FFT LIBRARIES PERFORMANCE BENCHMARKING ON COMMONLY AVAILABLE MICROCONTROLLERS

Alexandru-Viorel Pălăcean [1], Dumitru-Cristian Trancă [2], Victor-Valentin Stoica [3], Răzvan-Victor Rughiniș [4], Daniel Rosner [5]

The Fast Fourier Transform (FFT) is a widely used method for signal analysis in smart devices, such as wearables, power meters, or vibration monitoring systems, providing information about signal components, fundamental frequency, and harmonics amplitudes. With the emergence of concepts like IoT, Smart City or Smart Metering, the rapid development of new smart devices has accelerated, necessitating the constant optimization of signal analysis methods such as FFT. In this paper, we perform a benchmark of the most popular FFT libraries available for commonly used microcontrollers (MCUs). These microcontrollers can be integrated into various IoT devices, being responsible for the acquisition and processing of signals from various sensors or transducers. The FFT analysis applied to these signals provides relevant information about the signal components. We used multiple FFT libraries on a wide range of microcontrollers, analyzing two sets of electrical signal samples taken from a power meter. Based on the obtained results, we classified the libraries according to several performance criteria.

Keywords: FFT, THD, IoT, ESP32, Arduino, RP2040, Matlab

1. Introduction

The recent technological advancements have led to the development of new technologies aimed at improving the quality of services and life. Internet of Things (IoT), Smart City, or Industry 4.0 are part of this evolutionary trend, relying on a series of interconnected smart devices capable of providing information about the parameters of their operating environment. The processing of the information gathered from the various sensors connected to these

[1]Teaching Assistant, National University of Science and Technology POLITEHNICA Bucharest, Romania, e-mail: alexandru.palacean@upb.ro

[2]Assistant Professor, National University of Science and Technology POLITEHNICA Bucharest, Romania, (Corresponding author), e-mail: dumitru.tranca@upb.ro

[3]PhD Student, National University of Science and Technology POLITEHNICA Bucharest, Romania, e-mail: victor.stoica0203@stud.trans.upb.ro

[4]Professor, National University of Science and Technology POLITEHNICA Bucharest, Romania, e-mail: razvan.rughinis@upb.ro

[5]Associate Professor, National University of Science and Technology POLITEHNICA Bucharest, Romania, e-mail: daniel.rosner@upb.ro

devices is carried out either in the cloud or locally, at the network edge. In the latter case, the processing power of the devices must match the complexity of the conducted operations. One of the most widely used techniques for signal analysis, which provides information on their frequency, components, and corresponding amplitudes, is the Fast Fourier Transform (FFT).

Multiple implementations of this technique have been developed for various embedded system architectures, reflecting considerable research interest in integrating and optimizing them within smart systems used in both industrial and domestic environments. A. Buzachis et al. [1] proposed an IoT as a Service (IoTaaS) smart metering solution for electrical grid signals harmonic analysis using the FFT algorithm, running on a Raspberry Pi 3 computer. The authors aimed to characterize the network's non-linear loads in order to prevent possible malfunctions. In [2] Qi-Lin Mao and Ming-Yue Zhai presented a novel natural timestamp generation technique based on the grid frequency estimation, using the fractional cepstrum domain transformation model.

The challenge of optimizing signal analysis algorithms for energy constrained devices has been approached by B. Mazzoni et al., who proposed in [3] a parallel architecture for the Short Time Fourier Transform (STFT) and Discrete Wavelet Transform (DWT) methods for ultra-low-power IoT devices. In [4] the authors present a low-complexity Power Spectral Analysis (PSA) algorithm based on the Fast Gaussian Gridding (FGG), designed for common wearable devices equipped with low power processors. The results indicate a 92.99% performance increase compared to traditional methods and a 91.7% energy consumption reduction, without affecting the analysis accuracy. G. Fabregat et al. [5] developed an optimized Direction-Of-Arrival (DOA) algorithm that runs on ESP32 microcontroller. This low cost system uses largely available, off the shelf microphones and performs real-time localizations in under 3.3ms.

The amount of memory utilized by signal analysis algorithms is often substantial. Tsung-Ying Sun and Yu-Hsiang Yu [6] proposed a method to optimize the memory usage of the Discrete Fourier Transform (DFT) by optimizing the cache and reducing the size of the twiddle factors by almost 50%. J. Mauzet et al. [7] proposed a Dynamic Partial Reconfiguration (DPR) system for radar applications. A Discrete Fourier Transform (DFT) sum or a Fast Fourier Transform can be selected for the analysis. The authors compared the two methods, highlighted their advantages and proposed a self-adaptive decision method.

In [8] Afzeri et al. developed an online control and monitoring solution for industrial induction motors. The system monitors the temperature and vibration using dedicated sensors and uses the FFT algorithm to analyze the acquired signals parameters. Another solution for monitoring the vibrations of mechanical elements has been proposed by N. Nistor et al. in [9]. A FFT

algorithm was used on an ATmega microcontroller to analyze the signal acquired from a broadband acoustic sensor. D. Pérez [10] outlined a solution based on the FFT transform and an Arduino development board for a power meter. The device was designed to measure the power quality of the residential voltage and to calculate the Total Harmonic Distortions (THD), including up to the 25th harmonic in the analysis. The results are plotted through the serial interface at a 2M baud rate.

Several benchmarks have been conducted on various implementations of the FFT on multiple hardware platforms, indicating a strong interest in this type of analysis. B.K. Vinay et al. [11] performed a performance analysis of FFT algorithms running on ARM and DSP cores. In [12] A. Ayala et al. performed parallel multi-dimensional FFT performance benchmarks on the Exascale Computing Project (ECP) of the United States. P. Steinbach et al. [13] presented *gearshifft*, an open-source FFT benchmarking system that integrates various FFT implementations that can be run on different hardware platforms.

As can be observed, the FFT transform is frequently used in various low-power applications where energy consumption reduction and processing optimizations are essential. In this paper, we aimed to determine the performance of the most commonly used FFT libraries for commercially available microcontrollers, our contributions being as follows:

- A comprehensive benchmark, evaluating the execution performance of FFT libraries available for commonly used microcontrollers and comparing the results with those obtained using dedicated Matlab functions,
- The classification of microcontrollers based on the execution performance of the FFT analysis, relevant information for selecting a microcontroller for a given application,
- A comparative presentation of the results in the form of detailed and easily interpretable tables and charts.

2. Background

Among our areas of research interest is the field of smart metering, represented by intelligent IoT devices capable of measuring various electrical parameters of the grid. In another paper, currently under review, we presented a power meter capable of measuring both standard and specialized electrical parameters, such as the Total Harmonic Distortion (THD) factors of the voltage and current grid signals. During the development stage of the device we had to choose a microcontroller responsible for the signal samples acquisition, processing, measured values calculation and data transmission through multiple IoT protocols.

Because we wanted to compute the THD factors of the signals, the microcontroller also had the task of performing the FFT analysis of the samples. Both the voltage and current signals needed to be sampled over a period of

one second, with the samples saved in the microcontroller's memory and subsequently processed. Therefore, we had to choose a library capable of performing the analysis in the shortest possible time, preferably under one second, concurrently with the acquisition process of the next set of samples, ensuring that we do not lose signal components due to processing delays.

Since we aimed to include the first 40 harmonics of the signal in the THD factor calculation and because we set a sampling frequency of 8 kSPS (kilo samples per second), we needed to identify a library that executes the FFT analysis in the shortest possible time, with high accuracy in estimating the fundamental frequency and the amplitudes of the signal harmonics, while consuming minimal microcontroller memory. Since the execution speed of the algorithm depends on the processing performance of the microcontroller, it was necessary to test multiple largely available models to identify the one that meets the imposed requirements.

Before testing and selecting the FFT libraries, we needed to sample the input signals. To accomplish this task, we used our power meter's microcontroller to acquire the samples from an ADC (Analog to Digital Converter) module and then to save the samples' values in a text file on a microSD card. Based on these data, we can run the FFT algorithms and determine the signal frequency and harmonic components amplitude, including the fundamental. To verify the amplitude estimation accuracy of the libraries, we compare the results with the RMS (Root Mean Square) value of the signal. Figure 1 presents the sampling process and the components of a sinusoidal signal, which can be calculated using (1).

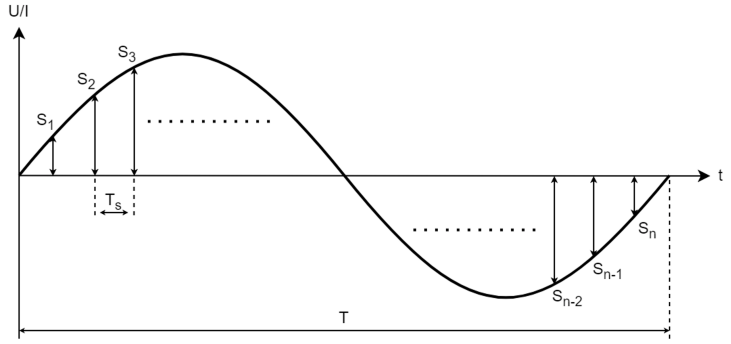


FIGURE 1. Voltage/current signal sampling

$$\begin{aligned} RMS_{value}[V/A] &= \sqrt{\frac{\sum_{i=1}^{n} S_i^2}{n}} \\ f_s[Hz] &= \frac{1}{T_s} \\ f[Hz] &= \frac{1}{T} \end{aligned} \tag{1}$$

where:

S_i represents a signal sample,
n is the total number of samples,
RMS_{value} is the resulting RMS value of the signal,
T_s represents the period of time between two samples,
f_s is the sampling frequency,
T is the signal period,
f represents the signal frequency.

To calculate the THD factor of a signal, first the signal harmonic spectrum must be determined and the harmonics amplitude extracted using a FFT transform. The resolution of the THD analysis is determined by the maximum number of harmonics examined (e.g., up to the 15th or 31st harmonic). Figure 2 presents the result of the FFT analysis applied on a 50Hz sinusoidal signal with harmonic components and superimposed with white noise. The largest peak corresponds to the fundamental component of the signal, with a frequency of 50 Hz. However, significant peaks can also be observed for the harmonic components at frequencies that are multiples of the fundamental frequency. The THD value is determined using (2).

$$THD[\%] = \frac{\sqrt{\sum_{i=2}^{n} H_{iRMS}^2}}{H_{fRMS}} \times 100\% \tag{2}$$

where:

n represents the maximum number of analyzed harmonics,
H_{fRMS} represents the fundamental harmonic RMS amplitude,
H_{iRMS} represents other harmonic components RMS amplitude.

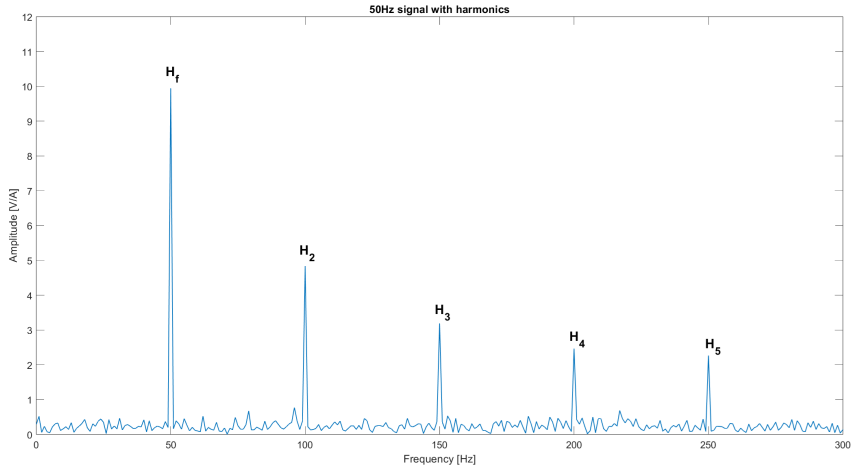


FIGURE 2. FFT on 50Hz signal with harmonics

The following section outlines the process of selecting the tested libraries and microcontrollers, and describes how the testing procedures were conducted and the criteria that were followed.

3. Benchmarking process description

The electrical power quality measurement device that we designed, mentioned in the previous section, needed to be equipped with both wired (RS485) and wireless (WiFi) communication interfaces, to allow the integration of a microSD card slot for data logging, and to be provided with as many GPIO pins as possible to enable the connection of various peripheral devices. Moreover, it needed to be available on the market in large quantities and to have a minimal cost. Following the analysis of available models from various manufacturers, we selected the Raspberry Pi RP2040 microcontroller and several models of ESP32 microcontrollers from Espressif Systems. Thus, we utilized a set of development boards equipped with the selected microcontrollers and tested several FFT libraries to evaluate their performance. The boards are presented in Figure 3 and their specifications in Table 1:

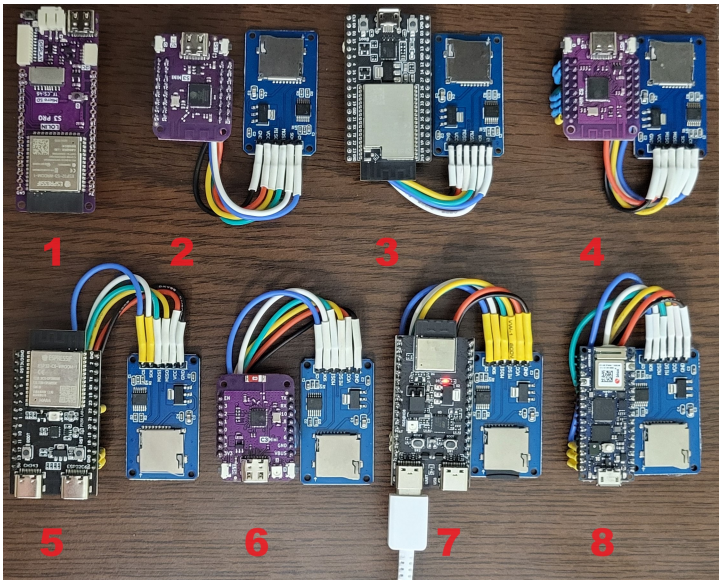


FIGURE 3. Boards used for benchmarking

(1) Wemos S3 Pro [14], (2) Wemos S3 Mini [15], (3) Espressif ESP32-DevKitC-VE [16], (4) Wemos S2 Mini [17], (5) ESP32-C6-DevKitC-1 [18], (6) Wemos C3 Mini [19], (7) ESP32-H2-DevKitM-1 [20], (8) Arduino Nano RP2040 Connect [21]

TABLE 1

Selected microcontrollers specifications

Board	MCU	CPU		Memory			
		Type	Frequency [MHz]	ROM [KB]	SRAM [KB]	PSRAM [MB]	FLASH [MB]
1	ESP32-S3-WROOM-1	Dual Core	240	384	512	8	16
2	ESP32-S3FH4R2	Dual Core	240	384	512	2	4
3	ESP32-WROVER-E	Dual Core	240	448	520	8	8
4	ESP32-S2FN4R2	Single Core	240	128	320	2	4
5	ESP32-C6-WROOM-1	Single Core	160	320	512	N/A	8
6	ESP32-C3FH4	Single Core	160	384	400	N/A	4
7	ESP32-H2-MINI-1	Single Core	96	128	320	N/A	4
8	Raspberry Pi RP2040	Dual Core	133	18	264	N/A	16

Our goal was to test multiple FFT libraries on all these microcontrollers in order to identify the one that performs this type of analysis fastest. To provide the input data for the algorithm, in our case the voltage and current signal samples from the grid, we connected microSD memory card modules to the development boards, except for the first one, which already included this interface. The modules are connected to one of the Serial Peripheral Interface (SPI) of the microcontrollers.

The microcontrollers were programmed using the Microsoft Visual Studio Code and PlatformIO IDE. To select the libraries used in the test, we analyzed all popular FFT libraries available for these microcontrollers within the PlatformIO's libraries manager. Among these, we highlight: *fix_fft* [22], *KickFFT* [23], *FFT_C* [24], *Adafruit Zero FFT Library* [25], *OsakanaFft* [26], *DokuFFTPACK* [27].

We examined the types of input and output data variables they utilize, as well as their potential limitations regarding the maximum number of samples analyzed. Most of the libraries highlighted in their documentation as very fast, utilize lookup tables and variables with reduced sizes (8 or 16 bits). Additionally, from our tests, many of these libraries can analyze a maximum of 512 samples, being primarily developed for audio signal analysis. The *fix_fft* library integrates functions for both direct and inverse FFT analysis and utilizes int8 data type buffers to store the sampled signal values in order to enhance processing speed. *KickFFT* accepts any data type for the samples buffer but has a limit of maximum 512 samples that can be analyzed. Similar constraints are present for the *FFC_C* library, which uses int buffers but limits the maximum number of samples to 512. The *Adafruit Zero FFT Library* was designed to run on ARM cortex M0 CPUs, accepts multiple data types for the input and output buffers, but limits the maximum numbers of signal samples to 4096.

Other libraries, such as *Audio* [28], *ESP32Sampler* [29], *Seeed Arduino Mic* [30], *FrequencyDetector* [31] or *AudioTuner* [32] have been developed particularly for audio applications, specifically for processing signals with a limited number of samples, acquired from microphones with analog or digital interfaces.

In our case, the datasets contain at least 8000 samples, each represented using 24 bits. Therefore, it was necessary to select the libraries that work with int, float, double, or complex type variables, and that do not have limitations regarding the total number of samples. Following the analysis, we identified another three libraries that meet the specified criteria: *ESP32 FFT* [33], *ArduinoFFT* [34], and *Fast4ier* [35]. The first library uses float type variables, the second one double type variables, and the third complex type variables. None of these libraries have a limitation related to the maximum number of samples, only requiring the number to be a power of 2. Therefore, during the tests, we chose the number of samples in a dataset to be 8192 (2^{13}). Each library was integrated into our benchmarking algorithm and uploaded to each tested development board.

In the initialization step of the program, we allocate memory for the buffers used to store the signal samples and the buffers used by the FFT libraries. We initialize the SD card module and an UART (Universal Asynchronous Receiver-Transmitter) interface which will be used to display the tests results. Then, a set of 8192 voltage and current signal samples are read from a text file residing on the microSD card and stored in the buffers. In the next stage, the integrated FFT analysis function from the used library is called for execution. Once the results are available, a function that calculates the THD factors is called. Finally, the results are printed using the serial communication interface.

Before and after the execution of the FFT and THD functions, we record the current timestamp returned by the *millis()* function in order to calculate the total time required for each operation. Also, we recorded the available microcontroller's memory before and after the libraries buffers memory allocations using the *esp_get_free_heap_size()* function to determine the total memory usage for each tested library. We applied the same steps for each set of signal samples, with a total of 10 sets being analyzed during the tests. The operations total time and the libraries memory usage information is also transmitted through the serial interface. *RealTerm* was used as a terminal program due to its message capturing function and ability to save them in a text file. This functionality allowed us to process and compare the results more easily.

The function that calculates the THD factor uses the results of the FFT analysis, identifying the largest peaks among the signal harmonic amplitudes, located at frequencies that are multiples of the fundamental frequency. Both voltage and current THD factors are calculated.

As a reference in our comparisons, we used the results from the Matlab *FFT* and *THD* functions, which are dedicated to these types of analyses. The program implemented in Matlab applies the same steps as the microcontroller algorithm, described earlier. The signals samples are read from a copy of the text file available on the microSD card used for the microcontrollers, the file being stored on the computer used to run the program. The execution time of

the *FFT* and *THD* procedures is measured using the *tic* and *toc* functions. We also implemented the same THD function as in the microcontroller algorithm, to compare the THD results generated using the FFT libraries output with the THD value based on the Matlab's *FFT* function. It also provided us with the capability to generate comparative graphs of the FFT analysis results executed by the different libraries.

4. **Results and discussion**

The results obtained by running each FFT library on each of the previously mentioned microcontrollers were compared with one another, as well as with the results generated by the algorithm developed in Matlab. Table 2 presents the output of the test algorithm run on the Raspberry Pi RP2040 microcontroller, using the *ESP32 FFT* library. Each line represents the results for a 8192 samples set. The second column contains the estimated signal frequency and in the third one we can find the estimated fundamental harmonic amplitude RMS value. The next column contains the measured signal RMS value, calculated using equation (1). The THD factor value is displayed in the next column, followed by the total execution time required by the FFT and THD functions. The last line presents the average values of interest for the dataset. The same information was generated for all the libraries and microcontrollers tested. Based on this data, we calculated the library estimation error using:

$$Err[\%] = \frac{|Val_N - Val_{est}|}{Val_N} \times 100 \tag{3}$$

where:

Val_{est} is the estimated value (algorithm result),

Val_{ref} represents the reference value (50Hz for the frequency error, the measured signal RMS value for the amplitude estimation error).

Table 3 presents the estimation errors for the *ESP32 FFT* library, for both voltage and current samples. As one can see, the error values remain approximately constant for all the samples of the same signal. Furthermore, we can observe that the error in estimating the amplitude of the fundamental harmonic is greater than that in estimating the signal frequency.

TABLE 2

ESP32 FFT library results for the voltage samples on the Raspberry Pi RP2040 microcontroller

Set	f [Hz]	E_{RMS} [V]	M_{RMS} [V]	THD [%]	FFT_t [ms]	THD_t [ms]
1	49.805	216.874	232.495	2.282	375.0	68
2	49.805	217.469	233.014	2.476	371.0	68
3	49.805	216.899	232.584	2.227	374.0	68
4	49.805	217.688	233.171	2.268	375.0	68
5	49.805	216.886	232.555	2.455	375.0	68
6	49.805	217.618	233.055	2.320	374.0	68
7	49.805	217.620	232.968	2.565	371.0	68
8	49.805	217.595	232.504	2.258	374.0	68
9	49.805	218.833	233.000	2.282	375.0	68
10	49.805	218.300	232.491	2.427	374.0	68
Avg	49.805	217.578	232.784	2.356	373.8	68

TABLE 3

ESP32 FFT library signal frequency and amplitude estimation error

Set	Estimation error [%]			
	f		H_f	
	Voltage samples	Current samples	Voltage samples	Current samples
1	0.391	0.391	6.719	8.066
2	0.391	0.391	6.671	8.021
3	0.391	0.391	6.744	8.006
4	0.391	0.391	6.640	7.846
5	0.391	0.391	6.738	8.020
6	0.391	0.391	6.624	7.974
7	0.391	0.391	6.588	7.940
8	0.391	0.391	6.412	7.683
9	0.391	0.391	6.081	7.308
10	0.391	0.391	6.104	7.386
Avg	0.391	0.391	6.532	7.825

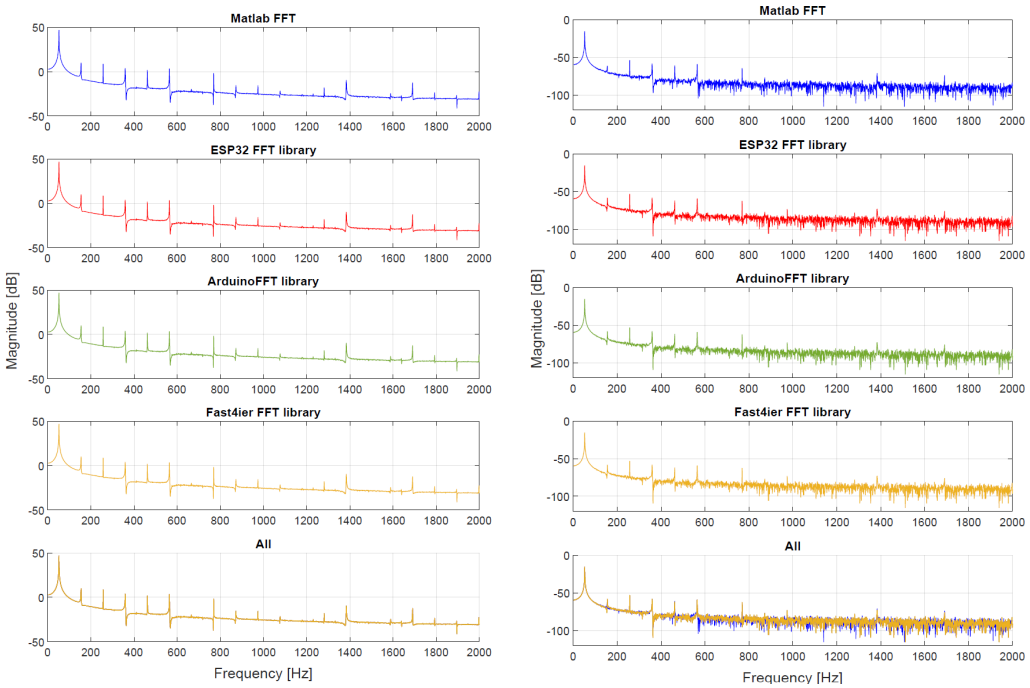
The same operations were performed for the other libraries as well as for the algorithm developed in Matlab. Figure ?? presents the FFT analysis results for all the libraries and Matlab algorithm for both voltage and current samples. The last chart of each signal contains overlapping curves of the results. Even though there is a significant difference in the RMS values of the two analyzed signals, 232V for voltage and 0.18A for current, we can observe that the curves of the library results and Matlab overlap almost perfectly in both cases.

The comparative estimation errors are presented in Table 4. We can observe that both frequency and fundamental harmonic amplitude estimation errors are similar across all three libraries. In contrast, the estimation errors of the Matlab algorithm are significantly smaller. This indicates that the Matlab *FFT* function may apply additional corrections to the input signal, such as filters and windowing functions that can reduce the effect of spectral leakage and increase the estimations accuracy.

TABLE 4

FFT libraries signal frequency and amplitude estimation error

Library	Estimation error [%]			
	f		H_f	
	Voltage samples	Current samples	Voltage samples	Current samples
ESP32 FFT	0.391	0.391	6.532	7.825
Arduino FFT	0.327	0.327	6.532	7.594
Fast4ier	0.327	0.329	6.532	7.594
Matlab	0.007	0.007	0.102	1.435



(A) Voltage samples (B) Current samples

FIGURE 4. Libraries FFT analysis results for voltage and current samples

TABLE 5

FFT analysis execution time on different boards/microcontrollers using the ESP32 FFT library

Set	Board/Microcontroller FFT execution time [ms]							
	ESP32 S3 Pro	ESP32 S3 Mini	ESP32 Wrover-E	ESP32 S2 mini	ESP32 C6-Wroom-1	ESP32 C3 Mini	ESP32 H2 Mini	RPi RP2040
1	40	48	53	115	136.5	219.5	374	386.5
2	39.5	48.5	52.5	114.5	136.5	219	372.5	383
3	40	48	52.5	115	136.5	219	372.5	385.5
4	39	48.5	53	114.5	136	219	372	385.5
5	39.5	48.5	52.5	114.5	136	219	372.5	387
6	39	48	53	115	136	219	373.5	385.5
7	40	48.5	52.5	114.5	136	219	373	383
8	39.5	48	52.5	115	136	218.5	373	385.5
9	40	48	52.5	115	136.5	219	372.5	385.5
10	39	49	52.5	115	136	219	372.5	386

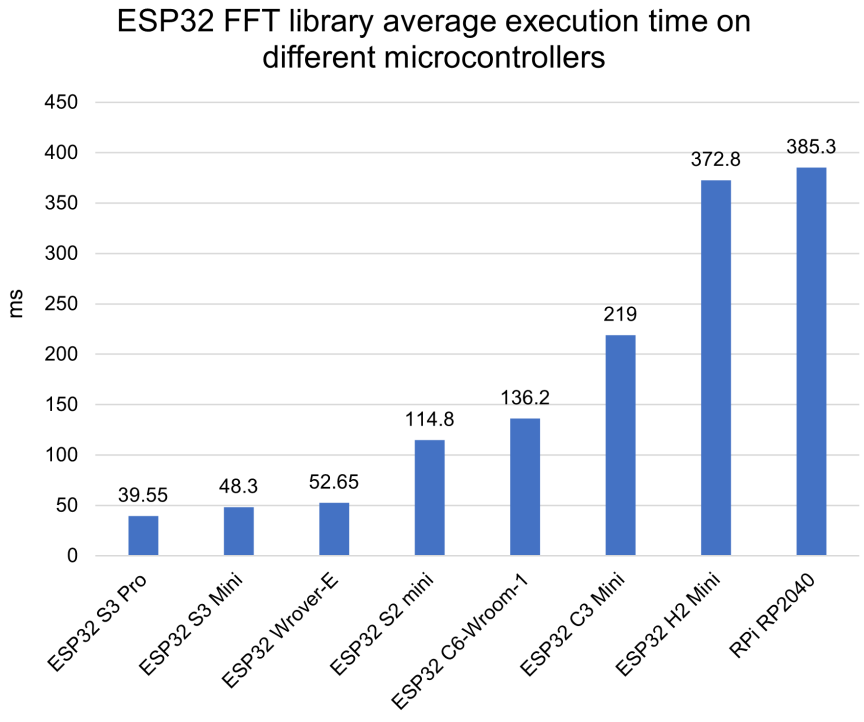


FIGURE 5. ESP32 FFT library average execution time on different boards/microcontrollers (lower is better)

TABLE 6
FFT and THD analysis execution time on different boards/microcontrollers for all libraries

Board / Microcontroller	Library					
	FFT time [ms]			THD time [ms]		
	ESP32 FFT	Arduino FFT	Fast4ier	ESP32 FFT	Arduino FFT	Fast4ier
ESP32 S3 Pro	39.55	385.55	155.8	16.55	5.45	4.6
ESP32 S3 Mini	48.3	520.3	256.3	16.9	5.5	4.7
ESP32 Wrover-E	52.65	703.8	321.3	17.95	5.9	4.95
ESP32 S2 mini	114.8	526	377.2	22.3	8.1	7.85
ESP32 C6-Wroom-1	136.2	436.7	390.55	27.2	5.3	5.35
ESP32 C3 Mini	219	719.35	724.65	34.5	8.75	8.05
ESP32 H2 Mini	372.8	1224.7	1235.15	55.65	13.95	12.5
Raspberry Pi RP2040	385.3	1282	1142.9	66.15	19.2	15.9

Table 5 displays the FFT analysis execution time of the *ESP32 FFT* library on the microcontrollers. It can be observed that the execution time varies significantly across different microcontroller models, indicating that their performances are not similar. In Figure 5, we have graphically represented the average execution time on the studied microcontrollers, calculated based on the datasets in Table 5.

Using the total execution time of the FFT and THD functions implemented using all the tested libraries, we sorted the microcontrollers based on their performance, information presented in Table 6.

The best performance was achieved with the ESP32 S3 microcontrollers, followed by the ESP32 Wrover-E. Next in line are the other ESP32 models, while the longest execution time, indicating the weakest performance, was obtained by the Raspberry Pi RP2040 microcontroller. This result was expected, as the RP2040 integrates a low-power Cortex M0+ processor, while the ESP32 utilizes more advanced *Xtensa* processors. Additionally, it can be observed that the shortest execution time was achieved with the *ESP32 FFT* library. The second-best performance was achieved by the *Fast4ier* library, which obtained shorter execution times than *ArduinoFFT* across most of the tested microcontrollers.

TABLE 7

Libraries memory usage

Library	No. bytes
ESP32 FFT	196716
ArduinoFFT	131104
Fast4ier	65552

From the perspective of the libraries memory usage, the comparative results are presented in the Table 7. The *ESP32 FFT* library has the highest memory usage, while *Fast4ier* is the most efficient, using the least amount of memory. This behavior can be explained by the number, size and data type of the buffers used by the libraries. Using all the performance indicators calculated for the three libraries (estimation errors, memory usage, and total execution time), we created a ranking chart, presented in Figure 6. To highlight the differences between the libraries, the values were calculated after running the algorithms on the same development board: ESP32 S3 Pro. The fundamental harmonic amplitude and harmonic frequency estimation errors were calculated as the average of the estimation error values for the voltage and current samples for each library, extracted from Table 4. The execution time of each FFT analysis algorithm was taken from the first row of Table 6 (for the ESP S3 Pro board), while the total memory usage was extracted from Table 7. We selected the maximum value of each indicator and used it as a reference to normalize the remaining values accordingly. For example, the memory usage indicator was calculated using equation (4).

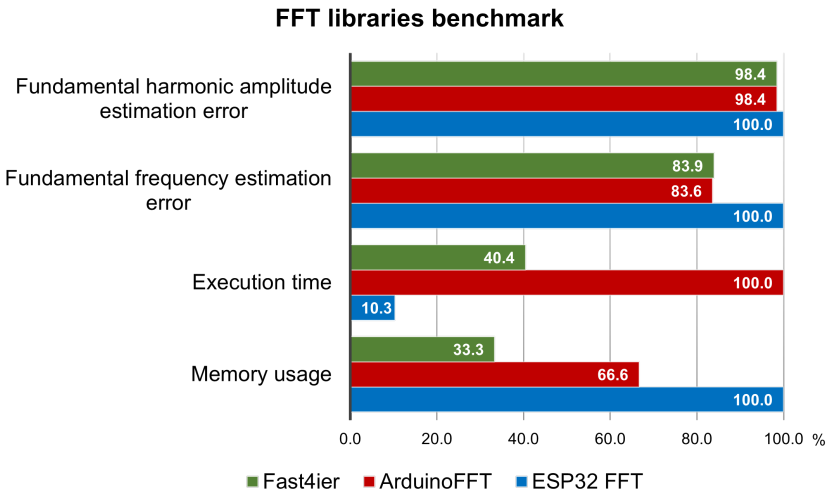


FIGURE 6. FFT libraries benchmark

$$LibX_{Mem}[\%] = \frac{LibX_{Mem}[bytes]}{MAX(LibX_{Mem}, LibY_{Mem}, LibZ_{Mem})[bytes]} \times 100 \quad (4)$$

The results indicate that the *ESP32 FFT* library is the most efficient in terms of execution time, while the *Fast4ier* library is the most recommended when constraints related to memory usage are most important. The *ArduinoFFT* library shows average performance across all analyzed aspects.

The THD factor errors, based on the FFT analysis results provided by the three libraries, were computed using (3), with the Matlab dedicated *THD* function's output value used as the reference. The results are displays in the last three columns of Table 8. The second column presents the error of the same THD function from the microcontroller algorithm, implemented by us in Matlab. From the last row of the table, which presents the average error across the 10 tested sample sets, we can observe that the THD factor obtained based on the FFT results of the three libraries is similar, with the *ESP32 FFT* and *ArduinoFFT* libraries yielding the same result, while *Fast4ier* shows a slightly higher error. Additionally, the error obtained using our implemented THD function in Matlab, based on the results of the dedicated *FFT* function, is larger compared to using the FFT libraries running on microcontrollers. This could be due to the larger errors in estimating the harmonic amplitudes generated by the tested libraries, which may bring the THD factor value closer to the one generated by the Matlab's *THD* function.

TABLE 8

THD error for the different libraries compared to Matlab's standard THD function

Set	THD error [%]			
	Matlab custom THD	ESP32 FFT library	ArduinoFFT library	Fast4ier library
1	2.107	2.082	2.082	3.514
2	6.865	6.354	6.354	7.414
3	6.448	5.404	5.404	6.604
4	5.280	5.225	5.225	1.398
5	2.929	1.972	1.972	5.674
6	3.804	2.784	2.784	9.941
7	10.172	9.731	9.731	1.447
8	5.454	5.386	5.386	0.750
9	0.329	0.595	0.595	1.984
10	7.522	6.012	6.012	9.866
Avg	5.091	4.555	4.555	4.859

5. Conclusion

In this paper, we analyzed the performance of several FFT libraries on commonly available microcontrollers. We compared the memory usage and the accuracy of the fundamental frequency and fundamental harmonic amplitude estimation. Based on the execution time of the analysis, we classified the microcontrollers in terms of performance.

The source signals used during the tests were acquired from the local grid network using a power meter designed by us and presented in another paper. The amplitudes of the grid and current signals harmonics resulting from the FFT analysis were used to calculate the Total Hamonic Distortion (THD) factors. The results were compared with the output of Matlab's dedicated *FFT* and *THD* functions, and based on this, the THD estimation error was determined.

The results obtained indicate significant differences in the performance of the microcontrollers, including execution time and memory usage by the tested libraries. However, the accuracy of the FFT analysis is similar to that provided by Matlab dedicated function. Compared to the results provided by Matlab, the calculation error of THD factors is similar across all tested libraries. This outcome suggests that the difference is caused by the accuracy in identifying signal harmonics and the error in estimating their amplitudes.

This benchmark provides an overview of the execution performance of advanced signal processing procedures across various models of microcontrollers available on the market. It can serve as a starting point in selecting a suitable FFT library or microcontroller during the design stage of an IoT device.

REFERENCES

[1] *A. Buzachis, A. Galletta, A. Celesti, M. Fazio and M. Villari*, Development of a Smart Metering Microservice Based on Fast Fourier Transform (FFT) for Edge/Internet of Things Environments, 2019 IEEE 3rd International Conference on Fog and Edge Computing (ICFEC), Larnaca, Cyprus, 2019, pp. 1-6, https://doi.org/10.1109/CFEC.2019.8733148

[2] *QL. Mao, and MY. Zhai* A new grid frequency estimation algorithm based on the fractional FFT for IoT nodes time stamps, Cluster Computing **22** (Suppl 4), 8155–8160 (2019), https://doi.org/10.1007/s10586-017-1653-2

[3] *B. Mazzoni, S. Benatti, L. Benini and G. Tagliavini*, Efficient Transform Algorithms for Parallel Ultra-Low-Power IoT End Nodes, IEEE Embedded Systems Letters, vol. **13**, no. 4, pp. 210-213, Dec. 2021, https://doi.org/10.1109/LES.2021.3065206

[4] *C. Eleftheriadis and G. Karakonstantis*, Energy-Efficient Spectral Analysis of ECGs on Resource Constrained IoT Devices, IEEE Transactions on Biomedical Circuits and Systems, https://doi.org/10.1109/TBCAS.2024.3406520

[5] *G. Fabregat, J. A. Belloch, J. M. Badía and M. Cobos*, Design and Implementation of Acoustic Source Localization on a Low-Cost IoT Edge Platform, IEEE Transactions on Circuits and Systems II: Express Briefs, vol. **67**, no. 12, pp. 3547-3551, Dec. 2020, https://doi.org/10.1109/TCSII.2020.2986296

[6] *T.-Y. Sun and Yu-Hsiang Yu*, Memory usage reduction method for FFT implementations on DSP based embedded system, 2009 IEEE 13th International Symposium on Consumer Electronics, Kyoto, Japan, 2009, pp. 812-815, https://doi.org/10.1109/ISCE.2009.5156962

[7] *J. Mazuet, M. Narozny, C. Dezan and J.-P. Diguet*, A Seamless DFT/FFT Self-Adaptive Architecture for Embedded Radar Applications, 2020 30th International Conference on Field-Programmable Logic and Applications (FPL), Gothenburg, Sweden, 2020, pp. 115-120, https://doi.org/10.1109/FPL50879.2020.00029

[8] *Afzeri, Deni Kurnia, Muhammad Nur Hakim and Indriani Siti Nurmala*, Implementation of FFT in Industrial Induction Motor Monitoring via Mobile Phone, Journal of Applied Science and Advanced Engineering (JASAE), vol. **1**, no. 1 (2023): JASAE: March 2023 https://doi.org/10.59097/jasae.v1i1.8

[9] *N. Nicusor, L. Baicu and B. Dumitrascu*, Vibration spectrum analysis using FFT in the microcontroller, 2022 IEEE 28th International Symposium for Design and Technology in Electronic Packaging (SIITME), Bucharest, Romania, 2022, pp. 176-180, https://doi.org/10.1109/SIITME56728.2022.9987820

[10] *D. Pérez*, Power Quality Monitor for Residential Voltage, Electrical Engineering and Systems Science, Cornell University arXiv, https://doi.org/10.48550/arXiv.2005.06045

[11] *B.K.Vinay and Manjula N Harihar*, Comparative Performance Analysis of Fast Fourier Transform on ARM and DSP Core Using Standard Benchmarks, International Journal of Science and Research (IJSR), Vol. **3**, Issue 6, June 2014, https://www.ijsr.net/archive/v3i6/MDIwMTQ4MjU=.pdf

[12] *A. Ayala, S. Tomov, P. Luszczek, S. Cayrols, G. Ragghianti and J. Dongarra*, FFT Benchmark Performance Experiments on Systems Targeting Exascale, ICL Technical Report ICL-UT-22-02, Innovative Computing Laboratory, The University of Tennessee, Knoxville, 2022, https://icl.utk.edu/files/publications/2022/icl-utk-1548-2022.pdf

[13] *P. Steinbach, and M. Werner*, gearshifft – The FFT Benchmark Suite for Heterogeneous Platforms, In: Kunkel, J.M., Yokota, R., Balaji, P., Keyes, D. (eds) High Performance Computing. ISC High Performance 2017. Lecture Notes in Computer Science(), vol. **10266**. Springer, Cham. https://doi.org/10.1007/978-3-319-58667-0_11

[14] Wemos S3 Pro, Development board specifications and microcontroller datasheet, wemos.cc, https://www.wemos.cc/en/latest/s3/s3_pro.html

[15] Wemos S3 Mini, Development board specifications and microcontroller datasheet, wemos.cc, https://www.wemos.cc/en/latest/s3/s3_mini.html

[16] ESP32-DevKitC, Development board specifications and microcontroller datasheet, Espressif Systems, https://docs.espressif.com/projects/esp-idf/en/latest/esp32/hw-reference/esp32/get-started-devkitc.html

[17] Wemos S2 Mini, Development board specifications and microcontroller datasheet, wemos.cc, https://www.wemos.cc/en/latest/s2/s2_mini.html

[18] ESP32-C6-DevKitC-1, Development board specifications and microcontroller datasheet, Espressif Systems, https://docs.espressif.com/projects/espressif-esp-dev-kits/en/latest/esp32c6/esp32-c6-devkitc-1/user_guide.html

[19] Wemos C3 Mini, Development board specifications and microcontroller datasheet, wemos.cc, https://www.wemos.cc/en/latest/c3/c3_mini.html

[20] ESP32-H2-DevKitM-1, Development board specifications and microcontroller datasheet, Espressif Systems, https://docs.espressif.com/projects/espressif-esp-dev-kits/en/latest/esp32h2/esp32-h2-devkitm-1/user_guide.html

[21] Arduino Nano RP2040 Connect Product Reference Manual, Arduino S.r.l., https://content.arduino.cc/assets/ABX00053-datasheet.pdf
[22] *Dimitrios P. Bouras*, fix_fft, Arduino, 2024, https://www.arduino.cc/reference/en/libraries/fix_fft/
[23] *Linnes Lab, Orlando S. Hoilett, Akio K. Fujita*, KickFFT, Arduino, 2024, https://www.arduino.cc/reference/en/libraries/kickfft/
[24] *AlexGyver*, FFT_C, Arduino, 2024, https://www.arduino.cc/reference/en/libraries/fft_c/
[25] *Adafruit*, Adafruit Zero FFT Library, Arduino, 2024, https://www.arduino.cc/reference/en/libraries/adafruit-zero-fft-library/
[26] *Nobuhiro Kuroiwa*, OsakanaFFT, Arm Limited 2024, https://os.mbed.com/users/hamling_ling/?utm_source=platformio&utm_medium=piohome
[27] *Doku Newon*, DokuFFTPACK, Arm Limited 2024, https://os.mbed.com/users/dokunewon/code/DokuFFTPACK/
[28] *Paul Stoffregen*, Teensy Audio Library, PJRC, https://www.pjrc.com/teensy/td_libs_Audio.html
[29] *Michiel Steltman*, ESP32 Sampler, 2024 GitHub, Inc., https://github.com/MichielFromNL/ESP32Sampler?utm_source=platformio&utm_medium=piohome
[30] *Seeed Studio STU (Dmitry Maslov)*, Seeed Arduino Mic, Arduino, 2024, https://www.arduino.cc/reference/en/libraries/seeed-arduino-mic/
[31] *Armin Joachimsmeyer*, FrequencyDetector, Arduino, 2024, https://www.arduino.cc/reference/en/libraries/frequencydetector/
[32] *Colin Duffy*, AudioTuner, 2024 GitHub, Inc., https://github.com/duff2013/AudioTuner?utm_source=platformio&utm_medium=piohome
[33] *R. Scheibler*, FFT, Library for FFT, Arduino, 2024, https://www.arduino.cc/reference/en/libraries/fft/
[34] *E. Condes*, arduinoFFT, A library for implementing floating point Fast Fourier Transform calculations on the Arduino framework, Arduino, 2024, https://www.arduino.cc/reference/en/libraries/arduinofft/
[35] *J. Mercier, LIBROW*, Fast4ier, An FFT and IFFT library, Arduino, 2024, https://www.arduino.cc/reference/en/libraries/fast4ier/