EEE 212-Microprocessors Off-Lab Assignment 6

Due Date: 21.12.2022 - 13.30

1 Motivation

Switching between analog and digital domains is an essential concept of Electrical and Electronics Engineering. Recall that, during your laboratory studies of EEE211 (Analog Electronics), you have dealt with analog signals. You applied some set of operations over various type of signals. For example, to modulate voice signal onto a carrier wave you mixed (multiplied) these signals and to attenuate unwanted frequency components of various signals you implemented low-pass filters, high-pass filters and band-pass filters. So it is natural to look for means to conduct such operations in digital domain with the help of microprocessors, FPGAs, computers.

As a first step of digital signal processing, one should find a way to represent analog signals in digital domain. To do so, we use analog-to-digital converters (ADCs). Analog-to-digital converters are used to sample analog signals, that is, an ADC measures the voltage level of an analog signal at a certain time and reports digital representation of the measurement. Digital representation of a sample point is nothing but an n-bit number. After sampling a signal multiple times and recording samples into the memory of the digital logic device (microprocessor etc.) that is used, you will obtain digital representation of a signal for a certain duration of time. Then, you are ready to conduct discrete-time counterparts of continuous-time operations using the processor.

After processing samples in digital domain, for turning back to analog domain we use digital-to-analog converters (DACs). DAC is a device that takes digital input (an n-bit number) and generates a voltage that is proportional to the n-bit number.

For this lab assignment, you will use internal ADC and DAC of your FRDM-KL25Z to build your own signal generator! Though it seems as a challenging task, this assignment sheet will guide you thoroughly to ensure that you are not lost and efficiently use your time in order to complete the task. I strongly suggest you to read all parts of this paper before getting into the implementation of the task.

2 Detailed Description and Implementation

As it is mentioned, you will implement a signal generator by using FRDM-KL25Z, internal ADC and DAC of KL25Z, a push-button and a power supply. The signal generator that you will implement must be capable of:

- Generating sinus, sawtooth and square waves. Of course, you should not generate them simultaneously. Type of the wave that will be generated is to be determined by a push-button.
- Amplitude of the signal that you generate must be proportional to the reference voltage that will be sampled by ADC.
- Once the push-button is pressed, type of the wave that is generated should be changed, i.e., if the current signal is a sinusoid, once the button is pressed the circuit should generate a sawtooth wave. When the button is pressed again, a square wave should be generated. Hence, as the push-button is pressed, you should switch between the possible set of signals (sinusoidal, sawtooth, square).

At this point, you should realize that we are going to perform three tasks simultaneously. While instructing ADC to take samples and recording the conversion results, we should fed data to DAC and check if the push-button is pressed or not to take the appropriate action.

Before getting into implementation of the system, one should determine the flow of the program. Is the program going to wait until ADC generates conversion result or until the push-button is pressed? If the program is going to wait until such event, then how it can generate a signal continuously while polling for these events? Now, necessity for interrupts emerges.

It is evident that the program should fed data to DAC whenever it can. Hence, we conclude that we should load data into DAC registers continuously and we do not serve to DAC in an interrupt basis.

To respond when push-button is pressed, we have two options. Either the program will wait until the push-button is pressed or we serve based on interrupts. At this point you should question that do we really have to use interrupts in this case? Why we would have to wait until the push-button is pressed if we are not going to use interrupts? Can not the program serve for DAC and check if the button is pressed or not from time to time? The answer is: it can not. Because, we need to capture when the button is pressed. If the program is busy with other instructions when the button is pressed and when the time comes to check if it is pressed or not, it is probable that the button is released. As a result, the program can not detect the event. All in all, we need interrupts in these type of situations.

When it comes to dealing with ADC events, you can either use interrupts or check if ADC completed the conversion while serving to DAC and respond appropriately if a new conversion result arrives.

2.1 ADC

KL25Z has an onboard ADC which you can configure and access conversion results directly from your main program. If the MCU does not have an onboard ADC, to perform analog-to-digital conversions one has to obtain an external ADC and interface the external ADC through the pins of the MCU. Having an onboard ADC brings advantages together with disadvantages. Needless to say, bringing external components to the project environment makes the debugging and development process more prone to error. Hence, having ADC as an onboard component not only makes the implementation phase easier but also makes the sampling process more stable. Though, the onboard ADC comes with too many functions and usually learning to configure and operate such complicated components requires detailed understanding of the component's technical documents. If you are interested in capabilities and have a complete understanding about what is actually going on, you should refer to the datasheet of KL25Z.

For purposes of this assignment, we do not expect you to learn how to configure the onboard ADC. Instead, an example program will be uploaded that is capable of sampling a reference voltage and changing the color of onboard led of FRDM-KL25Z with respect to the sampled value. You should give it a try and ensure that everything works fine. One important remark is that, to generate reference voltage that is going to be sampled, you can use an adjustable power supply. If you have access to a potentiometer, you can use a fixed power supply together with it to generate an adjustable reference voltage to be sampled. After you flash the example program to the memory, locate the appropriate pin to connect reference voltage. As the reference voltage varies, color of the led should change. If it is the case, you are good to go.

If you examine the example ADC program, you should notice that the program waits until the ADC finishes conversion and does nothing while waiting. As it is discussed above, you should modify the code such that the program checks if the conversion is completed or not while dealing with other "responsibilities". We argued that such an approach will not work for push-button problem because there is the possibility of "missing" the moment that the button is pressed. But we can not "miss" the conversion result of ADC (Why it is like that?). So you can use the latter approach safely.

Another way is that using an ISR to handle ADC events. If you can make it work, it is also fine. But I see it challenging because of various reasons.

VERY IMPORTANT REMARK: You should not feed more than 3.3 Volts to the pins that you will be using as an input to ADC.

2.2 DAC

You will use the onboard DAC of FRDM-KL25Z in order to generate a sinus wave, sawtooth wave or square wave. To do so, you should evaluate one period of all three functions and store them in separate arrays (or in a multidimensional array). Frequency of the signal that you generate should be equal to ten times one of your group members' Bilkent ID's last two digit. For examlpe, my ID ends with 15. Hence, frequency of the signal that my program will generate should be equal to 150 Hz. Note that frequency of the resulting signal depends on not only the instructions that is related to DAC but also other instructions that are in the same scope with them.

You will also be given the sample code that operates the onboard DAC. Modify it such that the program is able to generate not just a sine wave, but also triangle and square waves.

Finally, what you should do is integrate ADC and DAC components. Recall that what you will be sampling with ADC is a reference voltage fed to the MCU by an adjustable power source. Amplitude of the signal that you will be generating using DAC **MUST** be directly proportional to the reference voltage that you have sampled. Amplitude of the signal can be any multiple of the reference voltage. You are free to choose the coefficient.

VERY IMPORTANT REMARK: One important parameter in the procedure above is number of samples that you will be taking in one period. Note that even though you can compile a program without errors, you are subject to hardware limitations. As a result, if you try to store too much samples, you may encounter runtime errors such as **HardFault**. So, try to keep everything simple.

2.3 Push-Button

You should define an ISR (Interrupt Service Routine) to act when the button is pressed. This is the final step of your implementation. As the button is pressed, type of the signal that is generated by DAC should be switched between sinusoidal, triangle and square. To do so, you may need to define a global variable (Why?).

3 Grading Rubric

- Sampling a reference voltage and demonstrating it somehow: **10 points**
- Generating a sinusoid and demonstrating the resulting wave via oscilloscope: **10 points**
- Acting on push-button event and demonstrating it somehow: **20 points**
- All components of the system works as described: **60 points**