NTNU – Trondheim
Norwegian University of
Science and Technology

TDT4258 Low-Level Programming
Laboratory Report

Assignment 2 - Making sounds

Group 16

Petter John Fotheringham Sørlie
Odd Kristian Kvarmestøl
Håvard Aspen Løvik

October 18, 2017

1 Overview

The exercise required us to get familiar with how to make sounds using the development board. Learning how to use interrupts, GPIO and the DAC in C was central in completing the assignment. We were supposed to make three sound effects and a start up melody that we may want to use in our game. Our solutions ended up having 4 sound effects and one melody. The melody plays at start up and all sound including the melody can be played by pressing buttons on the gamepad. We have also made a stop button to stop the current sound playing. A feature that we implemented that may not be normal is the ability to play multiple tracks at once, so one track can be the base while another can be the melody. The baseline solution use polling to read the button values and the timer so we know when to play the sounds and when to send information to the DAC. No energy saving methods were implemented in the baseline solution. The improved solution uses interrupts instead of polling, in addition to other energy saving methods as deep-sleep.

1.1 Baseline Solution

Our solution is based around a sequencer. The sequencer takes a list of notes, consisting of a frequency and duration, and plays them, one after the other. The sequencer also supports multiple tracks, called voices, that can be played simultaneously. A set of voices are called a song. A song can be started by passing one to the procedure *seq_play_sound*. This procedure resets the song and sets it as current.

The song is started, both at the start of the program and when a button is pressed. The button pressed logic is located in the main loop of the program.

When the timer has reached a certain number of ticks, the main loop will trigger the procedure for generating a new sample and pass it to the DAC.

1.2 Improved Solution

The primary difference between the baseline and the improved solution is the use of interrupts. Where the baseline solution use polling for both feeding the DAC and handling the user input, the improved solution utilize interrupts for both.

To further improve the performance of the improved solution, we disable the DAC and timer when no sound is playing. We also set a higher energy saving mode (deep sleep) when no song is playing. We are not able to set the energy saving that high while a song is playing because that would prevent the timer interrupt from firing.

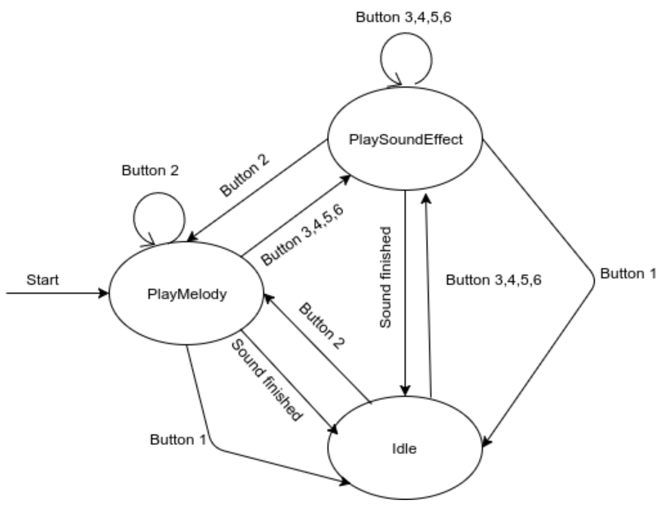

Figure 1.1: The FSM to our solution.

Other than these improvements the improved solution is equal to the baseline solution.

2 Energy Measurements

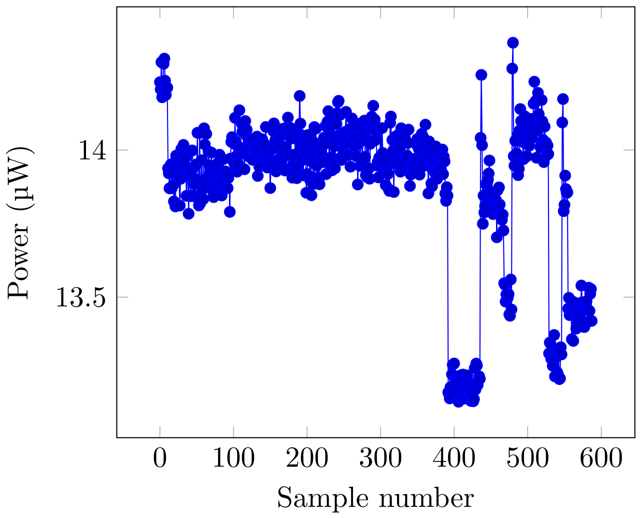

Figure 2.1: Energy consumption for the baseline code. The points hovering around 14 μW are sampled when we are playing a sound while the points below 13-5 are sampled when idle.

The power measurement samples were taken while we first played the start up melody, and then played some sound effects. As we can see in Figure 2.2 the energy saving methods helped a lot when idle. The main reason for the energy usage being so low when idle is the deep sleep mode. If we compare Figure 2.1 with Figure 2.2 we can see that the idle power usage is about 13 μW less on the improved solution. The graph also shows that the energy usage when playing a sound isn't that grate with a difference around 3 μW. This is mainly because the two solutions do pretty much the same when they are playing. The reason we can see a difference is because we use timer interrupts with SLEEPONEXIT instead of polling the timer to know when to send data to the DAC. We are pretty happy with the energy usage when the system is idle, but we would like to have the power usage when playing lower. We think this can be solved by implementing the DMA methods so the timer and the DAC can communicate without interrupting the CPU while playing. This is not implemented yet, but is an option in the future.

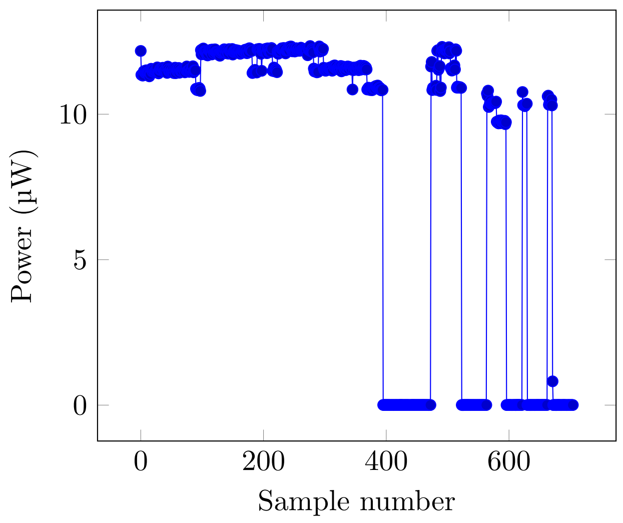

Figure 2.2: Energy consumption for the improved code. The points hovering around 10 are sampled when we are playing a sound. The points close to 0 μW are sampled when idle and clearly shows the effect of the sleeping mode.