NTNU – Trondheim
Norwegian University of
Science and Technology

TDT4258 Low-Level Programming
Laboratory Report

Exercise 2

Group 9:

Toms Stūrmanis
Matej Spetko

October 18, 2017

1 Overview

Main task is to implement sound generation using DAC, by sending digital samples to it, which gets converted to analog signal. Different buttons makes different sounds. Upon rebooting, main sample is played.

Task was implemented using two different approaches. First one used simple looping mechanism to read registers and write samples. Second approach used interrupts to generate samples.

1.1 Usage

Upon restart intro song is played. To replay intro press button 1. To play other songs press button 2 and 3. To stop current song, press any other button.

1.2 Music implementation

Sounds are created using two timers, because there is not enough space in MCU to use wav files for example. Files with large sample count are large.

TIMER1 creates period of required note, TIMER2 changes notes period after note has been played for needed time, then it disables TIMER1 for pause period, so change of note, or pause is hearable.

So for example to play note A on 440 Hz for one second, it only needs three integers. Note frequency, play duration, pause duration.

1.3 Baseline Solution

Baseline solution uses polling, to check all needed registers. Needed registers are push-button register, and TIMER1 and TIMER2 counter registers. Because polling takes time to check each if statement, it's hard to calculate needed frequencies, that's why in baseline solution, music is played only in theory.

GPIO pins 8-15 on portA are set to output, for LEDs. Pins 0-7 on portC are set as input, for buttons.

DAC is set to sample frequency 437.5 kHz, with differential output to channels 0 and 1.

TIMER2 is set to minimum frequency of 214 Hz, by setting TOP as $2^{16}-1$, and using default HFCLOCK = 14 MHz. TIMER1 is set depending on needed note frequency.

State machine for baseline solution is seen in Figure 1.1.

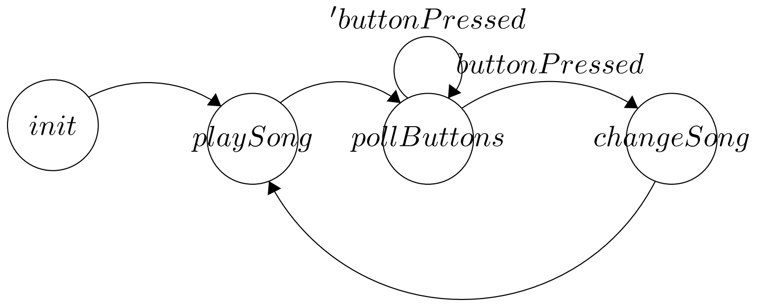

Figure 1.1: Polling solution state machine

1.4 Improved Solution

Improved solution uses interrupts instead of polling, so it's possible to acquire needed frequencies and timings. Also it's possible to save energy by sleeping, when no music is played. Also all RAM blocks are powered down, except first 32 KB. When music has done playing, it goes to sleep mode EM2.

Solution is seen in state machine in Figure 1.2.

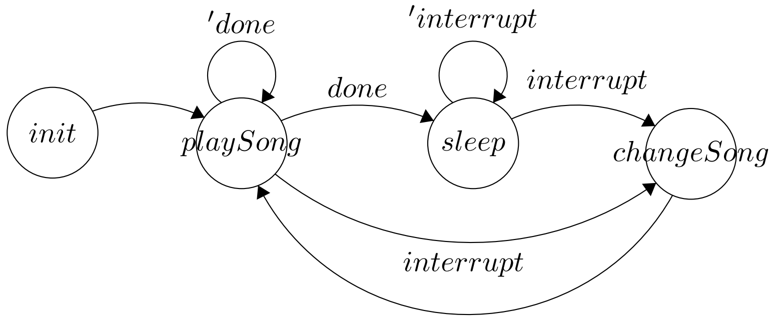

Figure 1.2: Improved solution with low power mode EM2, state machine.

2 Energy Measurements

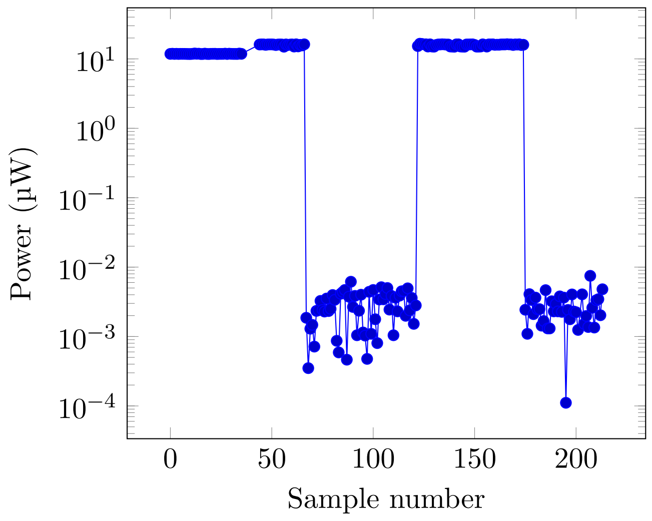

Figure 2.1: All 2 solutions in one plot.

As seen in first solution power consumption is staying high all the time, because of polling.

For improved solution, power consumption is only high for amount of time sound is played, after that it goes back to sleep mode.