NTNU – Trondheim
Norwegian University of
Science and Technology

TDT4258 Low-Level Programming
Laboratory Report

Assignment 2

Group 1:

Sigvart M. Hovland
Alf Martin Eggan
Karl Andreas Eggan

November 7, 2016

1 Overview

In this Exercise we were supposed to implement Sound effects using the on chip Digital-to-Analog-Converter(DAC) that were connected to an amplification circuit on the Development Kit(DK) using the C programming language. We were supposed to produce the Sound using one of the timer circuits on the chip to feed (generated) sound-data/samples to the DAC. The program should play a short start up melody when it started. Also it should react to input from the GPIO pins connected to the buttons on the additional board and play a sound/sound-effect when they buttons were pressed.

Improvements to the solution were to put the processor into deep sleep(EM2) and disabling parts of the SRAM[4] while it was not feeding samples to the DAC. The final product became a sampler which plays different sound samples when keys are pressed. This way you can actually play the full PPAP[5] song using the buttons on the board.

1.1 Baseline Solution

To implement the baseline solution we used the skeleton code. We expanded this code by implementing the setupGPIO, setupNVIC, setupDAC and setupTimer. To see how these methods were set up see the source code. The first two methods are very similar to the methods in exercise 1. Then we made 3 modes for sounds(0-square, 1-saw, 2-reverse-saw). This mode were set in the busy-wait loop. This value was read in the timer interrupt. In the TIMER1_IRQHandler() we first cleared the pending interrupts by writing 1 to TIMER1_IFC. Then we calculated the values based on the sound type mode from the busy-wait loop and pushed these to DAC0_CH0DATA and DAC0_CH1DATA. This gave us 3 sounds triggered by the buttons.

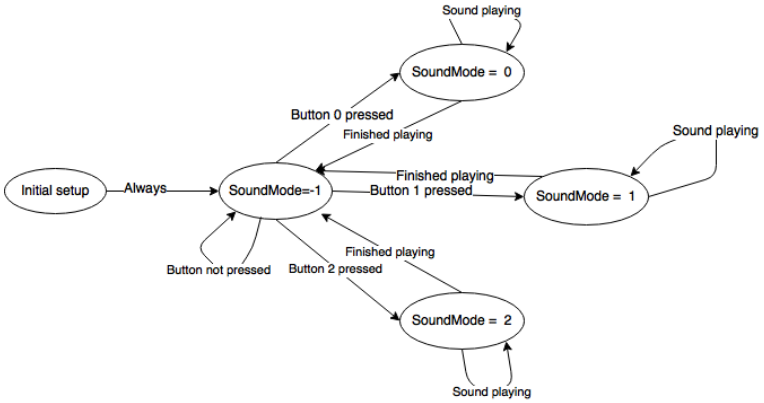

Figure 1.1: Baseline solution state machine.

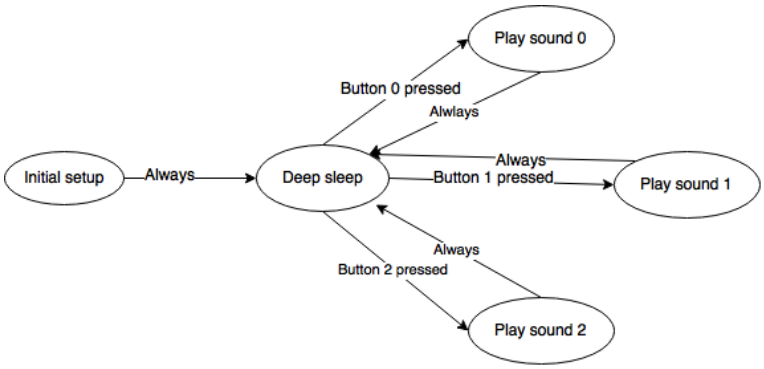

Figure 1.2: Improved solution state machine.

1.2 Improved Solution

In the improved solution we replaced the code from the busy-wait to the interrupt functions for GPIO and but the processor into Deep sleep[3]. In addition we made methods to play raw-files from wav. To make it more energy efficient we disabled RAM block 1-3 by writing 7 to EMU_MEMCTRL register. We also had to make some changes to the linker script so that it wouldn't allocate heap in the area which we had disabled.

The .raw files were converted to linkable objects using the command 'arm-none-eabi-ld -r -b binary -o filename.o filename.raw' unfortunatly this created objectfiles with .data sections which the compiler then tried to put into RAM due to instructions in the linkerscript which was too small. Hence we had to rename the sections using the command 'arm-none-eabi-objcopy –rename-section .data=.rodata ppap.o' which then assigned it to .rodata which will be put into the flash by the linker script.

2 Energy Measurements

We used the energy Profiler in Simplicity Studio to measure the power consumption of both the baseline and improved over 50 samples. You can see both of the idle measurements in Figure 2.1. As you can see the difference is quite large as the improved solution seems to go towards 0µW.

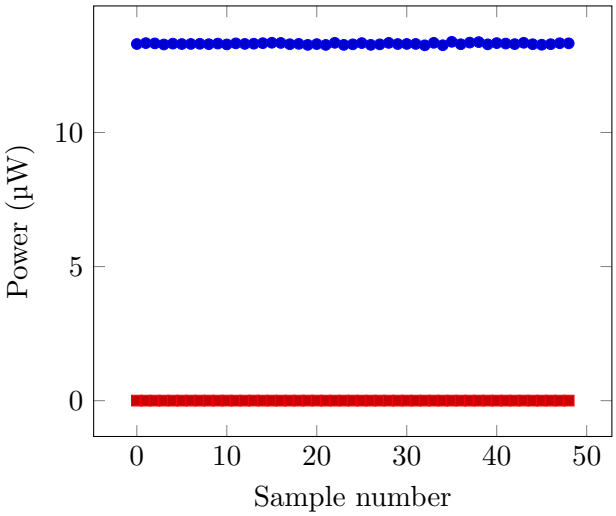

Figure 2.1: Baseline vs. improved solution power consumption

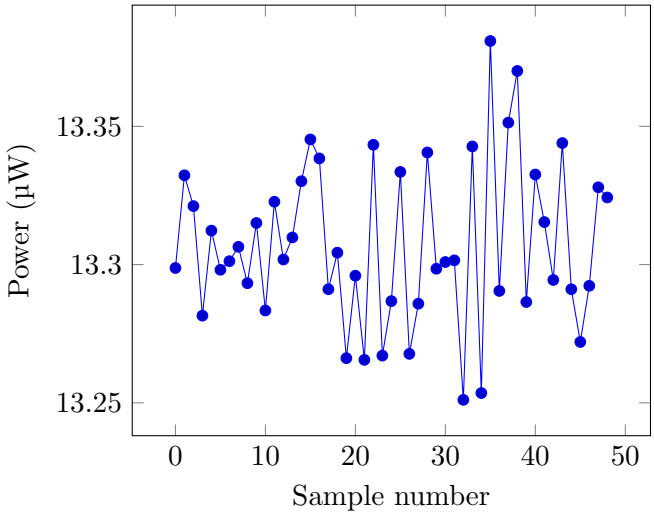

Figure 2.2: Baseline idle power consumption.

As you can see from Figure 2.2 the average power consumption for the baseline solution is about 11µW.

Where the blue plot in Figure 2.3 It's about $3 * 10^{-3}\mu$W. The Red plot in 2.3 show the energy consumption of the improved solution without the SRAM block 1-3 powered off. So you can see that powering of the SRAM block saves about $2 * 10^{-3}\mu$W in power consumption. Both these plots represent the processor in an idle state where nothing is happening.

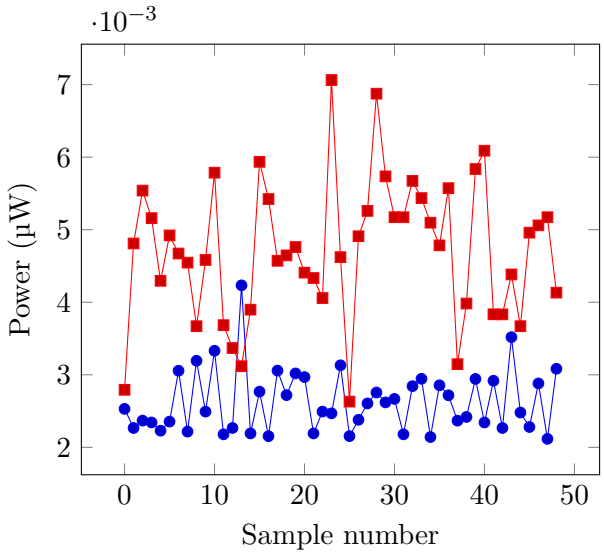

Figure 2.3: Idle power consumption of the improved solution with SRAM block 1-3 turned off vs the improved solution with SRAM 1-3 turned on.

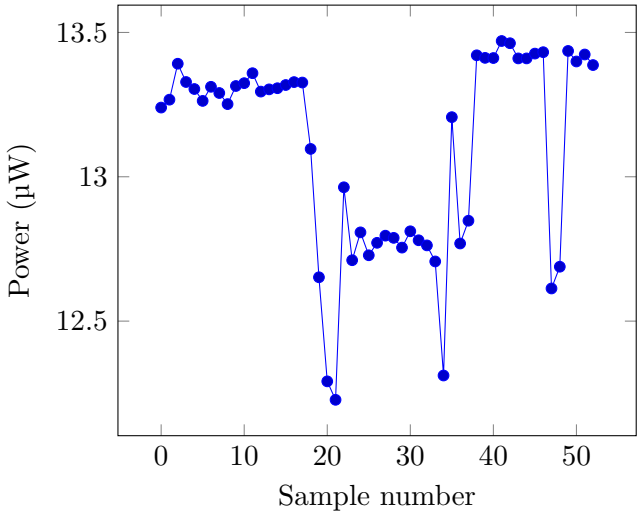

Figure 2.4: Baseline button pressed power consumption.

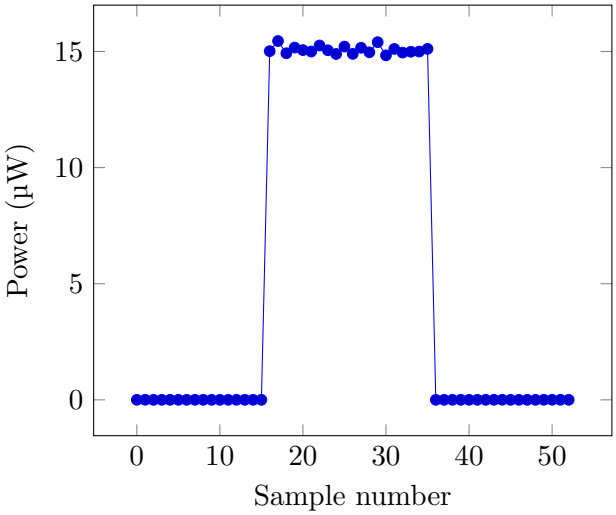

Figure 2.5: Power consumption plot for button press in improved solution.

Looking at the plot 2.6 between button pressed in the baseline and the button pressed in the improved version is quite interesting. As we can see that a lot more power is consumed in the improved version on a button press. While in the baseline the power consumption actually goes down when a button is pressed. This is due to the fact that in the improved version we use audio samples which are sampled at 8000 Hz and have a lot of variation demanding a lot more processing from the CPU and switching from the DAC. While in the baseline we generate quite simple samples and the processor is probably halted/(nop instructions) a few times when it writes to the DAC waiting data and confirmation causing the power consumption to drop compared to the busy-loop constant drain. However the overall power consumption in the improved is better due to it entering sleep mode in between playing samples.

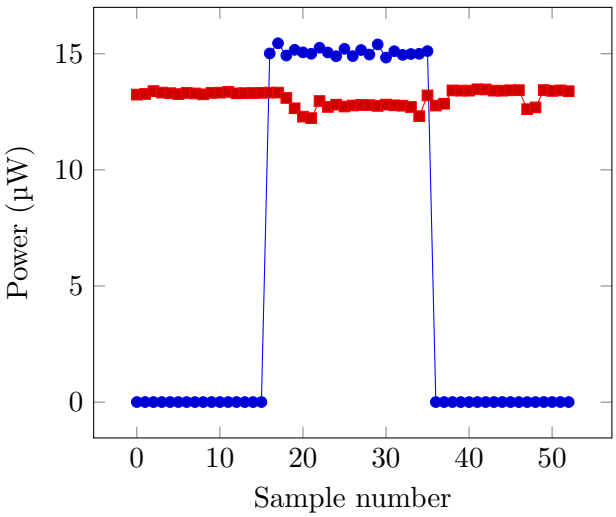

Figure 2.6: Power consumption plot for button press in improved solution vs baseline

2.1 Conclusion

In conclusion the improved solution reduced the amount of power consumed by the processor in an idle state. Turning of unused hardware/circuits such as SRAM blocks further increased the power saving in deep sleep(EM2). However a lot of improvements could be made to the improved solution and disabling SRAM blocks only saves power in idle mode. So in this use case it's not making that much of a difference except if you had left the device on for a long period of time.

The best way to save power in our improved solution would be to enable DMA[2] in Ping-pong mode connected to the DAC[1] through the Peripheral Reflex System (PRS) pre-scaling the DAC to around 16-8 KHz and make it request a new sample each time it's done processing a sample. This would lend it self very well to our solution since we stored all of our audio data in the flash memory. With pointers to the start and end memory address of the audio sample. Using this method would off load the processors a lot and we could have just put it into EM1 while a sample is playing and woken it up only to change the start and end memory address for the DMA/DAC. Also ofcours putting it in deep sleep(EM2) in between and wake it up when a button is pressed which we allready do..

Other things that could be done is using a Low energy timer instead of a regular timer and go into EM3 letting only the low frequency clock tree be enabled. We could also have put the DAC in another mode than continuous such as sample-and-hold. Since this consumes less energy than having it in continuous mode.

Another thing if we were not using predefined sampled we could have used the sine generator on the DAC which is what uses the least power of all the modes on the DAC. And just frequency modulate sine waves with interrupts with a LETIMER through the PRS system. However then we would have quite a limited amount of sounds we could have generated.

Bibliography

[1] Digital to analog converter an0022 - application note. page 11.

[2] Direct memory access - an0013 - application note. pages 1–19.

[3] Energy modes an0007 - application note.

[4] Energy optimization, an0027 - application note. page 4.

[5] Ppap - youtube video.