happening on the DAC output, so it will linearly interpolate between consecutive, non-identical voltages. You should keep all of this in mind while analyzing the Waveforms oscilloscope display.

Change the sampling rate to 50 kHz and observe the DAC's output with the DAD/NAD's oscilloscope. Make an additional output pin to view the toggling frequency of the sampling rate with the second channel of the oscilloscope. Take a screen shot of the DAC's output and the sampling rate's frequency.

PART D – PRE-LAB QUESTIONS

2. How many samples per period did you produce at 50 kHz? Suppose we input a 1 kHz signal sinusoid; what would be the minimum frequency needed to reproduce this signal?

PART E: USING THE SPEAKER

Note: There is nothing to turn in for this section, however you cannot move on to the other parts without understanding how to properly set up the speaker. Following this guide will greatly benefit you moving forward.

Now that we understand sampling different frequency sinusoids you will output these sine waves to a speaker (see Figure 4). Re-install the Analog Backpack onto the uPad. Use the headers labeled AIN– and AIN+ as analog inputs. These

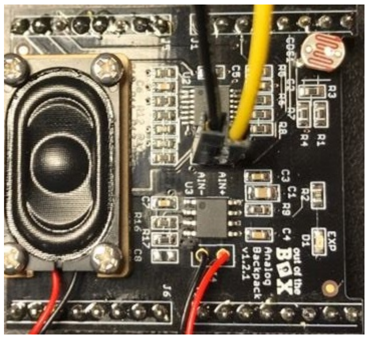


Figure 4: AIN+ and AIN– setup

two headers are inputs to a differential amplifier (as shown on the Analog Backpack Schematic v1.2) that allows input voltages of ±5V. (Note our uPad pins can't read in voltages higher than a specified maximum of 3.3V). The differential output from amplifiers allow the analog inputs to be in a range from -2 V to 2 V (in differential mode). Each individual signal (IN0+ and IN0–) are 0 to 2.0 V relative to ground. Since the ADC system is differential, we will have to take this into account.

Using your DAD/NAD board's WaveGen Function, input –5.0 V into the AIN+ signal and GND into the AIN– signal (see Figure 4). Use a multi-meter or the DAD/NAD to measure the individual signals IN0+ and IN0–, each with respect to ground. Take note of the voltage reading for each signal. Since the ADC system is differential, the actual voltage reading would be the difference between the + and – terminals. Repeat the same procedure for a +5.0 V input.

Now that you understand how the differential amplifier works, set up the ADC system to read in the analog signals IN0+ and IN0– when 5.0 V is input to the system as previously stated. The ADC system must be properly configured to read the analog values to continue. View the ADCA_CHx_RES registers to verify the digital representation is correct in correlation with a 5.0 V input.

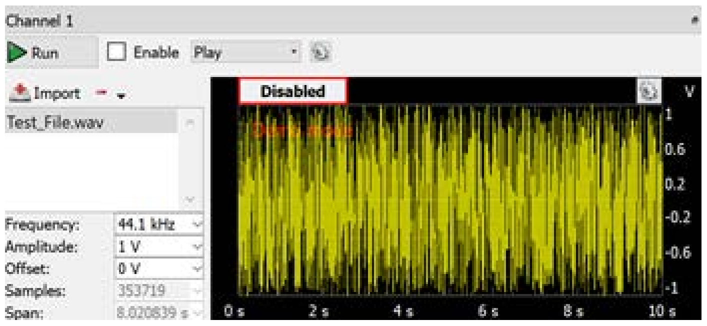
With a working ADCA system to read in analog values, you can now input sine waves. Input a sine wave with a 1 V amplitude and -1 V offset; you will use several different frequencies for the sinusoid. Keep in mind that when sampling different frequency sinusoids, the sample rate should be great enough to properly recreate the waveform. Use 100 kHz. Like previously, collect ADC samples and output them to the DACA at the "same time".

Look at the Analog Backpack Schematic. You will outputting the sine waves to an 8-Ohm (8 Ω) speaker. (Make sure the speaker wires are connected properly via multi-meter continuity test). The speakers input is from one of the channels of the DACA. Our speaker we will be using is connected to an audio amplifier that is **ONLY** powered through an external source, i.e., this source is **REQUIRED**. The external power source allows us to drive enough current to play the speaker. In our case, we must plug in the AC/DC adapter to power the board and amplifier to make the speaker work. You may have also noticed from the Analog Backpack Schematic that there's a signal that is being driven to the amplifier called **/Power_Down**, you'll need to set the proper voltage (Vcc) for it to enable the speaker.

Once the system is properly set-up, you will toggle the frequency of the input sinusoid from 100 Hz to 1 kHz, you can verify the signal going into the speaker is correct by properly probing the DACA channel with one of the oscilloscope's pins of your DAD/NAD board and using the measurement tools to view the frequency.

PART F: SAMPLING A .WAV/MP3 FILE

In this part of the lab, you will write a program that plays a song of your choice through the speaker (`Lab5f.c`). The DAD/NAD board software, "Waveforms 2015", allows you to import a .wav, .mp3, or .m4a files and samples it at the rate the audio was produced. MP3, M4a and WAV files are used in all areas of audio. MP3 files generally are more compressed than WAV files, preserving memory. Since Waveforms transforms these files to the desired samples, you do not need to worry about formatting the data or directly using the sampling theorem discussed previously.



Test_File.wav

Frequency: 44.1 kHz
Amplitude: 1 V
Offset: 0 V
Samples: 353719
Span: 8.020839 s

Figure 5: Waveform Setup

Generating the Sampled Data

1. Open Waveforms.
2. Click on Wavegen tab on the side and navigate to the "Simple" menu.
3. Switch "Simple" to "Play".
4. Import the WAV file on our Lab webpage available at https://mil.ufl.edu/3744/labs/Never_Underestimate.wav. After using this one, you can try another file of your own choosing**.** If the .wav/.mp3 file is big, it might take a while for all the samples to load.
5. Once the waveform is generated, you'll be able to send this to the A/D system of the XMEGA.

Figure 5 show the Wavegen output for `Test_File.wav`.

NOTE: You will need to follow proper initializations for the song to play on the speaker clearly.

If you do not have any .mp3 or .wav files, websites such as **`Soundcloud`** or **`archive.org`** have free legal downloadable music. **House35**, a local band in Gainesville has provided access to their music at the following URL: *https://soundcloud.com/house35*.

Specifications

Sample the audio at the rate specified by the waveform program on your XMEGA processor. Set up an available pin as an output to view the correct frequency.

With the Analog Backpack on the uPAD, place the wavegen signal (Yellow wire) from the DAD/NAD board onto the AIN+ signal and respective ground on the AIN- signal.

You will need to configure the A/D system for a **12-bit right-adjusted and unsigned** result to read in IN0+. Why would we use a 12-bit resolution over an 8-bit resolution? You will also need to configure the system in **single-ended input** mode.

Make sure the DACA is properly set-up to convert the 12-bit unsigned result to its decoded analog value. You need to make sure your conversion from A/D to D/A is precise or information will be lost.

To make an efficient program you want to avoid unnecessary lines of code that delays the ADC or DAC systems. To avoid CPU time looping, you will need to use an A/D interrupt to trigger when the conversion is complete (after it is started). Make your ISR execution fast as possible. If you are to sample at 44.4 kHz, each sample comes at 22.5 us intervals. Each instruction approximately takes 4+ cycles since we are dealing with 16-bit values (two 8-bit concatenated registers) adding 4/32 MHz = 1.25 ns each time between samples as well the conversion time for each bit in the A/D system. Time adds up quickly if you are not careful in your code.

Power the speaker with the AC/DC adaptor and have the necessary code to turn it on. Once you run the waveforms, you should hear the MP3/WAV file on the speaker. You can increase the volume of the signal from 1 to 5V since the signal is constrained from 0-2V on the backpack.

PART G: AUDIO RECORDER

In the Part F, you sampled and output a song at the same time. You do not want to do this most of the time, rather collect all the samples and process them accordingly. Unfortunately, our microprocessor does not have a dedicated memory in storing large amounts of data (although your uPAD Memory Base does have a SPI interfaced Flash ROM). In this part of the lab, we will be designing an audio recorder in storing 32K samples and playing back the audio in three different modes: Fast, Normal, and Slow.

A sampled audio recording is provided for you for this lab on our Lab webpage. However, you have the option of creating your own by using your laptop/phone in producing a 3 s to 4 s recording.

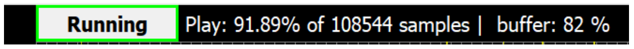

Running Play: 91.89% of 108544 samples | buffer: 82 %

Figure 6: Data Buffer

We will combine three sub-systems in making the recorder: USART, TC and EBI.

Specifications

- Configure the USART as previously at 1,000,000 bps (1 Mbps), 1 stop, 1 start bit, and no parity.
- Set up the EBI at the base address 0x8000 with a chip select of 32 K size. (See EBI example on website to understand pointers and memory.)
- Place the audio in the wavegen and visually see how many samples were produced at the frequency referenced

Using the number of samples and sampling frequency provided by Waveforms you need to figure out an equation to solve for the frequency to collect 2^{15} samples on your processor. For example, the audio recording provided has 42,756 bytes or samples which doesn't fit the external SRAM. We need to sample at a rate where we can skip some data to fit the data into the 32 K SRAM. The audio might be a little fuzzy, depending on the number of samples when you play it back since every sample is not collected.

- Initialize a TCxx to work with the sampling frequency you calculated.
- Set up the A/D and D/A systems like Part F, but configure the resolution to 8-bits for the ADC.

Setting/Mode	Functionality
'p' or 'P'	Playback recorded audio (infinitely)
'r' or 'R'	Record Audio into **external** SRAM
's' or 'S'	Slow down the play backed audio (half the normal sample rate)
'f' or 'F'	Speed up audio in playback mode (twice the normal sample rate)
'q' or 'Q'	Quit playing the audio on the speaker
'n' or 'N'	Normalize the audio back to the original sample rate

1. Create a flowchart/pseudocode for this program **BEFORE** you begin to write code. The first step in beginning is making the recorder mode work independently of the other modes, once you know you have the audio in memory integrate all the mode together.
2. Create a C file called `Lab5g.c`. **NOTE:** There is multiple ways to design this program. I recommend following the guidelines below.

Record Mode

Once pressing 'r' or 'R' you want to start the timer system (Stated above) in collecting 32k samples to fill the external SRAM.

IMPORTANT: Recording is **TIME CRITICAL**, you may have not noticed in Part F but the samples are pushed out from a buffer at timer intervals. You want to run your Wavegen and wait until the buffering is back to the beginning so you do not lose data and **THEN** press the recording button. See Figure 6.

Once you press 'r' or 'R' the audio is getting sampled and being stored into the SRAM. To prevent overwriting data, pressing 'r' or 'R' once will fill the SRAM completely one time. To record another audio segment you must press the record button again.

Play Mode

If you press 'p' or 'P' before you record any audio, you will be playing random noise already stored in your SRAM memory. After a successful recording, the recorded audio can be played back normally using the normal sample rate calculated and controlled by another timer system.

To speed up the audio, you will need to increase the frequency in which you play the audio.

To slow down the audio, you will need to decrease the frequency in which you play the audio.

To stop the audio, you can pause the timer system in preventing the next sample to be played.

PuTTY Terminal

You should have some type of indication of what mode you are using when you press a button. Create your own menu or display for this program following the mode/functionality table.

PART G – PRE-LAB QUESTIONS

3. Why did we set up the resolution to be 8-bits rather than 12-bits?

PRE-LAB REQUIREMENTS

1. Answer the pre-lab questions.
2. Configure the ADC properly for Channel 0 (CdS cell)

3. Display proper voltages (in decimal and hex) on the terminal.
4. Recreate a sinusoidal input waveform by using the ADC/DAC system to sample at a certain rate.
5. Play a song of your choice using the processing capabilities of the XMEGA.
6. Create an audio recorder using EBI, USART, TC, A/D, and D/A systems.

IN-LAB REQUIREMENTS

1. Demonstrate Parts F and G. If these parts do not work, be prepared to demonstrate as much as you can from Parts A through E.