https://doi.org/10.31891/csit-2022-2-4
UDC 621

Bohdan HUNKO
Lviv Polytechnic National University

HARDWARE AND SOFTWARE SYSTEM OF LIGHT VISUALIZATION OF SOUND SIGNALS

The paper proposes a hardware and software system for light visualization of sound signals. The paper goes through performance requirements, system design process and practical solutions for audio visualization. The paper differs from the present solutions because it not only shows the end results, but also goes through design process, decision-making and performance measurements.

There are many methods that are practically used for audio visualization: amplitude visualization, spectral visualization, frequency visualization etc. One of the most interesting and common methods is spectral visualization of audio signals. This method is based on a mathematical model of obtaining the frequency spectrum of an audio signal using fast Fourier transform (FFT) and subsequent visualization of this spectrum.

The proposed solution is designed with high performance and low latency in mind and shows practical applications of hardware and software-based optimization techniques. Also, the paper describes several visualizations technics and gives an overview on possible visualization improvements.

Key words: hardware and software, performance, audio signal visualization, fast Fourier transform (FFT), analog signal processing.

Богдан ГУНЬКО
Національний університет «Львівська політехніка»

АПАРАТНО-ПРОГРАМНА СИСТЕМА СВІТЛОВОЇ ВІЗУАЛІЗАЦІЇ ЗВУКОВИХ СИГНАЛІВ

Системи опрацювання сигналів є надзвичайно важливою складовою сучасної електроніки. Саме аналогові сигнали є основним джерелом інформації про навколишній світ. Аудіо сигнали як різновид аналогових сигналів є надзвичайно поширеним способом передачі інформації.

Системи обробки та візуалізації аудіо сигналів займають значну частину сфери обробки сигналів. Візуалізація аудіо сигналів допомагає вирішувати багато практичних завдання та дозволяє людям краще усвідомлювати природу та властивості аудіо сигналів. Системи візуалізації аудіо сигналів можна зустріти як в професійних програмах, наприклад програмах для обробки аудіо треків, так і в програмах загального призначення, таких як наприклад аудіо плеєри.

Є багато способів які практично застосовуються для візуалізації аудіо: амплітудна візуалізація, спектральна візуалізація, частотна візуалізація та інші. Одним з найцікавіших та найбільш поширених способів є спектральна візуалізація аудіо сигналів. В основі цього методу лежить математична модель отримання частотного спектру аудіо сигналу за допомогою швидкого перетворення Фур'є (ШПФ) та подальша візуалізація цього спектру.

В цій статті розглядаються практичні методи обробки та візуалізації аудіо сигналів, способи оптимізації програмного забезпечення, критерії оцінювання продуктивності системи та способи досягнення необхідних системних параметрів. У процесі дослідження проведено аналіз існуючих методів обробки та візуалізації аудіо сигналів, визначено оптимальні параметри апаратного забезпечення та компонентів системи а також визначено технічні характеристики розробленої системи.

Результатом проведеної роботи стала програмно апаратна система для світлової візуалізації звукових сигналів. Створена система використовує мікроконтролер як апаратну базу для опрацювання сигналів та LED стрічку для світлової візуалізації спектральної характеристики сигналу.

Ключові слова: апаратне та програмне забезпечення, продуктивність програми, візуалізація звукового сигналу, швидке перетворення Фур'є (ШПФ), обробка аналогового сигналу.

Introduction

Demand on audio signal processing systems is rising each year. Systems that visually represent audio signal are very important part of current digital world because they expand the perception of audio signals which helps humans better understand the signal nature, simplify the process of audio editing and makes the exploration process easier.

There are several popular technics for audio visualization, the most popular being amplitude visualization and spectral visualization. This article goes into details of spectral visualization of audio signals. The project uses a microcontroller unit (MCU) to process audio signal. MCU resources are limited, so a high level of optimization is required to ensure high system performance can be reached. Signal spectrum is calculated using Fast Fourier transform (FFT) [1]. After FFT calculation, spectral characteristic of the signal is visualized on the LED strip.

There are many parameters which impact system performance, most significant being: system architecture, FFT size, audio sampling frequency, complexity of visualization function and number of LEDs in LED strip. It is critical to make the architecture as optimized as possible, so this article describes the investigation on which settings can be used to result in a system which satisfies real time visualization requirements.

Related works

There are several works related to applications of fast Furrier transformation (FFT) on microcontrollers:

- Basics of FFT algorithm on MCU [2] describes the basic ideas and shows the results of FFT algorithm performed on MCU.
- Develop FFT apps on low-power MCUs [3] goes into math behind the FFT and gives a general idea on audio sampling and signal processing.
- Practical FFT on microcontrollers using Common Microcontroller Software Interface Standard (CMSIS) Digital Signal Processing (DSP) [4] talks about usage of DSP library, its benefits and possible problems.

However, described works does not provide solution for light visualization of audio signals, instead they give general idea on FFT algorithm, its usage and expected results. Also, described works does not go into topic of audio sampling, design of visualization system and its characteristic.

Therefore, the purpose of this study is to design the high performance, low latency system which will combine the FFT algorithm with audio sampling to prepare the data which will be visualized on LED strip.

Proposed methodology

There are several approaches that are used to visualize the audio signal, the most popular being amplitude visualization and spectral visualization.

Amplitude visualization isn't the best solution when visualizing audio on LED strips because amplitude of a signal at a given moment of time does not carry much information, for example playing the same song on different volume levels will result in different amplitudes even though the song is the same.

The better approach is to sample audio during some period of time, then apply fast Fourier transformation (FFT) [1] to it, which will convert the signal from amplitude domain into frequency domain, and then visualize frequency domain of the signal.

Microcontrollers are very limited in terms of available system resources, hence choosing the right system parameters and making the design performance and latency optimized is critical.

Following chapters describe what the design of such a system might be, which limitations are present and how to overcome them.

Choosing the hardware

There are several requirements when choosing the microcontroller for this project, most curtail being: fast CPU core to calculate FFT, availability of fast peripheral interphases to output the data to the LED strip, presence of high-performance ADC to sample audio signal. Also, presence of the DMA channels is preferable so audio buffering operations can be offloaded from the CPU core.

PSoC 64 (CY8CKIT-064B0S2-4343W) satisfies all the requirements described above and also provides user-friendly APIs alongside with great documentation, therefore this microcontroller was chosen as hardware platform for this project.

Here is a list of PSoC 64 hardware that is used in this project:

- 100 MHz ARM Cortex-M4 (CM4) core
- 12-bit ADC capable of sampling in speeds up to 2 Giga samples per second
- SPI interphase
- DMA channels

When choosing an LED strip, the choice fell on the WS2812B, as this type of strip has a relatively simple control interface, provides full coverage of RGB spectrum and has a low price compared to other types of LED strips, which makes the WS2812B ideal for this project.

Generally, other types of LED strips can be used, but one thing to note is that it is preferable to have a LED strip with fast response time and rapid data transfer speeds to minimize data transition time.

Overall system architecture

To satisfy high performance and low latency requirements, there are several hardware and software optimizations that need to be done.

Fig. 1 shows the overall system architecture for this project.

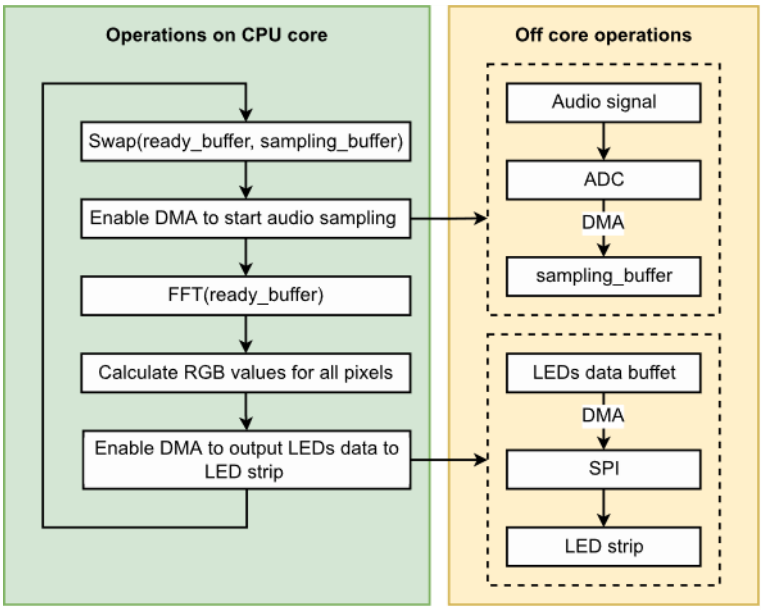

Fig. 1. Overall system architecture

As can be seen from the Fig. 1, a DMA channel is used to put ADC samples into audio buffer and another DMA channel is used to output LEDs data buffer through SPI to the LED strip. Such usage of DMA offloads a lot of work from CPU, which has noticeable positive impact on system performance.

Two buffers with same size are used for audio samples. While one buffer is getting processed by CPU, another buffer is being filled with ADC samples. This means that audio signal is sampled continuously and no data is lost as long as FFT and RGB values calculations are executed faster than new audio buffer is ready (later this article describes which system parameters will satisfy this requirement).

Utilization of DMA channels and usage of two buffers for audio signal effectively divides the system into 3 truly parallel tasks:

- FFT and RGB values calculation
- Audio sampling
- outputting LEDs data buffer to the LED strip

Fig. 2 shows execution timeline for these tasks and interactions between them.

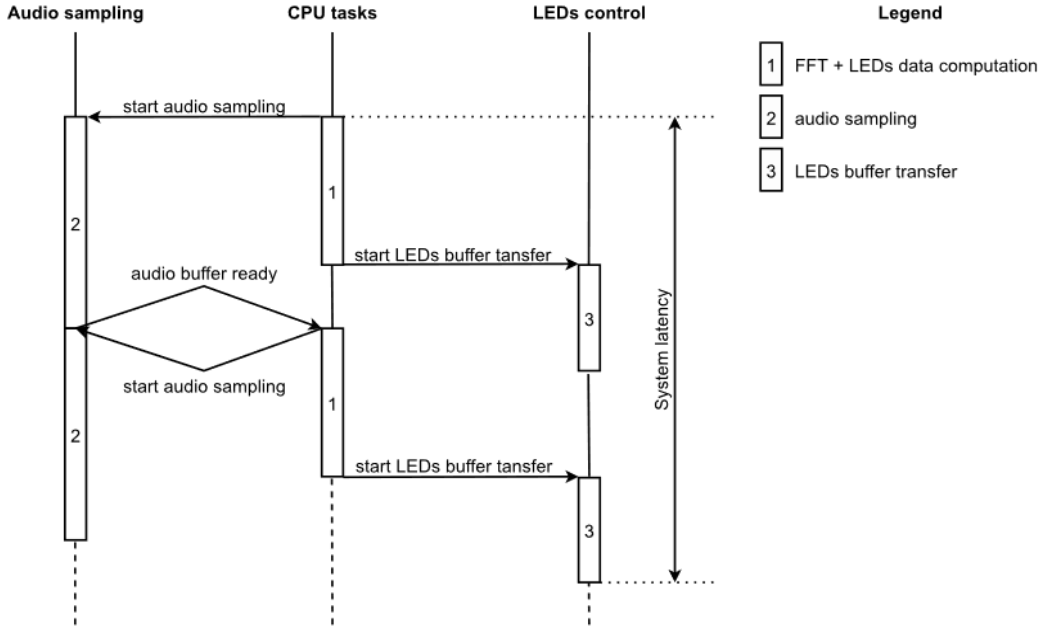

Fig. 2. Execution timeline for system tasks

Key system requirements can be formulated from Fig. 2:

- Requirement 1: Task 1 (FFT + LEDs data computation) must execute faster than Task 2 (Audio sampling). This requirement is needed to ensure that audio is sampled continuously without delays or wait time.

- Requirement 2: Task 3 (LEDs buffer transfer) must execute faster than Task 2 (Audio sampling). Again, this is needed to ensure that there are no delays in the system and audio is sampled continuously.
- Requirement 3: System latency should not exceed 45ms. Audio visualization can be treated as video signal, because visual image corresponds to audio (same as in video track), therefore video industry standards can be used to determine acceptable latency. Video industry standards [5] specify that audio should lead video by no more than 15 milliseconds and audio should lag video by no more than 45 milliseconds. Therefore, 45ms delay is used as largest acceptable latency.

Also, it is worth mentioning that semaphores are used to synchronize the tasks and ensure that the system behaves as expected.

System parameters optimization

As per Nyquist–Shannon sampling theorem [6] to capture full signal spectrum sampling rate must be at least twice the occupied bandwidth of the signal. From [7] it is known that humans can detect sounds in a frequency range from about 20 Hz to 20 kHz, so audio should be sampled at least at 40kHz frequency.

ISO/IEC 13818-3:1998 [8] define a set of frequencies for audio sampling: 44.1kHz, 48 kHz, 88.2 kHz, 96 kHz and 192 kHz. It is always a good idea to follow the industry standards, so this set of frequencies will be used in further calculations.

PSoC 64 ADC unit is capable of up to 2 GHz sampling rate in up to 12-bit resolution, so sampling audio fast enough should not be the problem.

Industry standards [9] shows that audio signal amplitude typically ranges between -1.228 V and +1.228 V. This amplitude range is not absolute because there is professional rated equipment (like studio microphones and audio systems) which is capable of outputting significantly higher voltage levels. Also, this range is maximum values at maximum volume, and if volume is lover the voltage will also be lower.

So, there is a tradeoff when configuring ADC:

- if ADC voltage boundaries are set high (e.g. from -3 V to +3 V) to account for professional equipment then discretization step will be quite big which will make ADC reading for similar amplitudes the same. So low or medium volume sound signal produced by consumer's electronic will be very limited in terms of values range which will result in poor samples quality.
- if ADC voltage boundaries are set low (e.g. from -1.2 V to +1.2 V) to only account for consumers rated electronics then discretization step will be smaller and low or medium volume sound will result in better samples quality. But in this case audio signal produced by professional rated equipment will produce voltage levels that are outside the range which will lead to poor samples quality.

This project is meant to be used with standard phone or computer (which is consumers rated electronic) therefore ADC is configured to be able to sample voltage range from -1.2 V to +1.2 V.

Having highly optimized implementation of FFT library is critical to ensure high system performance and low latency requirements are satisfied.

ARM provides FFT algorithm implementation as part of Common Microcontroller Software Interface Standard (CMSIS) Digital Signal Processing (DSP) library [10]. This library is highly optimized for Cortex-M processors and utilizes vector instructions to speed up FFT calculation, therefore this library perfectly suites the needs of this project, so it will be used to calculate the FFT.

The following input buffer sizes are supported by DSP FFT library: 32, 62, 128, 256, 512, 1024, 2048 and 4096 bytes.

FFT duration measurements are needed to choose the right buffer size.

Table 1

Performance measurements of FFT algorithm for each input buffer size

FFT size	**32**	**64**	**128**	**256**	**512**	**1024**	**2048**	**4096**
FFT duration [us]	44	89	164	351	731	1414	3061	6462

As previous mentioned PSoC 64 operates on 100MHz CM4 core therefore 1 us is 100 processor cycles. It should be noted that different CPU core might result in different cycles count as it may have different instructions set which may result in more/less optimized code, but I don’t believe that there will be a huge difference in cycles needed to calculate the FFT algorithm.

Table 1 will later be used to choose system parameters.

There are many ways to visualize the FFT, but a lot of them require 2-dimensional space for visualization. LED strip on the other hand is 1-dimensional. So graphing FFT like normal equalizers do will not work.

To solve this problem, this project uses LEDs color value to visualize the FFT. LED has 3 colors: red, green and blue, therefore FFT spectrum is divided into 3 intervals: low, medium and high frequency intervals. Each color then is assigned to frequency spectrum: red – low frequencies, green – medium frequencies and blue – high

frequencies. FFT values in each spectrum are averaged and mapped to value from 0 to 255, then these RGB values are shown on the LED strip.

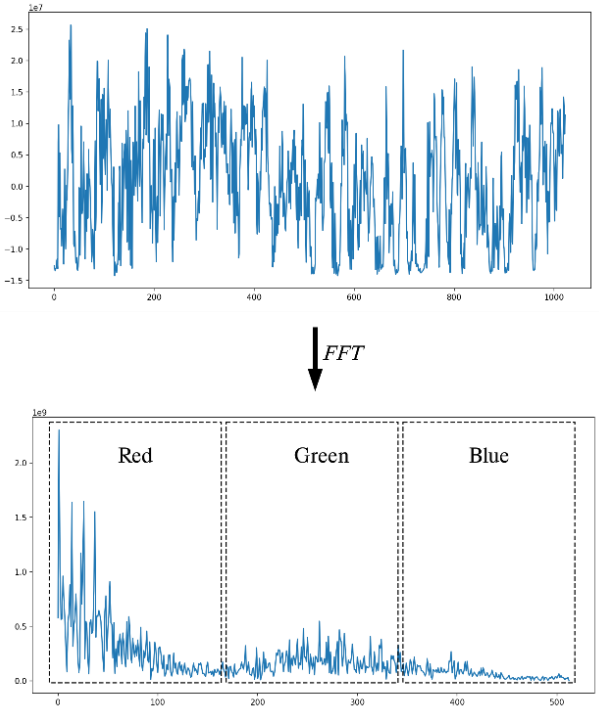

Fig. 3. FFT transformation results and their mapping to LEDs color

Visualization function utilizes functions from ARM DPS library to find the mean value of each of the intervals, this function uses vector instructions so overall visualization function does not take much time to execute.

Table 2

Performance measurements of visualization function

FFT size [bytes]	**32**	**64**	**128**	**256**	**512**	**1024**	**2048**	**4096**
FFT output size [bytes]	16	32	64	128	256	512	1024	2048
Visualization time [us]	17	18	21	27	38	61	107	199

Table 2 shows visualization function execution time for all possible FFT buffer sizes. Should be noted that an input buffer of size N produces FFT of size N/2.

Table 2 will later use these results to choose system parameters.

As mentioned previously, WS2812 is chosen as LED strip for this project, so this chapter describes the communication protocol of this LED strip.

WS2812 data sheet [11] shows that WS2812B LED strips are an almost arbitrary length string of pixels that can be cascaded together via a serial line.

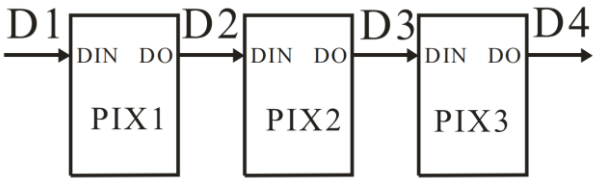

Fig. 4. LEDs cascading mechanism for WS2812 LED strip

In WS2812 each LED has 3 individually controlled diodes: red, green and blue. Each diode has 8-bit color range (0 - 255) which in total gives 3 * 8 = 24 bits (3 Bytes) per LED.

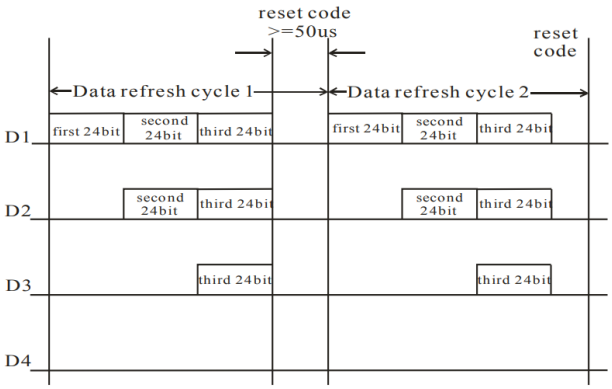

Fig. 5. Data transmission sequence for WS2812 LED strip

Fig. 5 shows that when communication starts, a pixel takes its Red, Green and Blue values from the data stream, then passes on the rest of the bytes to the next pixels.

G7	G6	G5	G4	G3	G2	G1	G0	R7	R6	R5	R4	R3	R2	R1	R0	B7	B6	B5	B4	B3	B2	B1	B0

Fig. 6. Composition of 24 bits of data for WS2812 LED strip

As can be seen from Fig. 6, WS2812 expects color codes in order of Green Reg Blue (GRB) which is different from RGB which everybody is used to.

Color codes sequence isn't the only weird thing in this communication protocol. WS2812 data sheet documents that a “bit 1” is actually encoded as a long pulse of 1 followed by a short pulse of 0. And a “bit 0” is a short pulse of 1 followed by a long pulse of 0. Fig. 7 shows data encoding for WS2812 LED strip and Table 3 shows exact timing requirements for these codes.

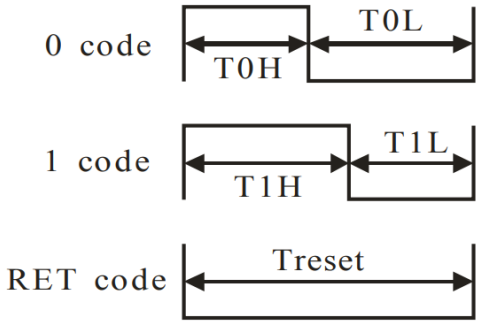

Fig. 7. Data encoding for WS2812 LED strip

Table 3

WS2812 data codes timing requirements

Name	Description	Required time [us]	Deviation [ns]
T0H	0 code, high voltage time	0.4	±150
T1H	1 code, high voltage time	0.8	±150
T0L	0 code, low voltage time	0.85	±150
T1L	1 code, low voltage time	0.45	±150
RES	low voltage time	Above 50	

As can be seen from the Table 3 T1H is double a T0H and a T0L is almost exactly double a T0H. That means that the fundamental unit of time in this system is 0.4uS. Therefore, WS2812 code for "1" can be encoded as 110 and code fore "0" can be encoded as 100.

So, there are 3 colors in one LED, each color is 8-bit value and each bit from that value is encoded as 3 bits in WS2812 protocol therefore 3*8*3 = 72 bits are needed to represent one WS2812 LED.

It is really hard to satisfy timing requirements from Table 3 when using CPU and delays to set the pin value to drive the LEDs. A much better way is to use serial peripheral interphase to transmit the data.

As described previously, bit transmission time for WS2812 is 0.4us, which means that transmission rate is 1/0.4uS = 2.5 Mbps. There are 3 types of peripheral interphases in PSoC: UART, I2C and SPI. Only SPI and I2C are capable of speeds that high. SPI does not require any pull-up resistors and also it can be easily used in custom data transmission rates, so in this project it is used as interphase to transfer the data.

Table 3 also shows that to update the LEDs, the data line must be pulled low for 50 us. SPI data line stays low while transfer is not in progress, so the only necessary thing is to ensure that there is at least 50 us pause between two consecutive transmissions.

Now let's calculate how long does it take to transfer the data for 1 LED. Formula (1) can be used to calculate data transmission time for 1 LED.

$$t_{1\_led} = \frac{bits\_per\_led}{transmission\ rate} = \frac{72}{2.5*10^6} = 28.8\ us \quad (1)$$

where *bits_per_led* is the number of bits per one WS2812 LED, *transmission_rate* is the transmission rate of SPI interphase.

From the Formula (1) it can be seen that it takes 28.8 us to transfer data for one LED, so let's see how much LEDs can be used and which number of Frames Per Second (FPS) can be achieved.

$$t_{n\_leds} = t_{1\_led} *N+50us = 28.8us *N+50us \quad (2)$$

where t_{1_led} is the data transmission time for 1 LED, N is the number of LEDs

Note that extra 50us in Formula (2) are the LEDs update window.

And Formula (3) can be used to calculate FPS.

$$FPS = \frac{1}{t_{n\ leds}} \quad (3)$$

where t_{n_leds} is the data transmission time for N LEDs.

From [12] it is known that human eye tolerates minimum of 24 FPS, but 30 FPS is more preferable target. Using Formulas (2) and (3) it can be calculated that at most 1150 LEDs can be used to produce FPS grater or equal to 30 FPS. WS2812 LEDs density is from 30 LEDs per meter up to 144 LEDs per meter, therefore 1150 is at least 7 meters (with 144 LEDs per meter) up to 38 meters (with 30 LEDs per meter).

This project uses 200 LEDs long WS2812 with 30 LEDs per meter, therefore using Formula (2) transmission time for 200 LEDs is:

$$t_{200\_leds} = 28.8us *200\ 50us = 5810us \quad (4)$$

t_{200_leds} will later be used to choose system parameters.

Let's put all system pieces together and determine which settings can be used to result in a high performance, low latency system.

System parameters should be chosen according to system requirements, so let's go through each system requirement and determine which parameters can be used to satisfy it.

Task 3 (LEDs buffer transfer) execution time is equal to transmission time for 200 LEDs. From Formula (4) it is known that transmission time for 200 LEDs (t_{200_leds}) is 5810us.

Task 1 (FFT + LEDs data computation) execution time is equal to sum of execution times of FFT calculation and visualization function duration. Combining the data from Table 1 and Table 2 gives the following results:

Table 4

Task 1 execution time

FFT size	**32**	**64**	**128**	**256**	**512**	**1024**	**2048**	**4096**
FFT duration [us]	44	89	164	351	731	1414	3061	6462
Visualization time [us]	17	18	21	27	38	61	107	199
Total [us]	61	107	185	378	769	1475	3168	6661

Task 2 (Audio sampling) execution time depends on sampling frequency and FFT buffer size. Formula (5) can be used to calculate Task 2 execution time.

$$t_{task\_2} = \frac{buffer\_size}{sampling\_frequency} \quad (5)$$

where *buffer_size* is the size of the audio buffer, *sampling_frequency* is the frequency used to sample audio.

Requirement 2 states that Task 3 (LEDs buffer transfer) must execute faster than Task 2 (Audio sampling). From Formula (4) it is known that Task 3 execution time is equal to t_{200_leds} (5810us), therefore Task 2 should take longer than 5810us to execute.

Requirement 3 states that system latency should be as low as possible and be less than 45ms. Fig. 2 shows that system latency is the sum of execution times for all three system tasks. Task 3 takes 5810us therefore to ensure that

the system latency is less than 45ms total execution time for Task 1 and Task 2 should be less than 45000-5810=39190us.

Knowing the argumentation for Requirements 2 and 3 heatmap for Task 2 execution time can be created.

		FFT size [bytes]							
		32	64	128	256	512	1024	2048	4096
Sampling frequency [kHz]	44.1	726	1451	2902	5805	11610	23220	46440	92880
	48	667	1333	2667	5333	10667	21333	42667	85333
	88.2	363	726	1451	2902	5805	11610	23220	46440
	96	333	667	1333	2667	5333	10667	21333	42667
	192	167	333	667	1333	2667	5333	10667	21333

Fig. 8. Task 2 execution time heatmap

From Fig. 8 can be seen that configurations that do not satisfy Requirement 2 are marked with red ( ) color and configurations that do not satisfy Requirement 3 are marked with orange ( ) color. Only the configurations that satisfy both Requirement 2 and Requirement 3 are marked with green ( ) color.

Also, from the Fig. 8 and Table 4 it can be seen that that regardless of FFT size and sampling frequency, Task 1 (FFT + LEDs data computation) always executes faster than Task 2 (Audio sampling). Therefore, system Requirement 1 (Task 1 (FFT + LEDs data computation) must execute faster than Task 2 (Audio sampling)) is always satisfied, regardless of settings.

So now it can be seen that all green values from Fig. 8 can be used to result in a system which will satisfy all system requirements. It is better to have audio sampling take a bit longer because then it will capture audio signal over a bigger period of time which will result in FFT which will better represent the signal.

Code and results

Considering everything said above, following system settings were chosen:

- LEDs count: 200
- FFT size 1024
- Audio sampling frequency 44.1 kHz

In this configuration, the system satisfies all the requirements and has a latency of: 5810us+1414us+23220us =30444us.

Overall there is some space for experiments with system parameters because as can be seen from Fig. 8 there are quite a few valid settings that can be used. If smaller latency is needed then smaller FFT buffer size can be chosen and alternatively if higher audio resolution is needed then higher sampling rate can be used.

Code for this project is licensed under GPL-3.0 license [13] and can be found in GitHub repository [14].

It is impossible to show the real time visualization in static paper, so I have prepared a video clip [15] and uploaded it to YouTube.

Conclusions

During this study, methods for light visualization of audio signals were developed. Ways of visualizing audio were investigated and described. System architecture for the project was developed and key system requirements were formulated. Using these requirements investigation on possible system parameters was done, this investigation has shown that there is a wide range of audio sampling frequencies, buffer sizes and LEDs quantities that can be used.

Overall, this article shows practical ways of using fast Fourier transformation on microcontrollers to obtain signal spectrum and methods that can be used for visualization of this spectrum.

There are a number of things to explore/improve for example add verity of visualization functions, add Bluetooth functionality and create a mobile app to allow the user to cycle through visualization modes, explore different hardware, visualize the audio on 2D array matrix of LEDs, etc.

References

1. Fast Fourier transformation: Wikipedia article. URL: https://en.wikipedia.org/wiki/Fast_Fourier_transform. (Accessed May 15, 2022)

2. DiCola T. Fun with Fourier Transforms: online article. URL: https://cdn-learn.adafruit.com/downloads/pdf/fft-fun-with-fourier-transforms.pdf. (Accessed May 15, 2022)

3. Develop FFT apps on low-power MCUs: online article. URL: https://www.embedded.com/develop-fft-apps-on-low-power-mcus/. (Accessed May 15, 2022)

4. Practical FFT on microcontrollers using CMSIS DSP: online article. URL: https://m0agx.eu/2018/05/23/practical-fft-on-microcontrollers-using-cmsis-dsp/. (Accessed May 15, 2022)

5. Audio-to-video synchronization: Wikipedia article. URL: https://en.wikipedia.org/wiki/Audio-to-video_synchronization. (Accessed May 15, 2022)

6. Nyquist-Shannon sampling theorem: Wikipedia article. URL: https://en.wikipedia.org/wiki/Nyquist%E2%80%93Shannon_sampling_theorem. (Accessed May 15, 2022)

7. Hearing range: Wikipedia article. URL: https://en.wikipedia.org/wiki/Hearing_range. (Accessed May 15, 2022)

8. ISO/IEC 13818-3:1998(en) Information technology: Generic coding of moving pictures and associated audio information Part 3: Audio: ISO standard. URL: https://www.iso.org/obp/ui/fr/#iso:std:iso-iec:13818:-3:ed-2:v1:en. (Accessed May 15, 2022)

9. Line level: Wikipedia article. URL: https://en.wikipedia.org/wiki/Line_level. (Accessed May 15, 2022)

10. Digital Signal Processing using Arm Cortex-M based Microcontrollers: ARM online documentation. URL: https://www.arm.com/-/media/global/resources/education/textbooks/dsp-sample-chapter.pdf?revision=0a9768b9-0a7a-42fe-aba9-6304f240275b&la=en. (Accessed May 15, 2022)

11. WS2812 Datasheet: online device datasheet. URL: https://pdf1.alldatasheet.com/datasheet-pdf/view/553088/ETC2/WS2812.html. (Accessed May 15, 2022)

12. Bakaus P. The Illusion of Motion: online article. URL: https://paulbakaus.com/tutorials/performance/the-illusion-of-motion/. (Accessed May 15, 2022)

13. GNU General Public License: online copy of license. URL: https://www.gnu.org/licenses/gpl-3.0.en.html (Accessed May 15, 2022)

14. Hunko B. GitHub repository with code for light visualization of audio signals: GitHub repository. URL: https://github.com/hunkob/music_synched_LED_PSoC (Accessed May 15, 2022)

15. Hunko B. Visualization results: YouTube video. URL: https://youtu.be/rnnqywi4_kU. (Accessed May 15, 2022)