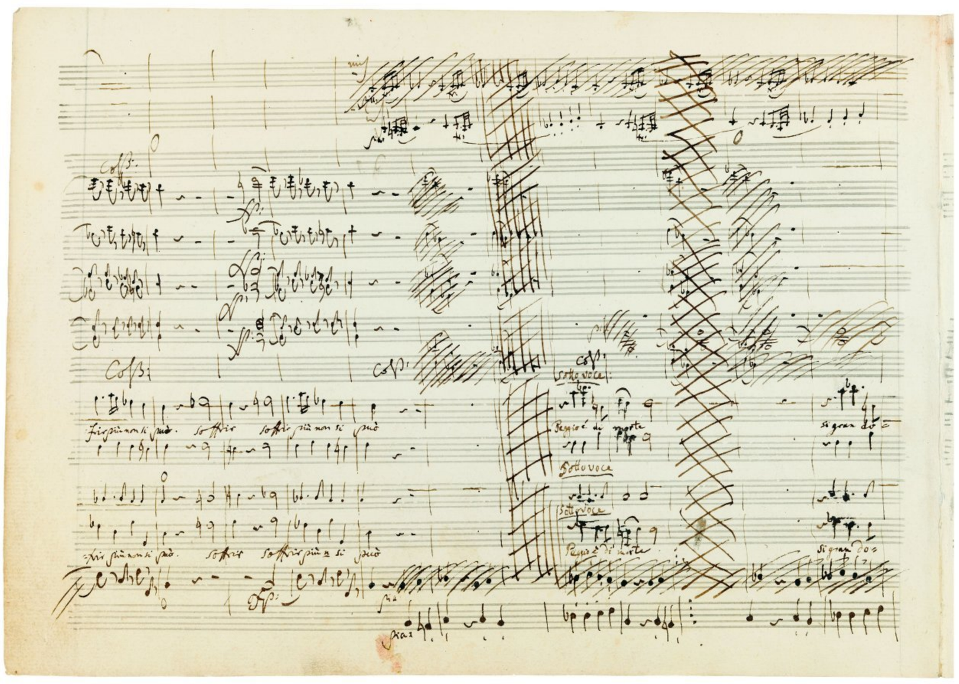
C8

Timers/Counters and Pulse-Width Modulation

In this exercise you will learn how to use the timer/counter hardware of your microcontroller to produce accurately timed output signals. You will build a minimalist sound synthesizer that uses a timer as tone oscillator with adjustable frequency. Making use of pulse-width modulation (PWM) to produce an analogue signal you will control the amplitude of the sound signal.



Schedule

Preparation Time	:	3 hours
Lab Time	:	3 hours

Items provided

Tools	:	
Components	:	4066 Quad Analogue Switch IC, Stereo In-ear Headphones, 3.5mm Stereo Jack Socket, 2×20kΩ potentiometer, 2×220μF/10V Capacitor, 2×Signal diode (preferably Schottky), 6×220Ω Resistor, Hook-up wire. [*Components to be retained by the student after the exercise.*]
Equipment	:	Oscilloscope[8]
Software	:	`avr-gcc, ASCII text editor`

Items to bring

Il Matto

Identity card

Laboratory logbook

Toolkit

Version: November 22, 2017

Before entering the laboratory you should read through this document and complete the preparatory tasks detailed in section 2.

> **Academic Integrity** – *If you wish you may undertake the preparation jointly with other students. If you do so it is important that you acknowledge this fact in your logbook. Similarly, you will probably want to use sources from the internet to help answer some of the questions. Again, record any sources in your logbook.*

You will undertake the exercise working with your laboratory partner. During the exercise you should use your logbook to record your observations, which you can refer to in the future – perhaps to write a formal report on the exercise, or to remind you about the procedures. As such it should be legible, and observations should be clearly referenced to the appropriate part of the exercise. As a guide the ✎ symbol has been used to indicate a mandatory entry in your logbook. However, you should always record additional observations whenever something unexpected occurs, or when you discover something of interest.

For each Task you should create a new directory so that you have a working version of every program at the end of the lab. Remember to place comments in your code.

You will be marked using the standard laboratory marking scheme; at the beginning of the exercise one of the laboratory demonstrators will mark your preparatory work and at the end of the exercise you will be marked on your progress, understanding and logbook.

Notation

This document uses the following conventions:

✎	An entry should be made in your logbook

1 Introduction

This lab introduces you to the configuration and use of timers to produce accurately timed wave forms while keeping the CPU-core free for other processing.

1.1 Outcomes

At the end of the exercise you should be able to:

- Configure a timer for a desired mode of operation
- Use a timer to produce a rectangular wave of a given frequency
- Use a timer to produce a pulse-width modulated wave
- Use pulse-width modulation to produce an analogue output signal

1.2 Overview

Most current sound-synthesis systems use sampled wave forms which are converted by digital-to-analogue (D/A) converters into audio signals. Before memory became so affordable that this technique could be widely employed, analog synthesizers were common. Figure 1 shows a sketch of the typical configuration of such a synthesizer. An oscillator (OSC) produces a periodic wave of audible frequency (about between 20Hz and 18kHz) that is then filtered by a voltage-controlled filter (VCF) and its amplitude is varied over time by a voltage-controlled amplifier (VCA). How the amplitude and the filtering varies is controlled by a different wave form that is produced by a low-frequency oscillator (LFO) and shaped with a specific envelope in the ADSR[1] unit. The synthesizer could be controlled by a keyboard or with a sequencer (SEQ).

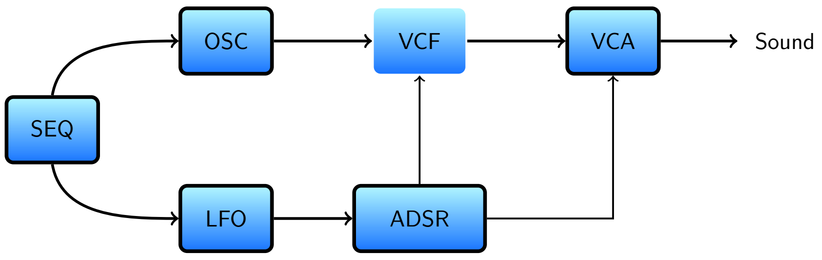


FIGURE 1: Typical components of an Analogue Synthesiser

In this lab you will use the `Il Matto` board to create the different components in software (LFO, SEQ), on-chip hardware (OSC), and a small circuit (ADSR) build around the CMOS IC 4066 [7] that contains four analogue switches (VCA). To keep things simple we will leave out the VCF.

[1]Attack-Decay-Sustain-Release

2 Preparation

First read the "Laboratory work" (section 3) below to understand the context of the preparation and see where each item of preparation will be required. Include in your logbook any calculations you make during the design process.[2]

2.1 Timer Choice and Configuration

2.1.1 Tone Oscillator

If one considers the requirements for frequencies (section 1.2), the different capabilities of the timers available on the ATMEGA644P (see the datasheet [3, pp. 93–160][3]), and the clock frequency of the `Il Matto` Board (see the `Il Matto` Quick Reference Card[4]), one will arrive at the conclusion that Timer 1 is most appropriate as tone oscillator.

1. Briefly explain why Timer 1 is a good choice. ✎
2. Determine from the ATMEGA644P datasheet[3] how to configure Timer 1 to work in Phase and Frequency Correct PWM Mode and toggle Compare Match Output A when the counter matches the corresponding output compare register. Draw in your logbook each register that needs to be set, labelled with its proper name[4] and the bit-setting required to configure the timer. ✎
3. Select a prescaler value such that Timer 1 can cover the frequency range from 50Hz–5kHz. ✎
4. Write in your logbook the C-code you need to configure Timer 1. ✎
5. Write in your logbook a C function that takes a frequency as argument and sets Timer 1 to output a square wave of that frequency at Compare Match Output A . ✎

2.1.2 PWM Oscillator for Amplitude Control

1. Select one of the remaining timers as PWM source that can provide two individually controlled outputs. Consider the circuit diagram of the `Il Matto` Board (see the `Il Matto` Quick Reference Card[4]) in your design decision. Briefly explain your motivation for the choice made and explain why Timer 0 may be a bad choice. ✎
2. What is the maximal frequency the PWM can run at on your `Il Matto` board if you want at least 8 bit of resolution for the modulation? ✎
3. Determine how to configure the timer for PWM with the maximal frequency possible at 8 bit resolution. Draw in your logbook each register that needs to be set, labelled with its proper name and the bit-setting required to configure the timer according to your design. ✎

[2]Write out the formula, then write the formula with values *and units* inserted, followed by the result.
[3]If you find this difficult to understand the application notes AVR130[1] and AVR131 [2]) may help
[4]Fill in the values for the n and x variables used in the naming in the datasheet.

2.2 Port/Pin Assignment

1. Make a table with the pins of the `Il Matto` board you will use. The table should lists: description of the signal, port, and pin number. It is a good idea to include such a table in the header comment of any code you write.

2.3 Review

To accomplish the tasks in the allotted time you require knowledge covered in previous lectures and laboratory exercises. If you feel unsure about any of the following topics, please review them and discuss them with your peers before the C8 exercise:

1. How to connect components in parallel and how to connect components in series.
2. How does one use resistors to divide a voltage?
3. How does one use an oscilloscope to determine the frequency of a periodic signal?
4. How can one set or clear a specific bit in a register of the ATMEGA644P?
5. How does one configure a specific pin of the ATMEGA644P as output?

3 Laboratory Work

Create a new folder named "C8" to store the source files for this laboratory.

3.1 Producing an Audio Frequency Square Wave

Your first task is to use a timer to toggle a pin at a specified audio frequency.

- Download the C-code file `square.c` provided at https://secure.ecs.soton.ac.uk/notes/ellabs/1/c8/ into your newly created folder "C8". The code provided in `square.c` has stubs for two functions you need to fill in.
- Following the code written in your preparation, write the function `init_tone()` which will be called once at the start of the program and should configure Timer 1 to use as tone oscillator according to the design you have prepared.
- Write the function `tone()` that can be called to set the output frequency.
- Insert a comment near the top of the file that explains what output pin should be used with this program.
- Set up the program so that it will play at 262Hz ($\approx$ a middle C note) indefinitely.

- Use the oscilloscope to determine the frequency of the output pin. Make an entry in your logbook about the frequency you have set in your program and the frequency you have measured (include any calculations for your measurements in your logbook). By how much differs the measured frequency from the set frequency? What could explain the difference? *If the difference is large debug your program before continuing.*

- Use two 220Ω resistors to build a voltage divider between the port pin you have chosen as output and ground.

- Place your headphones on the table (i.e., not in your ears). Connect your stereo-headphones with both channels in parallel to each other and parallel to *one* of the resistors in your voltage divider. Check the volume of your headphones— if it is too loud divide the voltage further by inserting more resistors in your voltage divider. You can connect both headphone sets in parallel so that both member of the team can listen at the same time.

3.2 Generate a Tone Scale

Make a copy of your current (filled in) `square.c` and name it `scale.c`.

Your second task is to play a scale of musical notes.

- Download the C-header file `et_scale.h` provided at `https://secure.ecs.soton.ac.uk/notes/ellabs/1/c8/` into your folder “C8”.

- Take a look at the `et_scale.h` file and note how your program can determine how long the scale is.

- Edit `scale.c` to include `et_scale.h`.[5]

- Modify `scale.c` so that it plays all tones on the scale up and down indefinitely.

3.3 Play a Tune

The next step is to play a melody and then to improve the sound by adding a bit of circuitry.

If we can play a scale, it is not hard to play simple tunes. But what to play? Fortunately quite few tunes have been encoded into an ASCII text format called abc-notation[6] and can be downloaded through a search interface [9]. This gives us a start at playing something more interesting than scales or random notes.

- Create a new empty folder “C8-3-3”.

- Download the C-header file `et_scale.h` provided at `https://secure.ecs.soton.ac.uk/notes/ellabs/1/c8/` into your newly created folder “C8-3-3”.

[5]Don’t even think of copying the .h file into the .c file; use the `#include` preprocessor command.

- Download the C-code file `melody.c` provided at https://secure.ecs.soton.ac.uk/notes/ellabs/1/c8/ into your "C8-3-3" folder. This file contains a very simple and crude player for abc-files that plays them partly wrong... but is so small that its code is easy to understand.
- Install your `init_tone()` and `tone()` functions in the C-code.
- Run the program on `Il Matto` and see whether it plays a melody. Feel free to look for another tune [9] and insert it in the code (you need to change the line endings to `\n\`).

The hard on-off amplitude of the note playing does not sound very nice. In real instruments each note has a certain envelope function that detrmines its amplitude over time. This envelope depends on the physics of the instrument and how it is played. One of the simplest envelopes to emulate is the sudden rise (called "attack" in music) and exponential decay of the amplitude of percussion instruments.

The discharge of a capacitor over a resistor would provide the type of decay that is common in instruments. If one charges the capacitor quickly one could also get the sharp rise.

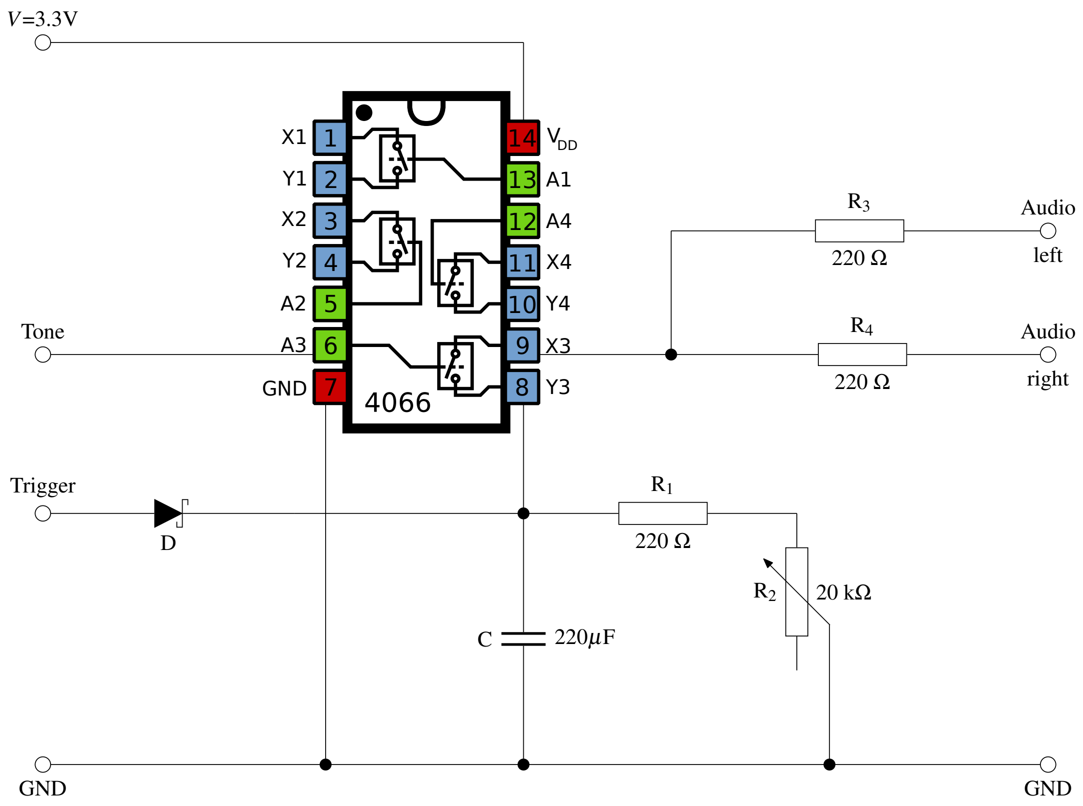


FIGURE 2: Circuit to control the envelope of the amplitude of the sound. Switch 3 was convenient for laying out this diagram; use the switch most convenient for the circuit layout on your breadboard. This circuit is kept simple for the small breadboard on the `Il Matto`; normally the unused inputs of the CMOS IC (A1, A2, A4) and the dangling connector of the potentiometer would be connected to ground. Pinout of IC reproduced from [5].

1. Assemble the circuit shown in Figure 2. Pay particular attention to the fact that the audio signal from the tone generator is *not* fed into the analogue input of the switch, but instead used to turn the switch on and off.

2. What do you expect the output waveform of Figure 2 to look like when a brief pulse is applied at the diode (trigger) and a the tone oscillator is connected to the control input of the switch? Sketch it in your logbook. What is the diode for?
3. Modify your code so that it will send a short (5-15ms) pulse on one of the output pins every time when it plays a new note. Connect this trigger signal to the diode through which it can charge the capacitor.
4. Adjust the potentiometer so that a note will end before the next one starts.
5. Experiment with some settings of trigger pulse length, potentiometer setting and delay between notes. What happens for very short trigger pulses?

3.4 Volume Control with PWM

So far we have focused on the frequency of the output signal. Another parameter that can be varied is the relationship between the length of time the signal is high and the length of time it is low. A signal that changes so quickly between high and low that the receiver cannot follow the change will appear as a time average. For instance if we have a signal that is 3V when it is high and 0V when low and is half the time high and half the time low it will appear to a slow receiver as a 1.5V signal. Now, if the proportion it is on is changed, let's say to be at 3V only 10% of the time, then the signal will appear to a slow receiver as 0.3V. Our eyes are quite slow receivers, so this can also be used, for example to dim an LED. The change in the length of the on-pulse relative to the off-state can thus be used to obtain an analogue output from a single digital output pin, by pulsing it in a specifically timed way. This concept has many applications and accordingly the timers in the ATMEGA644P are set up to provide PWM output.

You will now use a PWM signal to control the amplitude of the sound output. Copy the file `melody.c` and name the copy `volume.c`

1. Add and initialisation function `init_volume()` for the timer you chose in the preparation to be used as PWM source.
2. Add a function `volume(uint8_t x)` that will set the width of the pulse proportional to `x`.
3. Disconnect the trigger signal from the one channel circuit build in the previous task. (This is the connection from the ATMEGA644P to the diode.)
4. Connect the PWM output pin of your timer to the Trigger input of your circuit (cf. Fig. 2).
5. Set the pulse width to a fixed value (e.g. 30%) and check with the oscilloscope that your timer works as intended.
6. Modify the program so that with each note that is played the pulse-width of the PWM output is changed to increase the volume. What will happen in your program when the maximum volume is reached?
7. Verify with the headphones that the volume control works as intended.

4 Optional Additional Work: Stereo Panning

Create a copy of your folder “C8-3-3” and name it “C8-4”. In folder “C8-4” rename `volume.c` into `panning.c`

1. Build up the circuit for a second channel identical to the channel from the previous tasks and wire the headphone in a stereo configuration.
2. Add and initialisation function `init_pan()` for the timer to be used as PWM source. Configure two output pins for the timer.
3. Add a function `pan(uint8_t x)` that will change the pulse width in the two output channels in an inverse way. (When one channel reaches its maximal pulse width the other channel should reach its minimal pulse width.)
4. Connect each of the Trigger inputs to one of the two PWM output pins.
5. Determine the frequency of the PWM signal. Does this correspond to your expectations?
6. Modify the program so that with each note that is played the pulse-width of the PWM output is changed to increase the amplitude in one channel and decrease it in the other channel. Your program should smoothly pan the melody from left to right and back.
7. Adjust the panning speed so that you can easily demonstrate the panning with the headphones and connect the oscilloscope to show both PWM signals during the panning. (Leave this setup in place for marking.)
8. As you will notice from listening to your program, human perception of loudness is not a linear function of signal amplitude. Modify your program to compensate (somewhat) for this. What trade-off needs to be considered?

References

[1] Atmel Corporation. AVR130: setup and use the AVR timers. Application Note 2505A-AVR-02/02, 2002. URL http://www.atmel.com/Images/doc2505.pdf.

[2] Atmel Corporation. AVR131: using the AVR’s high-speed PWM. Application Note 2542A-AVR-09/03, 2003. URL http://www.atmel.com/Images/doc2542.pdf.

[3] Atmel Corporation. ATMEGA164PA/324PA/644PA/1284P Datasheet. Datasheet 8152G-AVR-11/09, 2011. URL http://www.atmel.com/Images/doc8152.pdf.

[4] Steve R. Gunn. Quick Reference: Il Matto, 2012. URL https://secure.ecs.soton.ac.uk/notes/elec1201/IlMattoQR.pdf.

[5] Inductiveload. Pinout diagram of the 4066 quad bilateral switch IC. http://en.wikibooks.org/wiki/File:4066_Pinout.svg, February 2009. Released into Public domain.

[6] Steve Merrony. Abc quick reference card, 2011. URL http://www.stephenmerrony.co.uk/uploads/ABCquickRefv0_6.pdf.

[7] Fairchild Semiconductor. CD4066BC. Datasheet DS005665, 2005. URL http://www.fairchildsemi.com/ds/CD/CD4066BC.pdf.

[8] Tektronix. Digital Storage Oscilloscope (TDS1000C). User Manual Rev. A, 2006. URL https://secure.ecs.soton.ac.uk/notes/ellabs/reference/equipment/std-bench/Tektronix_TDS1000C_2000C_User_Manual.pdf.

[9] Chris Walshaw. Abc tune search, 2012. URL http://abcnotation.com/search.