LUND UNIVERSITY

DEPARTMENT OF ELECTRICAL AND INFORMATION TECHNOLOGY

MASTER OF SCIENCE THESIS

Wi-Fi Based Remote Control of an Audio-Platform

A System Approach

Author:
Anders Skoog
Fredrik Stolt

Supervisor:
André Spånberg
Examiner:
Joachim Rodrigues

Lund 2013

©

The Department of Electrical and Information Technology
Lund University
Box 118, S-221 00 LUND
SWEDEN

This thesis is set in Computer Modern 12pt,
with the LaTeX Documentation System

Printed in E-huset Lund, Sweden.
june. 2013

Abstract

Wi-Fi connectivity is becoming present in more and more places and devices. This opens up for new and innovative ways to use Wi-Fi technology compared to the Personal Computer (PC) to PC network it was first intended for.

In this thesis a solution for a Wi-Fi extension module together with software for using the extension module to remote control an audio platform developed by Böhmer Audio has been researched and constructed. The Wi-Fi expansion module solution was developed by researching what Wi-Fi chip solutions were available on the market and then choosing one solution that is optimised for low price, ease of use and bandwidth. A printed circuit board (PCB) for the TiWi-SL WiFi chip that was chosen has been developed and constructed. The software part of the thesis consist of low level serial peripheral interface (SPI) drivers developed in C code for an AVR microcontroller used in the communication with the WiFi extension module. To remote control the Audio platform, server and client software have also been implemented. The server side software consists of a HyperText Transfer Protocol (HTTP) server programmed in C code which lets the client access a web interface written in HyperText Markup Language (HTML) and Javascript that controls the audio platform. The web interface layout is automatically generated by a script written in Python that parses an Extensible Markup Language (XML) file provided by Böhmer Audio.

Acknowledgement

We would like to thank our thesis supervisor, André Spånberg, for all the help he has given during the thesis project and Böhmer Audio for letting us do the thesis at their company. We would also like to thank everyone else working at Böhmer Audio and Aditus Science for making us feel welcome and part of the team.

We would like to thank Joachim Rodrigues for his help on how to write a good report.

Finally, we want to especially thank our family and friends for supporting us during the five years of education at Lunds Tekniska Högskola.

Anders Skoog & Fredrik Stolt
Lund, May, 2013

Contents

List of Tables

List of Figures

List of Acronyms

AP	Access Point
API	Application Programming Interface
CAD	Computer Assisted Drawing
CSMA/CA	Carrier Sense Multiple Access/Collision Avoidance
CS	Chip Select
CSS	Cascading Style Sheets
CPU	Central processing unit
DHCP	Dynamic Host Configuration Protocol
DMA	Direct Memory Access
DOM	Document Object Model
DSP	Digital Signal Processing/Processor
DSSS	Direct-Sequence Spread Spectrum
EEPROM	Electrically Erasable Programmable Read-Only Memory
FEM	Front End Module
GPIO	General Purpose In Out
GPU	Graphic processing Unit
Hi-Fi	High Fidelity

HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
I2C	I-squared-C Protocol
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
IO	In Out
IRQ	Interrupt Request
LED	Light Emitting Diode
Mac	Macintosh
MISO	Master In Slave Out
MOSI	Master Out Slave In
NFC	Near Field Communication
OEM	Original Equipment Manufacturer
OFDM	Orthogonal frequency-division multiplexing
OS	Operating System
PC	Personal Computer
PCB	Printed Circuit Board
RAM	Random Access Memory
RF	Radio Frequency
RISC	Reduced instruction set computing
SCLK	Serial Clock

SMA	SubMiniature version A
SMD	Surface Mount Device
SPI	Serial Peripheral Interface
SRAM	Static random-access memory
SS	Slave Select
SSID	Service Set Identification
TCP	Transmission Control Protocol
UART	Universal Asynchronous Receiver/Transmitter
UDP	User Datagram Protocol
WEP	Wired Equivalent Privacy
Wi-Fi	see WLAN
WLAN	Wireless Local Area Network
WPA	Wi-Fi Protected Access
WPA2	Wi-Fi Protected Access 2
XML	Extensible Markup Language

Chapter 1

Introduction

In todays world people expect to be able to do everything with their smartphone or tablet which is always within arm's reach. Due to this it is a big advantage for a product to have an easily accessible interface that works with such devices. Smartphones and tablets have a number of different ways to communicate wirelessly with other devices, some of the most common technologies are: Bluetooth, Near Field Communication (NFC), cell phone network and Wi-Fi. Of these technologies, Wi-Fi is the one present in almost all tablet and smartphones currently on the market(2013). Wi-Fi also makes it easy to communicate with desktop and laptop computers connected to the same network as the Wi-Fi enabled product.

In a larger perspective more and more devices get the ability for network connectivity which opens many new possibilities. There is a lot of talk about the internet of things to describe this phenomena where people no longer have only one special device for connectivity but have it conveniently built into most of the devices around them instead. It is in this light Ericsson [1] have predicted that by 2020 over 50 billion devices will be available online which can only be achieved if people have more than one device.

This creates a demand for inexpensive and easy to use solutions for adding wireless connectivity to products. Currently there is a high competition in many consumer electronics with many products seen as commodities. This leads to price sensitive customers so the solution has to be cheap and also simple to keep the development cost low.

Finding such a solution is at the core of this project. The solution has to be cheap and have a reasonable complexity. It should also be able to integrate with most embedded applications meaning it cannot take up too many resources. Although here we use it for remote control, the application could be anything from a sensor sending statistics to a server as well as streaming audio. The techniques used mostly stay the same, for example here we do not connect to the internet but still uses the Transmission Control Protocol (TCP) and Internet Protocol (IP).

The project have been done at the company Böhmer Audio that develops a digital platform for Hi-Fi audio applications that includes digital signal processing (DSP). For a simpler way to change various settings of the audio platform such as volume and input source they want to add the capability to connect over Wi-Fi for remote control. We have made a prototype by choosing one chip among many available on the market and developed the software needed to get the remote control to work, the prototype has been verified and works.

1.1 Thesis Outline

The remainder of this chapter gives an overview of the whole system to have in mind when reading the rest of the report.

Chapter 2 presents relevant theory for the project, including theory about different communications protocols but also about antennas and cross platform development.

In chapter 3 the hardware components used are described.

Chapter 4 describes how the hardware is connected and how the PCB is designed.

Chapter 5 describes the software, both the low level part handling communication with the hardware and the higher level handling the web server.

Finally, chapter 6 starts by listing some specific problems and challenges during the project and after that the ending comments and conclusion is presented.

1.2 Overview

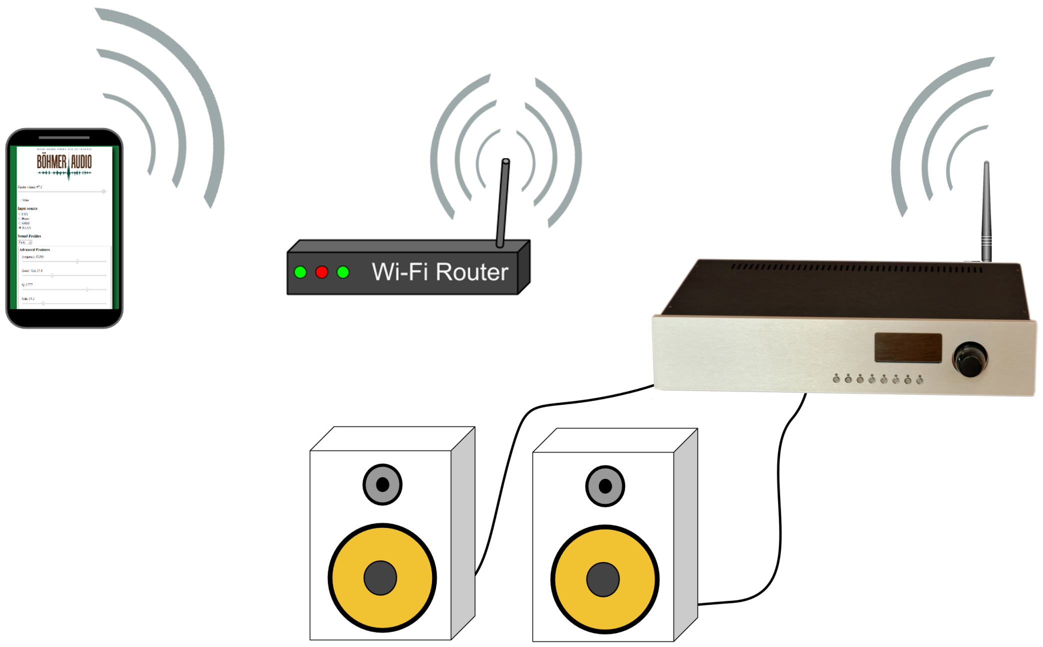


Figure 1.1: High level overview of the Wi-Fi remote control system that has been developed in this project and how it is used in the end product.

Figure 1.1 shows a high level overview of the Wi-Fi remote control system developed in this project. The figure illustrates how the communication goes from the device, i.e. a smartphone, via the Wi-Fi router and finally to the audio platform connected to the speakers. Both the device and the audio platform thus have to be connected to the same network provided by the Wi-Fi router to send messages between them and enable the remote control features. In this report the audio platform will be referred to as the server and the device as the client.

In Figure 1.1 the device and the router are already existing hardware, so is the Hi-Fi system except the part of it that handles the network communication. Figure 1.2 shows the inside of the Hi-Fi system with the part added in this project labeled as implemented hardware. For the Wi-Fi extension module

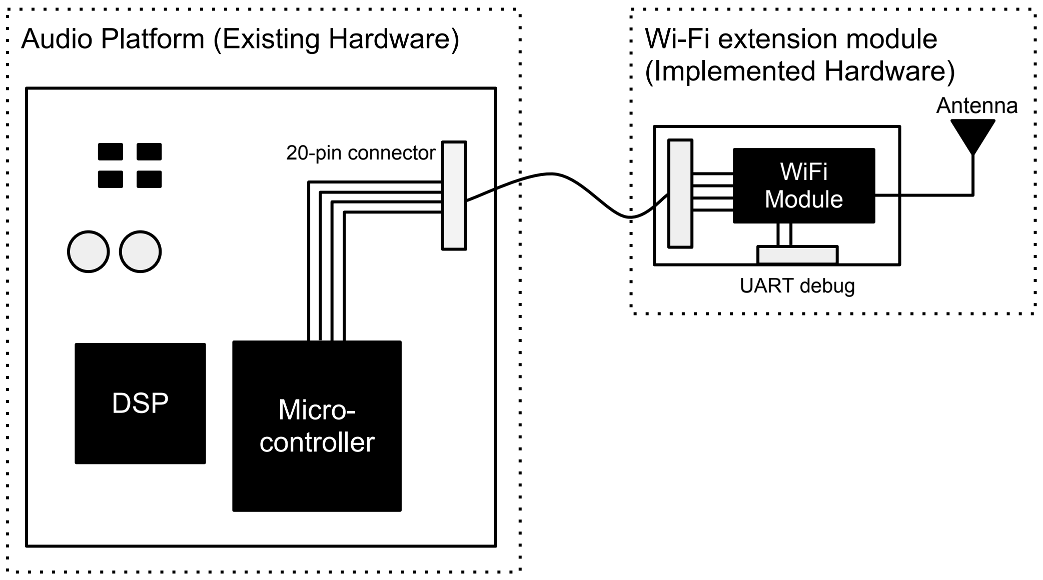


Figure 1.2: Low level overview showing how the implemented hardware fits in the system.

there is the constraint of needing to use the existing 20-pin connector and the selection of communication protocols that are available on the pins. The final design consists of a Wi-Fi module from LS Research which communicates through a modified version of SPI with the microcontroller on the main board.

The module handles receiving and transmitting of the wireless data as well as the network stack. It also handles connecting to the network, connection information is saved in the internal non volatile memory. The microcontroller handles the communication with the module and runs a server that clients can connect to and remote control through. The microcontroller also handles the rest of the audio system including coding and decoding audio data, handling USB and controlling the DSP. In order to handle all the tasks efficiently the microcontroller runs a real time operating system. Compared to the other tasks running the Wi-Fi solution have a low priority which means it can not use to much resources on the microcontroller.

1.3 Requirements

The overall requirement of the project is to provide a complete solution for wireless communication. That includes choosing a suitable chip, doing a PCB layout and finding a solution for sending control commands to the audio-platform that works on all common platforms such as smart phones, tablets and PCs. More detailed requirements are listed below.

- Find an off the shelf Wi-Fi-chip solution
- Design and manufacture a PCB for the Wi-Fi chip
- Create a cross platform remote control interface solution
- Implement drivers and control software for the Wi-Fi solution on the main microcontroller
- Implement a server solution for the remote control interface
- Make a remote control interface that is configurable via XML files
- Integrate software into the existing Böhmer Audio HiFi platform

Chapter 2

Theory

In this chapter the background theory needed for the rest of the report is presented.

2.1 Wi-Fi

Wi-Fi is a technology for wireless networks. Wi-Fi uses the different IEEE 802.11 standards and although these two are often used interchangeably Wi-Fi is a certification given by the Wi-Fi alliance organization after testing that the device is compatible with the standard. The standard uses Carrier Sense Multiple Access/Collision Avoidance (CSMA/CA) to allow multiple access by different clients. To avoid collisions the sender sends a *Request To Send* frame and receives a *Clear To Send* frame in a handshake operation. The wireless network can either use an Access Point or work in Ad-Hoc mode. An access point, often a wireless router, is a central node controlling the traffic while Ad-Hoc is a direct connection between two devices. Every network have an SSID standing for Service Set IDentifier for identification. In the standard are also different encryptions, first there was Wired Equivalent Privacy (WEP) followed by Wi-Fi Protected Access (WPA) and later WPA2. WEP and WPA is not considered safe so all new devices come with support for WPA2.

The frequency bands used are around 2.4 GHz or 5 GHz. These frequencies can be used freely without any license as long as the emission power is not too high. The first version of the standard were limited to data rates of 1-2 Mbps and is not used anymore because several new standards have made it obsolete. 802.11b uses the 2.4 GHz band and supports up to 11 Mbps. 802.11g uses the same frequency but supports speeds up to 54 Mbps. The difference being that 802.11g uses the more complicated Orthogonal Frequency-Division Multiplexing (OFDM) as modulation instead of Direct-Sequence Spread Spectrum (DSSS) used in 802.11b but they are backward compatible. 802.11a came at the same time as 802.11b using the 5 GHz band and supporting up to 54 Mbps but was not as widely accepted as the b/g standards. The most recent standard on the market is 802.11n that can use both 2.4 GHz and 5 GHz and multiple antennas and streams to achieve 600 Mbps. 802.11n is backwards compatible with 802.11b/g and also have almost twice the reach compared to the older standards. Note that all speeds here are theoretical and only obtained in best case scenarios.[2]

2.2 Network Protocols

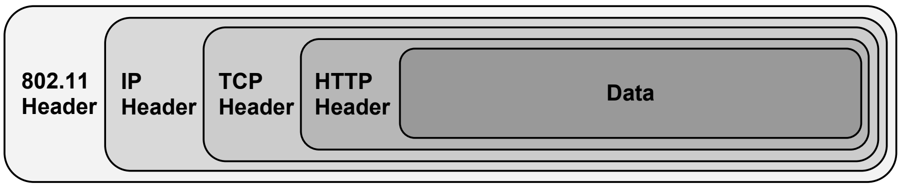


Figure 2.1: Data encapsulation when sending over network

Apart from the IEEE 802.11 standard some higher level network protocols are needed for real world applications. Each protocol add its own headers to the message resulting in some overhead over the useful data. This is called encapsulation and an example when sending data over a wireless network is shown in Figure 2.1. The figure shows how headers are added in turn to the data, starting with the HTTP header and ending with the IEEE 802.11 header, with the header from the higher level protocol being encapsulated as data for the lower level protocol. The protocols are briefly described in this section.

The Internet Protocol (IP) transmits blocks of data called datagrams in this case over the wireless connection but over wired connections as well such as ethernet. It uses IP-addresses to move the data packages over networks to their destination. The protocol have the ability to fragment a datagram into smaller packets so it can be transported through all networks. IP is not a reliable protocol meaning that packets can be lost but it includes a checksum to detect errors in the transmission.[3]

Another layer is the the Transmission Control Protocol (TCP) which is a connection oriented protocol meaning it has some status information including how much has been sent etc. As well as it uses an IP-address it also uses a port number to address the receiver, these two combined is called a socket address and the connection is called a socket. TCP is a reliable protocol and ensures that everything is sent correctly. To achieve this it uses acknowledgements and trigger retransmissions if they are not received or the package is damaged. Another simpler protocol is the User Datagram Protocol (UDP) that uses a socket address and just sends messages over IP without any control mechanisms meaning it is as unreliable as the IP protocol is. On the upside it is has higher throughput.[4][5]

HyperText Transfer Protocol (HTTP) is the protocol that is mainly used for internet web pages. It is a request/response protocol with messages sent as text. The request contains among other things protocol version, request modifiers as well as the data. The response is similar but adds a server status line in the beginning of the message. HTTP is sent over TCP/IP using port number 80.[6]

Dynamic Host Configuration Protocol (DHCP) is a protocol used for delivering configuration parameters. It is a client-server protocol and uses UDP to deliver messages. DHCP is useful for allocating an IP address to the client, either the client requests dynamic allocation and gets an available IP-address or it can also try to request a specific static IP address. With dynamic allocation the IP address is only valid for a certain amount of time but the DHCP server will as far as possible allocate the same address as before.[7]

2.3 SPI

Serial Peripheral Interface Bus more commonly know as SPI is a master slave protocol. As the name implies it is a serial interface and it has its own clock line, driven by the master, making it easy to synchronize between slave and

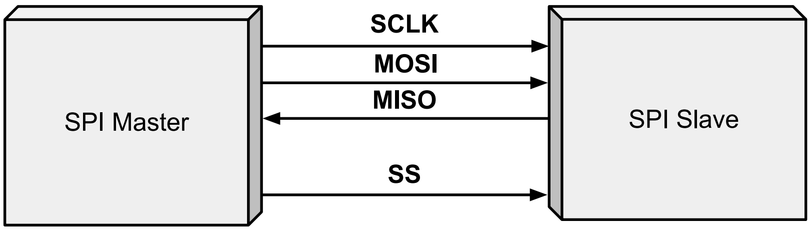


Figure 2.2: SPI signals

master. It is also full duplex and relatively simple to implement with high throughput. SPI is an industry standard way for serial communication but it is not a formal standard so there exists different flavors of SPI.

SPI consists of four signals seen in Figure 2.2 needed for communication to one slave. SCLK, Serial Clock, is a clock signal for synchronization. MOSI and MISO standing for Master Output Slave Input and Master Input Slave Output are for the data and SS, Slave Select to select the chip/slave. Another name for SS is CS standing for Chip Select. SPI also has the ability to share one bus with multiple slaves, this is done by a separate SS line for each slave unit. By asserting the slave select line of the unit that the master wants to talk to it can share the same MOSI, MISO, SCLK between all the slaves.

Since SPI is not a formal standard it can be configured in different ways in respect to how the clock signal is related to the data signals. To configure this two variables called CPOL and CPHA are commonly used. CPOL is the polarity of the clock and controls the base value of the clock to 0 or 1. CPHA is the phase and controls if the data is sampled on the first or second clock edge also specified by values 0 or 1. Combined there are four modes which can be seen in Figure 2.3 that shows the transmission of one byte. For example with CPOL=0 and CPHA=0 the clock would be 0 when not transmitting and the data would be sampled on the rising edge of the clock, changing to CPHA=1 would mean the data would be sampled on the falling edge instead.[8]

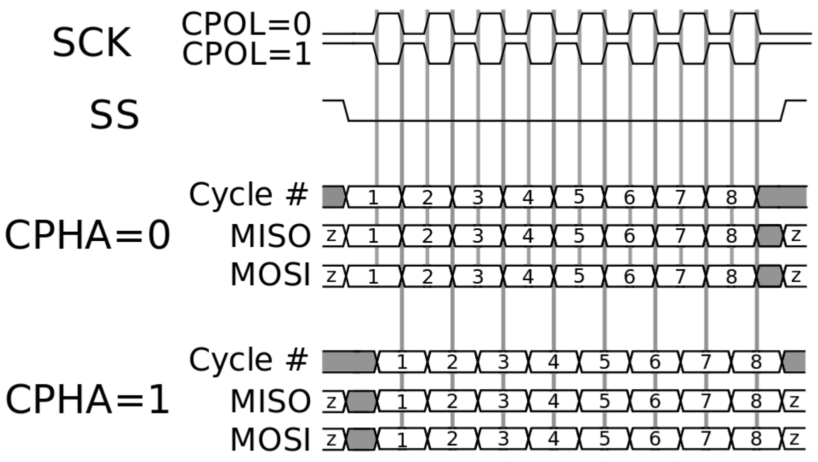


Figure 2.3: SPI timing with different settings for CPOL and CPHA for transmission of 8 bits. The vertical lines shows when the data lines are sampled in relation to the clock.[9]

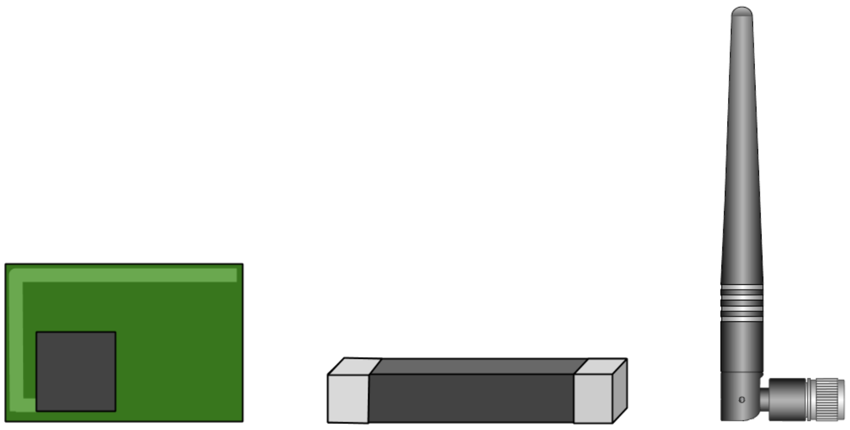


Figure 2.4: Different types of antennas: to the left is a PCB antenna, in the middle a chip antenna and to the right a half wave length antenna

2.4 Antenna Alternatives

For a wireless solution to work it will need an antenna to transmit and receive. Special care has to be taken when designing the antenna because it is critical for the wireless performance and because of the high frequencies involved. Here three possible approaches on how to design the antenna are described with their respective advantages and disadvantages.

The first possible solution is to add a trace in the PCB layout and adjust it to the frequency of the signal, in this case 2.4 GHz. An example how it can look is in Figure 2.4 to the left. Obviously this is a cheap solution since it only needs some extra PCB area. However it is also very sensitive to its surroundings. It has to be placed where there is no copper pour i.e. there can be no ground or power plane under it and it is sensitive to metal in its surroundings. Also small variations in the PCB manufacturing can detune it. To make the PCB antenna work it requires many iterations of the design and every time the PCB layout is changed the antenna may need to be tweaked, leading to increased development cost and time to market.[10]

Another solution is to use a ceramic chip antenna. Ceramic chip antennas are much smaller than trace antennas, they are just a few mm in each dimension compared to a trace antenna where a quarter wave length for 2.4 GHz is about 31 mm. It looks like a generic component as seen in the Figure 2.4 in the middle. Due to it being ceramic it is also less sensitive to nearby components and metal making it more forgiving to changes in the design. There is the need to add some extra components(inductors, capacitors) to match the impedance to the antenna but it still is an easier design than the antenna trace.[10]

The third solution is an external half wave length antenna as shown to the right in Figure 2.4. It connects to the board via a standard SMA(SubMiniature version A) connector. The design for this is the easiest, the only thing to think about is to keep the distance from the connector and the antenna output of the chip as short as possible[11]. It is also the most costly and bulky alternative but on the flip side it has the best performance.

2.5 Cross Platform Software

In this thesis cross platform software is defined as software that can be written once in one programming language and then be run on multiple operating systems, for example: Windows, Android, iOS and Linux. The cross platform solutions research in this thesis can be summarized in three different alternative approaches presented here.

The first solution is to write the software in one programming language and then use a compiler that has the ability to compile native code for each platform. Since the code is compiled into native code it has a speed advantage over other solutions, the disadvantage is that there has to be a compiler for each platform.

The next solution is to a use a scripting language and have interpreter software that is written for the different hardware platforms which runs the script code. This approach has the advantage of being able to have access to the native hardware functions such as the accelerometer and local memory depending on the interpreter software. The disadvantage is that it is slower and uses more resources than running native code.

The final solution is to not write an app but use HTML code together with javascript code as a normal website. The interface can then be loaded in the device web browser from a server. In a way it is similar to the previous solution only that the interpreter software is the web browser that is already included in the device. This has the advantage that it can run on any platform that has a web browser giving it a big advantage in the number of platforms supported. The disadvantages are that it is slower than the other solutions and that it can be hard to get access to native hardware functions.

Chapter 3

Hardware Components

This chapter introduces the main hardware components used in this project consisting of a Wi-Fi module, an antenna and a host microcontroller. The microcontroller is a part of the existing audio platform and thus not interchangeable while the Wi-Fi module and antenna had to be chosen and the criterias used for the choice are specified.

3.1 Wi-Fi Chip

The Wi-Fi chip is an off the shelf chip that handles the wireless communication. Since it will work with other devices it is important that it supports the standard described in section 2.1. First the requirements for the Wi-Fi chip is specified before the different alternatives are presented and then comes relevant information about the chosen chip.

3.1.1 Requirements

The requirements for the off the shelf Wi-Fi chip in this project are as follows:

- The chip should be an off the shelf Wi-Fi chip that is available at major electronic distributors.

- The chip has to have a minimum bandwidth of 1.4Mbit/s. The minimum bandwidth limitation comes from the fact that the Wi-Fi chip is planned to be used for audio streaming in the future, which according to equation (3.1) gives the minimum bandwidth of 1.4Mbit/s.
- The price should be as low as possible. The price of the Wi-Fi chip is important since it is going to be used in a consumer product. A fact that needs to be considered is that the end consumer might have to pay 5 to 10 times more for the component than the original purchase price due to development and manufacturing costs[12].
- The Wi-Fi chip uses one of the following communication standards: SPI, I2C (Inter-Integrated Circuit), or UART (Universal Asynchronous Receiver/Transmitter).
- The chip should have support for the different encryption schemes used by the standard. This can not be done on the microcontroller without using too much resources.

$$16bit \cdot 2channels \cdot 44100samples/s = 1.4Mbit/s \tag{3.1}$$

Some features were not considered as requirements but seen as beneficial:

- An integrated TCP/IP stack, would be useful since it frees up resources from the hosting microcontroller.
- Support for the newer IEEE 802.11n standard was considered beneficial not primarily because of the bandwidth increase but for the better range that the newer standard provides compared to the older 802.11b/g standards, see section 2.1.
- Example code and APIs, which speed up the development process.

3.1.2 Off the Shelf Wi-Fi Chip Alternatives

From the requirements, stated above in section 3.1.1, 13 off the shelf Wi-Fi chips were chosen and can be seen in Table 3.1. The table contains in addition to the name of the manufacturer and Wi-Fi chip four columns that contain the price, data rate, interface and comments about the Wi-Fi chips. The data rate column states the data rate for the Wi-Fi part of the communication while the data rates stated in the interface column are the speed of the communication protocol between the Wi-Fi chip and the microcontroller. The data rate stated in the interface column gives a more realistic view of the throughput of the Wi-Fi chip but could not be used to compare the different chips since only a handful of the chips stated the interface data rate in their data sheets.

From the Wi-Fi chips in the Table 3.1 the one called TiWi-SL was chosen for the project. Other than meeting the requirements for performance the TiWi-SL was chosen because of its extensive documentation and API. Another strong point of the TiWi-SL is that it is based on the same network controller as the Texas Instruments CC3000MOD so it could be replaced in the future. The reason the TiWi-SL was chosen over the the CC3000MOD, was the lack of availability at the suppliers when the thesis project began (February 2013).

3.1.3 TiWi-SL Wi-Fi-Module

The TiWi-SL is a Wi-Fi-module developed by LS Research. The module is based on Texas Instruments CC3000 SimpleLink^{TM} technology which means that all of the APIs (Application Programing Interface) that are provided by Texas Instrument for the CC3000 also work together with the TiWi-SL. A block diagram of the module can be seen in figure 3.1 and specifications are available in table 3.2.

The block named WLAN in Figure 3.1 is the part of the Wi-Fi module that uses the Texas Instrument CC3000 network controller. The CC3000 network controller has a CPU that runs the necessary operations to enable wireless access. It also has hardware for encryption and decryption of the different standards with support to handle WEP, WPA and WPA2. It handles all the network communication, such as the TCP/IP stack, radio baseband and SPI communication with the host microcontroller. Also included is a non volatile 32 KB EEPROM (Electrically Erasable Programmable Read-Only Memory) that stores information such as connection profiles, radio parameters and about 5 KB of the 32 KB is available for the user. A large part of the EEPROM is reserved for patching the internal software of the processor. The user has no access to the internal software but relies on the API that is supplied.

Table 3.1: Wi-Fi chip solutions considered

Name	Manufacture	Price	Data Rate	Interface	Comment
TiWi-SL [13]	LS Research	25 $	b/g, 54Mbps	SPI (16MHz)	detailed documentation
CC3000MOD [14]	Texas instruments	12.5 $	b/g, 54Mbps	SPI(16MHz)	detailed documentation, not available yet
WYSBCVGXA [15]	Taiyo Yuden	20 $	b/g/n, 150Mbps	SDIO	Bad documentation and support, no TCP/IP
RN171 [16]	roving networks	23 $	b/g, 54Mbps	SPI (2Mbps)	Easy to use
WF121-A [17]	Bluegiga Technologies	32 $	b/g/n, 72.2Mbps	I2C, SPI, UART (20Mbps)	No datasheet available
RS9110-N-11-24 [18]	Redpine	36 $	b/g/n, 65Mbps	SPI/SDIO	Good documentation
GS1011MEE-SMP [19]	GAINSPAN	43 $	b, 11 Mbps	SPI (3 Mbps)/ UART/ I2C	Easy to use
HDG104 [20]	H&D Wireless	47 $	b/g, 54Mbps	SPI	Recommended by atmel, no TCP/IP
HDG204 [21]	H&D Wireless	54 $	b/g, 54Mbps	SPI	Not available at suppliers, no TCP/IP
AR4100P [22]	Qualcomm Atheros	60 $	b/g/n	SPI	Expensive
SPB4100 [23]	H&D Wireless	79 $	b/g/n, 72Mbps	SPI	Expensive
AX22001 [24]	asix		a/b/g, 54Mbps	SPI, I2C, UART	Not available at suppliers, complete SoC
Ec32Sxx [25]	econais		b/g/n, 54Mbps	SPI (18Mbps), I2C	Not available at suppliers

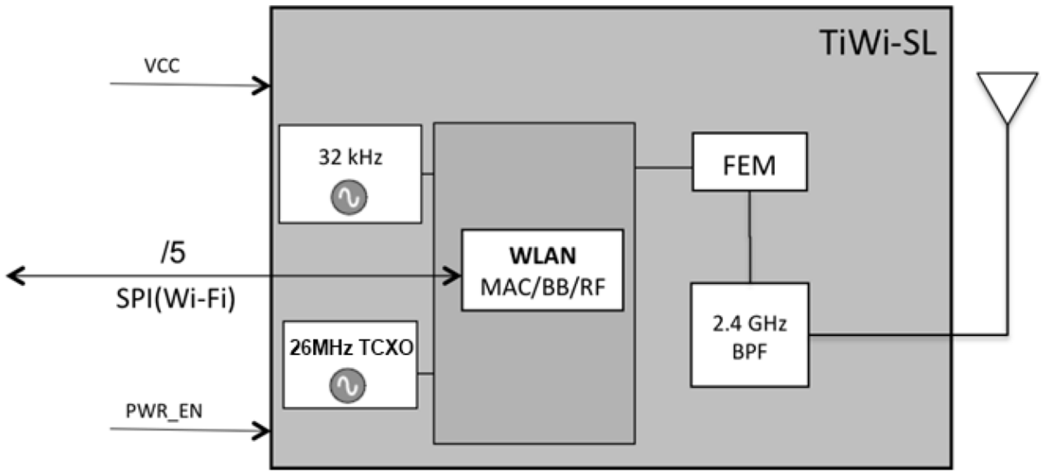


Figure 3.1: Block diagram of TiWi-SL [13]

Communication with the microcontroller is done via a modified version of the SPI protocol. The modification to the SPI protocol consist of an extra line called IRQ in addition to the normal signals, MISO, MOSI, SCLK, SS, for more information on SPI see section 2.3 and 5.1.1. On the Wi-Fi module there is an I2C interface for writing directly to the modules EEPROM but it is preferably done through commands over the SPI. For debug purposes of the internal software there is a 1.8 V UART port available to extract logs.

The FEM in Figure 3.1 block is the RF(Radio Frequency) Front End Module that takes care of the digital data coming from the baseband and converts it to an analog radio signal. After that the radio signal passes through a 2.4 GHz band pass filter indicated by the 2.4 GHz BPF block and finally the signal is transmitted through the antenna. In the block diagram there are also two crystal oscillators, one 26 MHz oscillator used by the WLAN block as its main clock source and a 32.768KHz used for the realtime clock on the chip.

3.2 Antenna

The antenna chosen is an external antenna that is sensitive to 2.4GHz. It is an omni-directional half wave-length antenna with 50 Ω impedance. It is

Table 3.2: Specifications for TiWi-SL Wi-Fi module

Standards Supported	IEEE 802.11 b/g
Host Interface	SPI
Vcc Min	2.9 Volts
Vcc Max	3.6 Volts
Temp Range	-40 to +85 C
Transmit Power	20.0 dBm, 11 Mbps, CCK (b), 16.9 dBm, 54 Mbps, OFDM (g)
Rx Sensitivity	-89 dBm, 8% PER, 11 Mbps, -76 dBm, 10% PER, 54 Mbps
Transmit Current	269 mA, CCK (b), 187 mA, OFDM (g)
Receive Current	92 mA, b/g
Power Down Current	<1 uA

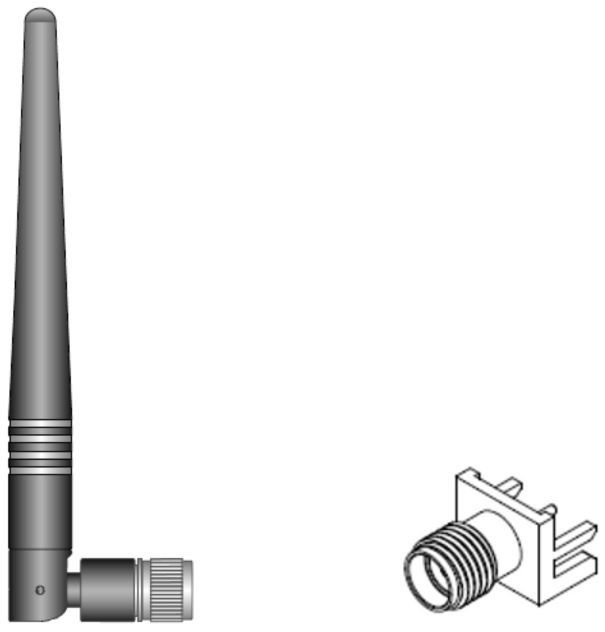


Figure 3.2: Sketches of the antenna and its SMA connector.

connected with a standard SMA connector. The used connector is fitted to the side of the board and soldered on both the top and the bottom of the PCB. Both connector and antenna can be seen in figure 3.2.

Different alternative of antennas are discussed in section 2.4. For this application there is the constraint that antenna will be located inside a metal casing meaning that the whole chip would be shielded hindering the wireless signal to reach it. That rules out the antenna trace and the chip antenna and leaves the external antenna as the only viable option.

3.3 Microcontroller

The microcontroller on the Böhmer audio control board is an AVR AT32UC3C0512C made by Atmel. It is a 32-bit RISC(Reduced Instruction Set Computing) architecture running at a clock speed of 60 MHz. For memory it has 64 kB of SRAM available with single-cycle access as well as 512 kB of flash RAM. It has a low latency interrupt controller with 4 different priority levels. There are support for a lot of interfaces including two modules for SPI which can handle 4 different slaves each. Also there is built in support for Direct Memory Access(DMA) operation. The peripheral DMA controller supports 16 channels and is compatible with the SPI modules.[26]

Chapter 4

PCB Design

A Printed Circuit Board (PCB) was designed to hold the Wi-Fi chip, antenna and some supporting components. Connection to the host microcontroller which is already located on the main control board of the audio platform is done through a 20-pin connector. How everything is connected was specified in a schematic. With help of the schematic a PCB layout was designed and later assembled. This chapter follows the same structure as described above.

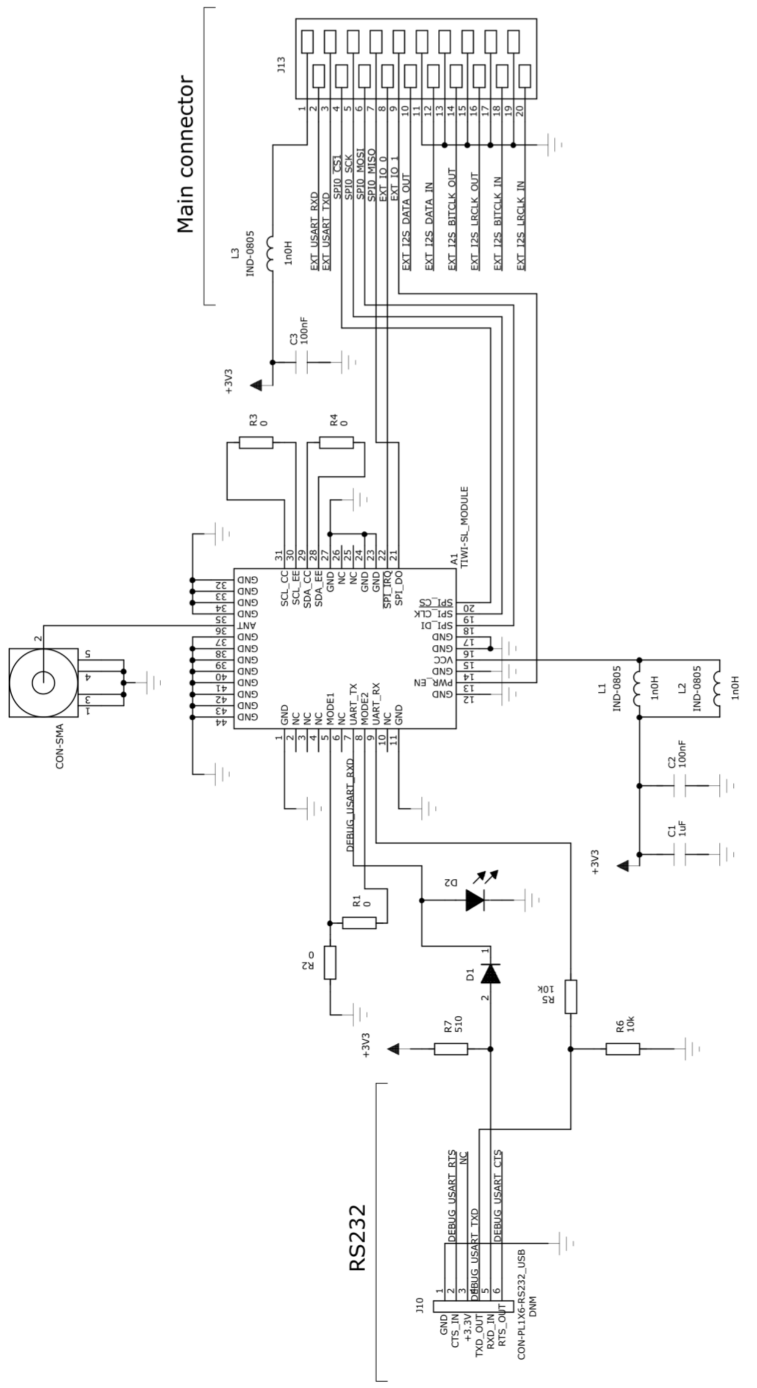


Figure 4.1: Schematic of the Wi-Fi extension module

4.1 Schematic

A complete schematic of the hardware can be seen in Figure 4.1. For component values and further details refer to table 4.1. The 20-pin connector is found to the right in the schematic and goes to the main control board and the microcontroller. It has all communications as well as power supply at 3.3 V. There is also a six pin connector used for debugging. Other than the main module there are some supporting components surrounding it described briefly here, all the names of the components refer to the schematic.

- Capacitors C1, C2 and C3, are used for decoupling of the power lines. C1, C2 are specifically for the Wi-Fi module and of different values to be effective at different frequency ranges.
- L1, L2 and L3, are ferrite beads also used at the power lines for filtering out high frequency noise.
- R1 and R2, are two jumpers that depending on which one is populated decides if the Wi-Fi module is in debug mode or normal mode.
- R3 and R4, are two jumpers used to short the I2C interface that is used for writing to the EEPROM. The I2C can be used to recover the module if it is bricked during a firmware update.
- R5 and R6, are two resistors in a voltage divider level shifting the debug signals.
- R7, a 510 Ohm pull up resistor used in the level shifting circuit.
- D1, a protection diode use in the level shifting circuit.
- D2, a red LED with a threshold voltage of 1.8V that is used to clamp the voltage in the level shifting circuit

Table 4.1: Bill of materials

Components	**Info**	**Package**	**Art. num farnell**
TiWI-SL		Custom	741-450-0067 (Mouser.com)
Antenna	2,4GHz	SMA	2143321
C1	1uF	1206	9227873
L1,L2,L3	0.2 Ohm	0805	1635706
C2,C3	100nF	0603	431989
R5,R6	10kOhm	0603	1653253
Antenna connector	SMA	SMA	1909295
LED	Red	0603	2112124
R1,R2,R3,R4	0 Ohm	0603	2131805
R7	510 Ohm	0603	1627741
Diode	Signal diode	0805	8150206
Main connector	SMD	20 Pin connector	1641873
Debug connector	Pin header	6 Pin connector	1654535

4.1.1 Level Shifting

In the schematic in Figure 4.1 the connector called J10 is used for debug communications. The debug cable available usees 3.3 V signals and the Wi-Fi module uses 1.8 V signals, this meant that some components were needed for shifting the levels of the signals. For this purpose two different circuits for doing the shifting in each direction were added [27].

On the the line going from 3.3 V output to 1.8 V input a voltage divider was used to reduce the 3.3 V to 1.7 V. The 1.8 V output to 3.3 V input side uses slightly more advanced circuitry and here the fact that a logic low does not have to be exactly zero to be interpreted as low is used. The circuit has three components, a pull up resistor connected to 3.3 V and a diode in series. The last component is a red LED to protect the input of the Wi-Fi module which has a threshold of 1.8 V which means that if the voltage is over 1.8 V the LED will light up and clamp the voltage. During high voltage there will be no current through the series diode and thus the pull up will give 3.3 V and during low voltage the diode will be fully open and give a diode threshold voltage as output.

4.2 PCB Layout

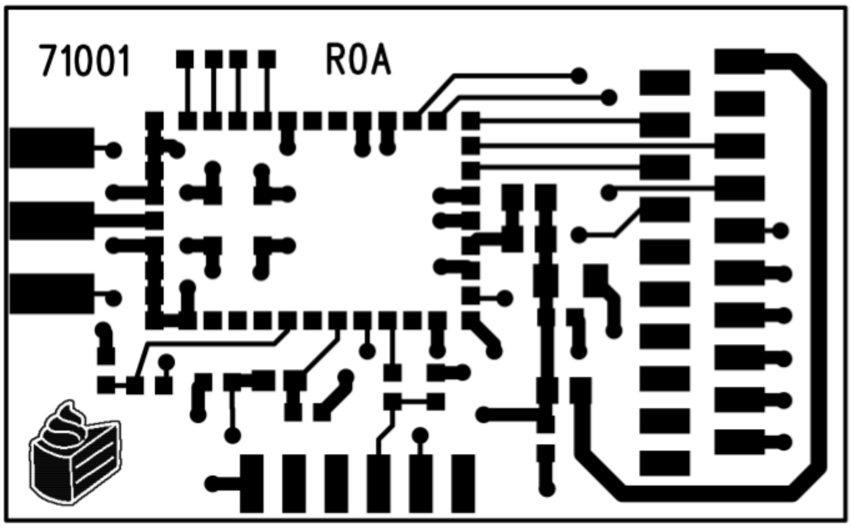


Figure 4.2: PCB Wi-Fi extension module top side

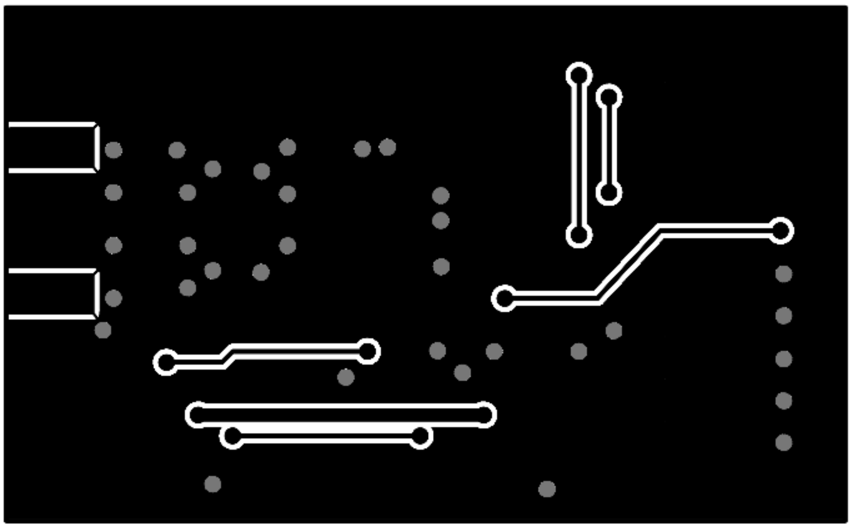


Figure 4.3: PCB Wi-Fi extension module bottom side

For the PCB layout a footprint of each component in the schematic was used. The footprint specifies how the pads of the components is to be drawn onto the PCB. Two terminal components like capacitors and resistors have standard footprints that are available in the library of the PCB Computer Assisted Drawing (CAD) software. For the the non-standard packages used by the Wi-Fi module and SMA connector, custom footprints had to be drawn in the CAD software. How the footprints needed to be drawn was found in the respective data sheet. PCB manufacturing have limited accuracy so to make sure that traces were not placed too close to each other suitable design rules were set to the CAD program.

The routing of the PCB was done manually. The antenna connector is placed at an edge of the board because of the mounting of the connector, see section 3.2. The module was placed close to the antenna connector to make the trace to the antenna short. A shorter trace between the antenna and module result in less parasitic inductance which is important because of the high RF-frequencies in the antenna signal. The components for the filtering of the power to the module are also put as close as possible to the Wi-Fi modules power pad to minimize parasitics. The rest of the components are placed close to the pads they would be connected to and in a way so as to make the routing simple. It was not possible to have the routing in only one layer so two were used, changing layers was done with vias. To avoid reflections of signals in the traces on the PCB, 90 degree angles were avoided in all traces when routing.

The PCB is designed as a two layer PCB although the manufacturer recommends a four layer PCB in the TIWI-SL Antenna Design Guide[11]. The choice to use two layers is based on the price and the ease of manufacturing of the PCB and due to the fact that only a few traces need to use the second layer. As much of the routing as possible is done on the top layer, shown in Figure 4.2. The unused area on the bottom layer is flooded with copper creating a partial ground plane. The traces on the bottom plane are drawn so that the partial ground plane is not divided in any significant way. No traces are drawn under the module itself as specified in the TIWI-SL Antenna Design Guide. The bottom layer is shown in Figure 4.3. To compensate for the loss of the power plane present in a four layer design the width of the power lines and power vias are increased.

Not only electrical considerations were made with the layout but also the heat dissipation was taken into account. Although the module is quite low power it produces some heat. The four pads connected to ground in the middle of the module as seen in the top layer are for thermal dissipation. As the ground plane is big it can handle the heat better which is one of the reason no traces were drawn under the module. The antenna connector pads on the bottom layer only have small connections in the corner to the ground plane. That is so that they can be soldered on, if the connection were larger the heat needed for melting the solder would dissipate in the ground plane.

On the top of the PCB there is a picture of a piece of cake, this was added to improve the aesthetic of the PCB.

4.3 Assembling the PCB

All the components used for the PCB are SMD (Surface-Mount Device) components. The choice of using SMD components is in part that they can be made much smaller than through hole components and the fact that the process of populating the PCB can be automated with the help of a so called pick-and-place machine. A pick-and-place machine is just as the name suggests a machine that after having been configured picks up a component and places it on the correct pads on the PCB. When the components have been placed the components need to be soldered, this is usually done either by wave soldering where the PCB is passed under a wave of molten solder or with reflow soldering where the PCB with its components are baked in a oven to let the solder paste on the pads reflow.

In this project only two prototype PCB were assembled, this means that it is not time or cost effective to use the manufacturing process described above, instead the PCB was assembled by hand.

Chapter 5

Software

Figure 5.1 shows an overview of the different software blocks that have been developed in this project. The dark gray blocks are APIs provided by Texas Instruments that handle the Wi-Fi connection configuration and socket networking, described in section 5.3.1. In these block only minor changes had to be made in order to get the APIs to compile with the GCC compiler used. The code developed can be found in the appendix A.

The SPI driver is the part of the software that control the lowest level functions of the communication between the microcontroller and the Wi-Fi module. The SPI driver also control the power enable signal of the Wi-Fi module. The SPI driver is described in section 5.1.

The block called HTTP Server and Wi-Fi Control interface contains the software for the HTTP server that is used to get access to the remote control functions from an external device. The HTTP server is described in section 5.3.2. The block also contains the commands that are used to control how the Wi-Fi module connects to the network and which network it chooses. The lower box called the Wi-Fi thread is the part that is run on the microcontroller connected to the Wi-Fi module. As the name implies it runs as a thread on the real time OS of the microcontroller with a low priority.

The interface builder generates the web page used by the HTTP server and only has to be executed once. It runs on a PC and takes the design files and and combines them with the help of a Python script, described in section 5.2.2.

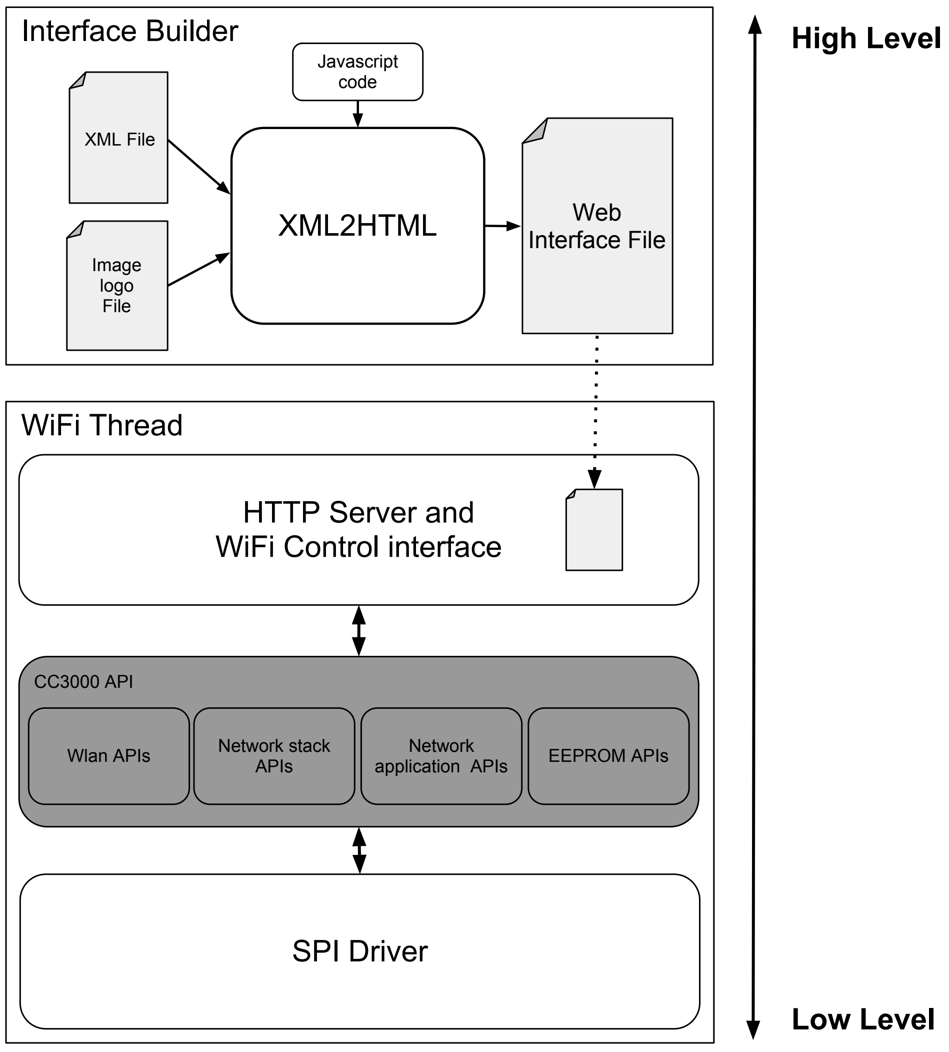


Figure 5.1: Overview of the software structure. White blocks have been implemented in the project, dark gray blocks are APIs from Texas Instruments and light gray pages are in- and output files used by the different software blocks

5.1 SPI Driver

In order to enable communication between the Wi-Fi module and the AVR microcontroller drivers for the SPI communication were implemented. The following subsections describe the extended SPI protocol used and how it was implemented on the microcontroller.

5.1.1 Extended SPI

The module communicates over SPI but with some modifications as mentioned earlier in section 3.1.3. The module operates in slave mode and the microcontroller in master mode but the module also needs to initiate communication with the microcontroller. The solution is an extra signal named IRQ that is an interrupt line that is pulled low by the module when it is ready or has something of interest to send to the microcontroller. How the IRQ works is described here:

- Startup: At startup the Wi-Fi module signals with the IRQ signal when it is ready to receive the first write operation over the SPI from the microcontroller. The first write operation is slightly different from normal in that it requires a pause after sending the first 4 bytes for 50 μs before continuing the transmission.

- Writing data to module: For writing to the Wi-Fi module other than the first write the IRQ is used to signal that the module is ready. In normal SPI after asserting the chip select it will start with the transmission directly. However the module may not be ready and therefore first has to reply that it is ready by asserting the IRQ line before the transmission from the microcontroller is allowed to start. Therefore the pattern when writing is, the CS line goes low, the module acknowledges by setting the IRQ line low and then the data is sent from the microcontroller.

- Reading data from module: Because the module works in slave mode there would be no way for it to signal when it has data to transmit with the normal SPI protocol. Instead of having the master microcontroller polling the Wi-Fi chip to see if there is new data the Wi-Fi module sets the IRQ signal low alerting the master that there is new data to read.

- Ending transmission: IRQ remains low during each transmission and goes high when it is done, thus alerting that it is ready for a new transmission.

Because the module operates in slave mode it can not do a transmission to the microcontroller on its own, the master has to send as many bytes as it gets during receiving to toggle the clock line. The first byte the microcontroller sends while receiving data is specified to be 0x03 to signal that the microcontroller wants to read data. The rest of the bytes sent to the Wi-Fi module are just dummy bytes and are set to zero.

The extended SPI version has one potential flaw where the master could misinterpret the IRQ signal if the slave tries to write something at the same time as the master tries to write. If the master starts a write operation it first make sure that the IRQ is high before asserting CS. If the module at the same time has something to transmit it can assert the IRQ before it notices the CS line. The master will then misinterpret that the module is ready to receive and not as a pending transmission from the module. The problem is resolved by the Wi-Fi module always looking at the first byte received to see if it is the read flag byte (0x03) to tell if the master wants to read or write.

5.1.2 Implementing the Driver on the Microcontroller

The AVR microcontroller has hardware support for communicating over SPI which is necessary to obtain the desired speed. SPI communication could also be done in software by bit banging but since the processor is only running at 60 MHz it would not be possible to get the SPI to run at the full 16 MHz the Wi-Fi module is capable of. The software solution would also take up all resources of the processor during the transfer. The SPI hardware generates the SPI clock by dividing the main clock. Naturally it is divided by an integer meaning that SPI clock might not be able to set an exact frequency and in this case it settles at 15 MHz when setting it to 16 MHz since it is the closest choice. The data is loaded byte by byte into a shift register which is transferred bit by bit by the SPI hardware.

Configuring the SPI involves mapping the pins to use and specifying the phase and polarity of the clock and the desired frequency. Some modifications had to be made with regard to the chip select line. When setting up the SPI hardware for a transfer, the data to be sent will be written to a 32-bit register and the same register also holds which CS signal is to be used since the same SPI hardware can be shared between up to four different slaves. When transmitting the SPI hardware will assert CS and then directly start the data transfer. This creates a problem since according to the communication protocol used the Wi-Fi module has to assert IRQ before transmission can begin. The solution is not to map the CS signal to a pin when configuring the SPI and instead handling the CS signal in software separately as a general purpose IO pin.

The first implementation of the SPI driver had the processor looping through each byte and transferring them one by one. That is not the most efficient way since the microcontroller has the capability of DMA transfers. DMA, standing for Direct Memory Access is specialized hardware that lets peripherals access the main memory without going through the CPU. It gets rid off the overhead needed when the CPU otherwise has to read from the peripheral into one of its registers and then transfer it from the register to the memory which is an unnecessary step. Not only does it free up the CPU for more useful work but also allows faster memory transfers when the CPU is not a bottleneck. When the DMA transfer is complete the DMA controller notifies the processor by triggering an interrupt.

Access when using DMA is made through continuous blocks of memory. With each transfer a starting address in the memory is given along with the number of transfers and the size of each transfers together specifying a memory block. For initializing the DMA channel it needs to know the hardware peripheral that are to be used and the direction, i.e. loading from or to the memory. It is specialized hardware therefore it can only be used with supported peripherals. Also the interrupt used when transfer is done has to be set up. Here, two DMA channels are used since it is needed in both directions one for sending and one for receiving.

5.1.3 Driver Operation

The SPI driver is implemented like a state machine shown in figure 5.2 (the code may differ slightly in names). Using a state machine is suitable because the driver uses a lot of interrupts and the state has to be stored between them. In the figure it is shown which interrupt triggers a change from one state to

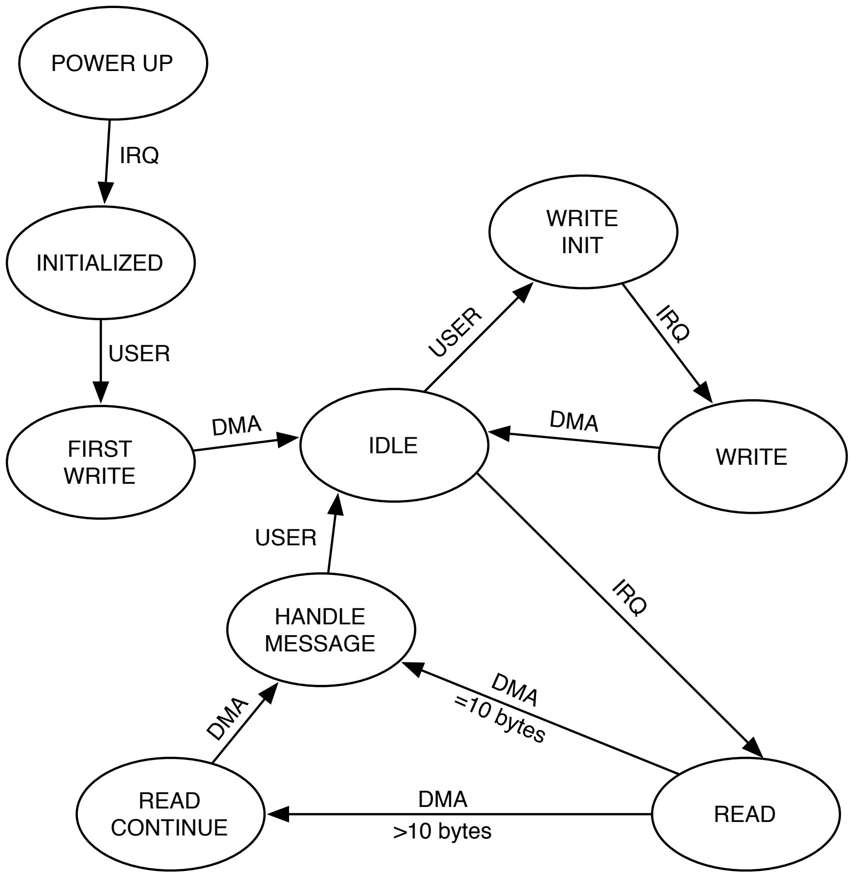


Figure 5.2: SPI State machine

another, the two different interrupts are interrupts from the IRQ line of the module and the other DMA transfer done interrupt. The only state changes that are not triggered by an interrupt are when: initiating a write, since that is triggered by the user of the driver and from HANDLE MESSAGE which returns after the higher level have processed the message.

On startup the initial state of the SPI driver is POWER UP where the power enable line is set high and it waits for the module to respond that it is ready. From there the state changes to INITIALIZED where the SPI driver waits for the user to trigger the first write operation, the Wi-Fi module will not send a message unless it has first received one. After the DMA controller signals that the first write has been completed the SPI driver changes to the IDLE state.

If the SPI driver receives a write command while in the IDLE state, it goes into WRITE INIT where the driver adds the SPI header and asserts CS. After getting a response from Wi-Fi module through an IRQ interrupt the driver initiates the data transfer in state WRITE and when the DMA transfer is complete the driver returns to IDLE.

Getting an IRQ interrupt when IDLE signals that there exists data to read and moves the driver into the state READ. To use DMA for reading, the length of the message have to be known beforehand. Because it is not known how many bytes should be read from the Wi-Fi module, the driver first reads only 10 bytes including the header which is the shortest message it will ever receive. After that transfer is done the SPI driver looks at the header to see the size of the whole message. If the message was only 10 bytes the SPI driver goes to HANDLE MESSAGE, if there is more data to read the driver moves to the READ CONTINUE state reading the rest of the data before going to HANDLE MESSAGE. In HANDLE MESSAGE the SPI driver pauses the SPI so no messages can be sent or received from the Wi-Fi module. When the message has been processed the SPI driver goes back to the IDLE state unpausing the SPI. For the full SPI driver code see appendix A.1.

5.2 Client Solution

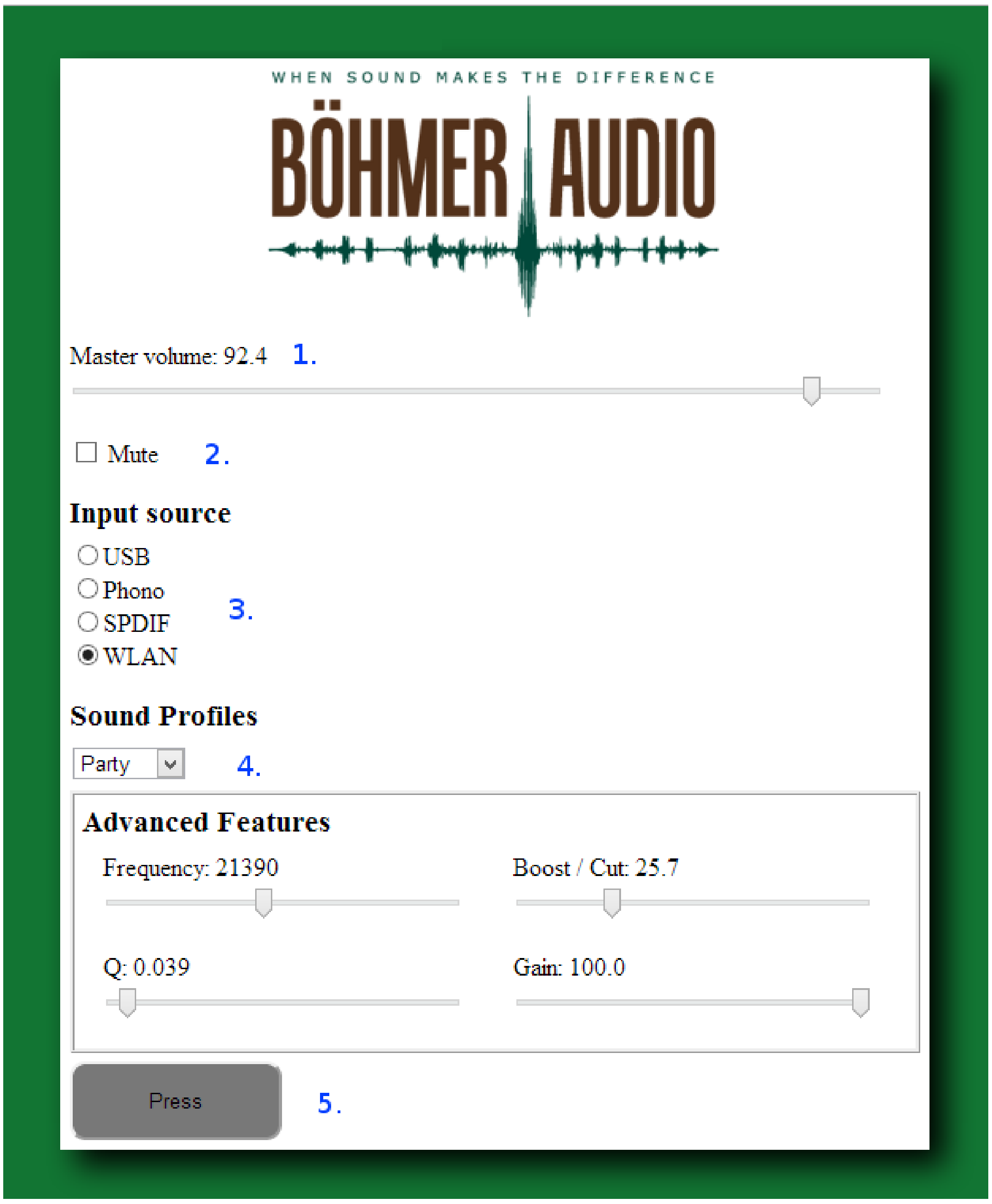
Table 5.1 is a collection of the different client solutions considered for the project. For cross platform development there were three different types of apps considered, for more information see 2.5, the one that was chosen was an HTML based interface accessed via a web browser from an HTML server. The reason being that the solution has better cross platform support since it supports both hand held devices and desktop/laptop devices without needing to have any special software installed on the different platforms. Another good reason for using a HTML based interface is that it relatively easy can be translated from an XML based layout file to HTML code, as specified in the requirements in section 1.3. For more information on how the XML to HTML translation works see section 5.2.2. Since there are no advanced graphics or other heavy calculations in our remote control application the slower speed of the HTML solution compared to the other solutions is not an issue.

Table 5.1: App programing languages considered for the project

Name	**Programming solution**	**Pros**	**Cons**
Kivy [28]	Interpreted Python code	Works on many platforms, GPU accelerated	No official support by Google and Apple
Sencha touch [29]	Interpreted HTML, Javascript, CSS code	Easy to use development tools	sencha touch slow on many phones, no support for PC/Mac
Phonegap [30]	Web browser based, HTML, Javascript, CSS	Uses standard web technologies	Needs to be built for every platform. No support for PC/Mac
Appcelerator [31]	Compiles to native code, HTML, Javascript, CSS	Good performance	No support for PC/Mac
Mosync [32]	Hybrid between interpreted native, C/C++ and/or HTML, Javascript, CSS	Able to mix web programing with C/C++	Only support for iOS 4.3 and earlier
HTML website	Web browser based, HTML, Javascript, CSS	Works on all platforms that have a web browser	Limited memory on host to save images and code
Xamarin [33]	Native code, C#	Good performance	Not free to use

5.2.1 Web Interface

The web interface consist of HTML and CSS code for defining the layout of the interface and Javascript working in the background to handle the communica-



Master volume: 92.4

☐ Mute

Input source

○ USB
○ Phono
○ SPDIF
● WLAN

Sound Profiles

Party

Advanced Features

Frequency: 21390

Boost / Cut: 25.7

Q: 0.039

Gain: 100.0

Figure 5.3: Web interface

tion with the server. The HTML part of the code defines the basic layout of the interface while the CSS part is used for the detailed layout such as background colors, organising the widgets into boxes and changing the layout according to screen size. Javascript, a coding language used for running code on the client side inside the browser, is used to send the user commands to the server. The HTML part of the code is automatically generated with the help of a special script, see section 5.2.2, so that the customer easily can choose which functions should be available to the end user.

The interface consist of 5 different kinds of input widgets, as can be seen in Figure 5.3. The widgets are:

1. Range sliders, which are used for inputs such as volume, filter cutoff frequency, gain and so on. To make it easier for the user to control the slider input an extra print out of the current value of slider has also been added.

2. Checkboxes, which are used for features that only have two options, like the mute command.

3. Radio Buttons, that are used for features that have many options where only one of the options can be activated at a time, for example when choosing input source.

4. Dropdown boxes are used for the same type of input as the radio buttons but is more compact which can be useful on devices with small screens.

5. Buttons, has no uses in our application but was added for future updates.

There is also the possibility to group together widgets with similar functions under a caption, see Advance Features captions in Figure 5.3.

The javascript is written in such a way that each kind of input widget, such as range slider, button, checkbox and so on, is tied to a specific javascript function and not the specific widget. The reason for doing it that way is that the javascript does not have to be changed when the layout changes. When the function is triggered it gets passed a DOM reference (Document Object Model) to identify which specific widget has triggered the function. The widgets are designed so that the name of the widget is the same as the command that is going to be sent to the server. The javascript takes the name of the widget and append the value of the widget to a text string and send it as a HTTP GET request to the server. For the full code of the Javascript script see appendix A.3.

5.2.2 XML to HTML Parser

According to the requirements in section 1.3 the remote control interface should be configured from an XML file. In order to fulfill the requirement a script that parses the XML file and translates it into HTML was developed in Python. Python is a scripting language developed to make code easy to read and write. Python also has a vast amount of libraries available making it a good language for rapid development.

The script takes a logo image file and an XML layout file that is generated by the Böhmer audio PC software and generates the web interface. The XML file contains data on which controls should be available to the user, which commands should be sent to the server when an action is triggered on the web interface and the layout of the controls. An example of how the XML code is translated to HTML code can be seen below. The XML code in the example below is a tag for a slider widget. A tag in HTML and XML code is marker for an object e.g. $< numeric >$ in the code below for the slider widget. The tag has a variable called cmd containing a string with the command that should be sent to the audio platform and min and max values for the range of the slider. When the XML code is translated into HTML the values from the XML tag are transferred to the HTML tag called input, from the transferred values an appropriate step size value is calculated and added to the HTML tag. Above the input tag in the HTML code an extra tag is added that show the current value of the slider to make it easier for the user to know the value of the slider.

XML code

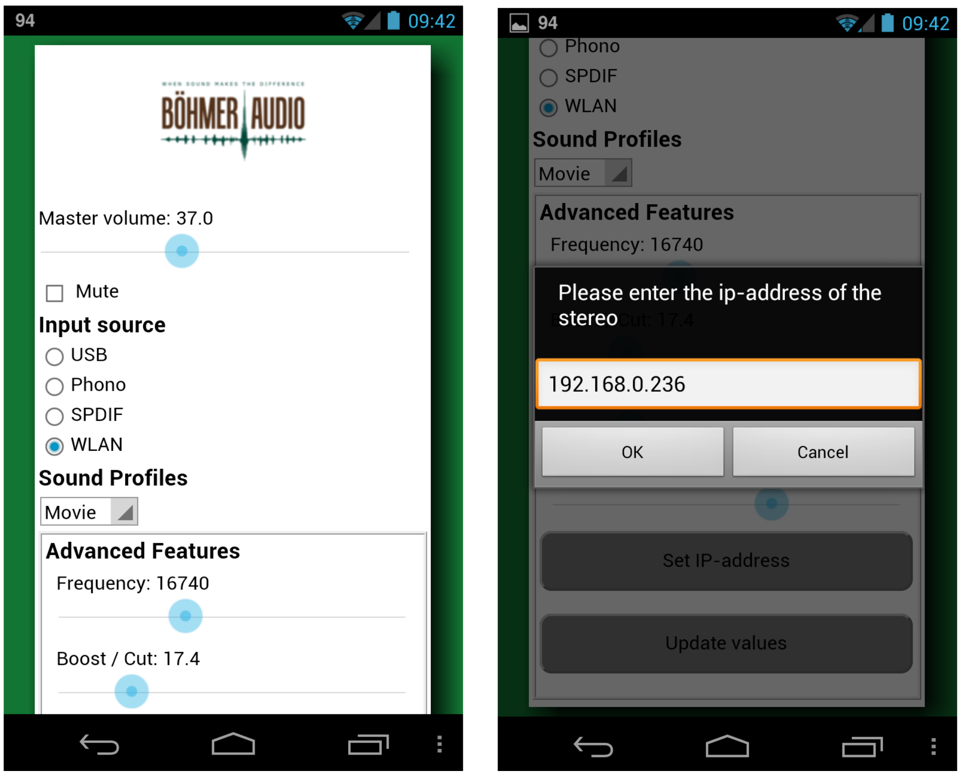
```
<numeric cmd="vol_set" min="0" max="100">Master Volume</numeric>
```

HTML code

```
<form>
<div style="float:left">Master Volume: </div><div id="Master_VolumeOut" style="float:left"
    >0</div>
<br/>
<input type="range" name="vol_set" id="Master_Volume" value="0" onchange="slChange(this)"
    max="100" min="0" step="0.1" >
<br/>
</form>
```

The script produces two files, one HTML file for previewing how the interface will look and one file containing the website coded to work with the micro-controllers file system. The second file is transferred to the file system on the microcontroller where it is accessed by the server software. For the full code for the XML to HTML script see appendix A.3.

5.2.3 Smartphone App Solution


Master volume: 37.0

☐ Mute

Input source

◯ USB
◯ Phono
◯ SPDIF
◉ WLAN

Sound Profiles

Movie

Advanced Features

Frequency: 16740

Boost / Cut: 17.4

Please enter the ip-address of the stereo

192.168.0.236

Figure 5.4: App interface on Android smartphone. Left picture showing the remote control interface. The right picture showing the dialog box for inputting the ip-address of the audio platform

A framework called Phonegap was used to also develop an app. Phonegap is a framework that makes it possible to take HTML and Javascript code and package it so that it can be installed on a smartphone like a native app. The existing code is reused but in order to make the HTML and Javascript interface work better as an app some changes to how it works were made. The changes in code that were made were mainly two extra buttons, one that updates all the current values of the audio platform and another one that opens up a dialog where the user can input the servers IP address and save it so there is no need to reenter it when rebooting the app.

Using an app instead of loading the interface in the handheld device web browser has some advantages and some disadvantages which can be seen below.

Advantages:

- No need to remember ip-address after initial configuration
- Customers can install app from app store/play store
- Good selling point when pitching product to customers
- Loading layout data from memory card instead of for the server

Disadvantages:

- Needs to be compiled individually for each OS
- Need to be distributed through an app store

5.3 The Server

As a result of choosing a HTML based interface, see section 5.2, an HTTP server was implemented as the server solution. The HTTP server consists of one part that handles the communication with the client through TCP sockets using the CC3000 API and another part that parses the messages that come from the client and sends appropriate responses.

5.3.1 Wi-Fi Module API

The available API have C functions to utilize the module. It consists of basic functions like starting and stopping the device, connecting and disconnecting to a Wi-Fi access point and scanning for access points. For wireless communications it uses the syntax of standard sockets also used in linux for example. API functions are mostly implemented by taking the arguments and packaging them in a frame and sending over the SPI.

Some changes had to be made to the supplied API to make it compile partly due to strict compiler settings and due to name conflicts. Also in the API it would at some points perform busy wait polling in a while loop waiting for a message from the module. Since the microcontroller is running a real time OS it was instead implemented so that the thread goes to sleep and wakes up when a new message is received with the help of semaphores. The changes have no effect on the functionality of the code.

5.3.2 HTTP Server Functionality

The server software is written to handle three different kinds of requests, load interface request, set command request and get update request.

The load interface request is the request sent by the web browser in order to receive the HTML layout data and the javascript data to be able to render the web site. The HTML layout data is too big to send in one TCP message so it is divided into smaller chunks when sending.

The command request is used for controlling the parameters of the audio platform. The server parses the command request by looking for the start of the command indicated by a ? character. The parsed command is sent to the command queue of the system where it later will be processed. More than one command can be sent in one request by inserting a & character between the commands.

The update request is used to get information about the current state of the parameters used by the remote control. The update request is sent every time the web interface is reloaded. When an update request is received by the server it responds to the client with an answer in a special format seen in italic text below. The answer has an abbreviation for each parameter and the value is presented with a five text characters and can thus be an integer, float number or normal text.

?mute=true0&mvol=00012&rvol=00100&lvol=00100&vlfu=0.111&bfre=40000

& bbos=00.03&bqqq=0.018&bgai=000.9& inpt=USB00

If the server receives an unknown request, it will respond with a 404 not found error which is a standard error in the HTTP protocol.

5.3.3 HTTP Server Implementation

The whole HTTP server is written in C-code, see appendix A.2 for code, and uses the API for the WiFi-chip to set up TCP sockets, see section 2.2 for more info, needed to receive the requests for loading the web interface. TCP has to be used for it to work with the HTTP protocol.

The server is implemented in a standard way, the emphasised words in the following text corresponds to the respective API commands. At startup the server opens a socket and setting the address to port number 80 which is called *binding*. Nothing is set about the address of the receiver, instead the socket is set to *listen* for incoming connections from clients. That means that the Wi-Fi module will receive the connections and put them in a queue, and the server has to *accept* the connection. Accepting a connection returns a new socket to the client. The server then *receive* the request from that socket, parse the request and *sends* back a reply before *closing* the socket and thus the connection. When not handling connections to clients the server will constantly poll the Wi-Fi module for new connections to accept using the original socket.

Chapter 6

Conclusion and Ending Comments

6.1 Challenges

Here some challenges and problems encountered throughout this project are described. They are things that did not fit in the rest of the report and that took some time and effort to resolve.

6.1.1 SPI Slave Select Problem

As specified in section 5.1.1 the Wi-Fi module does not use normal SPI and some changes had to be made in regard to that in section 5.1.2 by controlling the chip select separately. These changes made it not as straightforward to implement as it otherwise would, adding some not so obvious timing issues. For example when running the code without the extra delays of debug outputs the chip select went high before the transfer was done. The thing happening was that when the DMA signals that the transfer has been completed it only mean it has transferred the last byte to the SPI hardware. So the chip select was put high but there was still some data left in the SPI shift register to send. That would have been taken care of with standard SPI but here the check that the send buffer was empty had to be implemented in the code.

6.1.2 Web Interface Touch Screen Event Triggering

To get the web interface to work on both touchscreen input devices and conventional mouse and keyboard setup and together with the relatively resource limited embedded system, certain problems had to be solved.

The problem that was encountered was the fact that many of the events available to trigger a javascript function from the slider did not work together with both touchscreens and conventional mouse/keyboard setups. After some research an event called "onChange" was found which would trigger independent of which input device is being used. OnChange has one big disadvantage by the fact that it triggers for every step that the slider is moved. This means that one command is being sent for each step putting stress on the server since each command opens and closes many TCP-link resulting in poor performance. The solution to the problem was to only send a command when the slider has been released. This was implemented by having a timer that is reset every time a onChange event happens, if the timer reaches 400 ms the slider is considered to have been released and the value of its current position is sent to the server.

6.1.3 Web Interface Loading Speed Optimization

An optimization made to make the web interface load faster in the web browser was to encode the logo image file as base64 encoded data[34] and including that data directly in the HTML data being sent. This gives a faster loading of the interface because normally the web browser will first open a TCP socket and request the HTML data and then close that socket just to open a new one and then request the image data, if the image-data is base64 encoded and included in the HTML-data there is no need for the second connection. This is good since handling many request takes more time than just sending one data stream.

6.1.4 Web Interface Screen Size Optimization

Since the interface can be accessed by many different devices with varying screen sizes, special consideration had to be taken to make the layout work with the different screen sizes. The solution found was to have two different layouts, one for screens under 600 pixels wide and one for screens that are wider than 600 pixels. For screens smaller than 600 pixels all the widgets are laid out in one column while for bigger screens some widgets are laid out so that they are next to each other on the same row using more of the screen and reducing the amount of scrolling needed in the interface.

6.1.5 Patching

The module supplied did not come with the latest version of its driver and firmware and during the time of the project new firmware versions came out. Updating was done by writing to the internal EEPROM that had part of the memory reserved for this use. The part originally allocated for the patches was too small to accommodate the newer patches so to update the module the file structure of the EEPROM had to be rewritten. To do that an earlier version of the firmware first had to be loaded to the module since the newer versions do not allow full access to the EEPROM leading to a more complicated update procedure. Texas Instruments supply a patch programmer to run on the microcontroller but that was only compatible with their own microcontrollers so the code had to be ported to the used microcontroller. To make it function correctly took some extra time but was important since the new firmware fixes some crucial bugs.

6.1.6 Bugs on Module Side

Since the module is fairly new there were some bugs that had to be worked around as with all new products. Most were fixed in patches from Texas Instruments, for example one bug when sending at full speed the different parts in a TCP stream were sent out of order. One elusive bug was that the module sometimes would accept a connection when there was no message to be received. The bug was hard to detect because it looked as the module hanged when what was really happening was that it tried to receive something that was not there. Receiving is a blocking operation and therefore it stalled. To find the bug all packets sent over the network were inspected with the help of a package analyzer program called Wireshark. Finally, it was solved by using another command called select that monitors the state of the socket and more specifically can check if the socket really has data to read before calling receive. The Texas Instrument support forum was helpful in how to solve these problems.

6.2 Conclusion

This thesis main focus has been to find a suitable solution to the requirements set by Böhmer Audio in the beginning of the thesis. Another aspect of the thesis has been how to take requirements from paper and turn it into a working product which has been a great learning experience for gaining the knowledge needed to complete all the different stages in the development process.

Böhmer Audios idea is to sell their audio platform to OEM manufacturers that then can configure the platform according to their needs. To integrate our thesis project into the same philosophy all the software has been designed so that it can be easily reconfigured, e.g. the web interfaces function and layout are configured via an XML file. Also the HTTP server is programmed in such a way that if new control commands are added in the main system there is no need to change anything in the server part of the software.

By doing extensive research on the different Wi-Fi chip solutions available on the market a price and performance effective solution has been found. Another factor that lowers the price of the Wi-Fi extension module is the fact that the Wi-Fi chip chosen requires only a few external components to work, reducing the total component cost and manufacturing complexity. The Wi-Fi chip has enough bandwidth to be used for streaming music in the future and the software remote control interface can, since it is HTML based, be used by most modern devices that have a web browser.

We believe this is part of the future for consumer products. It is much more convenient and easy to control the settings wirelessly from the comfort of the couch than being constrained to a few buttons and small numeric display on the front panel of the audio platform. The possibilities other than this application are numerous for example the module could be connected to a sensor and be used to load a web page with statistics.The technology is mature enough to be added to most products and this thesis proves that the effort needed is not too big. Hopefully the price of the WiFi chip will decrease further making it even more appealing.

Overall this thesis project has been a success, we fulfilled all the requirements set in the beginning of the project and hopefully we have shed some light on the process of adding wireless connectivity to an application.

Bibliography

[1] Ericsson, more than 50 billion connected devices, Februari 2011

[2] Data Communications and Networking 4th edition, Behrouz A. Forouzan

[3] RFC 791 Internet Protocol, http://www.ietf.org/rfc/rfc791.txt *acc: 2013-05-03*

[4] RFC 793 Transmission Control Protocol, http://www.ietf.org/rfc/rfc793.txt *acc: 2013-05-03*

[5] RFC 768 User Datagram Protocol, http://www.ietf.org/rfc/rfc768.txt *acc: 2013-05-03*

[6] RFC 2616 Hypertext Transfer Protocol, http://www.ietf.org/rfc/rfc2616.txt *acc: 2013-05-03*

[7] RFC 2131 Dynamic Host Configuration Protocol, http://www.ietf.org/rfc/rfc2131.txt *acc: 2013-05-03*

[8] SPI - Serial Peripheral Interface , http://www.mct.net/faq/spi.html *acc: 2013-05-02*

[9] Colin M.L. Burnett, http://en.wikipedia.org/wiki/File:SPI_timing_diagram2.svg, *acc: 2013-05-06*

[10] Lifländer J, Ceramic Chip Antennas vs. PCB Trace Antennas: A Comparison, http://www.pulseelectronics.com/file.php?id=3721 , *acc: 2013-05-02*

[11] LS Research LLC, Antenna Design Guide, 8 March 2012

[12] Burris M, Price of component in end product, http://components.about.com/od/Design/a/How-Component-Selection-Impacts-The-Final-Price-Of-A-Product.htm. *acc: 2013-04-04*

[13] LS Research LLC, TiWi-SL module Datasheet, 2011-2012

[14] Texas Instruments, TI SimpleLink^{TM} CC3000 Module – Wi-Fi 802.11b/g Network Processor, November 2012

[15] Taiyo Yuden, WYSBCVGXA Brief Data Report, 17 December 2012

[16] Roving Networks, RN-171 802.11 b/g Wireless LAN module, 10 February 2012

[17] , BlueGiga, Bluegiga WF121 Wi-Fi module, 2012

[18] Redpine Signals, RS-9110-N-11-24 Self Contained 802.11 b/g/n Module with networking stack, November 2012

[19] GainSpan, GS1011M-DS, 29 October 2012

[20] H & D Wireless, Data Sheet HDG104 WiFi SIP component, February 2011

[21] H & D Wireless, Data Sheet HDG204 WiFi SIP component, February 2011

[22] Qualcomm Atheros, AR4100 single-stream 802.11n SIP for the internet of everything, 12 March 2012

[23] H & D Wireless, Product brief SPB4100 Wirelesss LAN module

[24] Asix, AX22001 Single Chip Microcontroller with TCP/IP and 802.11 WLAN MAC/Baseband, http://www.asix.com.tw/products.php?op=pItemdetail&PItemID=106;72;104, *acc: 2013-05-03*

[25] Econais, WiSmart EC32Sxx Wi-Fi 802.11b/g/n Module Family, http://www.econais.com/wp/products/ec32wxx-family, *acc: 2013-05-03*

[26] Atmel, AT32UC3C, January 2012

[27] Microchip, Compiled Tips N Tricks Guide, Chapter 8, 2009

[28] Kivy, http://kivy.org/ *acc: 2013-05-24*

[29] Sencha touch, http://www.sencha.com/products/touch *acc: 2013-05-24*

[30] Phonegap http://phonegap.com/ *acc: 2013-05-24*

[31] Appcelerator, http://www.appcelerator.com/ *acc: 2013-05-24*

[32] Mosync http://www.mosync.com/ *acc: 2013-05-24*

[33] Xamarin http://xamarin.com/ *acc: 2013-05-24*

[34] S. Josefsson, Ed., The Base16, Base32, and Base64 Data Encodings, July 2003

Appendices

Appendix A

Code

A.1 SPI Driver Code

code/spiwifi.c

```
/*
 * spiwifi.c
 *
 *  Created on: 12 feb 2013
 *      Author: Anders Skoog & Fredrik Stolt
 */
#include "spiwifi.h"
#include <spi.h>
#include <gpio.h>
#include <intc.h>
#include "cc3000_common.h"
#include "hci.h"
#include "pinout.h"
#include "dbgmsg.h"
#include "FreeRTOS.h"
#include "task.h"
#include <semphr.h>
#include "printk.h"
#include "dmesg.h"
#include <pdca.h>

#define SPI_MASTER_SPEED   16000000 //16MHZ max SPI frequency
#define SPI_BITS           8
#define CPUHZ              FOSC0

#define SPI_IRQ_PIN   AVR32_PIN_PA16
#define POW_EN_PIN  AVR32_PIN_PA11
#define CS_PIN  AVR32_PIN_PA13
#define CS 1

#define SPI_HEADER_SIZE       (5)

//different states
#define eSPI_STATE_POWERUP            (0)
#define eSPI_STATE_INITIALIZED        (1)
#define eSPI_STATE_IDLE               (2)
#define eSPI_STATE_WRITE_WAIT         (3)
#define eSPI_STATE_FIRST_WRITE        (4)
```

```
#define eSPI_STATE_WRITE            (5)
#define eSPI_STATE_READ_IRQ         (6)
#define eSPI_STATE_READ_CONT        (7)
#define eSPI_STATE_HANDLE_MESSAGE   (8)

//used in SPI header
#define READ                        (3)
#define WRITE                       (1)

#define HI(value)                   (((value) & 0xFF00) >> 8)
#define LO(value)                   ((value) & 0x00FF)

volatile avr32_spi_t *spi;

typedef struct {
  gcSpiHandleRx SPIRxHandler;
  unsigned short usTxPacketLength;
  unsigned short usRxPacketLength;
  unsigned long ulSpiState;
  unsigned char *pTxPacket;
  unsigned char *pRxPacket;
} tSpiInformation;

volatile tSpiInformation sSpiInformation;

// Static buffer for 10 bytes used for reading with read code and 9 dummy bytes
unsigned char tSpiReadHeader[10] = { READ };

unsigned char spi_rx_buffer[CC3000_RX_BUFFER_SIZE];
unsigned char wlan_tx_buffer[CC3000_TX_BUFFER_SIZE];

xSemaphoreHandle DMAWriteSemaphore;
xSemaphoreHandle stateChangeSemaphore;
#define SPI_PDCA_CHANNEL_RX 0
#define SPI_PDCA_CHANNEL_TX 1

void SpiPauseSpi(void);
void SpiWriteInternal(void);
void SpiRead(void);
void SpiReadCont(void);
void SpiReadDone(void);

__attribute__((__interrupt__)) static void pdca_int_handler(void);

//has to be run after init_spi_synth
void init_spi_wifi(void) {

  //configure IRQ pin as input
  gpio_configure_pin(SPI_IRQ_PIN, GPIO_DIR_INPUT);

  //configure CS pin as output
  gpio_configure_pin(CS_PIN, GPIO_DIR_OUTPUT);
  gpio_set_gpio_pin(CS_PIN);

  spi = &AVR32_SPI0; //pointer to access hardware SPI

  spi_options_t spiOptions;
  spiOptions.reg = CS;
  spiOptions.baudrate = SPI_MASTER_SPEED;
  spiOptions.bits = SPI_BITS; //8 bits per transfer
  //standard values
  spiOptions.spck_delay = 4;
  spiOptions.trans_delay = 0;
  spiOptions.stay_act = 1;
  spiOptions.spi_mode = 1;
  spiOptions.modfdis = 1;

  //set options for slave
  spi_setupChipReg(spi, &spiOptions, CPUHZ);

  spi_enable(spi);

  // PDCA channel options needed to initiate
  static const pdca_channel_options_t PDCA_OPTIONS_FOR_TRANSFER =
    {
      .addr = (void *)wlan_tx_buffer,              // memory address
      .pid = AVR32_PDCA_PID_SPI0_TX,               // select peripheral
      .size = 0,                                   // transfer counter
      .r_addr = NULL,                              // next memory address, not used
      .r_size = 0,                                 // next transfer counter, not used
      .transfer_size = PDCA_TRANSFER_SIZE_BYTE     // size of each transfer 8 bits
    };

  static const pdca_channel_options_t PDCA_OPTIONS_FOR_RECIEVE =
    {
      .addr = (void *)spi_rx_buffer,
      .pid = AVR32_PDCA_PID_SPI0_RX,
      .size = 0,
      .r_addr = NULL,
      .r_size = 0,
      .transfer_size = PDCA_TRANSFER_SIZE_BYTE
    };


  /******* Interrupt config **************/
  // Disable all interrupts.
  Disable_global_interrupt();

  //register interrupt handler function
  INTC_register_interrupt(&pdca_int_handler, AVR32_PDCA_IRQ_1, AVR32_INTC_INT0);

  //initiate DMA channels
  pdca_init_channel(SPI_PDCA_CHANNEL_TX, &PDCA_OPTIONS_FOR_TRANSFER);
  pdca_init_channel(SPI_PDCA_CHANNEL_RX, &PDCA_OPTIONS_FOR_RECIEVE);

  // Enable all interrupts.
  Enable_global_interrupt();

  //enable DMA channels
  pdca_enable(SPI_PDCA_CHANNEL_TX);
  pdca_enable(SPI_PDCA_CHANNEL_RX);

  //semaphore used with DMA interrupt
  vSemaphoreCreateBinary(DMAWriteSemaphore);
  xSemaphoreTake(DMAWriteSemaphore, 0);

  //semaphore used with IRQ interrupt
  vSemaphoreCreateBinary(stateChangeSemaphore);
  xSemaphoreTake(stateChangeSemaphore, 0);

  //Need to wait for 1 second after powering up  before starting the module
  vTaskDelay(1000);
}

// Initializes SPI hardware and registers the SPI RX handler that is called on
// each packet received from the CC3000 device.
void SpiOpen(gcSpiHandleRx pfRxHandler) {
  sSpiInformation.SPIRxHandler = pfRxHandler;
  sSpiInformation.pRxPacket = &spi_rx_buffer[0];
  sSpiInformation.ulSpiState = eSPI_STATE_POWERUP;
  sSpiInformation.usTxPacketLength = 0;
  sSpiInformation.pTxPacket = NULL;
  sSpiInformation.usRxPacketLength = 0;
  tSLInformation.WlanInterruptEnable();
}

// Receives a pointer to data and the length of data and transmits a data
// packet to the SPI according to the SPI protocol,
// differentiating between the first write and subsequent write operations.
long SpiWrite(unsigned char *pUserBuffer, unsigned short usLength)
{
  //wait until the we are in a state that allows writing
  while (sSpiInformation.ulSpiState != eSPI_STATE_INITIALIZED
        && sSpiInformation.ulSpiState != eSPI_STATE_IDLE) {
    //put thread on hold until state is changed
    xSemaphoreTake(stateChangeSemaphore, portMAX_DELAY);
  }

  //check if padding is needed for even number of bytes
  unsigned short ucPad = !(usLength & 0x0001);

  //5-byte header including message length
  //user required to leave the first 5 bytes blank
  pUserBuffer[0] = WRITE;
  pUserBuffer[1] = HI(usLength + ucPad);
  pUserBuffer[2] = LO(usLength + ucPad);
  pUserBuffer[3] = 0;
  pUserBuffer[4] = 0;

  //number of bytes to send
  usLength += SPI_HEADER_SIZE + ucPad;
  sSpiInformation.usTxPacketLength = usLength;
  //message to send
  sSpiInformation.pTxPacket = pUserBuffer;

  while (1) {
    switch (sSpiInformation.ulSpiState) {
    case eSPI_STATE_INITIALIZED:
      sSpiInformation.ulSpiState = eSPI_STATE_FIRST_WRITE;
      vTaskDelay(7); //delay 7ms according to data sheet
      gpio_clr_gpio_pin(CS_PIN); //chip select low, request start transmission
      SpiWriteInternal();
      return 0;
    case eSPI_STATE_IDLE:
      //Disable interrupt because of potential race condition with read
      sWlanInterruptDisable();
      if (sSpiInformation.ulSpiState == eSPI_STATE_IDLE)
        sSpiInformation.ulSpiState = eSPI_STATE_WRITE_WAIT;
      else
        break;
      xSemaphoreTake(stateChangeSemaphore, 0); //reset semaphore
      sWlanInterruptEnable();
      gpio_clr_gpio_pin(CS_PIN); //chip select low, request start transmission
      //wait for change to write wait with the semaphore
      xSemaphoreTake(stateChangeSemaphore, portMAX_DELAY);
      SpiWriteInternal();
      return 0;
    default:
      break;
    }
    //if not valid state wait for change in state
    //wrong state can happen if there is an incoming message
    xSemaphoreTake(stateChangeSemaphore, portMAX_DELAY);
  }
  return 0;
}

//make transfer over SPI
void SpiWriteInternal() {
  switch (sSpiInformation.ulSpiState) {
  case eSPI_STATE_FIRST_WRITE:
    vTaskDelay(1); //Wait atleast extra 50uS before first write
    spi_selectionMode(spi,0,0,100); //Set selection mode to fixed peripheral
    spi_selectChip(spi, CS); //set right CS to load the right options

    //start with first 4 bytes
    for (unsigned short i = 0; i < 4; i++) {
      spi_write(spi, sSpiInformation.pTxPacket[i]);
    }
    vTaskDelay(1); //delay according to data sheet

    //Transfer the rest by setting up DMA
    pdca_load_channel(SPI_PDCA_CHANNEL_TX, (void*)(&sSpiInformation.pTxPacket[4]),
                      sSpiInformation.usTxPacketLength-4);
    pdca_enable_interrupt_transfer_complete(SPI_PDCA_CHANNEL_TX);
 break;

  case eSPI_STATE_WRITE:
    //Transmit by setting up DMA
    spi_selectChip(spi, CS);
    pdca_load_channel(SPI_PDCA_CHANNEL_TX, (void*)sSpiInformation.pTxPacket,
                      sSpiInformation.usTxPacketLength);
    pdca_enable_interrupt_transfer_complete(SPI_PDCA_CHANNEL_TX);
    break;
  default:
    break;
  }
  //wait until the DMA is done
  xSemaphoreTake(DMAWriteSemaphore, portMAX_DELAY);

  //when DMA is done there can still be data left in the SPI
  //shift register that have not been send
  while(!spi_writeEndCheck(spi));

  spi_unselectChip(spi,CS);
  sSpiInformation.ulSpiState = eSPI_STATE_IDLE;
  gpio_set_gpio_pin(CS_PIN); //chip select high, end transmission

}

//interrupt handler for DMA transfer complete
__attribute__((__interrupt__)) static void pdca_int_handler(void) {

  //Disable the interrupt
  Disable_global_interrupt();
  pdca_disable_interrupt_transfer_complete(SPI_PDCA_CHANNEL_TX);
  pdca_disable_interrupt_transfer_complete(SPI_PDCA_CHANNEL_RX);
  Enable_global_interrupt();

  if (sSpiInformation.ulSpiState == eSPI_STATE_WRITE ||
      sSpiInformation.ulSpiState == eSPI_STATE_FIRST_WRITE) {
    //Give semaphore to signal that write is done
    xSemaphoreGiveFromISR(DMAWriteSemaphore, NULL);
  }

  if (sSpiInformation.ulSpiState == eSPI_STATE_READ_IRQ) {
    //iinterrupt on transfer line so check that the receive is done, at most one byte left
    volatile avr32_pdca_channel_t* dma_rx_channel = pdca_get_handler(SPI_PDCA_CHANNEL_RX);
    while(dma_rx_channel->tcr);
    SpiReadCont();
  }
  else if (sSpiInformation.ulSpiState == eSPI_STATE_READ_CONT) {
    volatile avr32_pdca_channel_t* dma_rx_channel = pdca_get_handler(SPI_PDCA_CHANNEL_RX);
    while(dma_rx_channel->tcr);
    SpiReadDone();
  }
}

//interrupt handler for IRQ line
__attribute__((__interrupt__)) static void irq_arbiter(void) {

  //change to the right state
  switch (sSpiInformation.ulSpiState) {
  case eSPI_STATE_POWERUP:
    sSpiInformation.ulSpiState = eSPI_STATE_INITIALIZED;
    break;
  case eSPI_STATE_IDLE:
    sSpiInformation.ulSpiState = eSPI_STATE_READ_IRQ;
    SpiRead();
    break;
  case eSPI_STATE_WRITE_WAIT:
    sSpiInformation.ulSpiState = eSPI_STATE_WRITE;
    break;
  default:
    sSpiInformation.ulSpiState = eSPI_STATE_READ_IRQ;
    SpiRead();
    break;
  }
   // Clears the interrupt flag
  gpio_clear_pin_interrupt_flag(SPI_IRQ_PIN);

  //signal that the state changed
  xSemaphoreGiveFromISR(stateChangeSemaphore, NULL);
}

//Read the first ten bytes of incoming message
void SpiRead() {
  sWlanInterruptDisable(); //disable IRQ to not interrupt send
  gpio_clr_gpio_pin(CS_PIN); //ready for read
  spi_selectChip(spi, CS);

  //if already something in SPI read register empty it
  while(spi_readRegisterFullCheck(spi))
    spi_read(spi,NULL);

  //Start reading 10 bytes which is minimal transfer length
  pdca_load_channel(SPI_PDCA_CHANNEL_RX, (void*)spi_rx_buffer, 10);
  pdca_load_channel(SPI_PDCA_CHANNEL_TX, (void*)tSpiReadHeader, 10);
  pdca_enable_interrupt_transfer_complete(SPI_PDCA_CHANNEL_TX);
}

void SpiReadCont() {
  sSpiInformation.ulSpiState = eSPI_STATE_READ_CONT;

  //read the packet length in header
  sSpiInformation.usRxPacketLength = (spi_rx_buffer[3] << 8) | spi_rx_buffer[4];
  //already read 5 bytes excluding header
  unsigned long leftToSend = sSpiInformation.usRxPacketLength - 5;
  if (leftToSend) {
    //read the rest
    spi_selectChip(spi, CS);
    pdca_load_channel(SPI_PDCA_CHANNEL_RX, (void*)(spi_rx_buffer+10), leftToSend);
    pdca_load_channel(SPI_PDCA_CHANNEL_TX, (void*)sSpiInformation.pTxPacket, leftToSend);
    pdca_enable_interrupt_transfer_complete(SPI_PDCA_CHANNEL_TX);
  }
  else {
    SpiReadDone();
  }
}

//all bytes read so process message
void SpiReadDone() {
  SpiPauseSpi(); //pause to process message
  sSpiInformation.ulSpiState = eSPI_STATE_HANDLE_MESSAGE;
  sWlanInterruptEnable();
  //call the supplied callback with pointer to message
  sSpiInformation.SPIRxHandler(sSpiInformation.pRxPacket + SPI_HEADER_SIZE);
}

//Releases all resources used by the SPI and deinitializes the hardware.
void SpiClose(void) {
  if (sSpiInformation.pRxPacket)
    {
      sSpiInformation.pRxPacket = 0;
    }
  tSLInformation.WlanInterruptDisable();
}

//Resumes SPI communication under the assumption that it was
//previously paused within the SPI driver itself.
void SpiResumeSpi(void) {
  gpio_set_gpio_pin(SPI_IRQ_PIN);
  gpio_configure_pin(SPI_IRQ_PIN,GPIO_DIR_INPUT); //set pin as input
  gpio_set_gpio_pin(CS_PIN); //read done

  //change state to idle and signal with the semaphore
  sSpiInformation.ulSpiState = eSPI_STATE_IDLE;
  xSemaphoreGiveFromISR(stateChangeSemaphore, NULL);
}

//Pause SPI by forcing IRQ high
void SpiPauseSpi() {
  gpio_configure_pin(SPI_IRQ_PIN,GPIO_DIR_OUTPUT); //set pin as output
  gpio_clr_gpio_pin(SPI_IRQ_PIN); //set pin high
}
```

```
//The callback provided during wlan_init call and invoked
//to read a value of the SPI IRQ pin of the CC3000 device
long sReadWlanInterruptPin(void) {
  return (long) gpio_get_pin_value(SPI_IRQ_PIN);
}

//The callback provided during the wlan_init call and invoked to
//enable an interrupt on the IRQ line of SPI
void sWlanInterruptEnable(void) {
  gpio_configure_pin(SPI_IRQ_PIN, GPIO_DIR_INPUT); //set pin as input
  Disable_global_interrupt();
  gpio_enable_pin_interrupt(SPI_IRQ_PIN, GPIO_FALLING_EDGE);
  //register interrupt, INT0=Lowest priority
  INTC_register_interrupt(&irq_arbiter, AVR32_GPIO_IRQ_0+(SPI_IRQ_PIN/8),AVR32_INTC_INT0);
  Enable_global_interrupt();
}

//The callback provided during the wlan_init call and invoked
//to disable an interrupt on the IRQ line of SWLANINTERRUPTDISABLE
void sWlanInterruptDisable(void) {
  gpio_disable_pin_interrupt(SPI_IRQ_PIN);
}

//The callback provided during the wlan_init call and invoked to write a value to the
//enable pin of the CC3000 device, that is, entry or exit from reset of the CC3000 device
void sWriteWlanPin(unsigned char in) {
  // Enables gpio control for the pin
  gpio_enable_gpio_pin(POW_EN_PIN);
  if (in == WLAN_ENABLE) {
    gpio_set_gpio_pin(POW_EN_PIN);
  } else {
    gpio_clr_gpio_pin(POW_EN_PIN);
  }
}
```

code/spiwifi.h

```
/*
 * spiwifi.h
 *
 *  Created on: 12 feb 2013
 *      Author: Anders Skoog & Fredrik Stolt
 */

#ifndef _SPI_DRIVER_H_
#define _SPI_DRIVER_H_

#include "compiler.h"

typedef void (*gcSpiHandleRx)(void* p);

extern unsigned char wlan_tx_buffer[];

//Used by CC3000 API
void SpiReceiveHandler(void *pvBuffer);

//set power enable, init spi pins and DMA channels
void init_spi_wifi(void);

//Initializes SPI hardware and registers the SPI RX handler that is called on each
//packet received from the CC3000 device.
void SpiOpen(gcSpiHandleRx pfRxHandler);

// Receives a pointer to data and the length of data and transmits a data packet
//to the SPI according to the SPI protocol, differentiating between the first write
//transaction and subsequent write operations.
long SpiWrite(unsigned char *pUserBuffer, unsigned short usLength);

//Releases all resources used by the SPI and deinitializes the hardware.
void SpiClose(void);
```

```
//Resumes SPI communication under the assumption that it was previously paused
//within the SPI driver itself.
void  SpiResumeSpi(void);

//sWriteWlanPin - The callback provided during the wlan_init call and invoked to write a
//value to the enable pin, that is, entry or exit from reset of the CC3000 device
void sWriteWlanPin(unsigned char in);

//sWlanInterruptDisable - The callback provided during the wlan_init call and invoked
//to disable an interrupt on the IRQ line of SPI
void sWlanInterruptDisable(void);

//sWlanInterruptEnable - The callback provided during the wlan_init call and invoked
//to enable an interrupt on the IRQ line of SPI
void sWlanInterruptEnable(void);

//sReadWlanInterruptPin - The callback provided during wlan_init call and invoked
//to read a value of the SPI IRQ pin of the CC3000 device
long sReadWlanInterruptPin(void);

#endif // _SPI_DRIVER_H_
```

A.2 Server Code

code/remotewifi.c

```
#include "spiwifi.h"
#include "remotewifi.h"
#include "printk.h"
#include "httpserver.h"
#include "socket.h"
#include "wlan.h"
#include "hci.h"
#include "netapp.h"
#include "nvmem.h"
#include "security.h"
#include "cmdchan_input.h"

//semaphores used
xSemaphoreHandle waitEventSemaphore;
xSemaphoreHandle smartConfigSemaphore;
xSemaphoreHandle connectSemaphore;
xSemaphoreHandle DHCPSemaphore;
xSemaphoreHandle freeBufferChangeSemaphore;
xSemaphoreHandle sendingDoneSemaphore;

//variables for sockets
static int highsock = 0;
static int serversock;
static int connectlist[] = {0};

static volatile int DHCPfinished;

static void CC3000_UsynchCallback(long lEventType, char * data, unsigned char length);

static void handleHttpRequest(void);
static int openServer(void);
static void sendUpdate(void);

//function called with various events from the module
static void CC3000_UsynchCallback(long lEventType, char * data, unsigned char length) {
  if (lEventType == HCI_EVNT_WLAN_UNSOL_INIT || lEventType == HCI_EVNT_WLAN_KEEPALIVE) {
    printk("something is working\n");
  }
  else if (lEventType == HCI_EVNT_WLAN_ASYNC_SIMPLE_CONFIG_DONE) {
    xSemaphoreGiveFromISR(smartConfigSemaphore, NULL);
  }
  else if (lEventType == HCI_EVNT_ASYNC_TCP_CLOSE_WAIT){
    //printk("HCI_EVNT_ASYNC_TCP_CLOSE_WAIT\n");
  }
  else if (lEventType == HCI_EVNT_WLAN_UNSOL_DHCP) {
    xSemaphoreGiveFromISR(DHCPSemaphore, NULL);
    DHCPfinished = 1;
  }
  else if (lEventType == HCI_EVNT_WLAN_UNSOL_CONNECT) {
    xSemaphoreGiveFromISR(connectSemaphore, NULL);
  }
  else if (lEventType == HCI_EVNT_WLAN_UNSOL_DISCONNECT) {
    //printk("HCI_EVNT_WLAN_UNSOL_DISCONNECT\n");
  }
  //evnet after it has released and sent all buffers on the module
  else if (lEventType == HCI_EVENT_CC3000_CAN_SHUT_DOWN) {
    xSemaphoreGiveFromISR(sendingDoneSemaphore, NULL);
  }
}


void wifi_serverInit(){
  openServer();
}

void wifi_init() {
  //create semaphores
  vSemaphoreCreateBinary(waitEventSemaphore);
  vSemaphoreCreateBinary(connectSemaphore);
  vSemaphoreCreateBinary(DHCPSemaphore);
  vSemaphoreCreateBinary(freeBufferChangeSemaphore);
  vSemaphoreCreateBinary(sendingDoneSemaphore);
  //reset semaphores
  xSemaphoreTake(connectSemaphore, 0);
  xSemaphoreTake(waitEventSemaphore, 0);
  xSemaphoreTake(DHCPSemaphore, 0);

  sWriteWlanPin(0);
  vTaskDelay(50);

  init_spi_wifi(); //start by init SPI

  //init wlan by giving function pointers to handle the pins
  wlan_init(CC3000_UsynchCallback,0,0,0,&sReadWlanInterruptPin,
          &sWlanInterruptEnable, &sWlanInterruptDisable, &sWriteWlanPin);

  //start by sending the first messages
  wlan_start(0);
  //policy to auto connect to saved access point
  wlan_ioctl_set_connection_policy(DISABLE, DISABLE, ENABLE);
}

void wifi_smartConfig(unsigned char* key) {
  //create and reset semaphore to wait for the configuration
  vSemaphoreCreateBinary(smartConfigSemaphore);
  xSemaphoreTake(smartConfigSemaphore, portMAX_DELAY);

  // Reset all the previous configuration
  wlan_ioctl_set_connection_policy(DISABLE, DISABLE, DISABLE);
  wlan_ioctl_del_profile(255);

  //Wait until CC3000 is dissconected
  while (wlan_ioctl_statusget());

  xSemaphoreTake(connectSemaphore, 0); //reset

  //create entry for encryption key in EEPROM
  if (key) {
    nvmem_create_entry(NVMEM_AES128_KEY_FILEID,AES128_KEY_SIZE);
  }

  const char aucCC3000_prefix[] = {'T', 'T', 'T'};
  wlan_smart_config_set_prefix((char*)aucCC3000_prefix);

  //Trigger the Smart Config process
  if (key == NULL)
    result = wlan_smart_config_start(0); //without encryption
  else
    result= wlan_smart_config_start(1);  //with encryption

  xSemaphoreTake(smartConfigSemaphore, portMAX_DELAY);

  if (key) {
    // Decrypt configuration information and add profile
    long ret = wlan_smart_config_process();
  }

  // Configure to connect automatically to the AP retrieved in the
  wlan_ioctl_set_connection_policy(DISABLE, DISABLE, ENABLE);

  // reset the CC3000
  wlan_stop();
  vTaskDelay(50);
  wlan_start(0);
}

int wifi_status(unsigned char* ip_addr) {
  if (wlan_ioctl_statusget() != 3) { //disconnected
    return 0;
  }
  if (ip_addr != NULL) {
    tNetappIpconfigRetArgs report;
    //get ip-address
    netapp_ipconfig(&report);
    ip_addr[0] = report.aucIP[3];
    ip_addr[1] = report.aucIP[2];
    ip_addr[2] = report.aucIP[1];
    ip_addr[3] = report.aucIP[0];
  }
  return 1;
}

void wifi_reset() {
  wlan_stop();
  vTaskDelay(50);
  wlan_start(0);
}

//manual connection
void wifi_connect(int security, unsigned char* ssidPtr, int ssidLen,
                  unsigned char* decKeyPtr, int keyLen) {
  //delete previous connection information
  wlan_ioctl_set_connection_policy(DISABLE, DISABLE, DISABLE);
  wlan_ioctl_del_profile(255);

  //Wait until CC3000 is dissconected
  while (wlan_ioctl_statusget());

  switch (security) {
  case WLAN_SEC_UNSEC:
    wlan_add_profile(WLAN_SEC_UNSEC, ssidPtr, ssidLen, NULL, 1, 0, 0, 0, 0, 0);
    break;
  case WLAN_SEC_WEP:
    wlan_add_profile(WLAN_SEC_WEP, ssidPtr, ssidLen, NULL, 1, keyLen, 0, 0, decKeyPtr, 0);
    break;
  case WLAN_SEC_WPA:
  case WLAN_SEC_WPA2:
    wlan_add_profile(WLAN_SEC_WPA2, ssidPtr, ssidLen, NULL, 1, 0x18, 0x1e, 2,
                     decKeyPtr, keyLen);
    break;
  }
  //set policy to connect to the stored profile
  wlan_ioctl_set_connection_policy(DISABLE, DISABLE, ENABLE);
}

static int openServer() {
  serversock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
  sockaddr a;
  short port = 80;
  a.sa_family = AF_INET;
  a.sa_data[0] = (port & 0xFF00) >> 8;
  a.sa_data[1] = (port & 0x00FF);
  a.sa_data[2] = 0; //Ip-address 0.0.0.0 connects to all
  a.sa_data[3] = 0;
  a.sa_data[4] = 0;
  a.sa_data[5] = 0;
  memset(&a.sa_data[6], 0, 8);

  //bind socket to port 80
  if (bind(serversock, &a, sizeof(sockaddr)) != 0) {
    printk("bind failed\n");
    return -1;
  }
  //set socket to listen for new connections
  if (listen(serversock,0) != 0) {
    printk("listen failed\n");
    return -1;
  }
  if (serversock > highsock)
    highsock = serversock;
  return 0;
}

void wifi_checkHttpServer(){
  static cc3000_fd_set socks;
  static int clientsock = 0;

  //setup for using select to monitor socket
  CC3000_FD_ZERO(&socks);
  CC3000_FD_SET(serversock, &socks);

  if (connectlist[0] != 0) {
    CC3000_FD_SET(connectlist[0], &socks);
    if (highsock < connectlist[0])
      highsock = connectlist[0];
  }

  //minimum timeout at 5 ms
  timeval timeout;
  timeout.tv_sec = 0;
  timeout.tv_usec = 5000;

  //check if socket have something send
  int readsocks = select(highsock+1, &socks, NULL, NULL, &timeout);

  if (readsocks < 0){
    printk("error with select");
    return;
  }

  sockaddr clientaddr;
  socklen_t addr_length = 0;
  if (readsocks == 0){
    //accept new connection
    clientsock = accept(serversock, &clientaddr, &addr_length);
    if (clientsock >= 0) {
      if (connectlist[0] != 0) {
        //close existing
        closesocket(connectlist[0]);
      }
      connectlist[0] = clientsock;
      if (highsock < clientsock)
        highsock = clientsock;
    }
    //error need to restart server
    if (clientsock == -57) {
      closesocket(serversock);
      openServer();
    }
    return;
  }

  //This doesn't work, using select for serversocket gives no response
  if (CC3000_FD_ISSET(serversock,&socks)) {
    if (connectlist[0] != 0) {
      closesocket(connectlist[0]);
    }
    clientsock = accept(serversock, &clientaddr, &addr_length);
    if (clientsock >= 0) {
      connectlist[0] = clientsock;
      if (highsock < clientsock)
        highsock = clientsock;
    }
    else if (clientsock == -57) {
      closesocket(serversock);
      openServer();
    }
    return;
  }

  //if select flagged for thing to receive then handle request
  if (CC3000_FD_ISSET(connectlist[0],&socks)) {
    clientsock = connectlist[0];
  } else {
    return;
  }
  handleHttpRequest();
}

//function for sending over TCP, splitting it up in suitable chunks
//return 0 if managed to send and -1 otherwise
static int TCPsend(long sd, const char *buf, long len);
static int TCPsend(long sd, const char *buf, long len) {
  long offset = 0;
  long length = 0;

  while(len) {
    if (len > 1400)
      length = 1400;
    else
      length = len;

    xSemaphoreTake(freeBufferChangeSemaphore,0); //reset semaphore
    int sent = send(sd, buf+offset, length, 0);

    if (sent == -1) {
      return -1;
    }
    //sent == -2 signals no free buffers on the module so have to wait
    else if (sent == -2) {
      xSemaphoreTake(freeBufferChangeSemaphore,5000/portTICK_RATE_MS);
      sent = send(sd, buf+offset, length, 0);;
    }
    if (sent != length)
      return -1;

    offset += length;
    len -= length;
  }
  return 0;
}

static char allValues[] = {'H','T','T','P','/','1','.','0',' ','2','0','0',' ','O','K','\n
    ','\n','?','m','u','t','e','=','f','a','l','s','e','&','m','v','o','l','=','0','0','0'
    ,'0','0','&','r','v','o','l','=','0','0','0','0','0','&','l','v','o','l','=','0','0','
    0','0','0','&','v','l','f','u','=','0','0','0','0','0','&','b','f','r','e','=','0','0'
    ,'0','0','0','&','b','b','o','s','=','0','0','0','0','0','&','b','q','q','q','=','0','
    0','0','0','0','&','b','g','a','i','=','0','0','0','0','0','&','i','n','p','t','=','W'
    ,'L','A','N','0'};

#define VALUE_RES 5
typedef char vString[VALUE_RES];
```

```
//struct with settings stored as float values
//float values stored as 5 character string
typedef struct{
  vString volume;
  vString vlnfau;
  vString bq_freq;
  vString bq_bc;
  vString bq_q;
  vString bq_gain;
  char end;
} updateValuesFloat;

//stuct with settings stored as integers
typedef struct{
  int mute;
  struct{
    int usb;
    int phono;
    int spdif;
    int wlan;
  } input;
  int end;
} updateValuesInt;

updateValuesFloat fValues;
updateValuesInt iValues;
int *iValues_ptr;
char *fValues_ptr;
volatile int done_values;

//receives messages as strings from the other threads about different settings
void wifi_SendPartialPacket(const packet_t* p,uint16_t* sendOffset){
  *sendOffset = p->dataLength;
  if (!strncmp("200 OK",p->pData,6))
    return;

  //if waiting for integer
  if (iValues_ptr) {
    //convert to integer
    *iValues_ptr = atoi(p->pData);
    if(++iValues_ptr == &iValues.end)
      done_values = 1;
  }
  //if waiting for double
  else if (fValues_ptr) {
    //limit to number using 5 characters
    int length = strlen(p->pData)-1; //-1 for newline
    if (length > VALUE_RES)
      length = VALUE_RES;
    if (length < VALUE_RES)
      memset(fValues_ptr, '0', VALUE_RES-length);
    memcpy(fValues_ptr+(VALUE_RES-length), p->pData, length);

    fValues_ptr += VALUE_RES;
    if(fValues_ptr == &fValues.end)
      done_values = 1;
  }
}

static void sendUpdate(){

  //prepares message for updating values in the interface
  done_values = 0;
  fValues_ptr = NULL;
  iValues_ptr = (int*)&iValues;
  //fill up struct with answers in wifi_SendPartialPacket
  //have to call in same order as in struct
  CmdChan_ProcessRow_const(PACKET_TYPE_WIFI ,"preset test Mute.upr");
  CmdChan_ProcessRow_const(PACKET_TYPE_WIFI ,"preset test USB*.fpr");
  CmdChan_ProcessRow_const(PACKET_TYPE_WIFI ,"preset test Phono*.fpr");
  CmdChan_ProcessRow_const(PACKET_TYPE_WIFI ,"preset test SPDIF*.fpr");
  CmdChan_ProcessRow_const(PACKET_TYPE_WIFI ,"preset test WLAN*.fpr");
  while(!done_values);
  fValues_ptr = (char*)&fValues;
  iValues_ptr = NULL;
  CmdChan_ProcessRow_const(PACKET_TYPE_WIFI ,"vol get");
  CmdChan_ProcessRow_const(PACKET_TYPE_WIFI ,"uservalue get VeryLongNameForAUser");
  CmdChan_ProcessRow_const(PACKET_TYPE_WIFI ,"biquad get PEQUserTarget-1_1[0..4] F0");
  CmdChan_ProcessRow_const(PACKET_TYPE_WIFI ,"biquad get PEQUserTarget-1_1[0..4] BC");
  CmdChan_ProcessRow_const(PACKET_TYPE_WIFI ,"biquad get PEQUserTarget-1_1[0..4] Q");
  CmdChan_ProcessRow_const(PACKET_TYPE_WIFI ,"biquad get PEQUserTarget-1_1[0..4] G");
  while(!done_values);

#define RESPONSEH 17

  //Copy values into response string
  //Master volume
  memcpy(allValues+17+RESPONSEH, fValues.volume, 5);
  //right volume
  memcpy(allValues+28+RESPONSEH, fValues.volume, 5);
  //left volume
  memcpy(allValues+39+RESPONSEH, fValues.volume, 5);
  //vlfu
  memcpy(allValues+50+RESPONSEH, fValues.vlnfau, 5);
  //Biquad frequency
  memcpy(allValues+61+RESPONSEH, fValues.bq_freq, 5);
  //Biquad Boost Cut
  memcpy(allValues+72+RESPONSEH, fValues.bq_bc, 5);
  //Biquad Q
  memcpy(allValues+83+RESPONSEH, fValues.bq_q, 5);
  //Biquad Gain
  memcpy(allValues+94+RESPONSEH, fValues.bq_gain, 5);
  if(iValues.input.usb) {
    allValues[RESPONSEH+105] = 'U';
    allValues[RESPONSEH+106] = 'S';
    allValues[RESPONSEH+107] = 'B';
    allValues[RESPONSEH+108] = '0';
    allValues[RESPONSEH+109] = '0';
  }
  else if(iValues.input.phono) {
    allValues[RESPONSEH+105] = 'P';
    allValues[RESPONSEH+106] = 'h';
    allValues[RESPONSEH+107] = 'o';
    allValues[RESPONSEH+108] = 'n';
    allValues[RESPONSEH+109] = 'o';
  }
  else if(iValues.input.spdif) {
    allValues[RESPONSEH+105] = 'S';
    allValues[RESPONSEH+106] = 'P';
    allValues[RESPONSEH+107] = 'D';
    allValues[RESPONSEH+108] = 'I';
    allValues[RESPONSEH+109] = 'F';
  }
  else if(iValues.input.wlan) {
    allValues[RESPONSEH+105] = 'W';
    allValues[RESPONSEH+106] = 'L';
    allValues[RESPONSEH+107] = 'A';
    allValues[RESPONSEH+108] = 'N';
    allValues[RESPONSEH+109] = '0';
  }

  if(iValues.mute){
    allValues[RESPONSEH+6] = 't';
    allValues[RESPONSEH+7] = 'r';
    allValues[RESPONSEH+8] = 'u';
    allValues[RESPONSEH+9] = 'e';
    allValues[RESPONSEH+10] = '0';
  }else{
    allValues[RESPONSEH+6] = 'f';
    allValues[RESPONSEH+7] = 'a';
    allValues[RESPONSEH+8] = 'l';
    allValues[RESPONSEH+9] = 's';
    allValues[RESPONSEH+10] = 'e';
  }
}
```

```
//read, parse and respond to http request
static void handleHttpRequest(){
  int clientsock = connectlist[0];
  static char buff[1000]; //buffer to store received message
  char error_response[23]= {'H','T','T','P','/','1','.','0',' ','4','0','4',' ',
                            'N','o','t',' ','F','o','u','n','d','\n'};
  char OK_response[17]= {'H','T','T','P','/','1','.','0',' ','2','0','0',' ','O',
                         'K','\n','\n'};
  int bytes = recv(clientsock, buff, 1000, 0); //read response from wlan
  if (bytes <= 0) {
    closesocket(clientsock);
    printk("failed recv\n");
    return;
  }

  buff[bytes] = '\0'; //null terminate for search to work
  /*printk("bytes received: %d\nmessage:",bytes);
  for (int i = 0; i < bytes; i++) {
    printk("%c",buff[i]);
  }
  printk("\n");*/

  xSemaphoreTake(sendingDoneSemaphore, 0); //reset
  int sent = -1;

  if(strstr (buff, "HTTP/1.0") != NULL || strstr (buff, "HTTP/1.1") != NULL){
    if(strstr (buff, "GET") != NULL){
      char* request = strchr(buff,'?'); //check for ? to indicate settings request
      if (request != NULL) { //update values request
        request++; //remove ? from request
        if(strncmp(request,"update",4) == 0){ //sends update with all the current values
          sendUpdate();
          sent = TCPsend(clientsock, allValues,sizeof(allValues));
        }
        //Handle commands
        else{
          int i = (request[0] == '&' ? 1 : 0); //fixes a rare bug in javascript
          char* command = &request[i];
          while(i<bytes) {
            //replaces "~" with " " in request
            if(request[i] == '~'){
              request[i] = ' ';
            }
            //meaning there is another command after this one
            else if (request[i] == '&') {
              request[i] = '\0'; //add null termination
              //send message directly as a string
              CmdChan_ProcessRow(PACKET_TYPE_WIFI ,command,false);
              //printk("command: %s\n" , command);
              command = &request[i+1];
            }
            //
            else if (request[i] == ' ') {
              request[i] = '\0';
              CmdChan_ProcessRow(PACKET_TYPE_WIFI ,command,false);
              //printk("command: %s\n" , command);
              break;
            }
            i++;
          }
          //send response
          sent = TCPsend(clientsock, OK_response,sizeof(OK_response));
        }
      }
      else if (strstr(buff," / ")){ //site request
        sent = TCPsend(clientsock, site, SITE_SZ);
      }
      else { //nothing supported
        sent = TCPsend(clientsock, error_response, 23);
      }
    }
  }
  else{ //nothing supported
```

```
    sent = TCPsend(clientsock, error_response, 23);
  }
  //wait for sending done before closing socket
  if (sent == 0)
    xSemaphoreTake(sendingDoneSemaphore, portMAX_DELAY);
  closesocket(clientsock);
  connectlist[0] = 0;
}
```

code/remotewifi.h

```
#ifndef _REMOTE_WIFI_H_
#define _REMOTE_WIFI_H_

#include "packet_list.h"
#include <FreeRTOS.h>
#include <task.h>
#include <semphr.h>

extern xSemaphoreHandle connectSemaphore;
extern xSemaphoreHandle DHCPSemaphore;

void wifi_SendPartialPacket(const packet_t* p,uint16_t* sendOffset);

void wifi_init(void);

//return 1 if connected, zero otherwise
//ip_addr is 4 bytes long and is filled in by the function
int wifi_status(unsigned char* ip_addr);

//security - no security = 0, WEP = 1, WPA = 2, WPA2 = 3
void wifi_connect(int security, unsigned char* ssidPtr, int ssidLen, unsigned char* keyPtr
     , int keyLen);

void wifi_smartConfig(unsigned char* key);

void wifi_serverInit(void);

void wifi_checkHttpServer(void);

void wifi_reset(void);

#endif
```

A.3 Web Interface Code

code/remote.js

```
var pendingCommand = false; //Flag for pending commands
var sendingFlag = false; //blocking while sending
var command2Send = "?"; //pending command queue
var x; //xmlHttpRequeuest

//button event handler
function bClick(obj)
{
    if(sendingFlag){ //if already sending data put command in queue
        command2Send += "&"+obj.name;
                timeoutPendingCommand();
    }else{
            x = new XMLHttpRequeuest();

            x.open( "GET", "?"+obj.name, true );
            x.send( null );
            sendingFlag = true;

        x.onreadystatechange = checkPendingCommands;
    }
}

//radio button event handler
function radioClick(obj)
{
    if(sendingFlag){//if already sending data put command in queue
        command2Send += "&"+obj.value;
        timeoutPendingCommand();
    }else{
                x = new XMLHttpRequeuest();
                x.open( "GET", "?"+obj.value, true );
                x.send( null );

        sendingFlag = true;
        x.onreadystatechange = checkPendingCommands;
    }
}

//dropdown
function ddClick(obj)
{
    if(sendingFlag){//if already sending data put command in queue
        command2Send += "&"+obj.name+"~"+obj.value;
        timeoutPendingCommand();
    }else{
            x = new XMLHttpRequeuest();

            x.open( "GET", "?"+obj.name+"~"+obj.value, true );
            x.send( null );

            sendingFlag = true;
        x.onreadystatechange = checkPendingCommands;
    }
}

//checkbox event handler
function cbClick(obj)
{
    if(sendingFlag){//if already sending data put command in queue
        if(obj.checked){
                command2Send += "&"+obj.name;
            }else{
                command2Send += "&"+obj.value;
            }
                timeoutPendingCommand();
    }else{
            x = new XMLHttpRequeuest();
            if(obj.checked){
                x.open( "GET", "?"+obj.name, true );
            }else{
                x.open( "GET", "?"+obj.value, true);
            }

            x.send( null );
            sendingFlag = true;
            x.onreadystatechange = checkPendingCommands;
    }
}

//Slider event handler
var flag=0;
var pendingEvent = 0;
var pendingObj = null;
var timer = null;
function slChange(obj)
{
    //calculate how many decimals to use to optimaly fill the five byte avalble
        var decimals = Math.round(-Math.log(parseFloat((obj.step),10))/Math.LN10);
        if (decimals < 0)
                decimals = 0;
        var sliderValue = parseFloat(obj.value,10).toFixed(decimals);
        //updating the number over the slider
        var numberid = obj.id+"Out";
        document.getElementById(numberid).innerHTML = sliderValue;

    //checks if slider has changed the last 400ms and then sends
        clearTimeout(timer);
    timer = setTimeout(function(){
        //calculate how many decimals to use to optimaly fill the five byte avalble
        var decimals =  Math.round(-Math.log(parseFloat((obj.step),10))/Math.LN10);
        if (decimals < 0)
                decimals = 0;

        //updating the number over the slider
        var numberid = obj.id+"Out";
        document.getElementById(numberid).innerHTML = sliderValue;

        if(sendingFlag){ //if already sending data put command in queue
        if(parseInt(obj.value) > parseFloat(obj.max)){
                    command2Send += "&"+obj.name+"~"+obj.max;
            }else if(parseInt(obj.value) < parseInt(obj.min)){
                    command2Send += "&"+obj.name+"~"+obj.min;
            }else{
                    command2Send += "&"+obj.name+"~"+sliderValue ;
            }
            timeoutPendingCommand();
    }else{
            x = new XMLHttpRequeuest();
        if(parseInt(obj.value) > parseFloat(obj.max)){
                x.open( "GET", "?"+obj.name+"~"+obj.max, true );
        }else if(parseInt(obj.value) < parseInt(obj.min)){
                x.open( "GET", "?"+obj.name+"~"+obj.min, true);
        }else{
                x.open( "GET", "?"+obj.name+"~"+sliderValue , true );
        }
        x.send( null );
        sendingFlag = true;
        x.onreadystatechange = checkPendingCommands;
        }
    },400);
}

//Checks range slider support
function checkSLsupport(){
    var i = document.createElement("input");
    i.setAttribute("type", "range");
    return i.type !== "text";
}

//Removes extra zeros in begining of string
function removeZeroBegining(s){
        for (var i = 0; i < s.length; i++){
                if(s[i] != 0){
                        if(s[i] == '.')
                                return s.substring(i-1);
                        else
                                return s.substring(i);
                }
        }
        return "0";
}

window.onload = updateValues; //fire on wedsite load
//Update all values on website
var once = 0;
function updateValues(){

    //send requeuest for current status of DSP
    x = new XMLHttpRequeuest();
    x.open("GET","?update",false);
    x.send(null);

    var response = x.responseText;
    //Example response: ?mute=true0&mvol=00012&rvol=00100&lvol=00100&vlfu=0.111&bfre
```

```
        =40000&bbos=00.03&bqqq=0.018&bgai=000.9&inpt=USB00

        //Parse response
    if(document.getElementsByName("preset~recall~Mute.upr")[0] != null){ //check if widget
          is present in layout
        inputValue = response.substr(response.indexOf("mute")+5,5);
        if(inputValue.indexOf("false") != -1){
            document.getElementsByName("preset~recall~Mute.upr")[0].checked = true;
        }else{
            document.getElementsByName("preset~recall~Mute.upr")[0].checked = false;
        }
    }

        if(document.getElementsByName("vol~set")[0]  != null){
                var o = document.getElementsByName("vol~set")[0];
                var r = response.substr(response.indexOf("mvol")+5,5);
                r = removeZeroBegining(r);
                o.value = r;
                var s = o.id.toString();
                s= s+"Out";
                document.getElementById(s).innerHTML = r;
        }

        if(document.getElementsByName("vol~set~l")[0]  != null){
                var o = document.getElementsByName("vol~set~l")[0];
                var r = response.substr(response.indexOf("Lvol")+5,5);
                r = removeZeroBegining(r);
                o.value = r;
                var s = o.id.toString();
                s= s+"Out";
                document.getElementById(s).innerHTML = r;
        }

        if(document.getElementsByName("vol~set~r")[0]  != null){
                var o = document.getElementsByName("vol~set~r")[0];
                var r = response.substr(response.indexOf("rvol")+5,5);
                r = removeZeroBegining(r);
                o.value = r;
                var s = o.id.toString();
                s= s+"Out";
                document.getElementById(s).innerHTML = r;
        }

    if(document.getElementsByName("uservalue~set~VeryLongNameForAUser")[0] != null){
                var o = document.getElementsByName("uservalue~set~VeryLongNameForAUser")
                    [0];
                var r = response.substr(response.indexOf("vlfu")+5,5);
                r = removeZeroBegining(r);
                o.value = r;
                var s = o.id.toString();
                s= s+"Out";
                document.getElementById(s).innerHTML = r;
    }

    if(document.getElementsByName("biquad~set~PEQUserTarget-1_1[0..4]~F0")[0] != null){
                var o = document.getElementsByName("biquad~set~PEQUserTarget-1_1[0..4]~F0"
                    )[0];
                var r = response.substr(response.indexOf("bfre")+5,5);
                r = removeZeroBegining(r);
                o.value = r;
                var s = o.id.toString();
                s= s+"Out";
                document.getElementById(s).innerHTML = r;
    }

    if(document.getElementsByName("biquad~set~PEQUserTarget-1_1[0..4]~BC")[0] != null){
                var o = document.getElementsByName("biquad~set~PEQUserTarget-1_1[0..4]~BC"
                    )[0];
                var r = response.substr(response.indexOf("bbos")+5,5);
                r = removeZeroBegining(r);
                o.value = r;
                var s = o.id.toString();
                s= s+"Out";
                document.getElementById(s).innerHTML = r;
    }

    if(document.getElementsByName("biquad~set~PEQUserTarget-1_1[0..4]~Q")[0] != null){
                var o = document.getElementsByName("biquad~set~PEQUserTarget-1_1[0..4]~Q")
                    [0];
                var r = response.substr(response.indexOf("bqqq")+5,5);
                r = removeZeroBegining(r);
                o.value = r;
                var s = o.id.toString();
                s= s+"Out";
                document.getElementById(s).innerHTML = r;
    }

    if(document.getElementsByName("biquad~set~PEQUserTarget-1_1[0..4]~G")[0] != null){
                var o = document.getElementsByName("biquad~set~PEQUserTarget-1_1[0..4]~G")
                    [0];
                var r = response.substr(response.indexOf("bgai")+5,5);
                r = removeZeroBegining(r);
                o.value = r;
                var s = o.id.toString();
                s= s+"Out";
                document.getElementById(s).innerHTML = r;
    }

    //update drop down and radiobuttons position
    var inputValue = response.substr(response.indexOf("inpt")+5,5);
    inputValue = inputValue.replace(/0/g,""); //strip padding if needed

    inputValue = "preset~recall~"+inputValue+"*.fpr";
    var elementsName = document.getElementsByName("pre");
    var elementsId = document.getElementById("pre1");
    updateRadioAndDD(elementsName,elementsId,inputValue);

    if(!checkSLsupport() && once == 0){
        alert("No slider support, please choose another browser for a better experience");
        once=1;
    }

}

//Function for handling if there are commands waiting to be sent while waiting for server
    to anwser
function checkPendingCommands(){
    if(x.readyState == 4){
        if(pendingCommand){
                x = new XMLHttpRequeuest();
                x.open( "GET", command2Send, true );
                x.send( null );
                command2Send = "?";
                pendingCommand = false;
                x.onreadystatechange = checkPendingCommands;
        }else{
            sendingFlag = false;
        }
    }
}

/*timeout for pendingCommands
If no answer is recived 5 seconds after a command is sent to server
the pending command2Send queue  will be sent*/
function timeoutPendingCommand(){
         pendingCommand = true;
         setTimeout(function(){
                if(pendingCommand){
                x = new XMLHttpRequeuest();
                x.open( "GET", command2Send, true );
                x.send( null );
                command2Send = "?";
                pendingCommand = false;
        }else{
            sendingFlag = false;
        } },5000);
```

```
}

//Goes throught name array of radiobutton and Dropdown and updates the values
function updateRadioAndDD(elementsName,elementsId, changeVal){
    for (var i=0; i<elementsId.length; i++){
            if(elementsId.options[i].value == changeVal) {
                    elementsId[i].checked = true;
                    elementsId[i].selected = true;
            }
    }
    for ( var i = 0; i < elementsName.length; i++) {
            if(elementsName[i].value == changeVal) {
                    elementsName[i].checked = true;
                    elementsName[i].selected = true;
            }
    }
}
```

code/xml2html.py

```
import base64
from math import log, ceil



#Class used to build the HTML file
class htmlFile:

    def __init__(self,newfilename,backgroundColor,forgroundColor,title):
        self.inGroup = False
        self.f = open(newfilename, 'w')
        self.f.write('HTTP/1.0 200 OK\nContent-type: text/html\n\n<!DOCTYPE html>\n<html>\
            n<head>\n<title>'+title+'</title>')
        self.f.write('\n<script type="text/javascript">function bClick(e){if(sendingFlag){
            command2Send+="&"+e.name;timeoutPendingCommand()}
        else{x=new XMLHttpRequest;x.open("GET","?"+e.name,true);x.send(null);sendingFlag=
            true;x.onreadystatechange=checkPendingCommands}}
        function radioClick(e){if(sendingFlag){command2Send+="&"+e.value;
            timeoutPendingCommand()}else{x=new XMLHttpRequest;x.open("GET","?"+e.value,
            true);
        x.send(null);sendingFlag=true;x.onreadystatechange=checkPendingCommands}}function
            ddClick(e){if(sendingFlag)
        {command2Send+="&"+e.name+"~"+e.value;timeoutPendingCommand()}else{x=new
            XMLHttpRequest;x.open("GET","?"+e.name+"~"+e.value,true);
        x.send(null);sendingFlag=true;x.onreadystatechange=checkPendingCommands}}function
            cbClick(e){if(sendingFlag){if(e.checked)
        {command2Send+="&"+e.name}else{command2Send+="&"+e.value}timeoutPendingCommand()}
            else{x=new XMLHttpRequest;if(e.checked){x.open
        ("GET","?"+e.name,true)}else{x.open("GET","?"+e.value,true)}x.send(null);
            sendingFlag=true;x.onreadystatechange=checkPendingCommands}}
        function slChange(e){var t=Math.round(-Math.log(parseFloat(e.step,10))/Math.LN10);
            if(t<0)t=0;var n=parseFloat(e.value,10).toFixed(t);
        var r=e.id+"Out";document.getElementById(r).innerHTML=n;clearTimeout(timer);timer=
            setTimeout(function
        var t=Math.round(-Math.log(parseFloat(e.step,10))/Math.LN10);if(t<0)t=0;var r=e.id
            +"Out";document.getElementById(r).innerHTML=n;
        if(sendingFlag){if(parseInt(e.value)>parseFloat(e.max)){command2Send+="&"+e.name
            +"~"+e.max}else if(parseInt(e.value)
        <parseInt(e.min)){command2Send+="&"+e.name+"~"+e.min}else{command2Send+="&"+e.name
            +"~"+n}timeoutPendingCommand()}
        else{x=new XMLHttpRequest;if(parseInt(e.value)>parseFloat(e.max)){x.open("GET
            ","?"+e.name+"~"+e.max,true)}else if(parseInt(e.value)
        <parseInt(e.min)){x.open("GET","?"+e.name+"~"+e.min,true)}else{x.open("GET","?"+e.
            name+"~"+n,true)}x.send(null);sendingFlag=true;
        x.onreadystatechange=checkPendingCommands}},400)}function checkSLsupport(){var e=
            document.createElement("input");
        e.setAttribute("type","range");return e.type!=="text"}function removeZeroBegining(
            e){for(var t=0;t<e.length;t++){
```

```
if(e[t]!=0){if(e[t]==".")return e.substring(t-1);else return e.substring(t)}}
    return"0"}function updateValues(){
x=new XMLHttpRequest;x.open("GET","?update",false);x.send(null);var e=x.
    responseText;
if(document.getElementsByName("preset~recall~Mute.upr")[0]!=null){i=e.substr(e.
    indexOf("mute")+5,5);if(i.indexOf("false")!=-1)
{document.getElementsByName("preset~recall~Mute.upr")[0].checked=true}else{
    document.getElementsByName("preset~recall~Mute.upr")[0].checked=false}}
if(document.getElementsByName("vol~set")[0]!=null){var t=document.
    getElementsByName("vol~set")[0];var n=e.substr(e.indexOf("mvol")+5,5);
n=removeZeroBegining(n);t.value=n;var r=t.id.toString();r=r+"Out";document.
    getElementById(r).innerHTML=n}if(document.getElementsByName("vol~set~l")[0]!=
    null)
{var t=document.getElementsByName("vol~set~l")[0];var n=e.substr(e.indexOf("Lvol")
    +5,5);n=removeZeroBegining(n);
t.value=n;var r=t.id.toString();r=r+"Out";document.getElementById(r).innerHTML=n}
    if(document.getElementsByName("vol~set~r")[0]!=null)
{var t=document.getElementsByName("vol~set~r")[0];var n=e.substr(e.indexOf("rvol")
    +5,5);n=removeZeroBegining(n);t.value=n;
var r=t.id.toString();r=r+"Out";document.getElementById(r).innerHTML=n}if(document
    .getElementsByName("uservalue~set~VeryLongNameForAUser")[0]!=null)
{var t=document.getElementsByName("uservalue~set~VeryLongNameForAUser")[0];var n=e
    .substr(e.indexOf("vlfu")+5,5);
n=removeZeroBegining(n);t.value=n;var r=t.id.toString();r=r+"Out";document.
    getElementById(r).innerHTML=n}
if(document.getElementsByName("biquad~set~PEQUserTarget-1_1[0..4]~F0")[0]!=null){
    var t=document.getElementsByName("biquad~set~PEQUserTarget-1_1[0..4]~F0")[0];
var n=e.substr(e.indexOf("bfre")+5,5);n=removeZeroBegining(n);t.value=n;var r=t.id
    .toString();
r=r+"Out";document.getElementById(r).innerHTML=n}if(document.getElementsByName("
    biquad~set~PEQUserTarget-1_1[0..4]~BC")[0]!=null){
var t=document.getElementsByName("biquad~set~PEQUserTarget-1_1[0..4]~BC")[0];var n
    =e.substr(e.indexOf("bbos")+5,5);
n=removeZeroBegining(n);t.value=n;var r=t.id.toString();r=r+"Out";document.
    getElementById(r).innerHTML=n}
if(document.getElementsByName("biquad~set~PEQUserTarget-1_1[0..4]~Q")[0]!=null){
    var t=document.getElementsByName("biquad~set~PEQUserTarget-1_1[0..4]~Q")[0];
var n=e.substr(e.indexOf("bqqq")+5,5);n=removeZeroBegining(n);t.value=n;var r=t.id
    .toString();r=r+"Out";document.getElementById(r).innerHTML=n}
if(document.getElementsByName("biquad~set~PEQUserTarget-1_1[0..4]~G")[0]!=null){
    var t=document.getElementsByName("biquad~set~PEQUserTarget-1_1[0..4]~G")[0];
var n=e.substr(e.indexOf("bgai")+5,5);n=removeZeroBegining(n);t.value=n;var r=t.id
    .toString();r=r+"Out";
document.getElementById(r).innerHTML=n}var i=e.substr(e.indexOf("inpt")+5,5);i=i.
    replace(/0/g,"");i="preset~recall~"+i+"*.fpr";
var s=document.getElementsByName("pre");var o=document.getElementById("pre1");
    updateRadioAndDD(s,o,i);if(!checkSLsupport()&&once==0){
alert("No slider support, please choose another browser for a better experience");
    once=1}}function checkPendingCommands(){if(x.readyState==4){
if(pendingCommand){x=new XMLHttpRequest;x.open("GET",command2Send,true);x.send(
    null);command2Send="?";
pendingCommand=false;x.onreadystatechange=checkPendingCommands}else{sendingFlag=
    false}}}function timeoutPendingCommand(){
pendingCommand=true;setTimeout(function(){if(pendingCommand){x=new XMLHttpRequest;
    x.open("GET",command2Send,true);x.send(null);
command2Send="?";pendingCommand=false}else{sendingFlag=false}},5e3)}function
    updateRadioAndDD(e,t,n){for(var r=0;r<t.length;r++){
if(t.options[r].value==n){t[r].checked=true;t[r].selected=true}}for(var r=0;r<e.
    length;r++){if(e[r].value==n){e[r].checked=true;e[r].selected=true}}}
var pendingCommand=false;var sendingFlag=false;var command2Send="?";var x;var flag
    =0;var pendingEvent=0;var pendingObj=null;
var timer=null;window.onload=updateValues;var once=0</script>')
self.f.write('\n<meta name="viewport" content="width=device-width">')
self.f.write('\n<style>button[type="button"]{width:98%; height:50px; background-
    color:#787878;-moz-border-radius: 10px; border-radius: 10px;}
input[type="range"]{ background-color: #333; color: #fff; width:95%; } input[type
    ="range"]::-webkit-slider-thumb{ background-color:#888; height:20px; width:20
    px; }
body{background-color:'+backgroundColor+';} #wrapper{margin:auto; width:90%;
    background-color:'+forgroundColor+'; box-shadow:10px 10px 30px black; padding-
    bottom:1%;}
h3{margin:1%;} .DD{margin:1%;} .CB{margin:1%;} .SL{margin:1%;} .RB{margin:1%;} #
    logo{ margin:1%;} @media screen and (min-width: 600px) {.BT{margin-left:1%;
```

```
            width:25%;}
        .hbox{margin-left:3%; display: inline-block; width:45%;}} @media screen and (max-
            width: 601px) {.BT{margin-left:1%; width:100%;}
        .hbox{margin-left:3%; display: inline-block; width:98%;}} .gbox{margin:1%; border-
            style:ridge; }</style>')
        self.f.write('</head>\n<body>\n')
        self.f.write('<div id="wrapper">')
        self.img2base64()

    def closehtmlFile(self):
        self.f.write("</div></body >\n</html>")
        self.f.close()

    #Encode logo as Base64 for faster loading of web interface
    def img2base64(self):
        data_uri = base64.encodestring(open("logo.png","rb").read())
        img_tag = '<center><img alt="loading" src="data:image/png;base64,{0}">'.format(
            data_uri)
        self.f.write('<div id="logo">'+img_tag+'</div></center>')

    #Radio button
    def addRb2Html(self,val,cmd):
        if self.inGroup:
            addHboxStart()
        self.f.write('<input type="radio" name="'+cmd[0 : 3]+'" onclick="radioClick(this)"
             value="'+cmd+'">'+val+'<br/>\n')

        if self.inGroup:
            endDiv()

    def startRadio(self):
        self.f.write('<div class="RB"><form>\n')
    def endRadio(self):
        self.f.write("</form></div>\n")

    #Checkbox
    def addCb2Html(self,val,cmdOn,cmdOff):
        if self.inGroup:
            addHboxStart()
        self.f.write('<div class="CB"><form>\n')
        self.f.write('<input type="checkbox" name="'+cmdOn+'" onclick="cbClick(this)"
            value="'+cmdOff+'">'+val+'<br/>\n')
        self.f.write("</form></div>\n")
        if self.inGroup:
            endDiv()
    #Button
    def addButton2Html(self,cmd,val):
        self.f.write('<div class="BT">\n')
        self.f.write('<button type="button" name="'+cmd+'" onclick="bClick(this)">'+val+'
            </button></br/>\n')
        self.f.write("</div>\n")

    #Dropdown
    def addDd2Html(self,cmd):
        self.f.write('<div class="DD"><select id="'+cmd[0 : 3]+'1" onchange="ddClick(this)
            ">\n')

    #Add item to DropDown
    def addItemDD(self,val,cmd):
        self.f.write('<option value="'+cmd+'">'+val+'</option>\n')

    def endDropDown(self):
        self.f.write("</select></div>\n")

    #Range slider
    def addSl2Html(self,maxVal,minVal,text, cmd):
        if self.inGroup:
            self.addHboxStart()
        r = 10**ceil(log(float(maxVal)-float(minVal),10)-3); #calculate stepsize of slider
        if r >= 10:
            r = r / 10
        iDtext= text.translate(None, ' !.;,/-*')
        self.f.write('<div class="SL"><form>\n')
        self.f.write('<div style="float:left">'+text+':&nbsp </div><div id="'+iDtext+'Out"
             style="float:left">0</div><br/> '); #add sider text and slider value
            indicator
        self.f.write('<input type="range" name="'+cmd+'" id="'+iDtext+'" value="'+str(
            minVal)+'" onchange="slChange(this)"
        max="'+str(maxVal)+'" min="'+str(minVal)+'" step="'+str(r)+'" ><br/>\n')
        self.f.write("</form></div>\n")
        if self.inGroup:
            self.endDiv()

    #Add hbox start
    def addHboxStart(self):
        self.f.write('\n<div class="hbox">')

    #Add start groupbox
    def addGroup(self,lable):
        self.inGroup = True
        self.f.write('<div class="gbox">\n<h3>'+lable+'</h3>')

    #Add div end for groupbox
    def endGroup(self):
        self.f.write('</div>\n')
        self.inGroup = False

    #Add end div
    def endDiv(self):
        self.f.write('</div>\n')

    #add label
    def addLabel(self,label):
        self.f.write('<h3>'+label+'</h3>')

#Class that parses XML file
class parseXML:

    #Gets text that is between ">" (text) "<"
    def getText(self,line,xmlfile):

        #removes comments from line
        commentStart = line.find("<!--")

        if commentStart != -1:
            line=line[ : commentStart]
        text = "xml error"
        while 1:
            if line.find('>') != -1: break
            line=xmlfile.next()
             #removes comments from line
            commentStart = line.find("<!--")
            if commentStart != -1:
                line=line[ : commentStart]
        startIndex = line.find('>')+1
        if "<" in line[startIndex : ]: #text on one line
            text = line[startIndex : line.index('</',startIndex)]
            return text
        elif line[startIndex].isalpha(): #text on one line end on next
            text = line[startIndex : ]
            return text
        else:   #text on next line
            line = xmlfile.next()
            while 1: #jump intill text is found
                if line.strip(' \t\n\r   '): break
                line = xmlfile.next()
            if "</" in line: #end on same line as text
                text = line[ : line.index('</')]
                return text
            else:   #only text on line
                text = line
                return text

    #gets cmd attribute
    def getCMD(self,line,xmlfile,name):
        cmd="xml error"
        while 1:
            #print line
            if line.find(name) != -1: break
            line = xmlfile.next()
            #removes comments from line
            commentStart = line.find("<!--")
            if commentStart != -1:
                line=line[ : commentStart]
        startIndex = line.index(name+'="')+len(name)+2
        cmd = line[startIndex : line.index('"',startIndex)]
        cmd = cmd.replace(" ","~")
        return cmd

    #Parse xml file
    def ScanXml(self,filename,bg,fg):
            i = 0
            inDropDown = False
            inRadioButton = False
            xmlfile = open(filename, 'r')
            for l in xmlfile: #scan for back/foreground colors
                if "background" in l:
                    bg=self.getCMD(l, xmlfile, "color")
                if "forground" in l:
                    fg=self.getCMD(l,xmlfile, "color")
            xmlfile.close()
            xmlfile = open(filename, 'r')
            webHtml =  htmlFile('remote.htm',bg,fg,"Remote")
            for line in xmlfile:
                i += 1
                if "<title>" in line:
                    title = self.getText(line,xmlfile)
                    webHtml = htmlFile('remote.htm',bg,fg,title)

                elif "radiobutton" in line:
                    if not inRadioButton:
                        webHtml.startRadio()
                        inRadioButton = True
                    cmd = self.getCMD(line,xmlfile,"cmd")
                    val = self.getText(line,xmlfile)
                    webHtml.addRb2Html(val,cmd)

                elif "</vbox>" in line and inRadioButton:
                    webHtml.endRadio()
                    inRadioButton = False

                elif "numeric" in line:
                    minRange = 0
                    maxRange = 100
                    val = self.getText(line,xmlfile)
                    cmd = self.getCMD(line,xmlfile,"cmd")
                    if "min" in line:
                        minRange = self.getCMD(line,xmlfile,"min")
                    if "max" in line:
                        maxRange = self.getCMD(line,xmlfile,"max")
                    webHtml.addSl2Html(maxRange,minRange,val, cmd)

                elif "<groupbox" in line:
                    startIndex = line.index('title="')+7
                    text = line[startIndex : line.index('"',startIndex)]
                    webHtml.addGroup(text)

                elif "</groupbox" in line:
                    webHtml.endGroup()

                elif "<label>" in line:
                    val = self.getText(line,xmlfile)
                    webHtml.addLabel(val)

                elif "<combobox>" in line:
                    inDropDown = True
                    while not "<item" in line:
                        line = xmlfile.next()
                    cmd = self.getCMD(line,xmlfile,"cmd")
                    val = self.getText(line,xmlfile)
                    webHtml.addDd2Html(cmd) #add start to dropdown
                    webHtml.addItemDD(val,cmd) #add first item to dropdown list

                elif "</combobox>" in line:
                    inDropDown=False
                    webHtml.endDropDown()

                elif "<item" in line and inDropDown:
                    val = self.getText(line,xmlfile)
                    cmd = self.getCMD(line,xmlfile,"cmd")
                    webHtml.addItemDD(val,cmd)

                elif "<checkbox" in line:
                    cmdOn = self.getCMD(line,xmlfile,"on")
                    cmdOff = self.getCMD(line,xmlfile,"off")
                    val = self.getText(line,xmlfile)
                    webHtml.addCb2Html(val,cmdOn,cmdOff)
                elif "<button" in line:
                    val = self.getText(line,xmlfile)
                    cmd = self.getCMD(line,xmlfile,"cmd")
                    webHtml.addButton2Html(cmd,val)

            webHtml.closehtmlFile()

class data2Array:

    def __init__(self,filename,ArrayName):
        self.myfile = open(filename, 'r')
        self.ArrayName = ArrayName

    #Trasnforms HTML data into array with HEX values
    def transform2Array(self):
        htmlArray = ''
        cnt = 0
        br = 0
        while 1:
            char = self.myfile.read(1)          # read by character
            if char == "": break
            cnt += 1
            br += 1
            htmlArray += "\\x%02x" % ord(char)
            if br >= 19:
                htmlArray += "\\\n"
                br=0

        add_nulls = lambda number, zero_count : "{0:0{1}d}".format(number, zero_count)

        cnt = add_nulls(cnt,6) #padds cnt with leading zeros
        htmlArray = str(cnt) + "\n" + htmlArray #add cnt header to file
        self.myfile.close()
        file1 = open("httpremote", 'w')
        file1.write(str(htmlArray))
        file1.close()

filename = 'zygote.xml'
print 'Starting XML2HTML'
p=parseXML()
p.ScanXml(filename,'#127533','#FfFfFf') #Standard colors of nothing is specifide in XML
     file
site = data2Array("remote.htm","site")
site.transform2Array()
print "done ^__^"
```

code/zygote.xml

```
<window icon="bohmeraudio.ico">


  <background color="#127533">
  <forground color="#FfFfFf">
  <title>Remote control</title>
  <vbox>

    <numeric cmd="vol set" >Master volume</numeric>  <!-- "vol get": LED intensity -->

    <checkbox on="preset recall Mute.upr" off="preset recall Unmute.upr">
      Mute
    </checkbox>

    <hbox>
      <label>Input source</label>
      <vbox>
        <radiobutton cmd="preset recall USB*.fpr">USB</radiobutton><!-- "preset test USB*.
             fpr": LED color -->
        <radiobutton cmd="preset recall Phono*.fpr">Phono </radiobutton>
        <radiobutton cmd="preset recall SPDIF*.fpr">SPDIF</radiobutton>
        <radiobutton cmd="preset recall WLAN*.fpr">WLAN</radiobutton>
      </vbox>
    </hbox>

    <hbox>
      <label>Sound Profiles</label><!-- made up profiles -->
      <combobox>
        <item cmd="preset recall profile Movie">Movie</item>
        <item cmd="preset recall profile Speach">Speech</item>
        <item cmd="preset recall profile Music">Music</item>
        <item cmd="preset recall profile Pary">Party</item>
      </combobox>
    </hbox>


    <groupbox title="Advanced Features">
      <hbox>
        <numeric cmd="biquad set PEQUserTarget-1_1[0..4] F0" min="0" max="48000">Frequency
             </numeric> <!-- max min range added by anders-->
        <numeric cmd="biquad set PEQUserTarget-1_1[0..4] BC">Boost / Cut</numeric>
        <numeric cmd="biquad set PEQUserTarget-1_1[0..4] Q" min="0" max="1">Q</numeric>
        <numeric cmd="biquad set PEQUserTarget-1_1[0..4] G">Gain</numeric>
      </hbox>
    </groupbox>
    <button cmd="button press">Press</button> <!-- added by Anders-->
  </vbox>
</window>
```