LAB 6: ADC, DAC, AND MUSIC!

OBJECTIVES

- Understand how to utilize an **a**nalog-to-**d**igital **c**onverter (**ADC**), also known as an **A/D** system.
- Understand how to utilize a **d**igital-to-**a**nalog **c**onverter (**DAC**), also known as a **D/A** system.

REQUIRED MATERIALS

- DAD/NAD Analog Discovery board
- µPAD v2.0 Development Board
- µPAD v1.3 *Analog Backpack*
- USB A to B Cable
- Documentation:
  - XMEGA_AU_Manual (doc8331)
  - ~~µPAD Application Notes for Analog Inputs~~
  - AVR1300: Using the Atmel AVR XMEGA ADC (doc8032)
  - ATxmega128A1U_Manual (doc8385)
  - µPAD v2.0 Schematic
  - µPAD v1.3 *Analog Backpack* Schematic
  - *IS31AP4991 Audio Power Amplifier Datasheet*

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

NOTE: The amount of information given in this lab is much less than in earlier labs. Do **NOT** procrastinate!

INTRODUCTION

As you have seen throughout the previous labs, your microprocessor can produce and detect discrete binary values, i.e., ‘0’ corresponding to a voltage value close to your processor’s ground reference (normally 0V), and ‘1’ corresponding to a voltage value close to your processor’s voltage reference (e.g., 3.3V). However, the world is also filled with analog signals, i.e., signals that have more than two discrete values, such as those that represent velocity, temperature, sound, light, etc. Thankfully, it is possible for your processor to both interpret and generate these non-digital signals, through two separate systems: an **A**nalog-to-**D**igital **C**onverter (**ADC**), and a **D**igital-to-**A**nalog **C**onverter (**DAC**).

NOTE: Normally, sensors are used to convert an analog signal into a digital representation.

With an **A**nalog-to-**D**igital **C**onverter (**ADC**) system, an analog signal can be interpreted and converted to a digital value. This digital value, whose limits are determined by the hardware and software configurations of the ADC system, can then be interpreted by a microprocessor.

Similarly, a **D**igital-to-**A**nalog **C**onverter (**DAC**) system is a system that can convert a digital value (whose limits are also determined by the hardware and software configurations of the system) into an analog value (i.e., a voltage waveform). The precision of either system is determined by the smallest possible voltage change that affects the system’s digital value by one bit. The number of digital bits available to the system defines the **resolution** of the system, whereas the **accuracy** of the system is defined to be the actual change in voltage that, in the context of an ADC system, is measured per digital bit, and in the context of a DAC system, is produced per digital bit.

In this lab, you will first set up an internal ADC system of your XMEGA to sample a constant voltage waveform generated by your DAD/NAD. After this, you will create a more sophisticated program to utilize an ADC and create a voltmeter. The voltage measured by your voltmeter will be generated by your on-board CdS photocell (a simple light sensor), and a serial terminal (e.g., PuTTY) will be used to display voltmeter data.

NOTE: A **C**a**d**mium-**S**ulfide (**CdS**) **photocell** (also known as a **photoresistor**, or a **l**ight-**d**ependent **r**esistor [**LDR**]) is a resistor whose resistance is variable to the amount of light present on the surface of the photo-conductive cell (see Figure 1). The CDS cell located on your *Analog Backpack* is labeled ***CDS1***.

Figure 1: Two CdS photocells

Following this, you will hopefully be comfortable with using an ADC system. For the remainder of the lab, you will use an internal DAC system of your XMEGA.

To begin utilizing a DAC system, you will generate a waveform with a constant voltage. Then, you will generate (or more specifically, emulate) a sine waveform with a table of at least **256** predefined data points. This method of using a table of predefined data, otherwise known as a **look-up table (LUT)**, to generate a waveform is quite common, although it should not be surprising that with only a finite amount of data points, the generated waveform will amount some degree of error (although possibly negligible).

After emulating a simple sine waveform, you will begin to utilize the speaker on your *Analog Backpack*, after learning some basic music terminology. Then, you will learn some basic music theory regarding pianos, so that you will be able to finish the lab by creating a simple keyboard synthesizer. This program will utilize fourteen of your computer keyboard keys to generate specific musical notes within two selected musical octaves, namely the 6th and 7th octaves.

PRELAB REQUIREMENTS

You must adhere to the *Lab Rules and Policies* document for **every** lab. All program code for this lab must be written in C.

GETTING STARTED

NOTE: Unlike in previous labs, for this and the following lab, it is **REQUIRED** that you use group configurations, bitmasks, and any other useful macros defined in the <avr/io.h> header file.

NOTE: You should **ALWAYS** create a flowchart or pseudocode for a software design, **BEFORE** writing any code. Pseudocode and flowcharts are **NO** longer due 72 hours prior to your lab.

NOTE: If a program/design does not work, utilize the debugging capabilities on your µPAD and your DAD/NAD board, along with your prior electrical and computer engineering knowledge to fix any errors in your hardware and/or software (code). This should occur **BEFORE** you come to lab. If necessary, visit a teaching assistant or Dr. Schwartz, but come to lab prepared!

NOTE: As you have been doing throughout all the labs in this course (and hopefully how you will continue to do so in your engineering career), you should slowly build up any of your designs, by adding one manageable piece at a time, ensuring that new piece be fully functional before adding anything additional.

NOTE: As always, analyze any necessary schematics in tandem with the Alternate Pin Functions section within the 8385 datasheet to determine which pins are relevant.

PART A: INTRODUCTION TO ADC

1. Read Section 28 of the 8331 manual as well as the 8032 datasheet, both from Atmel, to begin to learn about how to configure and start a conversion on an ADC system. Also read the *µPAD Application Notes for Analog Inputs* document from OOTB for important notes regarding the µPAD.

The following are some general guidelines to follow when configuring an ADC system:

- Set the necessary data direction of the pins that you wish to use for the chosen ADC module.
- Configure the chosen ADC system for an appropriate mode (8-bit or 12-bit, signed or unsigned, single-ended or differential, gain or no gain), and use an appropriate voltage reference (refer to the *µPAD Application Notes for Analog Inputs* document when configuring an ADC system within your given processor).
- Find an appropriate equation of a line that represents the linear relationship between the range of measurable analog voltage values and the range of digital values of the ADC system (e.g., $\mathbf{V_{ANALOG} = f(V_{DIGITAL})}$). For example, with an analog voltage range of −2.5 V to +2.5 V, if the analog voltage is 1V, the digital representation should be about 51_{10} = 0x33 for an 8-bit ADC system, and 819_{10} = 0x333 for a 12-bit ADC system. Moreover, if the voltage is 1.5V, the digital representation should be about 77_{10} = 0x4D for an 8-bit ADC system, and 1228_{10} = 0x4CC for a 12-bit ADC system.

PRE-LAB QUESTIONS:

1. What is the main benefit of using an ADC system with 12-bit results, over an ADC system with 8-bit results? Would there be any reason to use 8-bit results instead of 12-bit results? If so, explain.
2. What is the **resolution** of a 12-bit signed ADC system, with a voltage range from -1V to 3V? What about the **accuracy** of the system?
3. What voltage references can your XMEGA be configured to use, taking the µPAD into account? For each possible voltage reference, describe a situation in which you would want to use that specific reference.

You will now create a simple C program to configure an available ADC system on your XMEGA. This chosen ADC channel will be used to measure a constant voltage generated by your DAD/NAD. To generate a voltage waveform with your DAD/NAD, use the *Wavegen* function within your *Waveforms* software. **Make sure to use the appropriate pins on your DAD/NAD**, as well as an ADC system for which you have access to on the µPAD.

2. Create a program (`lab6a.c`) to configure an ADC system and channel for which you can measure a voltage waveform with an available pin on your µPAD. Configure this ADC system to be able to measure signed values.
3. Use the *Wavegen* function of your *Waveforms* software to generate a constant **0.5 V** voltage waveform. Utilize your processor along with your program (`lab6a.c`) to verify that your ADC system is functional, applying any technique to do so, e.g., using a *Watch* window within Atmel Studio, etc. ~~Do the same for a constant **-1.5 V** voltage waveform. No screenshot is needed.~~
4. **OPTIONAL**: Peruse Table 37-8 in section 37.1.6 of the 8385 datasheet to determine what the minimum possible input voltage is for an ADC system within your XMEGA. Identify that this is ***different*** than the minimum possible voltage conversion value for your ADC system. Somehow, measure a constant **-1.5 V** voltage waveform with your ADC system, with V_{IN+} (positive terminal) remaining more positive than V_{IN-} (negative terminal), and without supplying a negative

LAB 6: ADC, DAC, AND MUSIC!

voltage to your V_{IN+} pin. **Take a screenshot**, proving that you can measure a **-1.5 V** voltage waveform.

PART B: CREATING A VOLTMETER

In this part of the lab, you will create a basic voltmeter to measure the voltage generated by your on-board CdS photocell.

A simple voltage divider circuit could be used to interface with a CdS cell and ADC system, as shown in *Figure 2*. However, a slightly more sophisticated technique using a Wheatstone Bridge was chosen to interface the CdS cell located on your *Analog Backpack* with your µPAD.

In darker (i.e., lower-light) environments, after the internal resistance of the CdS cell becomes large, this Wheatstone Bridge circuit will cause the voltage measured between the two terminals of the CdS cell to decrease. The opposite is also true, i.e., in brighter environments, after the internal variable resistance of the CdS is lowered, the Wheatstone Bridge circuit will cause the voltage measured between the two CdS terminals to be larger. Note that voltage would vary much more in a simple voltage divider circuit.

1. Identify the Wheatstone Bridge circuit within the *Analog Backpack* schematic. Determine which pins should be used to measure the voltage across the CdS cell, and how you will use an ADC system to effectively measure this voltage.

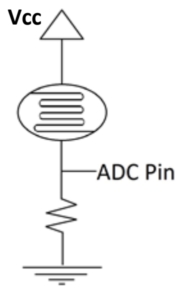

Figure 2: CdS cell within a voltage divider circuit configuration

Before using an ADC system to effectively measure the voltage across the terminals on your on-board CdS cell, you will need to determine the voltage range of your CdS cell with a voltmeter.

2. Analyze the *Analog Backpack* schematic to determine where to probe the terminals of the on-board CdS cell. Use your DAD/NAD or a multimeter to measure the voltage across the terminals of your on-board CdS cell. The voltage will change as the the photo-conductive cell is exposed to light. Record the minimum and maximum voltages and whether this corresponds to maximum or minimum light. Use the *Analog Backpack* schematic to determine the point at which you should probe the CdS cell.

NOTE: It is likely that using a finger to cover the CdS photocell will not *completely* remove light exposed to the photo-conductive cell. It is recommended to instead use something such as a thick, dark fabric.

After doing this, you will be able to create your voltmeter program (`lab6b.c`). Overall, your program must be able to measure the voltage drop experienced by the CdS cell located on your *Analog Backpack* every 100 ms (like a real voltmeter), and then display the results, in terms of both decimal and hexadecimal, within a serial terminal program on your computer (e.g., PuTTY). More specifically, the output displayed within your terminal program **must** include *at least* the following to describe the voltage measured:

(+/-)voltage_decimal V (0xvoltage_hex),

where ***voltage_decimal*** is the measured ADC digital value corresponding to the voltage drop experienced by the CdS photocell, in terms of a decimal value with two decimal places, and where ***voltage_hex*** is the measured ADC digital value corresponding to the voltage drop experienced by the CdS photocell, in terms of a hexadecimal value with two digits.

For example, if an 8-bit ADC system with a voltage range of -5 V to +5 V was used, and the measured voltage was to correspond to a decimal value of 1.37 V, your output should include "**`+1.37 V (0x22)`**", without the quotes.

Moreover, if for the same system, a measured voltage value was to correspond to a decimal value of -2.52 Vs, your output should include "**`-2.52 V (0xC0)`**", also without the quotes.

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

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'

3. Create a voltmeter program (`lab6a.c`), as defined above. Anything undefined is left at your discretion.

For the remainder of the lab, you will be using an internal DAC system within your XMEGA.

PART C: INTRODUCTION TO DAC

As you will soon discover, using a DAC system is very straightforward after having utilized an ADC system.

1. Read Section 29 of the 8331 datasheet from Atmel to learn how to configure and use a DAC system within the XMEGA.

Now, to demonstrate your understanding of the DAC system, you will generate a waveform with a constant voltage, and then measure this waveform with the *Scope* feature of your *Waveforms* software.

2. Create a program (`lab6c.c`) to generate a waveform with a constant voltage of **1 V**, using a DAC system for which you have access to probe the necessary pin on the µPAD. Measure the waveform with the *Scope* feature of *Waveforms*, and then **take a screenshot** of the oscilloscope.

PART D: GENERATING A WAVEFORM WITH A LOOK-UP TABLE

Sometimes, a DAC system can be used to output analog voltages based on sampled ADC values. In other cases, instead of sampling a waveform first, it can be desired to simply recreate a known waveform, often at a specified rate. (To handle any timing, a built-in timer/counter system is generally used.)

Normally, the data of this known waveform is stored within a data structure such as an array or table, so that it can be easily accessed within a program. When a data structure such as an array or table is used for this purpose, the data structure is often referred to as a **l**ook-**u**p **t**able (**LUT**).

NOTE: A data structure refers to a method of organizing units of data within larger data sets, e.g., an array in the C programming language is simply a contiguous block of memory (either program, data, BSS, stack, or heap memory), where the reference to the array is a pointer to the first element within the block of memory.

In this part of the lab, you will create and store a look-up table of at least **256** data points for a specified sine wave within one of your processor's memories (e.g., within program memory by utilizing the ***const*** identifier), and then utilize this table along with a DAC and TC system to **continuously** recreate the sine waveform at a given rate.

To generate the necessary data points for your look-up table, you may use *MATLAB*, *Excel*, or an online look-up table calculator such as the following: Sine Waveform Data Calculator. Make sure that this data in the same format expected by your DAC, i.e., left-adjusted or right-adjusted..

1. Write a C program (`lab6d.c`) to generate a sine wave of at least 512 data points, with voltages ranging between 0V and AREFB, and with a frequency of 1760 Hz (acceptable error of ±2%). Your program must utilize a built-in DAC and TC system.
2. Use the *Scope* feature of your *Waveforms* software to measure the created sine waveform. **Take a screenshot** of the waveform, identifying its frequency.

PRE-LAB QUESTIONS:

4. What is the correlation between the amount of data points used to recreate the waveform and the overall quality of the waveform?

PART E: INTRODUCTION TO MUSIC

In the previous part of this lab, you created a 1760 Hz sine waveform. In musical contexts, this waveform represents the note **A_6**, one of the 12 notes in the 6th octave. To further understand what this means, some basic music terminology is defined below.

In music, a **note** is a sound defined by a pitch and duration, where a **pitch** is a frequency of vibration (Hz), and where **sound** is created from vibrations that travel through the air or another medium.

A special collection of notes, ordered by pitch, defines a musical **scale**, where a single scale can be manifested at many different pitch levels. The most common musical scales are typically written using eight notes, where the first and last notes are an octave apart. An **octave** is defined to be an interval of notes where any given base note within the interval has a frequency of vibration twice that of the same base note in the previous octave (or half of the same base note in a following octave). For example, the base note **A** in the 6th octave (**A_6**), has a frequency of 1760 Hz, whereas the base note **A** in the 5th octave (**A_5**) has a frequency 880 Hz. Similarly, the base note **A** in the 7th octave (**A_7**) has a frequency of 3520 Hz.

Now, in this part of the lab, you will begin to use the speaker located on your *Analog Backpack* to create your processor's first musical tone! (They grow up so fast.) Though, to be able to use your on-board speaker, a few points about audio electronics will first be mentioned.

In the field of **audio electronics**, the main goal is to handle and produce either sound or pressure wave signals. For a device to handle such signals, an analog signal must first be encoded into a digital signal, with an analog-to-digital converter. (Often, a **preamplifier** is used to amplify any weak digital signals into more manageable signals for any processing components). Conversely, to produce sound or pressure wave signals, a digital signal must be sent from a digital-to-analog converter to a speaker system. To be able to drive a speaker system with enough power, an amplifier circuit is almost always needed.

NOTE: Normally, processors optimized to handle digital signal processing (known as **d**igital **s**ignal **p**rocessors [**DSP**s]), built with specialized ADC and DAC systems, along with other specialized components, are used to manage sound.

On your *Analog Backpack*, there exists a IS31AP4991 audio power amplifier used to drive your on-board speaker (labeled J4 on your *Analog Backpack* schematic) with enough power, by amplifying any signals sent from an internal DAC system within your processor. Since amplifiers are not completely efficient, i.e., the level of power output is less than that of what is input, your amplifier component will need to be supplied with additional power than that of what is input. To provide the IS31AP4991 component located on your *Analog Backpack* with additional power, you will need to connect your DC Barrel Jack power adapter to the VIN supply, and also connect the other end of the adapter to an AC wall outlet.

Furthermore, the amplifier component has a pin used to enable shutdown. To utilize the IS31AP4991, shutdown will have to be prevented.

1. Locate the IS31AP4991 audio power amplifier within your *Analog Backpack* schematic. Identify the DAC system that the amplifier input originates from. Identify how you will prevent the IS31AP4991 from entering its shutdown mode.

Now, you will use your processor to continuously output an $\mathbf{A_6}$ note (1760 Hz) to the speaker located on your *Analog Backpack*.

2. Create a program (`lab6e.c`) to continuously output an $\mathbf{A_6}$ note to your on-board speaker.

PART F: MAKING A MUSICAL INSTRUMENT

In this part of the lab, you will create a small interactive synthesizer keyboard, using your computer's keyboard.

A **synthesizer** (often abbreviated as synth) is an electronic musical instrument that generates electric signals which are then converted to sound through amplifiers and speakers. Synthesizers typically imitate traditional musical instruments like pianos, organs, flutes, or even vocals; they may also imitate any other wide range of sounds such as ocean waves, animal noises, simple electronic timbres, etc. The interface for a synthesizer is commonly in the form of a piano keyboard.

Within a synthesizer keyboard (as well as a piano keyboard), sound is generated electronically, with either simple switches or dedicated circuits associated with each key, which is unlike a normal acoustic piano, where upon a key being pressed, sound is generated by hammers hitting finely tuned strings.

Most modern pianos contain 88 keys. This means that the piano can generate around 88 different pitches (with each pitch corresponding to a musical note). Each key's frequency can be calculated through the following equation, where n represents the nth key (for n = 1, …, 88):

$$f(n) = 440 \times 2^{\frac{n-49}{12}} \text{ Hz}$$

NOTE: $\mathbf{A_4}$ is a standard pitch that is often used to tune various musical instruments. $\mathbf{A_4}$ is generally the 49th key on a keyboard, i.e., n = 49 in the above equation.

Every twelve keys on the piano make up an octave (see the previous part of this lab for the definition of an octave). Each octave on a piano is organized into a group of two black keys, a group of three black keys, and a group of seven white keys. The **C** note within each octave generally designates the start of the octave (when using a **C** scale), and is always placed before the group of two black keys. When you create your synthesizer keyboard later in this part, you will emulate notes from the 6th octave (see *Figure 3*). The 6th octave is chosen because it is the lowest octave in which all twelve notes have frequencies within the allowable range of our speaker's rated frequencies.

In addition to emulating the 6th octave, your synth will also emulate two notes in the 7th octave, C_7 and $C^{\#}_7$. To play each of these fourteen notes, you will use a specific key on your computer's keyboard (see *Table 1* for details about the layout of these twelve notes on your computer keypad, as well as each of the notes' frequencies). Your synthesizer must be able to play each of these note frequencies in two different **modes** (i.e., with two different waveforms): **sine**

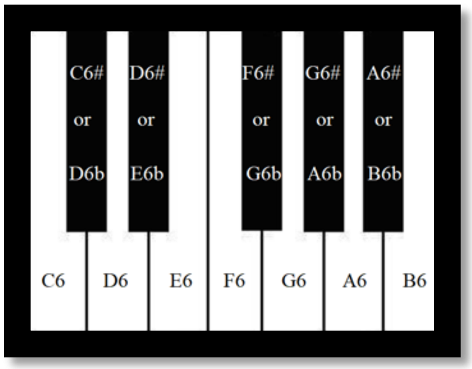

Figure 3: Piano key layout for 6th octave

and **sawtooth**. To accomplish this, your program should use **two** separate look-up tables.

To switch between each of the two modes (**sine** and **sawtooth**), you **must** press 'S' on your computer keypad.

Computer Key	Musical Note	Note Frequency (Hz)
'W'	C_6	1046.50
'3'	$C^{\#}_6/D^b_6$	1108.73
'E'	D_6	1174.66
'4'	$D^{\#}_6/E^b_6$	1244.51
'R'	E_6	1318.51
'T'	F_6	1396.91
'6'	$F^{\#}_6/G^b_6$	1479.98
'Y'	G_6	1567.98
'7'	$G^{\#}_6/A^b_6$	1661.22
'U'	A_6	1760.00
'8'	$A^{\#}_6/B^b_6$	1864.66
'I'	B_6	1975.53

Table 1: Computer synthesizer keyboard details

Each note on your synth should only play for *roughly* as long as you hold the corresponding key, and there must be no noticeable delay from the time a key is pressed to the time the corresponding note is output to your speaker **(see the appendix for more details).** You are **not** expected to play more than one key at the **same** time.

NOTE: It is recommended that you try to output a single frequency of both a **sine** and **sawtooth** waveform, switching between the two when 'S' on your computer keyboard is pressed, before you attempt to complete the entire program.

1. Write a C program (`lab6f.c`) to create the specified synthesizer keyboard program. **Each of the outputted note frequencies must have no more than ±2% error from the given frequency/period**.

PRE-LAB PROCEDURE SUMMARY

1. In Part A, configure an ADC system for the first time, and then measure two constant voltage waveforms generated from your DAD/NAD. Take two screenshots, one of each waveform measured.
2. In Part B, create a voltmeter program.
3. In Part C, configure a DAC system for the first time, and then generate a constant voltage waveform. Measure this waveform with your DAD/NAD, and then take a screenshot of the measured waveform.
4. In Part D, emulate a sine waveform with a look-up table and a DAC system.
5. In Part E, create a program so that your processor outputs its first musical tone!
6. In Part F, create a synthesizer keyboard program.

IN-LAB PROCEDURE

1. Demonstrate your voltmeter program, as described in Part B.
2. Demonstrate your synthesizer program, as described in Part F.

APPENDIX

A: COMPUTER KEYBOARD REPEAT DELAY

Your computer can most likely detect a keypress made to a connected keyboard in the order of picoseconds. To prevent a single keypress from being repeated unintentionally, your computer's operating system likely has a built-in repeat delay for any connected keyboard. Sometimes (*like in this lab*), we want our computer to recognize a single keypress multiple times, as **QUICKLY** as possible.

To change the "Repeat delay" settings within the Windows operating system, perform the following steps:

1. Navigate to the *Keyboard Properties* dialog box from the Start menu (or alternately, through *Devices and Printers*). (In Windows 10, type *Run* at a Cortona prompt, then type *main.cpl @1*. The third letter in *cpl* is a lowercase L and the number 1 follows the @.)
2. Click on the *Speed* tab (see *Figure 5*)
3. Use the sliders beneath *Repeat delay* to increase/lower the delay used before a single keypress is repeated
4. Click *Apply* to confirm any changes made

NOTE: The "Repeat rate" setting also available in the *Speed* tab is the frequency at which a single keypress is repeated, after the initial repeat delay. The configuration of this setting is **not** vital to this lab.

REMINDER OF LAB POLICY

Please re-read the *Lab Rules & Policies* so that you are sure of the deliverables that are due prior to the start of your lab. You must adhere to the *Lab Rules and Policies* document for **every** lab.

LAB 6: ADC, DAC, AND MUSIC!

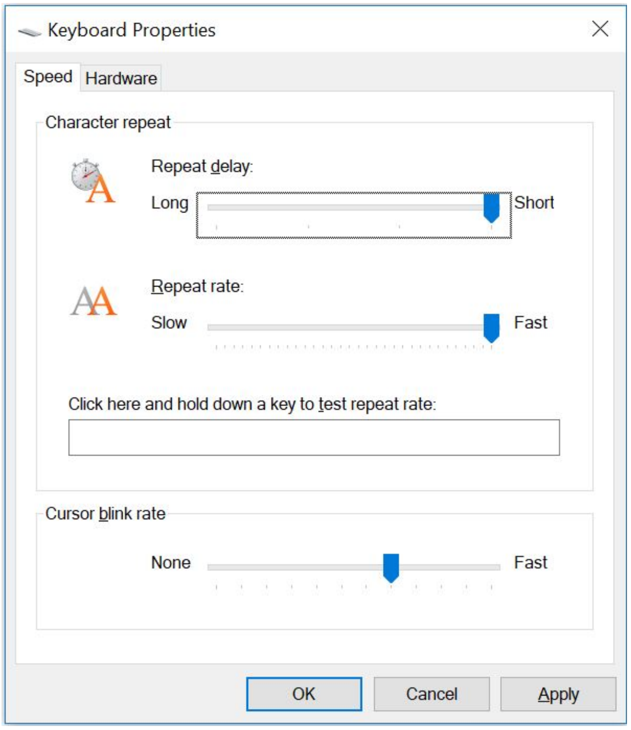

Figure 4: Repeat delay setting in Keyboard Properties