

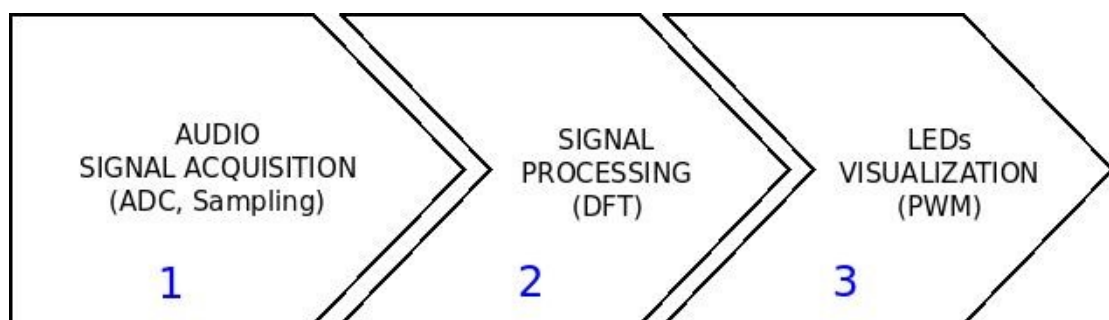
# ATtiny13 – dance lights with DFT

2016-11-20 by Łukasz Podkalicki

This experimental project shows how to use ATtiny13 to synchronize live music and lights with only a few additional components required. Project uses single ADC (*Analog to Digital Converter*) channel for signal acquisition and some DSP (*Digital Signal Processing*) calculations that in real-time separate three frequency bands and make the LEDs dance in sync with the music. This project also proves that real-time DFT (*Discrete Fourier Transformation*) is possible on such small MCU like **ATtiny13**, which offers only **64B of RAM** and **1024B of FLASH**. Project doesn't require any specific equipment and the parts costs are below 3\$. The code is on Github, [click here](#).

## Project Overview

In this project ATtiny13 has a three major tasks: read audio signal, process signal and visualize frequency bands using LEDs. Simplified block diagram is presented bellow.



### 1. Audio Signal Acquisition (ADC, Signal Sampling)

The input signal source can be any audio signal. MCU uses ADC to read analog signal (signal acquisition) with required FS (*Sampling Frequency*). This phase produces N signal samples – discrete signal in the time domain.

### 2. Signal Processing (DFT)

Next, signals are converted from time domain form to the frequency domain using

a DFT algorithm. The product of that phase is N values that represents amplitudes of N frequencies, where particular frequency is,  $F(n) = FS/N$ . Because of the *Nyquist frequency* rule, which is a property of a discrete-time system, we can use only N/2 of samples for further signal analysis. Please note, that to simplify algorithm complexity, only a real part of complex numbers has been used.

### 3. LEDs Visualization (PWM)

At last, MCU put some visualisation based on information from previous phase. Each LED represents a frequency bandwidth. MCU checks all frequency amplitudes, and by toggling LEDs It simulates Software PWM, what in result gives quite good control of LED brightness.

## Top Project Challenges

### 1. DSP calculations require real numbers what consumes a lot of ATtiny13 resources

Calculations on float/double take much of RAM, CPU time and FLASH space. It was a critical issue for the project realization. Solution was to make it possible to replace all real numbers by integers.

### 2. FFT algorithms require a lot of RAM, more than offers ATtiny13

Eliminations of real numbers was a good optimization trick but still, It was not possible to implement efficient algorithm. Solution was, adding precalculated *twiddle factors* for 6-point DFT, using array of short integers and focusing on calculations only for real part of complex numbers (sic!). In result 31 bytes (48.4%) of RAM has been reserved for arrays and 33 bytes (51.6%) was left to use by program.

For some smartasses that are interested in why *twiddle factors* are stored not in PROGMEM but in RAM. It's an optimization. Accessing data from RAM is a bit faster as it takes extra steps to get data from PROGMEM. For more information click [here](#).

### 3. FFT algorithms require more space than offers ATtiny13

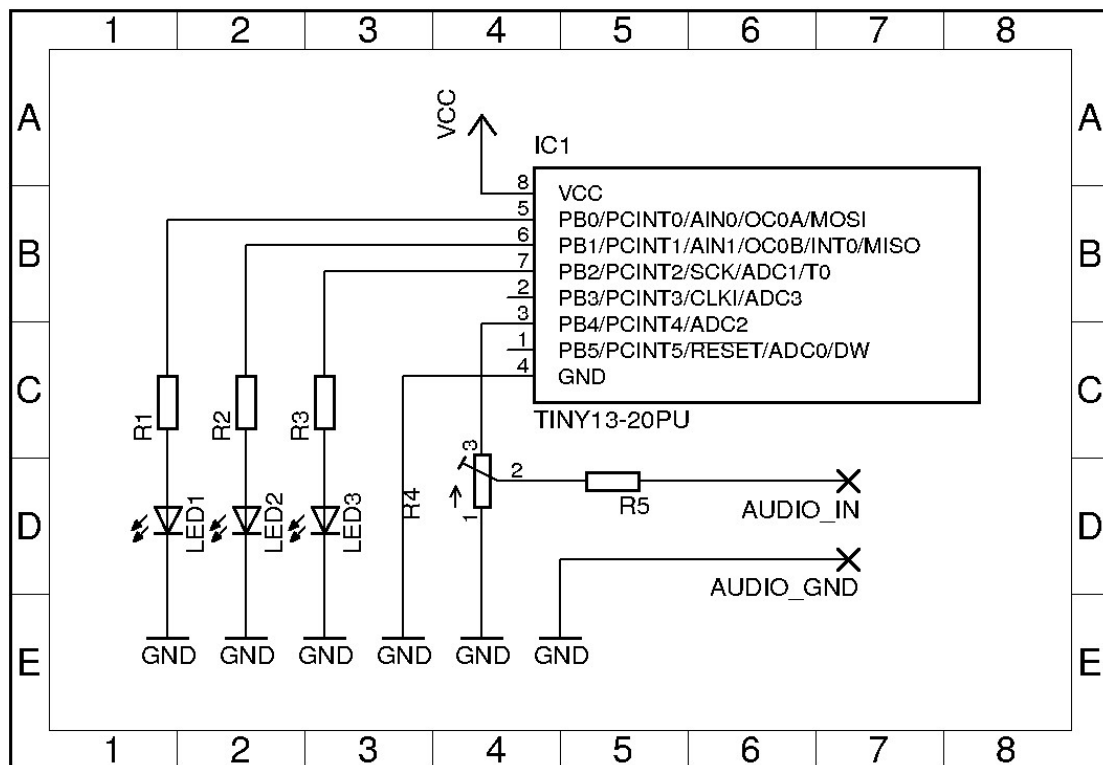
Popular implementations of FFT (Fast Fourier Transformation) require more

program space then offers ATtiny13. It's worth to note, that originally, algorithm was designed to do calculation for both parts of complex numbers and calculate a signal power as a product for next phase. However, it require ~20% more program space and >50% more RAM. Solution was, combine some features of naive version – it is DFT with precalculated twiddle factors using in FFT algorithms and do calculation for only real part of complex numbers what is good enough, as for this project. In result program size is ~464 bytes (45.3%).

## Parts List

- ATtiny13 – i.e. [MBAVR-1 \(Minimalist Development Board\)](#)
- R1, R2, R3 – resistor 560Ω, see [LED Resistor Calculator](#)
- R4 – trimmer 10kΩ
- R5 – resistor 100kΩ
- LED1, LED2, LED3 – red, green and blue (or other colours of your preference) LEDs

## Circuit Diagram



# Firmware

This code is written in C and can be compiled using the avr-gcc. More details on how compile this project is [here](#).

```
/**
 * Copyright (c) 2016, Łukasz Marcin Podkalicki <lpodkalicki@gmail.com>
 * ATtiny13/018
 * Dance lights with simplified DFT (Discrete Fourier Transformation)
 * --
 * Settings:
 * FUSE_L=0x6A
 * FUSE_H=0xFF
 * F_CPU=1200000
 */

#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>

#define LED_RED PB0
#define LED_GREEN PB1
#define LED_BLUE PB2
#define BASS_THRESHOLD (300)
#define MIDRANGE_THRESHOLD (40)
#define TREBLE_THRESHOLD (30)
#define N (6)

const int8_t W[N] = {10, 4, -5, -9, -4, 5};
int16_t samples[N] = {0};
int16_t re[N];
volatile uint8_t acquisition_cnt = 0;

ISR(ADC_vect)
{
    uint8_t high, low;

    if (acquisition_cnt < N) {
        low = ADCL;
        high = ADCH;
        samples[acquisition_cnt] = (high << 8) | low;
    }
}
```

```

        acquisition_cnt++;
        ADCSRA |= _BV(ADSC);
    }
}

static void
dft(void)
{
    uint8_t a, k, n;

    for (k = 0; k < N; ++k)
        re[k] = 0;

    for (k = 0; k <= (N>>1); ++k) {
        a = 0;
        for (n = 0; n < N; ++n) {
            re[k] += W[a%N] * samples[n];
            a += k;
        }
        if (re[k] < 0)
            re[k] = -(re[k] + 1);
    }
}

int
main(void)
{
    /* setup */
    DDRB |= _BV(LED_BLUE) | _BV(LED_GREEN) | _BV(LED_RED); // set LED pins
    ADCSRA |= _BV(ADPS2) | _BV(ADPS1); // set ADC division factor 64;
    ADCSRA |= _BV(ADEN) | _BV(ADIE); // ADC interrupt enable
    ADMUX = _BV(MUX1); // set PB4 as audio input
    sei(); // enable global interrupts

    ADCSRA |= _BV(ADSC); // Start first signal acquisition

    /* loop */
    while (1) {
        if (acquisition_cnt == N) {
            dft();

            /* LED BLUE - Bass */

```

```

        if (re[0] > BASS_THRESHOLD) {
            PORTB |= _BV(LED_BLUE);
        } else {
            PORTB &= ~_BV(LED_BLUE);
        }

        /* LED GREEN - Midrange */
        if (re[1] > MIDRANGE_THRESHOLD) {
            PORTB |= _BV(LED_GREEN);
        } else {
            PORTB &= ~_BV(LED_GREEN);
        }

        /* LED RED - Treble */
        if (re[2] > TREBLE_THRESHOLD) {
            PORTB |= _BV(LED_RED);
        } else {
            PORTB &= ~_BV(LED_RED);
        }

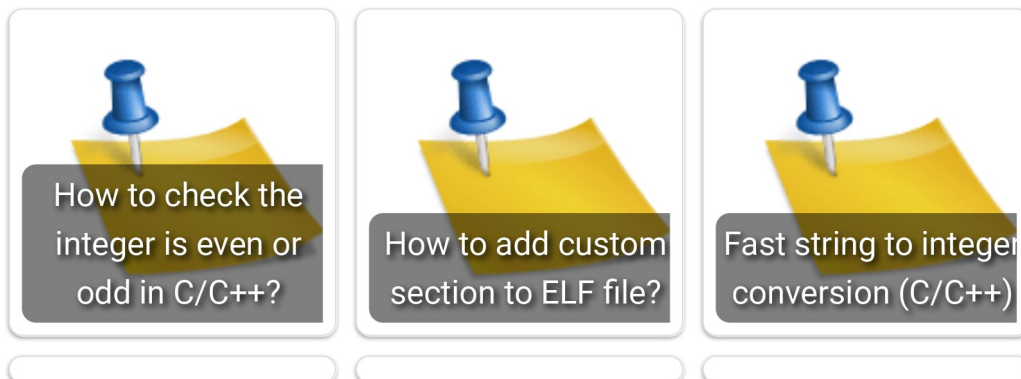
        /* Trigger next signal acquisition */
        acquisition_cnt = 0;
        ADCSRA |= _BV(ADSC);
    }
}
}


```

## References

- <https://blog.podkalicki.com/twiddle-factor-based-dft-for-microcontrollers/>

## Related Articles:





Capturing WiFi  
Beacon Frames,  
Probe Requests and  
Probe...



Packet sniffer in  
Python with Scapy



Markdown to PDF -  
quick howto for linux  
users (Ubuntu)

📁 AVR, DSP, Lab, Project

🔗 ADC, ATtiny13, Audio, AVR, DFT, DSP, FFT, LED, Math, MBAVR, Nyquist Frequency, PWM, Sampling Frequency, Signal, Signal Processing, Twiddle Factors

- ◀ ATtiny13 – tone generator
- ▶ Shiny and high quality PCB from OSH Park

## 13 thoughts on “ATtiny13 – dance lights with DFT”



**Andrey**

2020-11-20 at 12:29 PM | Reply

For example, I tried the following code, but it doesn't work. The LED\_YELLOW led does not go out

```
if (!(PINB & 0b00000111)) PORTB |= _BV(LED_YELLOW);  
else PORTB &= ~_BV(LED_YELLOW);
```



**Andrey**

2020-11-20 at 12:17 PM | Reply

Hello Łukasz!

Wonderful project! Thanks.

Tell me how to add a background channel to the PB3 output?



**Gaurang Shete**

2020-05-23 at 8:44 AM | Reply



can you explain the code brother?

---



**Ana Lia**

2018-11-26 at 10:18 PM | Reply

Hi,

Which microphone was used in this project? I'm using KY-038 sensor to input the audio but so far the LEDs don't react like in the video. Is this microphone sufficient for the project? Would you suggest amplification?

---



**Łukasz Podkalicki**

2018-12-06 at 8:52 AM | Reply

Hi, the project was tested with headphone audio output from my computer but you can use electret microphone and LM358 as amplifier.

/L

---



**Ting**

2017-03-12 at 5:01 AM | Reply

Hi! Just made this Dance Lights with FFT.... in "Stereo Vision" LOVED IT! and placed it beside my LM3916 VU meter project. Thanks for sharing the code and schematics!

---



**jb**

2016-12-11 at 8:10 AM | Reply

Currently you get a kind of PWM linked to your threshold, what do you think about using TCCR0A and TCCR0B and set up a true pwm ? ( Attiny861 has 3 PWM ).

I guess it would be nice with some kind of  $\log(re[n])$  and scaling before setting

PWM ?

<http://embeddedgurus.com/stack-overflow/2008/05/integer-log-functions/>

Have you seen this ?

[https://github.com/Arduinology/ardumower/blob/master/ardumower/fix\\_fft.cpp](https://github.com/Arduinology/ardumower/blob/master/ardumower/fix_fft.cpp)

( It would be nice selecting number of points, when you have more ram 😊 )



**Łukasz Podkalicki**

2016-12-11 at 10:21 AM | Reply

My goal is to create a 100+ projects on ATtiny13. On other chips with more RAM and/or PWM channels, a lot of improvements could be done in easy way. However, this one with "scaling effect" can be implemented using Software PWM, as well.

Regarding current algorithm and calculations for more points. Before I started this project I've produce some materials and scripts in order to make a simulations on local machine. I think it will be a part of other post, soon. But as for now – bellow you can find a python code that calculates Twiddle Factors of declared N what is the only required thing to extend the number of points on this project.

```
import math
N = 6
PI2 = 6.2832
# Calc twiddle factors
W = [math.cos(n*PI2/N) for n in range(N)]
# Convert W to list of integers
I = [int(n * 10) for n in W]
print("Twiddle Factors: {}".format(I))
```



**jb**

2016-12-11 at 10:45 AM | Reply

nice ! thanks !

fix\_fft seems to work with int8\_t, maybe you could save like 15 bytes 😊

My guess is you are close or equal to these with a lot less code  
<http://arduino-cool.blogspot.se/2012/09/arduino-light-organ-ii.html>  
<https://www.youtube.com/watch?v=tCmaOb-VAEo>



**jb**

2016-12-10 at 6:13 PM | Reply

```
>ADCSRA |= _BV(ADPS2)|_BV(ADPS1); // set ADC division factor 64;  
>F_CPU=9600000
```

Are you sampling ADC with 150kHz ? and if so why ?  
( I plan to run it on the default 1MHz )

A fun twist of your code would be to put it inside a flash-light driver  
[http://flashlightwiki.com/AVR\\_Drivers](http://flashlightwiki.com/AVR_Drivers)



**Łukasz Podkalicki**

2016-12-10 at 9:25 PM | Reply

Hi, after I read your comment I realized that I made a copy-paste mistake. Of course, this example will work with higher sampling rate, but it result in poor quality. Project has been tested using @1.2MHz internal clock source. ADC division factor is set to 64 so the ADC sampling frequency is 18750Hz (F\_CPU/64). Thank you!

NOTE: “copy-paste mistake” has been fixed on both blog and github.



**Marco Pfeffer**

2016-12-07 at 9:48 AM | Reply