Wireless Networking

FFT operation with an intermittent power-source

Thijmen Ketel - 4623258
y.m.t.ketel@student.tudelft.nl

Carlo Delle Donne - 4624718
c.delledonne@student.tudelft.nl

Dimitris Patoukas - 4625943
d.patoukas@student.tudelft.nl

March 9th, 2017

Contents

1 Introduction

The advancements in technology call for devices to be smaller, faster and less power-hungry. In this sense, devices are being developed that do not even need charging in a classical way. These devices can harvest energy from the environment by, for example, a small solar-panel or RF electromagnetic waves. Some of these are already being used today in for example NFC-chips in bank cards.

The technological challenge in this can be made more complex with the addition of larger and more extensive calculations. Most of these devices need to work with power-sources that are not stable. These intermittent sources cause devices to operate in a different way. The uncertainty of operation that is introduced with a fluctuating power-source needs to be resolved by a different way of coding. Challenges range from data-consistency to speed to energy consumption.

In this project a program will be developed that is able to perform a *Fast Fourier Transform* (FFT), or a slimmed down form of FFT to reduce calculations and complexity, while dealing with an intermittent power-source. This has to be implemented on a MSP430 platform from Texas Instruments and has to be written in the C programming language so it can be easily ported to different platforms.

The group consists of three students from the MSc Embedded Systems, from three different countries (Netherlands, Italy and Greece). This mixture of backgrounds and education levels can provide different views on how to approach the problem at hand.

2 First steps

In this chapter the first steps that are taken will be laid out. This includes the plans that are made, the approach that is taken by the team and the first implementations on real-world hardware.

2.1 Hardware: Texas Instruments MSP430

The very first step that was taken was the selection of the hardware that would be used, namely a micro-controller from Texas Instruments: the MSP430FR5994 [1]. This platform, as is clear in the name, is also used in the prototypes that are made at the Embedded Software (ES) department at TU Delft. This means that the code for the development kit can easily be ported to the platform used at ES-department.

2.2 Organisation

To create a correct and clear organisation for the project, multiple services are used:

- **GitHub**
  For code organisation and version control, GitHub will be used. A public repository is created hold the code and review additions from each of the team members. Data-sheets will also be kept on the repository to keep them available for reference.
- **WhatsApp & Slack**
  For communication, platforms as WhatsApp and Slack will be used. WhatsApp mainly for communication between the team members and Slack is meant for communication that is public for the rest of the class.
- **ShareLaTeX**
  The report of the project needs to be written with the help of LaTeX for concurrent and neat formatting. To keep this easy to use for every team member an online editer is used: ShareLaTeX. This editor is able to be used by multiple members at the same time and packages for formatting do not have to be downloaded.

Furthermore, meetings between the team members will be documented in a GitHub Markdown file for easy and simple formatting. The important parts will be transferred to the LaTeX document.

2.3 Discrete Fourier Transform

The next step to take is to implement a common *Discrete Fourier Transform* (DFT) on the development kit to get a feeling of the process [3]. The DFT is meant as a simple (in terms of math) operation so that the essence of the program is executed and the program can be reworked to an intermittent redundant system.

2.3.1 Signal creation

To implement the DFT, the first thing needed is a representation of an input signal, which we chose to be a combination of multiple sine waves. Because it is the discrete transform, the signal is represented by an *array* which holds a value for every time-step. The first problem that is encountered is the lack of a floating point unit (FPU) in the MSP430FR5994. This means either that floating point operations need to be handled by software (with libraries), resulting in a larger overhead, or that fixed point numbers should be used.

Fortunately, the MSP430FR5994 does have a *Low-Energy Accelerator* (LEA) which can be used for many mathematical operations and DSP-like applications. This LEA allows faster multiplications and even includes some *Fast Fourier Transform* utilities.

The way fixed points are handled on the MSP430FR5994 is with the help of two libraries that takes care of the representations and calculations of fixed point variables: the *IQmathLib* library and the *DSPLib* library. The way this is done can be configured by specifying the amount of bits that are be used for the decimals of the fixed point variable. In this way magnitude of variables can be sacrificed to gain precision in calculations [4].

2.3.2 DFT

Now that the input signal has been created, the implementation of the DFT can start. The code found in [3] is simple C-code but because it uses multiple floating point operations is has to be adapted to be compatible with the MSP430FR5994.

To do this, library documentation needs to be read and some simple functions to port the code will be used. The following decisions are made concerning the DFT:

- To conserve accuracy the decision is made to set the "fractional part" bits to 15. This means that numbers between -1 and 1 are allowed with an accuracy of 0.00003051758 (this can of course be changed later on);
- To calculate the magnitude of the Fourier coefficients the multiplications can cause an overflow. To counteract this problem the signal is scaled, and it is converted back after such operations;
- To decrease overhead as much as possible, the code will use many API function to perform standard mathematical operations. These functions would otherwise introduce large amounts of overhead due to the floating point operations.

When the code is ported and tested, the created input signal can be run through the DFT code to generate a spectrum. The code for this will be available on the repository. To keep the computation time relatively short, the essential runs are done with the following settings:

- The amount of samples (time-steps) that are given to the DFT is 32.
- The sample frequency is 32 *Hz*.
- Due to the chosen sample frequency, only input frequencies lower than 16 *Hz* are allowed for accurate processing. In this case frequencies of 2 *Hz* and 10 *Hz* are chosen.
- To keep the samples within the allowed range they are scaled down $\frac{1}{32}$ times their original value. This can be corrected after processing due to the linearity property of the Fourier transform.

Even though the DFT is applied on a relatively small sample size, low signal frequency and with use of the LEA, the resulted execution time is almost 5 seconds. To decrease this time the clock-speed of the micro-controller might be increased later on, for now this will be sufficient. The resulted discrete Fourier coefficients are stored in an array which the Texas Instruments Code Composer Studio IDE can form into a graph, which is shown in figure 1.

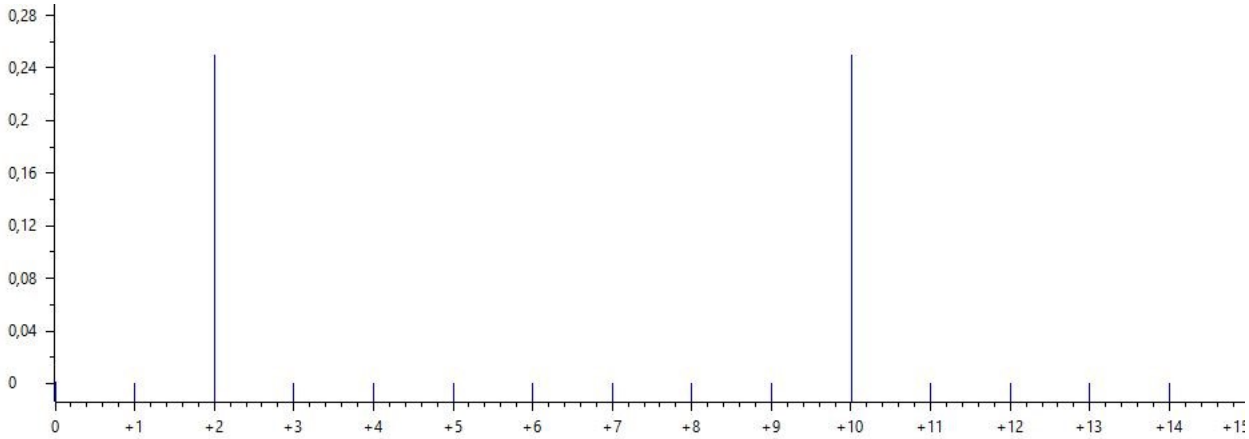

Figure 1: Discrete-Fourier Transform from the generated signal in section 2.3.1. The signal was created with two sinusoidal waves, one with a frequency of `2 Hz` and one with a frequency of `10 Hz`. This is visible from the spikes displayed in the graph (this picture is a screen-grab from the Code Composer Studio IDE)

These first steps are completed with the successful execution of the Discrete Fourier Transform (with maybe some room for optimisation). The next steps will be creating a first solution for operation under an intermittent energy-source condition. After which the MSP430FR5994 will be powered through another embedded platform which can simulate a randomly intermittent energy-source to test the proposed solution. In the next section the approach to create a solution will be laid out together with explanations of systems from which inspirations might be drawn.

3 Intermittent operation

The next step is to find a way to make the system redundant for intermittent operation. This means that the system can ensure data integrity even though it is rebooted during run-time. The system will need to be able to save its state and after a reboot check if a state has been saved and then reload that specific state. This will require an advanced way of writing and reading the current system state.

3.1 Mementos

One way to handle operations under an intermittent energy-source is to apply the *Memento* pattern to the code [5] (which of course is a small nod to Christopher Nolan's *Memento*). This software design pattern is developed to perform a dynamic check-pointing operation on the program during run-time. The system is able to sense the available energy that is left in the system and based on that, decide to checkpoint the program or continue computation. In case of a checkpoint the systems state (or only the essential data) will be stored in non-volatile memory. After a reboot the program can load the data from memory and continue its operation.

The benefits of a dynamic system like Mementos is that state saves and read/write overhead is reduced to a minimum due to the evaluation of available energy. Checkpoints can be introduced on only the most important parts of the program and less energy will be wasted by unnecessary read/write operations.

The drawbacks however do have a larger implication. Hardware support is needed to keep track of the available energy in the system, which will create extra overhead to deal with. This makes implementation of Mementos harder on the large scale as many micro-controllers and applications do not always have the hardware support needed or are able deal with the overhead that is added.

Due to the fact that Mementos is one of the first methods that offer a redundant way of executing code with an intermittent energy-source, it is not suited for this project.

3.2 DINO (Death Is Not an Option)

Another advanced method that can deal with operation under an intermittent energy-source is *DINO* (Death Is Not an Option) [2]. This is a coding style and compiler addition that uses, unlike Mementos, a static check-pointing system. The systems is a completely software-based solution, removing the need for hardware support. This makes it able to be implemented on almost any kind of platform.

With DINO, the programmer sets *task boundaries* to split the main program into sub-sections. *Tasks* are operations between two task boundaries, and they are dynamically defined by the execution flow. At a task boundary the program state is guaranteed to be consistent with the code executed thus far. Failures can occur, but in such cases the program restarts from the last consistent task boundary, making DINO a resilient and reliable approach.

3.3 Chain

coming soon...

3.4 Proposed approach

coming soon...

References

[1] MSP430FR5994 LaunchPad Development Kit.[Online]. Available: http://www.ti.com/tool/msp-exp430fr5994

[2] Brandon Lucia, Benjamin Ransford, *A simpler, safer programming and execution model for intermittent systems*, Newsletter ACM SIGPLAN Notices - PLDI '15, Volume 50, Issue 6, June 2015, Pages 575-585

[3] Simple DFT in C.[Online]. Available: https://batchloaf.wordpress.com/2013/12/07/simple-dft-in-c/

[4] Texas Instruments, MSP430 IQmathLib Users Guide version 01.10.00.05. January 19 2015

[5] Benjamin Ransford, Jacob Sorber, Kevin Fu, *Mementos: system support for long-running computation on RFID-scale devices*, Proceedings of the sixteenth international conference on Architectural support for programming languages and operating systems, Pages 159-170