International Journal of Engineering Research and Development
e-ISSN: 2278-067X, **p**-ISSN: 2278-800X, www.ijerd.com
Volume 6, Issue 9 (April 2013), PP. 67-72

FFT on ARM-Based Low-Power Microcontrollers

Ondrej Karpis[1]
[1]University of Zilina, Department of Technical Cybernetics, Zilina, Slovakia

Abstract:- This article discusses the processing of signals using fast Fourier transformation in low-power microcontrollers based on the ARM core. Simple microcontrollers do not support floating point numbers so fixed point data types must be used. Two ways of calculating FFT are analyzed: CMSIS library and special algorithm optimized for use on ARM processors (FFT-ARM). Both methods utilize a radix-4 decimation in frequency (DIF) algorithms. The radix-4 algorithm limits the length of signals that can be used. This limit can be overcome by signal resampling using linear interpolation. The performance of both algorithms is tested on the STM32F100 microcontroller.

Keywords:- FFT, low-power microcontroller, fixed-point numbers, ARM core

I. INTRODUCTION

Today we see massive penetration of electronics into all areas of our lives. Advances in technology of semiconductor manufacture, the emergence of new types of sensors (MEMS), increasing battery capacity and the development of technologies for obtaining energy from the environment, allow the deployment of digital systems in previously unprecedented extent. Virtually any device that uses electricity can be improved through the integration of control systems. Such embedded systems improve equipment performance, add new features, improve user comfort and can reduce energy consumption. In many cases, the tasks performed by the embedded system are not very challenging so simple microcontrollers can be used as the control element of the system. However, there are many applications that require some signal processing. This could be due to the need to measure consumption of the device, to evaluate voice commands of the user, to identify potential equipment failure and so on.

Another reason for the use of signal processing may be an effort to reduce energy consumption in devices that communicate wirelessly. For example, in the case of wireless sensor networks (WSN), it is often desirable to minimize the amount of data transmitted between network elements (nodes) because wireless communication is very energy intensive. Limiting communication can extend equipment lifetime. In these cases, it is appropriate to carry out signal pre-processing directly at the place of signal origin and to transmit only data relevant to the task solved. Algorithms for data compression often use frequency analysis of the data. In terms of price and low power consumption of network elements it is usually not possible to use powerful microcontrollers or digital signal processors as the core of the control system. The most commonly used are relatively simple microcontrollers with low power consumption, size and cost. These microcontrollers typically do not contain support for signal processing (such as support for floating point numbers). If we need some kind of signal processing, we must solve this problem. Basically there are two main options: to use floating-point emulation or to use fixed point math. Each of these options has advantages and disadvantages. Using floating-point emulation is quite time-consuming and usually cannot be used if we require real-time processing of the measured signal. On the other hand, the use of fixed point is quite fast but increases noise level in the calculations. This article is aimed to examine the possibility of calculating the FFT with low-power microcontrollers based on ARM core.

One of the motivations of this analysis is that at our university we often use low-performance microcontrollers in various applications: multi-agent systems [1], wireless sensor networks for traffic monitoring [2], energy harvesting [3] and others.

II. FFT ON LOW-POWER MICROCONTROLLERS

The vast majority of 32-bit microcontrollers are currently built on ARM cores. Low-power microcontrollers typically use cores Cortex-Mx. To support the use of these cores library Cortex Microcontroller Software Interface Standard (CMSIS) was developed by ARM Ltd. This library provides a single standard across all Cortex-Mx series processor vendors. It enables code re-use and sharing code across software projects and reduces time-to-market for new embedded applications. This library contains:

- mathematical functions,
- statistical functions,

- support for working with matrices, vectors and complex numbers,
- various types of filters and the control algorithms,
- transform functions (DCT, FFT) and more.

Complex Fast Fourier Transform (CFFT) utilizes a radix-4 decimation in frequency (DIF) algorithm and the size of the FFT supported are of the lengths 16, 64, 256 and 1024. The data can be in the fixed point (Q15, Q31) or floating-point (F32) format. The processing of real valued signals is realized by a real FFT. Real FFT of N-point is calculated using CFFT of N/2-point followed by a split process. Real FFT supports the lengths 128, 512 and 2048.

The second option to calculate the FFT is using one of the freely available or commercial source codes from the Internet. One of the best is the source code available at http://www.lartmaker.nl/projects/fft-arm/ (hereinafter referred to as FFT-ARM). The code contains a very minimal set of functions for radix 4/5 complex fixed point in-place FFT routines, optimized for the DEC/Intel StrongARM and other recent ARM cores. All that's provided by the original code are FFTs with size 20, 64 and 80. The core code is written in such a format that a C-compiler for ARM can directly translate each line into a single instruction, with optimal scheduling for StrongARM. It is optimized to make maximum use of the full ARM register set (to minimize loads/stores). The FFT-ARM code is designed so that with 12-bit numbers plus sign can be handled without overflow. The code is based on the algorithms from [4], [5]. The source code was adjusted by author of this article to calculate 256-point FFT. In order to minimize the code size and RAM consumption, unnecessary parts and redundant variables have been removed from the code (e.g. the 20 and 80-point FFT routines).

III. EXPERIMENTS

Both mentioned methods of calculating the FFT have been tested on board equipped with microcontroller STM32F100CBT6B (Fig. 1). This board was developed as a prototype node for traffic monitoring based on measuring changes in Earth's magnetic field using a magnetometer. More information about the node and first measurements made with it can be found in [6]. STM32F100CBT6B microcontroller is based on a Cortex-M3 core with 128 KB of Flash memory and 8 KB of RAM. The small size of RAM limits using of this microcontroller in applications that require signal processing. Microcontroller clock frequency during the tests was 24 MHz. To compile the source code was used development environment IAR Embedded Workbench for ARM 6.30. Size-limited evaluation license for this software is available free of charge.

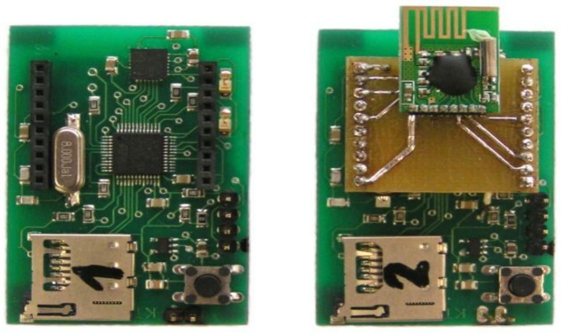

Fig.1: Node prototype for traffic monitoring

The main objectives of the experiments are:

- to evaluate the impact of using fixed point data formats on FFT accuracy,
- to compare the effectiveness of the CMSIS and FFT-ARM algorithms in terms of speed and code size,
- to evaluate the quality of signal spectrum obtained from signal resampled using linear interpolation.

A. FFT and Different Data Formats

Using fixed point numbers increases quantization noise and thus degrades the quality of the results. These tests should answer the question how serious is the reduction of the accuracy of FFT calculation. CMSIS library supports data formats Q15, Q31 and floating point. FFT-ARM algorithm uses only format Q12. To verify the accuracy of the calculations was used software SCILAB 5.4. FFT coefficients calculated using SCILAB are believed to be accurate and provide basis for comparison of all other calculations. Test signals were generated in double precision and then rounded to the tested data format. All tests were based on calculation of 256-point complex FFT.
As test signals were used:

- sinusoidal signal with a period of 26 samples,
- triangular signal with a period of 24 samples,

- saw signal with a period of 15 samples,
- human voice sampled at a frequency of 8 kHz.

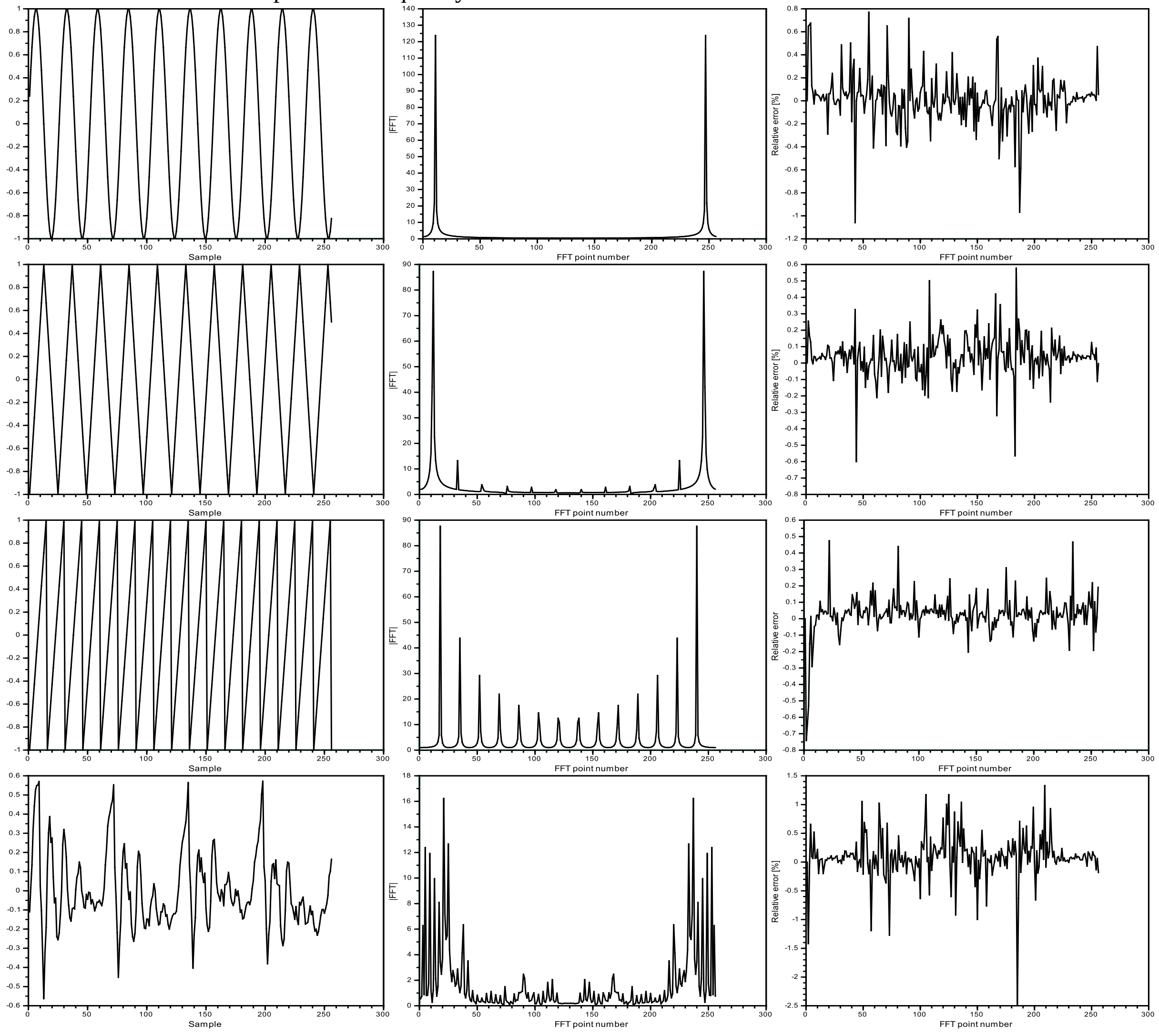

Fig.2: Test signals

The Fig. 2 shows the individual test signals, their spectrum and the relative difference between the actual and the calculated FFT expressed in %. The spectrum shown belongs to actual FFT as the calculated one looks virtually the same. The presented results are produced by the ARM-FFT algorithm, since it is expected to produce major differences from reality.

Magnitude of the relative error is usually up to 1 %. In special cases, the relative error is not suitable to evaluate the accuracy of the FFT calculations. For example, if the FFT coefficient is close to zero, even small absolute error can produce high relative error. This situation occurs especially when the base period is an integer multiple of the input signal period.

Table 1 shows the results of all tested data formats. For each number format is displayed maximum size of the relative error, the size of the standard deviation and median of relative errors.

Table 1: Relative Errors

Algorithm	Error [%]	sinus	triangle	saw	voice
CMSIS F32	max	67.5 e-05	38.6 e-05	26.4 e-05	35.8 e-05
	Stdev	9.58 e-05	5.97 e-05	4.56 e-05	5.83 e-05
	Median	1.9 e-05	1.72 e-05	1.37 e-05	1.01 e-05
CMSIS Q31	Max	10.2 e-05	7.68 e-05	10.8 e-05	27.4 e-05
	Stdev	2.21 e-05	1.44 e-05	1.55 e-05	4.57 e-05

	Median	0.99 e-05	0.6 e-05	0.46 e-05	1.52 e-05
CMSIS Q15	**Max**	3.48	1.62	2.64	10.5
	Stdev	0.81	0.44	0.63	1.66
	Median	0.45	0.23	0.23	0.52
FFT-ARM Q12	**Max**	1.06	0.6	0.74	2.48
	Stdev	0.22	0.12	0.12	0.38
	Median	0.07	0.06	0.04	0.13

The tests reveal two interesting facts:

1. The quality of the algorithm FFT-ARM with data format Q12 is higher than the CMSIS Q15 format. So far we are unable to determine the cause of this surprising result.
2. Data format CMSIS Q31 achieves slightly better results than CMSIS F32. In general, floating point data types have greater range than fixed point ones, but worse resolution. In this particular case, a higher resolution is perhaps more important than the total range.

Based on the obtained results it can be concluded that in most cases using of a fixed point FFT computation has not significant impact on the quality of results. In fact, greater source of error can be AD converter. For example, 12-bit converter integrated in the tested microcontroller has the maximum deviation between the actual and the ideal transfer function of up to ± 5 LSB.

B. Performance of Algorithms CMSIS and FFT-ARM

Comparing the performance of algorithms in terms of speed is based on measuring the time it takes to calculate given number of transformations. Timing was based on the timer interrupt triggered every millisecond. Number of calculated transforms was 10000. Both algorithms overwrite the input data with the output of the FFT. If the calculations are performed sequentially, each calculation works with different data. Table 2 shows the results for the algorithm FFT-ARM: the size of generated code, the total execution time and the number of FFT calculations per second. Speed of calculation and the size of the generated code depend on the type of optimization used. In this case, all the basic types of optimization provided by the IAR compiler were used.

Table 2: Performance of the FFT-ARM Algorithm

Optimization	**high speed**	**high size**	**high balanced**	**medium**	**low**	**none**
size [B]	6176	5940	6168	6024	6392	6728
time [ms]	9063	9013	9109	9208	11173	14506
speed [nr/s]	1103	1109	1097	1086	895	689

The fastest (and smallest) compiler-generated code was achieved with optimization focused on the size of the program. The measured values show that the performance of the FFT-ARM algorithm is very dependent on the applied optimization. However, even if there is used no optimization, the algorithm is able to perform nearly 700 FFT calculations per second.

Table 3 shows the results for the algorithm CMSIS. In this case, the speed and size of the code is practically independent on the applied optimization as the CMSIS library is already highly optimized. Therefore, only the results for high speed optimization are presented.

Table 3: Performance of the CMSIS Algorithm

Data type	**F32**	**Q31**	**Q15**
size [B]	14424	13388	9844
time [ms]	100940	24490	19806
speed [nr/s]	99	408	504

The results obtained are consistent with the assumption that the floating point emulation is quite time consuming. In this test, the emulation reaches only about one quarter of the performance achieved with fixed-point data type Q31. Taking into account the results of the first test, it seems that there is no reason to use floating point data types at all.

Analysis of the algorithms with comparable output quality (CMSIS Q15 and FFT-ARM) shows that FFT-ARM algorithm is about 120 % faster than CMSIS Q15 (1109 vs. 504 nr/s). Code size is similar in both cases (9013 B and 9844 B).

To enable real-time audio processing (assuming 8 kHz sampling frequency and 256-point FFT) the microcontroller must be able to calculate 31.25 FFT per second. In this respect, it seems that the performance of all tested algorithms is sufficient. However, we must realize that the FFT calculation is just one part of the sound processing and the microcontroller must perform other tasks too, such as reading data from the sensors, storing data on a storage medium, device management, communicating with other systems and so on. Another important factor is energy consumption. Consumption of the microcontroller varies greatly in different runtime modes. Fast execution of all necessary tasks and the subsequent transition to sleep mode can save a considerable amount of energy and thus increase the lifetime of the device (if it is powered by batteries). From this perspective, the clear winner is the ARM-FFT algorithm.

C. Resampling of the Input Signal

In practice it is often necessary to calculate the FFT with number of points other than any power of two. Although the general FFT algorithm can be applied to arrays of any size, both tested algorithms are optimized only to certain sizes. CMSIS algorithm can compute complex FFT of the arrays of size 16, 64, 256 or 1024. For the real FFT it can be 128, 512 or 2048 samples. FFT-ARM algorithm is currently optimized only for the calculation of the 256-point complex FFT. This limitation can be overcome by using signal resampling. In this case we can obtain required number of samples from the signal using some kind of interpolation. Interpolation can be done in different ways but the simplest is a linear interpolation. CMSIS library provides a function to calculate a single sample in a given interval using linear interpolation. After calculating the FFT from resampled signal it is possible to adjust the number of FFT points to the original number.
Fig. 3 shows the spectrum of the original signal consisting from 140 samples (the same human voice as in the previous test), spectrum of the signal resampled to 256 samples and the difference between the two spectra. Tested was only FFT-ARM algorithm.

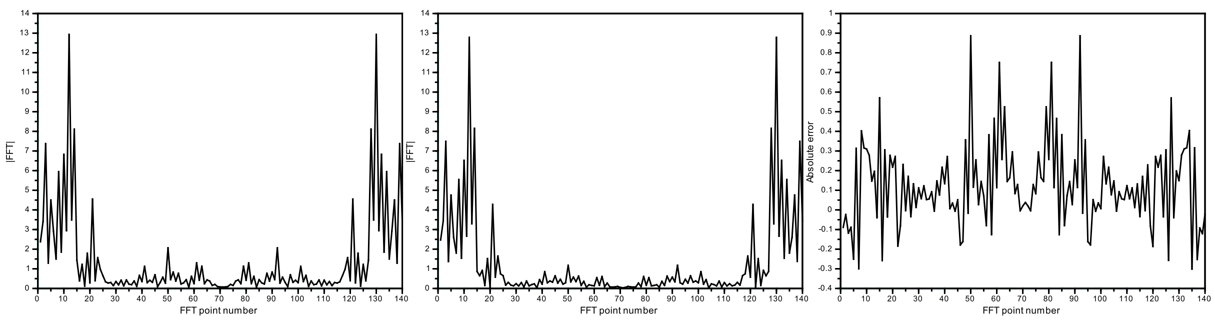

Fig.3: Comparison of frequency spectra

Despite the rather radical resampling (from 140 to 256 points), the obtained spectrum is very close to the original one. The differences are more visible only in the higher frequencies. Table 4 shows the results of resampling with different initial number of points (N_0). For each test is calculated the maximum absolute error, the standard deviation and median.

Table 4: Absolute Error After Resampling

N_0	**140**	**160**	**180**	**200**	**220**	**240**
max	0.887	0.963	1.002	1.072	1.144	1.146
stdev	0.217	0.236	0.205	0.198	0.201	0.201
median	0.138	0.094	0.094	0.091	0.086	0.084

The measured values shows that resampling from fewer points increases the size of the median. But it is interesting that it reduces the maximum error at the same time. Overall, the results obtained by resampling are pretty good. Of course there are more sophisticated ways of resampling (quadratic interpolation, interpolation using inverse FFT etc.), but they have far more computational complexity than linear interpolation. Even relatively simple linear interpolation caused decrease in the calculation rate from 1109 to 810 calculations per second (at maximum optimization).

IV. CONCLUSIONS

The presented experimental results demonstrate that calculation-intensive tasks, such as the FFT, can be implemented in low-power microcontrollers without major constraints. Of course, some optimization of the algorithms is required. Using of sophisticated algorithms for data processing directly in the place of their origin can significantly increase the lifetime of battery-operated devices. This applies particularly to devices communicating through wireless technologies such as wireless sensor networks. The results of this analysis will be used in WSN designed to monitor the traffic. Frequency analysis of the vehicle records shall allow better compression of measured data and their efficient transfer. This opens the way even to re-identification of vehicles in the transport infrastructure.

REFERENCES

[1]. J. Micek, M. Hyben, M. Fratrik, J. Puchyova, “Voice Command Recognition in Multirobot Systems: Information Fusion”, *International Journal of Advanced Robotic Systems*, Vol. 9, 181:2012

[2]. J. Micek, J. Kapitulik, “Wireless sensor networks in road transportation applications”, *Proceedings of MEMSTECH*, pp. 114-119, Polyana-Svalyava, Ukraine, 2011

[3]. M.Kochlan, P.Sevcik, “Supercapacitor power unit for an event-driven wireless sensor node”, *Proceedings of FedCSIS*, pp. 791-796, Wroclaw, Poland, 2012

[4]. W. H. Press et al., Numerical recipes in C:The art of scientific computing, 2nd edition, Cambridge university press, 1992

[5]. R. E. Blahut, Fast algorithms for digital signal processing, Addison-Wesley, 1985

[6]. O. Karpis, “Sensor for Vehicles Classification”, *Proceedings of FedCSIS*, pp. 785-789, Wroclaw, Poland, 2012