Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Measurement and Information Systems

Implementation of a webserver on a DSP processor

BSc Thesis

Author
Bálint Csengeri

Supervisors
Károly Molnár
György Orosz

December 14, 2012

Contents

HALLGATÓI NYILATKOZAT

Alulírott *Csengeri Bálint*, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy autentikált felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Budapest, December 14, 2012

Csengeri Bálint
hallgató

Acknowledgements

First of all, I would like to express my appreciation to ProDSP Ltd. for making it available for me to work on this project. I am much beholden to my exterior supervisors, Károly Molnár and László Balogh for their time, expertise and patience answering my questions and for their encouragement. In addition, I am specially grateful to Gergely Molnár for his help of building the client website with his great ideas, experience and helpfulness.

Abstract

The purpose of this project is to implement a portable solution for configuring and real-time monitoring an embedded control system, by a simple webpage interface. The reader of this thesis is going to get a review of the used technologies, programming guidelines, the architecture of the system, the performance of the application and the communication process. Furthermore, some implementation tricks, and future development opportunities are discussed.

The example application was created as desired, and its proper functioning has been verified.

The frame of the application is the communication flow of Javascript with a custom embedded HTTP-server that runs on the DSP. The client starts the communication periodically with an AJAX query. The server parses the request, initiates the configuration of itself based on the data received. Afterwards, the server sends the requested data or status information. In addition, the measurement of the parameters of an analog audio signal and other configuration processes were added to demonstrate the capabilities of the system. Moreover, a direct data (spectrum of the audio signal) visualization module was added to the client webpage.

This solution could be part of any medium-high performance industrial embedded project due to its fairly low resource need. The user can access the hardware and software resources of the system by a very easy-to-use webpage, which is a much more convenient way than using a separate program, because only a web browser is needed with a Javascript engine on the client machine.

Kivonat

A szakdolgozatom célja, hogy egy beágyazott irányító rendszerekben alkalmazható modult megtervezzek és megvalósítsak, ami lehetőséget biztosít arra, hogy a beágyazott rendszer konfigurálható és megfigyelhető legyen egy egyszerű weboldalon keresztül.

Az olvasó áttekintést kap a használt technológiákról, a programozási irányelvekről, a megvalósított rendszer felépítéséről és a használt kommunikációs csatornáról. Továbbá bemutatunk néhány trükkös megoldást a megvalósítás részleteiből és néhány ötletet a továbbfejlesztés lehetőségeiről.

A kezdeti kitűzött célt sikerült elérni, egy ténylegesen működő alkalmazás született.

A rendszer alapja egy kommunikációs csatorna létrehozása a kliens oldali Javascript program és a beágyazott, jelprocesszoron futó HTTP-szerver között. A kliens a kommunikációs csatornán aszinkron periodikus kéréseket (AJAX) küld. A szerver feldolgozza a kéréseket, végrehajtja a rendszeren a kérés által specifikált változtatásokat. Ezt követően visszaküldi a kliensnek a kért adatokat vagy státusz információt. A jelprocesszoron ezen funkción felül egy analóg jel mérése és feldolgozása, valamint további konfigurációs folyamatok is megvalósításra kerültek, annak érdekében, hogy demonstráljuk a rendszer lehetőségeit és képességeit. Továbbá a mért jelből előállított amplitúdóspektrum vizuális megjelenítését is elértük a kliens oldalon.

Az alacsony erőforrásigénye miatt, a létrehozott rendszer ipari projektekben is alkalmazható. A felhasználó tetszőleges hardver- és szoftvererőforrásokat érhet el egy nagyon egyszerűen használható weboldalon keresztül. Ez egy sokkal kényelmesebb megoldás, mint egy külön futtatandó program a számítógépen, mert nincs hozzá másra szükség, mint egy böngészőre, ami Javascript támogatással rendelkezik.

Chapter 1

Introduction

In this chapter, I shortly explain the tasks needed to create the working example application. Additionally, I list the motivation and the technologies that were covered. Finally, the basic structure of the document is introduced.

It is a common industrial need to monitor and configure embedded control systems in real-time. This can be done by implementing a simple webserver application on the control system which can monitor numerous signal and hardware parameters and configure the algorithms running or the hardware itself. The most important specification of the system is to implement a monitoring and configuration utility to provide the easiest interface possible for the management of an embedded system, where no special programs or runtime libaries are needed on the client side, just a common web browser. The embedded system provides a simple-to-use webpage which is managed dynamically depending on the current state of the control system. During this task, I could acquire knowledge from many different areas that was needed to solve the problems arised. It was essential to learn the development hardware board and its programming technologies, the use of an embedded real-time operating system, the TCP/IP protocol stack, web programming and some signal processing techniques. According to the knowledge requirements of the task, it is on the very edge of the embedded and the non-embedded fields, where technologies nowadays are not so standardized yet. The solutions for the problems that arise in this area are sometimes quite custom and special therefore they are not reusable efficiently. Thus I was trying to implement a technology that could be reused in different platforms. Additionally, I tried not to utilize the special properties of the system to avoid the probable extra difficulties of porting the solution to a different environment.

The example application contains some signal measurement features: frequency estimation, Root Mean Square (RMS) estimation, peak detection and real-time spectrum representation. Additionally, the project yields some hardware and algorithm configuration features from the webpage: frequency estimation method, FFT resolution and direct change of the state of the LEDs. In addition, a simple event detector module has been implemented, which counts the button push events. A simple character based UART console was added to the project as well, mainly for debugging purposes, but it could be used for direct changing and monitoring the state of the system by avoiding the Ethernet link and

the client side. All the specified functionalities were successfully created, the system was calibrated and the proper operation was verified.

The motivation for the project was primarily a potentially real industrial need, but it was also a very effective way of learning about embedded systems and about the very border of the embedded and the PC worlds. Specifically, all the following areas were covered:

- a HTTP server
  - basic structure,
  - operation,
  - configuration,
- the embedded system
  - software development,
  - reusable code implementation by using layers,
  - configuration,
  - performance,
  - limits,
- a certain embedded real-time operating system (RTOS)
- advanced driver development for the embedded system
  - Ethernet,
  - UART,
  - Audio Codec,
- some signal processing methods
- basics of advanced web programming
  - CGI,
  - Javascript,
  - AJAX.

In chapter 2, a review is given about programming and scripting technologies that could be used on the client and the server side, the webservers are introduced in general and the chosen one in detail.

In chapter 3, the questions related to the embedded and the client design of the example system are discussed. I expound the available hardware and software resources, the audio signal measurement layout, the software schematic and the flowcharts and their explanation of the processes at the embedded system, the client page and script design and the communication channel between the server and the client.

In chapter 4, a review of the coding policies and techniques is given, which were used for making the code easily readable, maintainable and understandable. Moreover, the tricks learned along the way are explained, too.

In chapter 5, the performance output of the application is discussed. Additionally, the global capabilites, the stability, the measured bottleneck of the system and the attributes of the measurement are presented.

In chapter 6, some suggestions are given about the usability of the solution and the possible enhancements. Some areas of the industry are also listed where it can be utilized successfully. I also demonstrate the shortcomings of the implemented application. Furthermore, the porting possibilities are covered.

Finally, in chapter 7, the work and the results are summarized.

Chapter 2

Technologies

In this chapter, the programming and scripting technologies are introduced that were used for client and server side development.

2.1 Embedded webservers

Webservers establish and control the information flow between the client and the server. This is achieved by generating and sending webpages to clients.

Embedded webservers try to provide similar functionality as normal webservers, but due to their embedded nature, they are obviously more difficult to configure, less dynamic and less advanced due to the lack of computing power and system resources. Using brute force algorithms are not possible, and it is a common need to achieve as low resource usage as possible.

On embedded servers, no real server-side high abstraction level script interpreters are available for server side programming (such as PHP, PERL, Python, etc.), everything has to be parsed and put together in the lower abstraction level language of the embedded system (typically C/C++).

Embedded systems tend to have a single controller (microcontroller or digital signal processor), which is responsible for multiple different tasks, for example real-time control of an embedded system. Hence, the monitoring and configuring interface that is realized by a webserver is preferred to run in the background when there is nothing more important to do, typically considered as low priority task.

Embeddable webservers have great variety, some of them are open source, and directly compilable only on Linux/Windows based systems, but not in the real-time operating system that we use, the VisualDSP++ Kernel. Analog Devices ported a quite lightweight and usable webserver (GoAhead) onto their own devices, but later they removed it from the usable modules[1] for commercial reasons, altough the old source code remained available. Hence, the GoAhead webserver was chosen, mainly because it was already ported to some similar devices (DSPs) and it was considered to be the fastest way to make progress.

[1] from VisualDSP++ 5.0 Update 9

2.1.1 GoAhead webserver

The GoAhead webserver [10] is lightweight enough to be used on a Blackfin project (even on lesser processors). It supports more functionalities than needed in this project. The basic functionalities of the GoAhead webserver are:

- Active Server Page Support,
- EJscript Support,
- EMF database compatability,
- form processing,
- URL requests from memory,
- basic security (User Management),
- Digest Access Authentication,
- SSL,

some of which need an active file system (SSL, User Management), however most of the features are able to run purely from memory.

As the usage of a file system is completely optional, we decided not to use it, as the server can run even faster, when it is running completely from fast access memory. Moreover, the presence of a file system is a great dependency, which might not be affordable on every embedded system. While not having a file system, it is emulated for the client (webpage at the browser), meaning information is accessed via standard URL requests and forms.

The GoAhead implementation (last modified in 2007) proved to be stable, altough the used source code contains several types of warnings, some of which could result in a runtime error. For example, a stack overflow error is able to occur quite easily when the server faces a relatively large amount of requests[2]. Other than this I did not experience any systematic error.

By now, some more recent GoAhead implementations exist, which have slightly different code structure, hence the already ported old revision was used to avoid possible incompatibities.

2.2 Server side technologies

2.2.1 C/C++

The project requires C/C++ as development languages as it runs on a DSP. The development environment of Analog Devices is called VisualDSP++. Under VisualDSP++ C, C++ and assembly languages are allowed to be mixed, every module should be written on

[2]This "large amount of requests" would be negligible load for a standard internet webserver, but we must never forget that the whole performance of the system is incomparably lower, and the intened use is not the public internet.

its most suitable abstraction level to best fulfil its purpose. VisualDSP++ contains a compiler that supports the features described just before, which is a quite common expectation in embedded systems.

VisualDSP++ lacks some important features, which could speed up the debugging process, therefore some algorithms and code blocks that will be platform independent or can easily be emulated by exterior software such as signal processing methods should be implemented in a more convenient environment, for example in Microsoft Visual Studio. The greatest advantage of Visual Studio is that some common coding errors (e.g. array indexing, memory violation, etc.) can be discovered and fixed incomparably easier.

2.2.2 CGI

Common Gateway Interface (CGI) is a standard in web programming. It is a common practice that server side processing programs are CGI scripts which are called by URL request with parameters from the client. The source code of CGI scripts are not reachable from the client, the server interprets the URL specified script by its interpreter engine with the parameters of the request. Common server side scripting languages are PHP, Perl, Python, etc. all of which are very high abstraction level and robust interpreted languages with high resource need. Thus, in our case, running any script interpreters are not possible, particularly due to the lack of resources. Fortunately, from the point of view of the client it is completely transparent how the results are created, the only thing that matters is the proper format of the output. This transparency allows the webserver to receive and parse the information and answer to the requests using a quite custom non-script based CGI interface. The consequence of this custom solution is that everytime a new functionality[3] is to be added, a new embedded side service function written in C/C++ has to be added instead of writing a separate file that contains a server side script.

GoAhead server has a module that provides a server side EJscript C interpreter engine which could have been used, but the custom solution was considered to be easier to use and to have much rapid runtime.

2.3 Client side technologies

In this section, the standards, languages and techniques available for client side development are introduce.

2.3.1 HTML

HyperText Markup Language (HTML) [9] is a markup language with which we can create webpages that are displayed by web browsers.

Switches, named tags configure the behavior of the text that is surrounded by them. HTML files contain HTML objects (e.g. tables, paragraphs, etc.) all of which is possible to be assigned with a unique ID or name.

[3] meaning a new type of request

Two main ways exist for the creation of HTML pages: What You See is What You Get (WYSIWYG) method (e.g. MS Frontpage), or writing direct HTML code in a text editor manually. Both methods have advatages but if an optimal and stable webpage is to be created, manual editing should be chosen. WYSIWYG HTML editors tend to put a high amount of unneccessary generated code fragments in the output HTML code, which can result in instability and unreasonably big code size, which rarely matters due the common speed requirement of webpages. The most efficient and frequent way of creating HTML pages is manual editing in a text editor. We can find great text editors (e.g. Notepad++), that can identify the language used, and provide numerous ways to aid our coding process:

- coloring code segments (data types, comments, etc.),
- highlighting language keywords,
- adjusting code structure automatically (text tabulation),
- basic syntax check.

Moreover, online editors[4] are available too on the internet, which tend to contain an editor panel and a viewer window, where the changes in the HTML code are displayed automatically. In addition, these online editors often provide a support for script (Javascript) editing, which is found to be very useful.

HTML pages can be made more dynamic by client-side scripts (such as JavaScript). There are tags and attributes in the HTML language that influence the style of the text, however webpage styling is frequently performed by Cascaded Style Sheets (CSS). HTML language primitives are considered to be easy to learn, many good and complex examples and copy-pastable templates are available on the internet on different HTML explanatory sites [17].

There are obvious problems with HTML. It is a quite unpleasant behavior of HTML (and generally of markup languages) is that no "debugging" opportunity is available, just basic sytax check. The HTML viewers (web browsers) are able to be considered as the debuggers as well. This is caused by the very nature of markup languages, as they just set how the structure of text and objects would appear. Consequently, in a rather massive project, complexity reduction techniques (CSS, embedding HTML pages into each other, etc.) should be used to make it easy to find errors.

2.3.2 CSS

Cascading Style Sheets (CSS) [9] is a style description language supported by all browsers. CSS is suitable for configuring the visual appearance of objects of the HTML/XHTML documents in detail.

CSS insertions could be put in the HTML document itself, but according to a more common practice, it takes place in a separate file (.css) accompanied with the HTML file.

[4]for example: `http://jsfiddle.net/`

The browser queries the specified CSS file after finding its include directive in the HTML source, just like a picture.

The main advantage of CSS is that the complexity of the HTML documents become greatly reduced by not containing any style setting information, consequently the maintenance of the style and the whole webpage become much easier as core objects and style description become disunited. Another significant advantage of using CSS is that certain style files are accessible from multiple HTML sources, hence the maintenance of the style become concentrated in one file instead of multiple complex attributes of HTML objects.

2.3.3 XML

Extensible Markup Language (XML) is a markup language as well as HTML, but it is not standardized or predefined at all, every data format is completely user specified. XML has a completely different goal than HTML.

The very purpose of creating XML is to store information in text based format. XML files can be put together or parsed at both client and server side. Tags may be defined for searchability or other purposes, but they are not obligatory.

The advantage of XML based information storage is that data is visible, meaning that it is readable for both humans and computers. Data structures and objects are defined manually, this results in logical and easily parsable data structures.

At this project I only use an alternative of XML when sending data to the client, in a simple no-tagged format. The exact format was originated from Java (JSON).

The use of a completely text based data structure may sound strange as it takes part in an embedded project where saving of resources is the biggest deal. In the PC world it is widespread because its robustness, easy processability and standardized and viewable appearance compared to a list of complex binary C-like data structs. At the embedded side, we must adapt to the client standards to provide an interface which is viewable from browsers.

Some examples for data storage format (these are the content of XML files):

```
Example 1 - XML file with tags:

<?xml version="1.0"?>
<email>
<from>sender@domain1.com</from>
<to>addressee@domain2.com</to>
<subject>greeting</subject>
<text>Hello World!</text>
</email>


Example 2 - Alternative of XML, used in this project with no tags:

[[0,12],[1,-2.3],[2,41.3],[3,-2],[4,32.1],[5,6.7]]\n
```

2.3.4 JavaScript

Javascript [11] is the most popular scripting language of client side web programming. Its syntax is similar to Java or C, but they are completely separate languages.

Javascript is considered to be an easy-to-learn language due to its simple syntax, high-level automatism and robustness.

Javascript is a very high abstraction level scripting language, many already implemented modules are created and hosted by different servers, all of which can be included directly from these hosts via internet.

It is rare that self-written functions are needed where no other libaries are used, just basic Javascript commands. Thus, before attempting to write a specific algorithm, a web search should be initiated in the commonly included Javascript modules for the function that might be suitable for the specific task. Searching is truely worth the time, because the commonly used modules tend to be more robust and probably faster than the self-written ones.

At the client side, Javascript runs in the background on the script interpreter engine of the web browser. The interaction to the user is realized by configuring and reading data in HTML objects. Javascripts start running on a specific event on the HTML page such as pushing a button, hovering over a box or finished loading the page.

Javascript is used for server side programming too, but not as widespread and efficient as PHP.

Browsers provide some effective ways for JavaScript debugging. In developer mode breakpoints can be added, stepping and running to the next breakpoint, monitoring the values of variables value in stepping mode, even in runtime, etc.

Sometimes browsers block the running of Javascript for security reasons, as countless malicious actions can be performed from JavaScript by highly elaborate tricks.

2.3.5 AJAX

Asyncronous Javascript and XML technique [18] (AJAX) is not a programming language, just a new way of using already created technologies and standards.

The greatest advantage of using AJAX is that it makes possible to change the content of webpages without reloading the whole webpage, or receiving a full HTML object (e.g. a HTML table), the reply of the server only contains the relevant data in XML or similar text format (e.g. a value to write in the cell of a table), and in the background Javascript parses and refreshes the already present objects of the HTML pages. This results in an increase in the speed of the service and the client itself, and it causes less internet traffic, therefore it results in additional rapidity.

In figure 2.1, we can see the basic idea for communication flow used by AJAX technique. AJAX functions typically post a request (as an URL) to the server and pend on the XML (or similar) answer for indefinite time. AJAX requests can be pending paralelly or sequentially, if more of them are present. If the response from the server has arrived, the AJAX functions tend to call another function which parses the received XML then execute

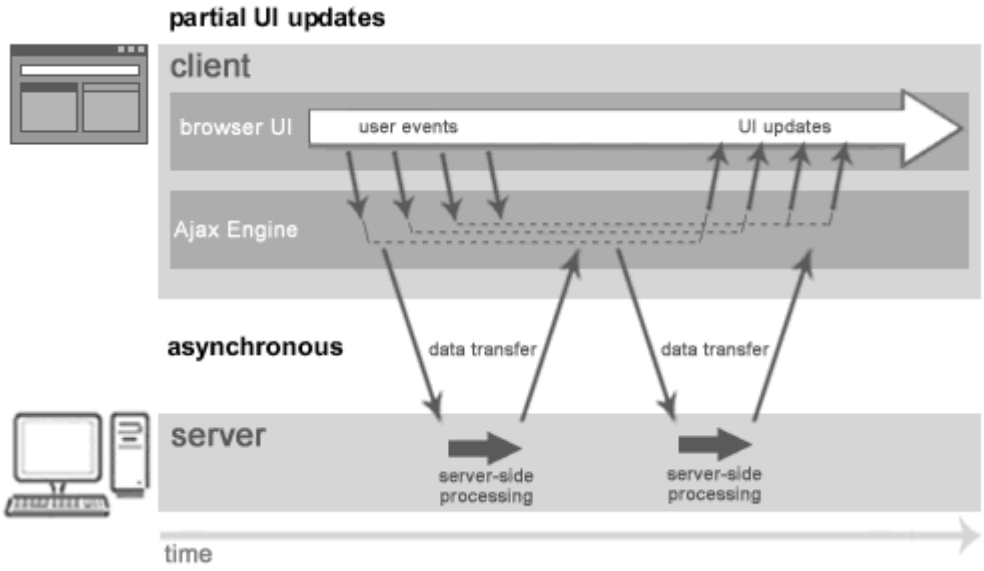

Figure 2.1: *AJAX information flow [16]*

the neccessary modifications in the webpage. The easiest way of using AJAX is including the Javascript libary that supports AJAX (e.g. jQuery), altough it is possible to write our own AJAX Javascript functions.

Common AJAX webpages are Gmail, Facebook or Google Earth, and others.

Chapter 3

Design

In this chapter, the design procedure is introduced, and the realized structure of the embedded and the client side is also presented in detail. The available hardware and software resources are considered as well.

3.1 Hardware components

3.1.1 ADSP-BF527C processor

Analog Devices Blackfin DSPs are medium-high performance, low cost, power efficient and widely usable DSPs that are highly popular along industrial applications for a wide range of purposes. Blackfin DSPs are designed to be the very fusion of digital signal processors and microcontrollers, because they have the typical peripherial set of microcontrollers, and the performance output of DSPs. Blackfin DSPs contain numerous peripherials, many of which are used in this project. Some examples could be found in the VisualDSP++ install directory for their basic configuration among the example codes. However, almost every peripherial driver needs to be rewritten by its user for the specific needs in order to reach the desired complexity and proper configuration of the application.

Blackfin processors tend to use quite high core clock frequencies (400-600 MHz). Their architecture allows a throughput of 2 MMAC[1]/1 MHz. According to the 32-bit address bus that Blackfin DSPs use, 4 GB of address space is available. Some models contain an embedded Ethernet Media Access Controller (MAC), that is needed by an Ethernet application. In this case, no external circuitry is needed for using the Ethernet.

The Blackfin DSP family has a wide range of DSPs, that have significantly different performance, peripherials and price. Therefore, developers can find a proper performance-price efficient DSP for the need of their applications.

ADSP-BF52xC cores [4] are designed for advanced connectivity and even lower power consumption than other family members. The letter C in their names refers to that they contain an integrated audio codec[2], meaning that a simple and complete audio hardware can be built from it with minimal external circuitry.

[1] Mega Multiply And Accumulate
[2] AD, DA and their configuration registers

3.1.2 ADSP-BF527 EZ-KIT LITE evaluation board

Evaluation boards are development boards provided by IC manufacturers to give their customers the possibility to try the capabilities of certain products. EZ-KITs are Evaluation boards from Analog Devices, which contain DSPs with many external peripherials (such as LEDs, buttons, memories, etc.).

EZ-KITs can be used for developing whole and complete applications too, but the usage of their peripherials is restricted due to the general purpose architecture of the boards. The most common usage of EZ-KITs are developing platform independent or easily portable applications or technologies, which are going to be put on a specifically designed hardware board later.

In order to use many configurable peripherials on a single EZ-KIT board, a very high amount of hardware configuration switches were added onto it. This causes the unpleasant and inevitable fact, that some peripherials are not able to operate simultaneously such as Ethernet and the DAC of the Audio Codec in our case. At this task, the situation was fortunate, all needed peripherials could work together, altough in the future, a specific board and control system may be developed for the webserver project. The peripherial and hardware configuration is not going to raise any questions due to the knowledge of the hardware engineer about the exact specification.

The hardware architecture of EZ-KITs are well documented, all the information for hardware configuration and programming can be found in the relevant Evaluation System Manual [3]. Many complex examples are provided with the EZ-KITs, therefore understanding the basic behavior and configuration of the DSPs is easy for a beginner. The configuration of some peripherials use the System Services libary described in section 3.3.1. System Services can be a bit difficult to comprehend at first. In addition, there are many weakly commented open-source driver examples (inside a certain dummy application) provided as well for EZ-KITs. These dummy applications can run standalone without the use of System Services, altough the core of these drivers must be extracted.

ADSP-BF527 EZ-KIT LITE provides a microphone in, a headphone out, a Line in and a Line out connectors for audio band signal measurement and generation.

Used hardware resources in our case (peripherials):

- SDRAM (64 MB),
- GPIOs (connected to LEDs and buttons),
- UART0,
- Audio Codec and its external circuitry,
- Ethernet MAC,
- SPI.

3.2 System overview

Figure 3.1 shows the physical layout of the whole system. The router was used to provide a local network to establish an Ethernet connection between the EZ-KIT and the PC. Applications are developed on the PC, and are downloaded to the EZ-KIT. There are two additional connections between the PC and the EZ-KIT. JTAG connection is used for development and debugging the application, UART-LINE is used for debugging, and additionally, a direct console is implemented via UART. The function generator is used for generating several signals for the measurement. The EZ-KIT has an audio channel input, where the analog signal is received. We use the digital multimeter to check the RMS voltage of the signal to ensure that it is not too high for the audio line.

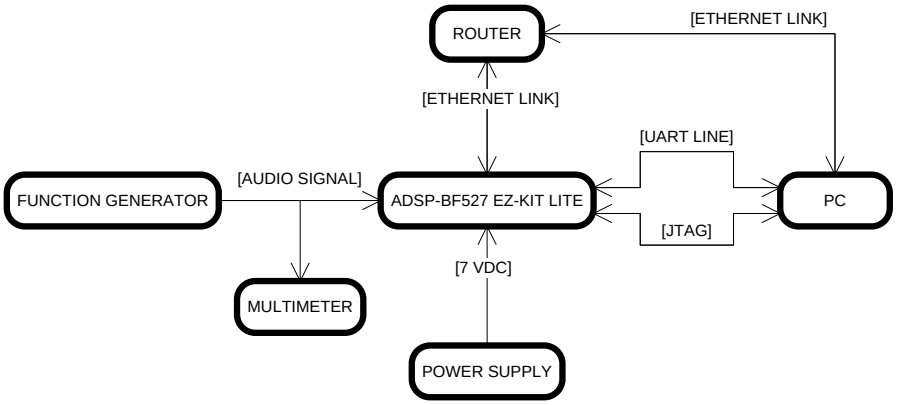

Figure 3.1: *System overview*

External equipment summary and their purpose in the measurement:

- *Function generator:* generating the analog audio signal,
- *BNC-jack cable:* function generator-board signal transfer,
- *DC power supply:* providing energy for the board,
- *DHCP server (router):* LAN connection,
- *Ethernet cable:* physical layer between the router and the board,
- *UART Serial Cable:* RS232-USB converter for server direct console,
- *USB cable:* development/debug via JTAG,
- *PC for development:* environment for development/debug,
- *Multimeter:* calibration and voltage level check.

In figure 3.2, we can see the physical system in the midst of a signal measurement procedure.

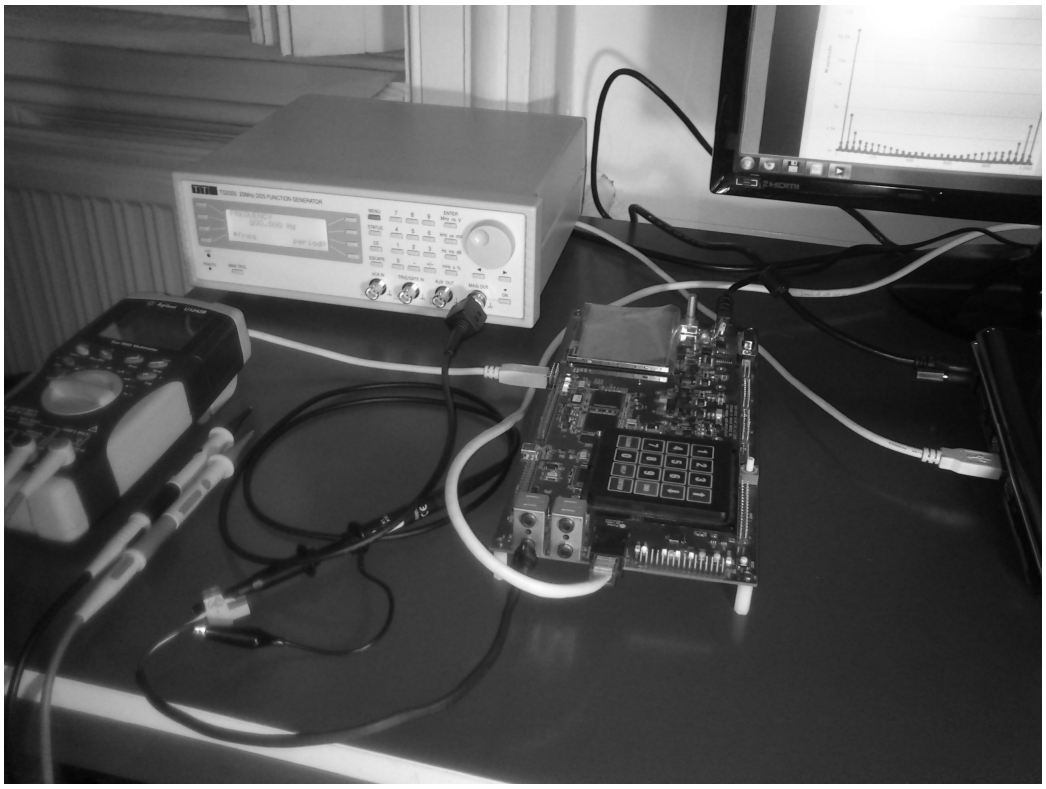

Figure 3.2: *System overview in reality*

3.3 Embedded software resources

3.3.1 System Services

Analog Devices provides a pre-written libary called System Services, where commonly used embedded system modules can be found which are already ported to all Analog Devices DSPs. These System Services can be configured for different DSPs by specific define statements and data structures.

System Services are quite complex and capable of implementing many different functionalities. Additionally, System Services has separate documentation [6].

System Services try to provide a platform independent initialization and control functionality to different DSP peripherials. This gives an additional higher abstraction level layer to the embedded application, but its usage can be too complicated.

System Services are sometimes very useful and result highly rapid application development, but in some cases understanding their usage is much more complicated than creating a self-written driver with the same functionality. Hence, it is worth to evaluate the complexity of each problem and consider whether to use the System Services. I considered using it at the initialization of the Ethernet MAC.

3.3.2 VDK

Before the introduction of VisualDSP++ Kernel (VDK), VisualDSP++ (VDSP) should be discussed shortly. VDSP is a good-old DSP development environment, it is quite easy

to use, and it has all the important and some extra features needed for embedded software development for Analog Devices DSPs. It lacks some serious comfort features (e.g. IntelliSense, function hyperlinks, etc.). However, the application development under VDSP is fairly satisfying, if someone can get used to the lack of these "luxurious" features, besides many workaround tricks exist for making the coding procedure easier.

Using an embedded operating system (kernel) at this project is inevitable, due to the need of a scheduler mechanism in order to run multiple different tasks described in section 3.3.

Choosing a specific kernel is often a complicated decision because of their wide variety, furthermore commercial, financial, experimental, modifiability and performance requirements should also be considered. For this project, VDK is the rational choice for the real-time operating system, because it is developed specificly for Analog Devices DSPs. More importantly, VDK can be used instantly, there is no porting procedure needed at all.

VDK [7] is a quite advanced real-time operating system, designed and implemented by Analog Devices. The design environment provided by Analog Devices (VisualDSP++) supports many VDK status and configuration modules (such as history, load, etc.), therefore implementation and debugging is easy and rapid while using VisualDSP++ and VDK combined. Altough, VDK is not open-source, it could be fully utilized by Analog Devices EZ-KIT owners. Additionally, a proper and well-commented skeleton code is provided for every VDK file created (thread, driver, interrupt service routine), which makes the coding procedure faster.

In the past (few years ago), there were neither really good, complex and usable working examples nor appropriate documentation given for VDK, but by now these deficiencies do not exist any more, and VDK became a reasonably competitive real-time operating system.

3.3.3 lwIP

The TCP/IP stack is a protocol stack, that realizes the different communication software layers for Ethernet based operations. Layers rely on the ones below them. According to the TCP/IP model[3], there are multiple layers with specific purpose:

- *physical or link layer:* contains communication technologies, e.g.: Media Access Control (Ethernet MAC)
- *network layer:* connects local networks, e.g.: Internet Protocol (IP)
- *transport layer:* handles host-to-host communication, e.g.: Transmission Control Protocol (TCP), User Datagram Protocol (UDP)
- *application layer:* process-to-process level communication, e.g.: Secure Shell (SSH), Telnet, Hypertext Transfer Protocol (HTTP)

[3]More models exist. Other models divide the roles of layers differently, or they may even contain additional layers for specific functionalities.

It is known, that applications based on Ethernet require a working TCP/IP stack. Fortunately, VisualDSP++ gives an option to embed a free and open-source TCP/IP stack in the application called lightweight IP (LwIP) [1].

LwIP focuses on low system resource consumption while providing full TCP/IP functionality. LwIP is a platform independent module, which has to be ported (specific driver needs to be written) to the specific system that it will be used on. This procedure has done for ADI DSPs by Analog Devices already, therefore LwIP is an includable and working module to the project.

There would have been other (commercial and open-source) implementations for a working TCP/IP stack. However, LwIP is considered to be the best alternative due to its stability, wide user support, moreover it has been ported to Blackfins already.

3.4 Client resources

In this section, the availabe modules are introduced that could be used for client side development.

3.4.1 jQuery

JQuery is an advanced libary based on Javascript. JQuery supports many commonly used features and methods of advanced web programming.

In this project, we have to refresh the client webpage asynchronously without reloading whole HTML objects. AJAX technique is exactly usable for this purpose, as introduced in section 2.3.5. The easiest way to use AJAX is including the jQuery Javascript libary, which has some functions that support AJAX. However, jQuery is usable for a wide range of functionalities, just a small fraction of the libary was used at this project.

The description of jQuery by its developers: "jQuery is a fast and concise JavaScript Library that simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid web development. jQuery is designed to change the way that you write JavaScript." [14]. JQuery is considered to be very easy to use, its functions and methods tend to be robust and documented properly. The source code is available for including or downloading from numerous internet hosts.

As it is includable, its source code is accessible, moreover according to its license terms [15], it is allowed to use anywhere even commercially without restriction.

3.4.2 Plotting charts

It was intended to visualize the real-time amplitude spectrum of a certain audio signal (array of direct data) on the client webpage in this project to prove the proper operation of the system. The amplitude spectrum is the absolute value of the result of the Fourier transform of the signal. Fourier transform can be realized with Discrete Fourier Transform (DFT) in digital systems, and Fast Fourier Transform (FFT) is a computation algorithm of the DFT.

An additional external libary should be included, if we would like to implement a direct data visualization module to the webpage for which there are two basic alternatives:

- Flot [13],
- Highcharts [12].

Flot is a basic and simple way of realizing the direct data visualization feature. It is very fast at refreshing the curves in runtime. According to its easy usage and simplicity, its appearance is quite modest. Highcharts is noticeably slower than flot, but it has far more configuration options, and also a much more attractive appearance. However, the speed of Highcharts on a recent PC configuration is satisfying. Flot is usable completely free of charge for any purpose. However, Highcharts is allowed to use freely only for personal purposes, a commercial licence has to be purchased for mercantile usage. I decided to use Highcharts, because it has a zoom feature by default, moreover it makes the webpage to look far more advanced, altough no commercial attitude has been considered yet.

3.5 Embedded software design

In this section, the questions about the design of the embedded software are discussed.

3.5.1 System structure

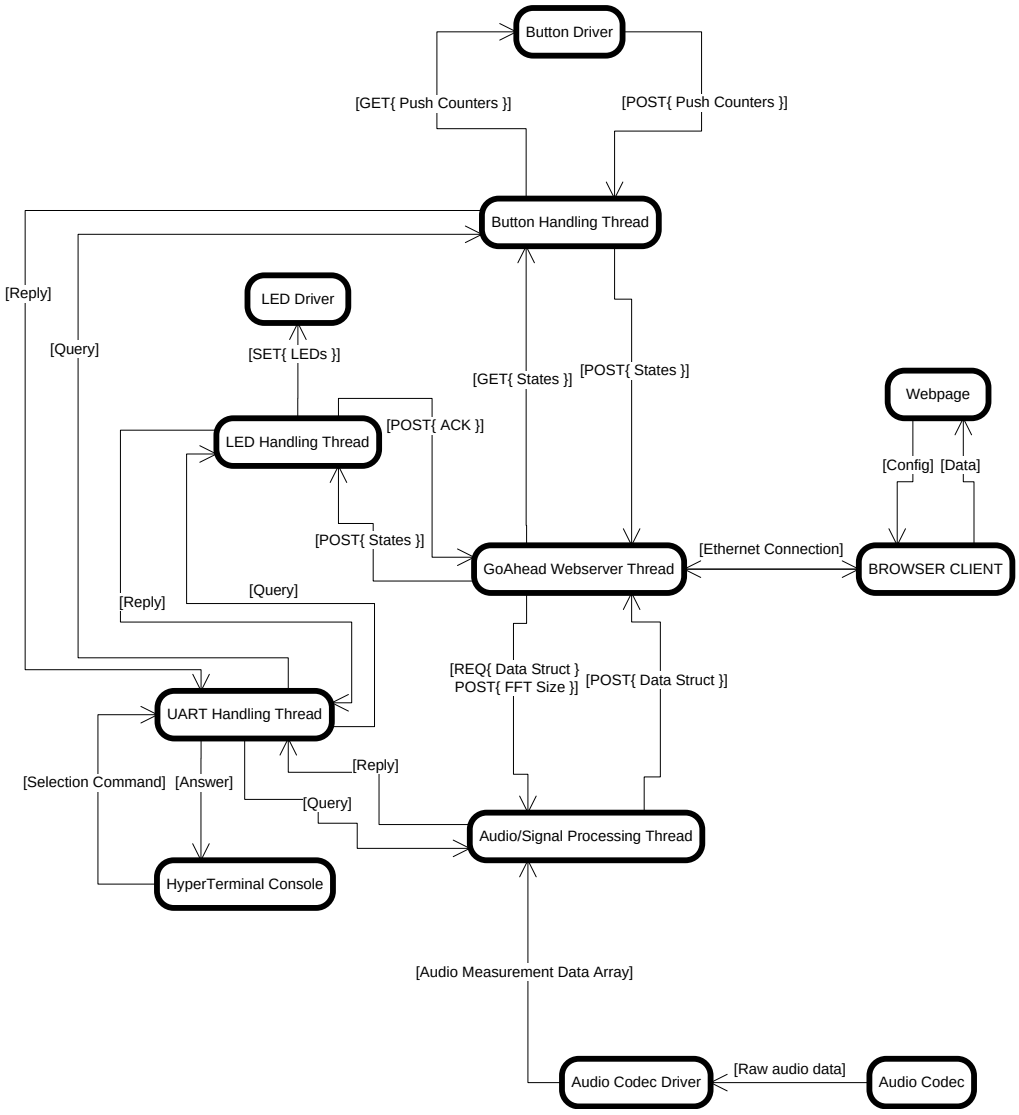

Figure 3.3: *System architecture*

In figure 3.3, we can see the block diagram of the system. Bubbles represent processes or abstract entities[4] and arrows show communication or control flow between them. Arrows have guard expressions, which specify the purpose or content of the arrows. The very center of the system is the webserver thread. The webserver thread establishes the connection between the webpage and the embedded control system. The webserver thread queries other threads for data in the embedded application. The implemented system contains real-time control and observation features, that are realized by hardware peripherial interrupts and periodic queries of hardware states based on timer interrupts. These functionalities

[4] such as external programs that take part of the project, drivers, etc.

are examples to demonstrate the capabilities of the implemented system, but any real-time control or observation (e.g. switched current control) feature could be realized instead.

3.5.2 Webserver

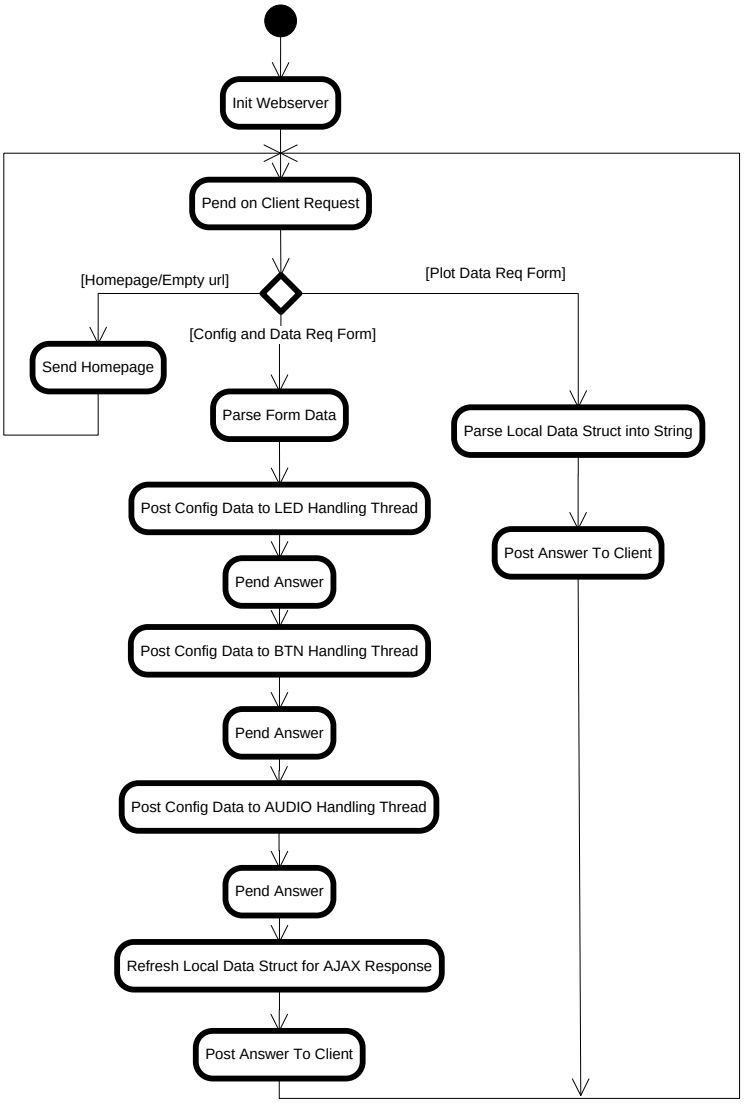

Figure 3.4: *Webserver thread process flow*

The main purpose of the webserver thread is to handle URL requests from clients. As we can see on figure 3.4, there are three basic types of requests that clients are planned to perform:

- *Request for file:* Typically the homepage HTML file, that is commonly requested by empty string or index.html, but this branch is performed in case of any file requests. This module is implemented by the server itself, no additional actions needed but putting files in the memory beforehand with the webcomp module, that is described in section 4.4.

- *Parameter set form:* Contains hardware (desired LED states by client) and algorithm configuration (FFT size, frequency estimation method) data, that was put together in Javascript at the client side. The running application should adjust its state as the request specifies. In this case, the server thread has to query each thread one-by-one for their specific data, afterwards it has to parse the responses of threads, then it has to send the answer to the client.

- *Direct data request form:* Indicates a request for the spectrum of the signal (DFT result). The server has to query the Audio handle thread for the data needed, then send it to the client.

3.5.3 Audio

The flowchart of the audio thread is shown in figure 3.5. The flowchart begins with the initialization of the audio codec. The communication between the thread and the audio driver is designed as follows:

- *global FLAG:* This indicates whether a new buffer has just become ready.

- *global POINTER:* This points to the most recent buffer that has been filled already.

The basic structure of the audio thread consists of two branches:

- *Executing signal processing:* The signal processing branch has to be performed if the FLAG indicates that new buffer has become filled.

- *Pending on a possible webserver request:* The audio thread pends on a possible query message from the webserver thread with a timeout[5] if the FLAG indicates that no new buffer has become filled yet. No action is performed if timeout of the pending has occured, the control goes back to check the FLAG. An answer with the most recent data has to be sent to the webserver thread if the query message from the webserver thread has arrived.

The signal processing branch of audio thread calculates numerous signal parameters as we can see in figure 3.5. First of all, the twiddle stride[6] must be adjusted for Fast Fourier Transform (FFT) size specified by the user. Afterwards, a FFT operation is done, from which we get the complex spectrum of the audio signal, then the absolute value of the spectrum is calculated. The next steps are to calculate the different frequency estimations, the RMS value of the signal, its peak value. Afterwards an averaging should be done to increase accuracy of some results (estimated frequencies by methods Simple Null Cross Detection and Interpolated Null Cross Detection as explained later). At the end of the branch, the communication struct needs to be updated with the most recent results, then

[5]Pending on a message also realizes the thread sleep.

[6]Twiddle stride is the leap distance between twiddle table indexes: if the twiddle table has a length of 1024, but the user specified a FFT of 256, the twiddle stride must be set to 4. The twiddle table consists of the values along the complex unit-circle, which are multiplied with the time function of the signal while calculating FFT.

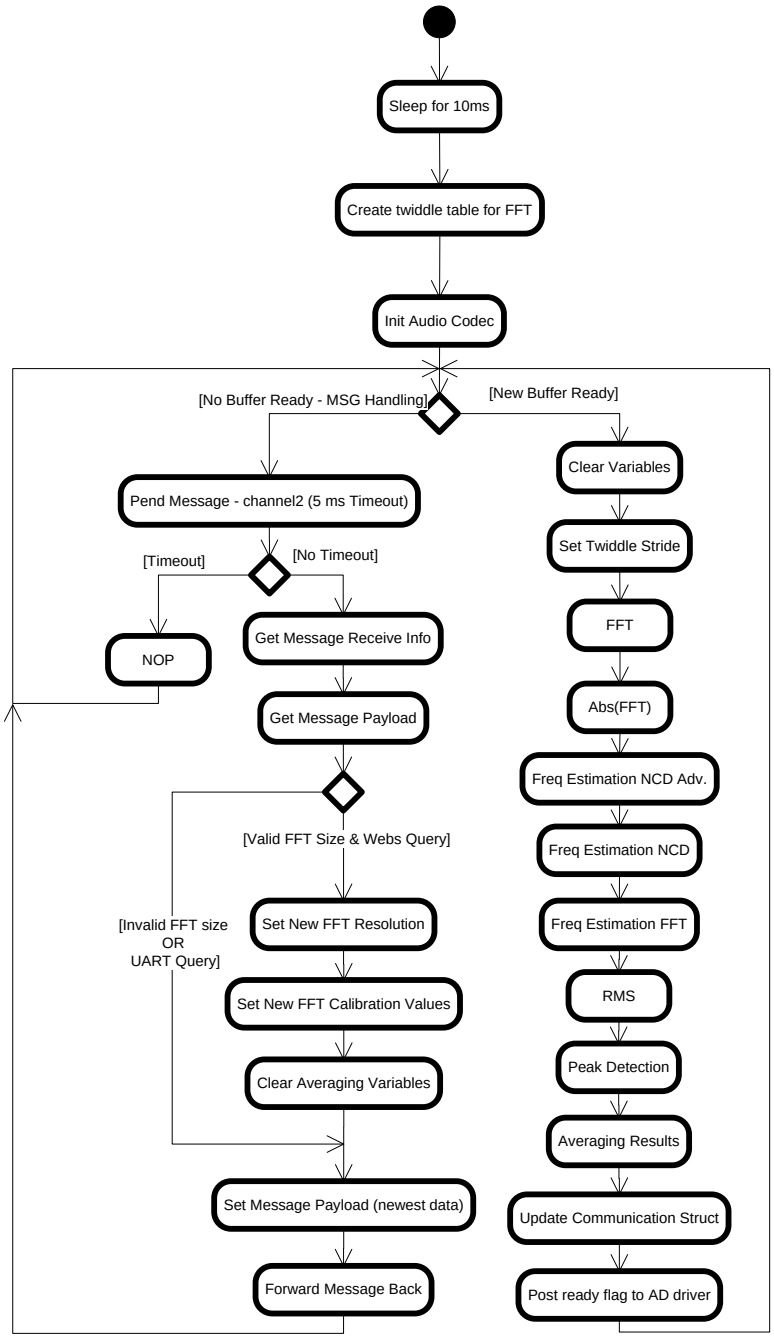

Figure 3.5: *Audio thread process flow*

a FLAG has to be posted to the AD driver to indicate that the signal processing has just finished.

The algorithms of different frequency estimation methods should be explained in detail:

- *DFT estimation (FFT):* The frequency is calculated from the index of the maximum value of the amplitude spectrum.
- *Simple null-cross detection (NCD):* This method measures the distances of the null-

crosses of the signal in integer index (signal period), averages them along the buffer, and calculates the frequency from it.

- *Interpolated null-cross detection (NCD Adv.):* The interpolated null-cross detection is almost the same as the simple null-cross detection, nevertheless it interpolates the null-crosses exact spot (in floating point context) with a linear curve, therefore a more accurate distance between them is estimated on each cross. Averaging is also done along the buffer.

No methods of these provide appropriate frequency estimation in case of a signal with non-harmonic components (such as a signal that contains 50 and 60 Hz components). Suggestions for solving this problem is explained in section 6.2.6. The accuracies of the estimations are expounded in section 5.5.

The pending action will not result in a timeout if the webserver thread sends a message to the audio thread. In this case, the No Timeout branch will be executed, in which the extraction of the configuration data (FFT size) from the webserver message is performed, then the local variables become updated, then the message is forwarded back to its original sender with the most recent data.

3.5.4 UART

The flowchart of the UART thread is shown in figure 3.6. The UART thread provides a diagnostic interface of the state of the system without the Ethernet link. Multiple commands can be sent on the serial line, each thread is allowed to be queried about its state by sending a character:

- *'A':* Queries the Audio thread, and sends back some signal parameters on the UART line.
- *'B':* Queries the Button thread, and sends back the current state of the buttons on the UART line.
- *'L':* Queries the LED thread, and sends back the current state of LEDs on the UART line.
- *'W':* Does nothing, just sends back a constant message (detection of possible fatal state of Kernel Panic occurrance).

The UART thread queries each thread the same way as the webserver does, altough two exceptions exist: the state of LEDs and the methods of the Audio thread are not allowed to be changed by UART console. The full and complete VDK device driver of the UART peripherial was given. It would be quick to implement the UART driver from scratch, but it was already written to the DSP.

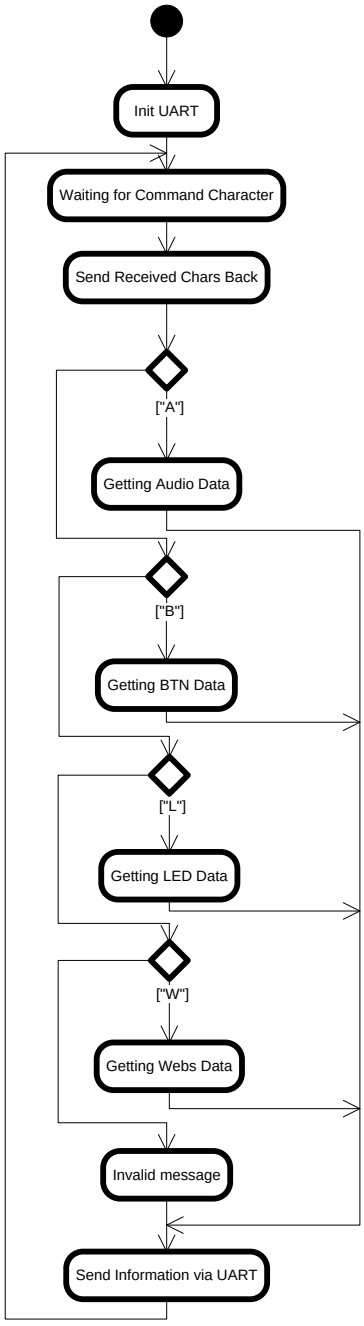

Figure 3.6: *UART thread process flow*

3.5.5 LEDs

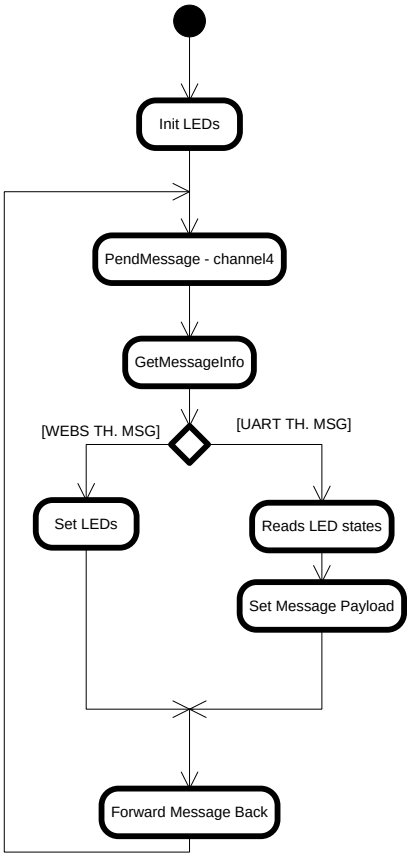

Figure 3.7: *LED thread process flow*

The flowchart of the LED handling thread is specified in figure 3.7. The structure of the LED handling thread is quite simple, it mainly pends on a message, and it sets the state of the LEDS based on the content of the message. As it is already mentioned in section 3.5.4, the reason of the two branches detached by the message sender is that it should not be possible to set any hardware configurations from multiple interfaces, only from the web browser.

3.5.6 Buttons

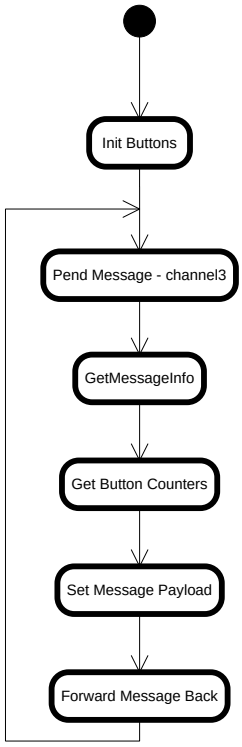

Figure 3.8: *Button thread process flow*

The flowchart of the button handle thread is shown in figure 3.8. The button handle thread is not very complicated either, it starts with an initialization of the driver. The driver performs the neccessary initialization steps of the buttons, furthermore it handles a timer interrupt, with which the push events are counted for both buttons. The driver realizes debouncing feature.

3.6 Client side design

3.6.1 Client webpage design

The design of the webpage of the client side was quite simple. It was mainly copied and set from different given templates, and it was modified following examples found at different explanatory websites [17]. There were no real design tasks for creating the webpage as it was given in a template accompanied with the CSS files, altough it has interesting structure described in section 4.3.1.

3.6.2 Script design

The design of scripts did not require any special design methods. A basic functionality is needed to detect the changes in HTML objects, and based on a timer interrupt the script has to send data periodically to the server with the freshest set of parameters, firstly the Parameter request, then the direct data visualization request. We can see the flowchart of the script in figure 3.9.

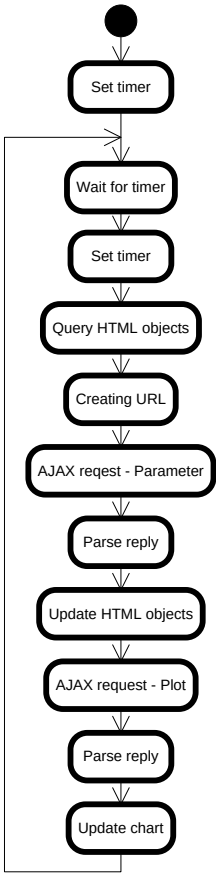

Figure 3.9: *Flowchart of the Javascript at the client*

3.7 Client - server communication process

In this section, the cooperation of the server and the client is summarized, while focusing on the client, to make the communication process clear. Requesting and receiving new data consists of the sequence below. The user can change any configuration parameter at any time. On the event of finished loading the webpage, a Javascript function is called to initialize a countdown timer to a user specified value (Data refresh period selection with a default value of 1 second). When the timer expires, a function is called which reinitializes the timer itself, in addition the following sequence is performed:

1. the Javascript reads all relevant configuration parameters from HTML objects,
2. the Javascript parses configuration parameters into a certain format (standard URL request with parameters),
3. the Javascript calls an AJAX request (parameter request) and the script blocks the running of parallel processes while the AJAX request has not finished,
4. when the answer has arrived, the Javascript extracts the data from it (LED states, BTN states, Audio signal parameters),
5. the Javascript refreshes all HTML objects with the latest data,
6. the Javascript sends another AJAX request (plot request), and the script blocks the running of parallel processes while the AJAX request has not finished,
7. when the answer has arrived, the Javascript reads and parses the spectrum data,
8. the Javascript updates the chart.

The server takes part in the two AJAX requests. The operation of the server is described in section 3.5.2.

Chapter 4

Implementation

In this chapter, the questions, guidelines and ideas related to the coding and implementation procedure of the project on both the server and the client side are discussed.

4.1 Guidelines

4.1.1 Reusable coding with layers

Resusability

The most important guideline along the coding procedure was the reusability of the code, which is a quite widespread attitude in system development. This view makes the coding procedure increasingly faster along time, because developers face very similar coding problems in significantly different systems. The functionalities that are already implemented should be reused in other projects as well, consequently this is going to decrease the time-to-market parameter of future projects.

Software layers

Code layers should be evolved, which have increasing abstraction level, thus the higher level layers give the possibility to reuse them, only the porting procedure of the lowest level is needed to the chosen processor. This results a more flexible, dynamic, forseeable and readable code at the cost of speed, because the different layers give the information to each other, and some more instructions get executed (function calls, returns, etc.) which are irrelevant along the context of the functionalities.

It is a known fact that not all codes can be written platform independently due to the specific system properties, architecture or development environment. These specific code fragments are considered as the lowest abstraction level layer of code, which should be adjusted to the system everytime.

All code that can be created in a higher abstraction level should be implemented platform independently, separated from lower-level layers. Sometimes code fragments cannot be separated into completely different layers. At this case, a systematic, well documented and commented format should be used, which is easily and quickly adjustable to any system.

If the coding procedure was done systematically and it was documented properly (see section 4.1.2), then joining the different level layers should not be a problem, but it is the responsibility of the developer oneself.

When some additional features are going to be implemented, the best thing that could be done is to modify or broaden the highest level layer that is possible to solve the problem on, while trying to avoid the modification of other layers.

4.1.2 Comments

Commenting in general

Commenting properly is one of the most important action while coding, comments are created not only for other people but for the developer oneself too. Creation of systematic and easily understandable comments are expected at software development companies. Software code is considered to exist, only if it is known how it works, how to use, maintain or improve it, all of which requires proper and up-to-date documentation.

Problems about commenting

Developers do not commonly like writing documentation or comments in their code, moreover they tend to put minimal effort in it to save time. Consequently the usage of documentation and comments of codes are not always adequate at all, altough it is essential for efficient system development. Another serious problem is a consequence of this, which is known as the code-comment-documentation inconsistency. The comments or the documentation often do not become updated, if additional features are added to or modified at a certain code.

Doxygen

A very useful tool called Doxygen [8] is available for creating automatic code-consistent documentation generation directly from the comments extracted from the code itself. There are some easy-to-learn commenting rules and conventions which must be used in order to use Doxygen. The developers just need to keep their code commented properly, afterwards the consistent, up-to-date and proper documentation becomes created by a some clicks with no real additional effort. Doxygen can generate PDF, RTF, HTML and LaTeX formats directly as output. Doxygen is available for several coding languages, altough the most important and widespread languages are supported (C/C++ in our case). Using Doxygen is a free and widespread solution for the common problems of commenting and documentation.

4.2 Embedded coding

4.2.1 Audio driver

Device drivers implement an interface to give the higher abstraction level software layers the possibility to use certain hardware resources by certain functions. Typical driver interface functions are the following:

- *Open the device*,
- *Read data from the device*,
- *Write data to the device*,
- *Close the device.*

Implementing an audio driver was required to make the audio codec operational. I have created a new audio driver based on a useful EZ-KIT example[1]. However, the example driver could not work simultaneously with the hardware and software configuration of the webserver by default. Hence, the example driver had to be rewritten and adjusted for the specific needs of the webserver application, because our measurement application does not require all of the features and modules (e.g. timers, DAC usage) that the default driver contains.

The modified audio driver consists of an interrupt handler and its initialization, in addition it contains the necessary initialization code of the audio codec. The driver fills a double buffer structure. One buffer is ready for higher level processing, while the other is becoming filled with audio data. The driver creates an interface for exterior functions to directly access the measured data via a continously updated pointer and a flag. The interrupt becomes triggered by the send request of the audio codec. The audio codec sends a packet of data, which is copied into a temporary buffer with Direct Memory Access (DMA). This packet contains two consequent measurement values from both LEFT and RIGHT channels. At the interrupt handler, we simply copy the data from the temporary buffer into the double signal buffer. The AD interrupt request causes an interrupt every $\frac{2\times1}{48000}$ s ≈ 42 μs. The occurrance of the interrupts could be rarefied by more advanced DMA utilization, but it was not neccessary, as the load caused by the frequent interrupts was low enough to cause no problems. Additional administration functions had to be implemented inside the driver:

- *Handle the double buffer structure:* This is implemented by a simple if-else branch.
- *Post the buffer switch event and the pointer to the latest buffer for higher level processes:* This is realized by a FLAG and a POINTER, the operations of which are described at section 3.5.3.

[1] $VDSP/Blackfin/Examples/ADSP-BF527 EZ-KIT Lite/Power_On_Self_Test/Parallel/Audio_test.c

There were two ways for the transmit of the initialization data for the audio codec:

- *Serial Peripheral Interface (SPI),*
- *Two Wire Interface (TWI).*

Finally, I used SPI, altough it has no significant advantage above TWI in this project. SPI seemed to be a little easier to implement than the other alternative.

4.2.2 LED driver

As the function of the LED handling thread is simple, it is easy to implement. There are three LEDs connected to three GPIO pins of the DSP. These pins must be configured to be GPIOs (not peripherial functions inside the DSP core) and outputs in order to use the LEDs. Afterwards, the states of the LEDs are able to be changed by writing 1 or 0 to the relevant bits of the GPIO control registers.

4.2.3 Button driver

The driver of the buttons is designed to detect each buttons push events by incrementing a counter. This feature requires bounce elimination. The measured bouncing frequency of the buttons is around 100 μs. In addition, a human being can push and release a button in 10 ms, therefore the event is present for at least 5 ms. The bounce elimination is realized with a timer interrupt with a period of 5 ms. The interrupt handler reads the state of the buttons, and detects their rising edge. The interrupt increments the timers in case of rising edge occurrance. The rising edge detection is implemented by comparing the previous state of the buttons with 0 and their current state with 1. A rising edge has just happened, if the two conditions are true at the same time. The counters are local static variables, which are able to be queried by a function that returns their values.

4.2.4 Communication between threads

The communication processes between threads are achieved by using VDK messages. VDK messages are kernel communication objects between threads. The advantage of VDK messages above other communication methods (globals, etc.) is that any kind of data structure is allowed to be sent by them safely. Therefore, they provide an unified interface for the data transfer between threads. The sending procedure is realized by casting a pointer to a data structure into a *void** pointer. The control flow arrows are realized by VDK messages between threads on the system schematic in figure 3.3.

4.3 Web programming

4.3.1 HTML

Firstly, it is a necessary convention that all HTML objects (selections, buttons, checkboxes, input fields) have to have unique IDs or name via which they are able to be referenced

for Javascript actions. The main part is a table, that uses *<iframe>* tag which embeds an additional HTML page (hereinafter inner-HTML). The inner-HTML document is divided into vertical slices with tag * *, this results in separate cells, where we can add different groups of cells of parameter configuration and visualization. Parameter cells contain tables, which have as many rows as many parameters we would like to visualize in the current cell. Parameter tables have two table cells in each row: [parameter_name parameter_value]. Four division groups (* * groups) are placed in the inner-HTML:

- *MEASUREMENT SETUP AND CONTROL*,
- *COUNTERS*,
- *NUMERIC RESULTS - SIGNAL ATTRIBUTES*,
- *Container for Highcharts.*

In figure 4.1, the structure of the implemented webpage can be seen.

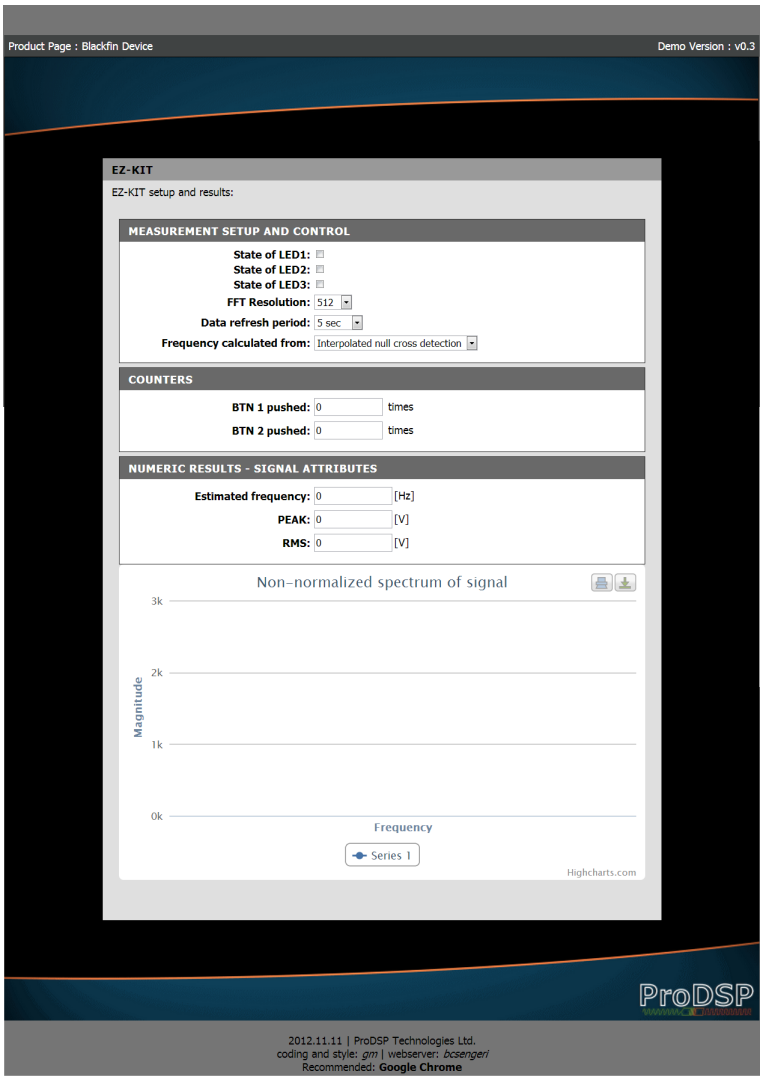

Figure 4.1: *The structure of the implemented webpage*

4.3.2 Javascript

The Javascript is a very high abstraction level language. We rarely have to implement own non-existent functions from scratch. I started searching for the functions that implement the blocks in the designed flowchart in figure 3.9 exactly. Default Javascript functions are found to be sufficient for basic the Javascript maintenance code.

In addition, jQuery and Highcharts additional external libaries are used according to section 3.4.

AJAX requests are implemented with the jQuery function *$.ajax(parameters.done(function())*, where parameters specify the requested URL, besides the function is executed on the arrival of the reply. For creating a valid Highcharts object, we need to run *new Highcharts.Chart();* constructor with a relevant parameter set which is able to be found at its documentation [12]. Updating the chart is implemented by using the *chart.series[0].setData(NEW_SET_OF_DATA); chart.redraw();* sequence.

Google Chrome supports Javascript debugging, which provides possibility to step through the code, while monitoring the actions. In addition, the function *console.log()* could be used for monitoring the behavior of the code in runtime.

4.4 Webcomp module

There is an executable file (webcomp.exe) provided with the source code of the GoAhead webserver. The executable file (webcomp.exe) converts the specified files into static standard C arrays of bytes before compile-time, which represent the files in binary form. These arrays can be included in the embedded project, and this feature allows to embed multiple files in the memory (RAM) for Ethernet service, without the use of a file system. The input of the webcomp module are the files to be compiled and a list that contains the names of the files. If a request is performed for a certain file, the webserver sends the elements of these arrays sequentially one to another, as they were a kind of byte stream.

The *files.dat* must specify the names and relative the paths of files correctly. An example is given for a filelist (named *files.dat*):

```
index.html
index2.html
router.css
style.css
/pictures/prodsp_header.gif
/pictures/prodsp_footer.gif
/scripts/jquery.js
/scripts/highcharts.js
```

The executable has to be run with these specific parameters from a command prompt:

```
webcomp.exe / files.dat > webrom.c
```

Afterwards, we must overwrite the webrom.c in the directory of the embedded project

with the one that we have just created. The content that we compiled become available for service after rebuilding the embedded project.

Chapter 5

Performance

In this chapter, the performance of the system is investigated, the questions related to the bottleneck and stability are discussed. In addition, the accurracy and range parameters of the signal measurement are described.

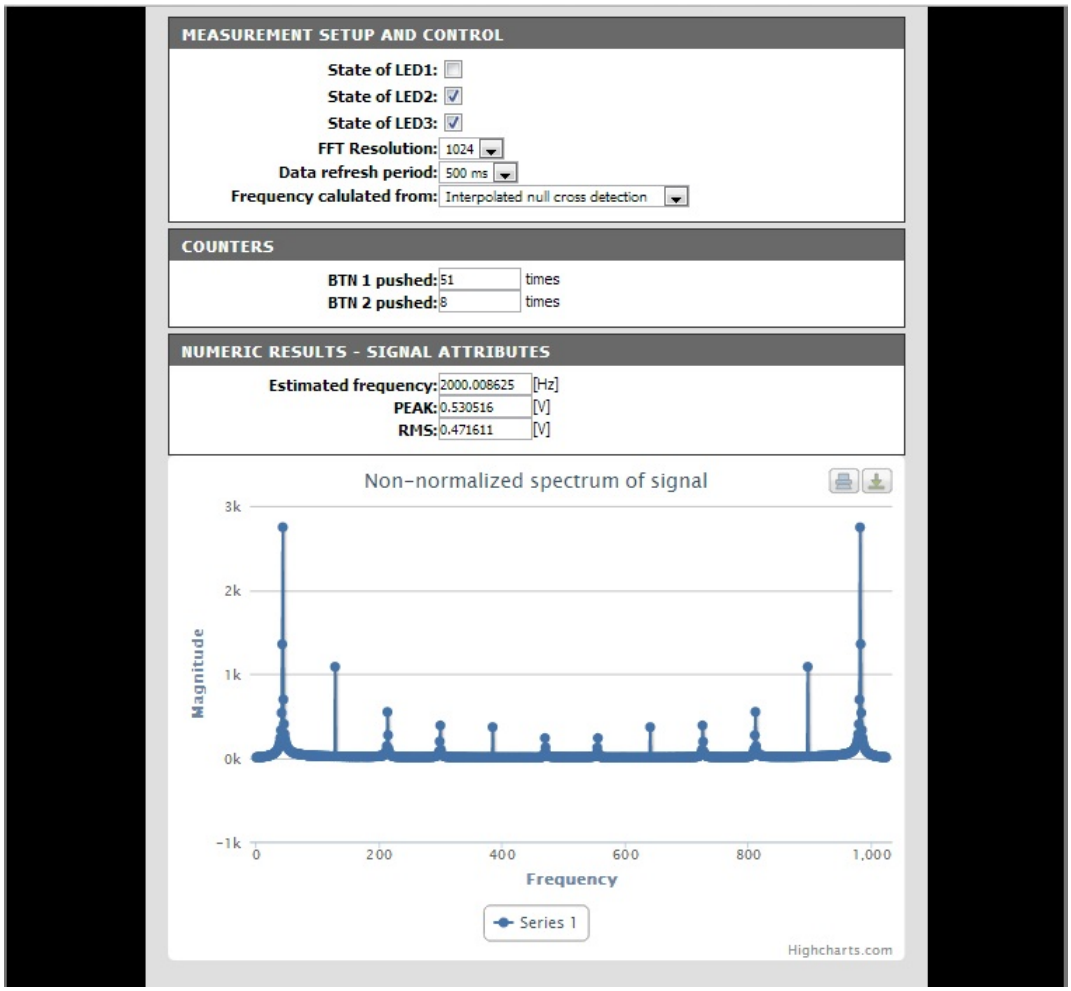

Figure 5.1: *Measurement on web interface*

In figure 5.1, we can see the final web interface while measuring a 2 kHz square wave signal created with a function generator. As we can see in the box *MEASUREMENT*

SETUP AND CONTROL, *LED1* is turned off, *LED2* and *LED3* is turned on. The size of the FFT calculation is set to 1024 and the refresh period of the Javascript refresh timer is set to 500 ms. Additionally, the frequency is calculated with the *Inperpolated Null Cross Detection* method. In the box *NUMERIC RESULTS - SIGNAL ATTRIBUTES*, the frequency (2000 Hz) of the signal, its peak (0.53 V) and RMS value (0.472 V) could be seen. In the box *COUNTERS*, we can see that the push event of *BTN 1* occured 51 times and *BTN 2* was pushed 8 times. In the box *Non-normalized spectrum of signal*, the amplitude spectrum of the square wave signal is shown.

5.1 Speed of the service

5.1.1 Client side

The performance of the implemented system proved to be far better than it was expected at the beginning of the project. Moving some data parsing methods off the server to the client side JavaScript contributed an additional increase in the performance as JavaScript engine runs on a high performance PC on which a lot more computing resources are available. However, the throughput of the client was significantly reduced after adding the Highcharts data plot module. This problem could be resolved by running the client application on a more recent PC configuration, or changing the plot application to a more lightweight one described in section 3.4.2.

The speed of the general Javascript code (not including the Highcharts) could not be measured relevantly, because it was so rapid, that other much slower processes (AJAX, Highcharts) dominated the response time of the system. However, flawless operation was observed with a refresh time of 30 ms before adding the Highcharts module. After adding Highcharts, a typical minimum refresh period was around 500 ms.

The implemented AJAX and Javascript functions have made to work above only Google Chrome yet. This incompatibility is probably caused by the slightly different structure of the Javascript engine in different browsers, and further research of the client side Javascript functions has to be initiated. It is possible that the problem is not in the Javascript itself, maybe the embedded application is not able to handle something, that other and more robust server-side CGIs are able to.

5.1.2 Server side

On the embedded application, the signal processing algorithm has to be started periodically. The algorithm is running for 5 ms typically. The complete application utilizes 20% of the total performance of the DSP. This 20% is dominated by the signal processing thread.

The Ethernet service of the application caused a negligibly small amount of the total load of the DSP (2%) under normal operation which was achieved by choosing a quite lightweight webserver and TCP/IP stack.

The embedded application was not able to handle the load by default, if I attempted to send continous requests. Consequently, a fatal state of KernelPanic occured. This was

probably caused by some kind of stack or heap error, which caused an unhandled exception in the kernel. This must be fixed at first anyway to reach safety critical stability. Three ways have became possible to fix this problem:

- User Management login, and request limit from the inside of the server core,
- handling kernel exception (maybe by reseting hardware, or reinitializing the whole webserver thread),
- decrease the reaction time (e.g. decreasing the tick period of the scheduler) of the application to serve the requests properly.

5.2 Bottleneck

The rapidity of the client side Javascript depends on the performance of the PC that it runs on, altough the PCs nowadays do not lack the computing power for handling this amount of load properly. The absolute bottleneck of the implemented system is the speed of the JavaScript engine of the client browser at normal operation. The Highcharts module dominated the slowness of the client side based on speed measurement tests.

The embedded application is not intended to be reachable from the open Internet, it can be placed in a local network where an unexpected request torrent will probably not occur. Therefore, the speed of the DSP was satisfying by far.

5.3 Stability

5.3.1 Client side stability

The client webpage was found to be very robust because of its simplicity. While restarting the embedded application, the client does not even need to be shut down or reloaded, unless the client webpage itself has been modified too.

5.3.2 Server side stability

The stability of the embedded server is currently not comparable to common webservers such as Apache. According to the deficiencies of the embedded server, this configuration is not suitable for safety-critical systems at this stage, but can be improved significantly by considering the suggestions given in chapter 6, however these do not affect the easy overloadability, hence this should be handled on another way. Moreover, it is possible that it is unsolvable by software and a custom and quite sophisticated solution should be invented in order to guarantee that kind of stability. However, it is a known fact that stable ordinary webservers can also be overloaded quite easily.

5.4 ADC voltage parameters

5.4.1 ADC voltage range

According to its documentation [5], the internal codec contains two (Left and Right channel) 16-bit wide Σ-Δ ADCs. The real ADC is 24-bit wide, but it is configured to use 16-bit word length, as the DSP supports 16-bit wide calculations in arithmetic level. Its datasheet states, that the maximal value of the ADC (32767) belongs to 1.4142 Vac.

5.4.2 ADC voltage calibration

ADC voltage calibration is realized by measuring and calculating the slope of the transfer characteristics of the ADC, afterwards we calculate its reciprocal (R). The final results could be calibrated by multiplicating the digital results with the calibration value of R.

According to the values mentioned in section 5.4.1, 1 LSB equals to $\frac{1.4142\ \mathrm{V}}{32767} \approx 43\ \mu\mathrm{V}$.

5.5 Frequency parameters

5.5.1 Frequency range

As audio channels are AC coupled and the sampling frequency is 48 kHz, therefore the theoretical frequency measurement range is 20 Hz - 24 kHz. However, the real frequency range of the measurement is 46.875-24000 Hz, as the AD buffer has a length of 1024 samples ($\frac{48000\ \mathrm{Hz}}{1024} = 46.875\ \mathrm{Hz}$).

5.5.2 Frequency measurement accuracy

As mentioned in section 3.5.3, I have implemented three different frequency estimation methods, all of which give estimations with different accuracies. The accuracy of Interpolated Null Cross Detection is the very best, altough it has quite big computation requirement as it uses floating point variables[1].

The accuracy of the DFT based frequency estimation is determined by the resolution of the DFT calculation. Its maximal error is $\frac{48000\ \mathrm{Hz}}{fft\_resolution}$.

The Simple Null Cross Detection uses integer distances between the positive null-crosses of the signal. Averaging is performed to a floating point variable if more periods of the signal fit in the AD buffer. Averaging increases the accuracy along the increase of the frequency. However, another effect decreases the accurracy at higher freqencies: the bigger the frequency is, the more relative error is present at the freqency estimation because of the integer nature of the time period. The measured typical relative accuracy was around 1000 ppm.

The accuracy of the Interpolated Null Cross Detection based estimation was found to be up to 10 ppm. This accuracy is comparable to the accuracy of a function generator I used.

[1] The Blackfin DSPs do not support floating point calculations in their basic aritmetic level.

Chapter 6

Future improvements

In this chapter, the additional improvement possibilities are discussed that arised during the implementation of the example application.

6.1 The potential of technology

The results at the final stage of this project verified that embedded systems are capable of using modern and rather advanced technologies via Ethernet. The state of the system is easily monitorable and configurable at unexpectedly low resource cost, though some hardware (Ethernet MAC) and software resources (existing ported TCP/IP stack) are essential in order to create the implementation. Based on the implemented features, any system parameter (hardware or software) became available to adjust and observe at once via a webpage. The web interface may be seen as an additional very high abstraction level regulation and observation layer at the top of an embedded system.

6.2 Further possibilities

6.2.1 EJscript investigation

GoAhead webserver has a module that is suitable for running an Embedded JavaScript (EJScript) interpreter, as mentioned at CGI description in section 2.2.2. EJScript has not been studied at this project, but it could be an additional improvement to the service procedure. It was considered to be slow beyond to use it appropriately for the first sight, but further and more accurate inspection should be done.

6.2.2 User authentication

Some kind of User Management feature should be added either done by using the file system based database module or done by custom only-from-memory procedure, if the application requires to provide safety critial stability. The user login data must be stored in a non-volatile memory such as EEPROM or flash, if no file system is available. The only-from-memory structure could to be specified, designed and implemented properly.

6.2.3 Configuration by several users

During the test and verification phase of the application, I experienced that the connection to the webserver from more clients raises some specification problems as hardware configuration is initiated from both clients. If two clients are present, and they set a state of a LED[1] differently, the LED will blink instead of staying still. There are more possible solutions for this problem:

- Priorities should be assigned for users, the hardware configuration must be declared valid by the user who has the highest priority.
- Only the login of one user must be allowed, every other attempt has to be rejected from the server core.

6.2.4 More recent GoAhead

I used an older revision of the server source code, it was considered to be proper for our intentions, but it has many bugs and problems, that has been solved already in a more recent revision. Therefore a newer version of the source code should be ported instead with some additional effort. The additional effort is required because the code architecture of the server changed along its revision history.

6.2.5 Jsocket

An alternative option was found to be available for data streaming at the beginning of the project. There is a Javascript libary available called JSocket [2], which is suitable for socket level communication directly from Javascript. The TCP/IP stack at the server side has the same functionality[2]. Consequently, a socket level communication method may be established between the server and the client side, which should result in a significantly faster data transmit channel than the implemented URL based commmunication link, altough this statement has not been confirmed yet.

6.2.6 Frequency estimation

An additional frequency estimation method could be added into the signal processing thread, which calculates the frequency of the signal directly from the definition of periodicity. This estimation method should be appriopriate to deal with complex signals that have more non-harmonic components. The suggested autocorrelation algorithm is the following:

1. Copy the first half of the audio buffer into another buffer, higher indexed elements should be zero.
2. Shift the content of the other buffer by one to the increasing indexes (inside the buffer)

[1] LED state changing is just a dummy feature for demonstration purposes, hardware configuration could have a far more dangerous purpose too, for example switching a current of a motor drive.

[2] lwip/sockets.h

3. Calculate its correlation (inner product) to the original signal.
4. Register whether the correlation was maximum so far, moreover if the number of shifts has reached the half of the length of the original buffer, go to step 6.
5. Go back to step 2.
6. Calculate the frequency from the index (discrete period) at the maximal correlation.

6.3 Porting to different hardware

The source code of the server consists of numerous standard C modules, all of which has been available and ported to ADSP-BF537 and ADSP-BF527 platforms already. The porting procedure is considered to be quick to similar Analog Devices DSPs, if VDK could be used. Porting the system to a significantly different controller core (e.g. Cortex) or above another kernel (e.g. uCLinux) would be much elaborate work due to the potential lack of any ported TCP/IP stacks, but the porting procedure is possible and deterministic.

Chapter 7

Summary

In this chapter, the work, the results and the benefits of working on this project are summarized.

I have achieved all the results that were desired, and even more:

- *a suitable webserver was found,*
- *the chosen webserver was ported,*
- *a sample measurement application was implemented,*
- *the parameters of the application are configurable,*
- *in addition, a graphic module for direct data (signal spectrum) visualization was added.*

Many additional improvements should be performed on the project to be a real usable industrial module. We can find some suggestions in section 6. However, the functionality is quite satisfying already.

During the implementation of a whole and complete embedded application by myself, I acquired much valuable experience about:

- *embedded application development,*
- *use of an embedded real-time operating sytem,*
- *driver development,*
- *some signal processing algorithms,*
- *calibration of a measurement utility,*
- *basic understanding of socket programming,*
- *using a certain webserver,*
- *webpage editing,*
- *Javascript programming,*

- *AJAX technique.*

This project with a few additional effort could take part in an industrial application, where a simple interface on a web browser is needed to configure and monitor a whole real-time control system.

List of Figures

Bibliography

[1] Adam Dunkels. lwIP. http://savannah.nongnu.org/projects/lwip/.

[2] aidamina. jSocket libary. http://code.google.com/p/jsocket/.

[3] Analog Devices, Inc. ADSP-BF527 EZ-KIT Lite Evaluation System Manual. "http://www.analog.com/".

[4] Analog Devices, Inc. ADSP-BF52x Blackfin Processor Hardware Reference. "http://www.analog.com/".

[5] Analog Devices, Inc. Blackfin Embedded Processor with Codec. "http://www.analog.com/".

[6] Analog Devices, Inc. Device Drivers and System Services Manual for Blackfin Processors. "http://www.analog.com/".

[7] Analog Devices Inc. VisualDSP++ 5.0 Kernel (VDK) User's Guide. "http://www.analog.com/".

[8] Dimitri van Heesch. Doxygen. http://www.stack.nl/ dimitri/doxygen/.

[9] Jon Duckett. *HTML and CSS: Design and Build Websites.* Wrox Press, 2011. ISBN-10: 1118008189.

[10] Embedthis Inc. GoAhead Web Server. http://embedthis.com/products/goahead/index.html.

[11] Dacid Flanagan. *JavaScript: The Definitive Guide: Activate Your Web Pages (Definitive Guides).* O'Reilly Media, 2011. ISBN-10: 0596805527.

[12] Highsoft Solutions AS. Highcharts. http://www.highcharts.com/.

[13] IOLA and Ole Laursen. Flot charts. http://www.flotcharts.org/.

[14] jQuery Foundation and other contributors. jQuery. http://www.jquery.com.

[15] jQuery Foundation and other contributors. jQuery License Terms. https://github.com/jquery/jquery/blob/master/MIT-LICENSE.txt.

[16] QuinStreet Inc. AJAX operation. http://www.developer.com/img/2005/08/Wei2.gif.

[17] Refsnes Data. Web programming resource. http://www.w3schools.com.

[18] Nicholas C. Zakas, Jeremy McPeak, and Joe Fawcett. *Professional Ajax, 2nd Edition.* Wrox Press, 2007. ISBN-10: 0470109491.