TALLINN UNIVERSITY OF TECHNOLOGY
Faculty of Information Technology
Thomas Johann Seebeck Department of Electronics
Electronics and Bionics

Sven Kautlenbach

MSP430 Based Wireless Sensor Network

Master Thesis (30 ECTS)

Supervisor: Olev Märtens

Tallinn 2014

Abstract

Structural health monitoring has been the extensive research topic for years. Solutions are mostly applied to the civil engineering constructions, where the monitoring could avoid accidents, caused by the malfunction and cracking of the structures. With the rapidly developing semiconductor technologies the researches have started to design and evaluate wireless monitoring nodes. The advantages of wireless sensing nodes over wired ones are disparate. Using wireless data communication enables more mobility, less dependability on environment, faster installation times, less costs, more expandability and in general more dynamic system. The small size associated with the wireless nodes mean that they could also be integrated into the materials.

The problems with the energy consumption, wireless link data throughput and latency are the main issues halting the widespread adaptation of these systems. Alleviating the cons of wireless systems are in focus of the research topics. Therefore testing new semiconductor devices and designs is needed. This work gives condensed overview of different wireless sensor designs covering different approaches to the problems. The state of the art of transceiver devices of different manufacturers and their portfolio is given.

Finally the Texas Instruments MSP430 platform is taken into the focus of this work. The CC430 family of transceiver microcontrollers is evaluated. The Texas Instruments SimpliciTI wireless network library, which is part of Chronos development kit, is modified and adapted for the requirements set in the work. The work grades the performance of the system, measures the basic properties and proposes solutions to overcome the shortcomings and the problems of the system.

The work was conducted as a part of the Archimedes project AR 12139 "Smart Composites – Design and Manufacturing".

Table of contents

Table of figures

List of tables

Abbreviations

A/D – Analog-Digital

ACK – Acknowledgment

ADC – Analog-to-Digital Converter

API – Application Programming Interface

ASK – Amplitude Shift Keying

CCA – Clear Channel Assessment

COM – Communication port

CRC – Cyclic Redundancy Check

CSV – Comma-Separated Values

DAC – Digital-to-Analog Converter

DMA – Direct Memory Access

DRAM – Dynamic Random-Access Memory

DSP – Digital Signal Processor

DSSS – Direct-Sequence Spread Spectrum

FCC – Federal Communications Commission

FCS – Frame Check Sequence

FFT – Fast Fourier Transform

FHSS – Frequency-Hopping Spread Spectrum

FIFO – First In, First Out

FSK – Frequency-Shift Keying

GCC – GNU Compiler Collection

GUI – Graphical User Interface

HTTP – Hypertext Transfer Protocol

I/O – Input/Output

I^2C – Inter-Integrated Circuit

IC	Integrated Circuit
IDE	Integrated Development Environment
IPv6	Internet Protocol version 6
ISM	Industrial, Scientific, Medical
JTAG	Joint Test Action Group
LCD	Liquid-Crystal Display
LIN	Local Interconnect Network
LoS	Line-of-Sight
LQI	Link Quality Indicator
MIPS	Million Instructions per Second
MOTE	sensor mote/node
MSK	Minimum-Shift Keying
NFC	Near Field Communication
OOK	On-Off Keying
OS	Operating System
PCB	Printed Circuit Board
PWM	Pulse Width Modulation
RAM	Random Access Memory
RISC	Reduced Instruction Set Computing
ROM	Read-Only Memory
RSSI	Received Signal Strength Indicator
RTC	Real-time clock
SCI	Serial Communication Interface
SHM	Structural Health Monitoring
SoC	System on Chip
SPI	Serial Peripheral Interface
SRAM	Static Random-Access Memory

TCP/IP – Transmission Control Protocol/Internet Protocol

UART – Universal Asynchronous Receiver/Transmitter

UDP – User Datagram Protocol

ULP – Ultra Low Power

USB – Universal Serial Bus

WLAN – Wireless Local Area Network

1 Introduction

An Archimedes funded project AR 12139 “Smart Composites – Design and Manufacturing” at the Tallinn University of Technology was led by then post-graduate Henrik Herranen for his doctoral studies in the mechanical engineering field. The goal of the group was to develop and research algorithms and devices for monitoring the health of the composite materials. This includes everything from determining material properties (like natural frequencies) to researching different sensing elements and exploring technologies to electronics packaging materials. One of the sub-topics of the project is exploring different possibilities to do the structural health monitoring using the current state of the art electronic devices. The functionality in the very scope of this is the wireless data transmission using low power digital transceivers. Among many researched platforms is the Texas Instruments MSP430. Input for this work is the requirements of the sensor node stated by the work of the structural mechanics in the project. The output of the work is the assessment of the MSP430 platform in the context of the requirements.

1.1 Motivation

The benefits of the SHM systems are remarkable. They provide less costs with enhanced performance and structural overview of the objects. Developing such systems is beneficial to the research of the smart materials too, since the size of the devices nowadays will approach the limit where they could be embedded into the structure of the material. Continuing miniaturization is needed to overcome the problems which hamper the successful adoption of wireless SHM systems. Minimizing the size reduces the power consumption, decreases the footprint and therefore enables more functionality on the area. Increasing the functionality, which most probably could be signal processing and therefore computing power would enable one to run algorithms locally, which means less wireless data transmission.

With wireless sensing technology still in its infancy, much work remains for bringing this promising technology to widespread use. In particular, more research studies are needed on challenging issues such as power consumption, time synchronization, multi-scale network topologies, decentralized data processing within large-scale networks, and formulation of power-efficient data driven usage strategies. [1] With the latest semiconductor technology the embedment of the radio circuitry into the same housing as the MCU is possible. The low power digital radios have become good enough so that it satisfies the requirements for low overhead data communication (up to 50 kbps) that is needed for home automation and security systems. Higher frequencies enable more throughput, while decreasing energy efficiency and range. At the same time lower frequencies will increase the range and therefore making radio link more robust and immune to scattering and attenuation. But with lower frequencies the size of the antenna increases too, which means increased footprint, which is not wanted. So the fine balance between data rate, range and power consumption must be determined with the current state of art devices.

The development process of this work focuses on wireless data transmission using relatively new technology of sub 1GHz transceivers. The Texas Instruments ultra-low power MSP430 MCU family with integrated transceiver CC430 series is taken into the focus. The CC430 RISC microcontrollers do not have the native support to execute instructions and arithmetics specific to signal processing (floating point calculations). Computing machines able to do that (like DSPs) are too power hungry and usually lack wireless communication which render them unusable. The mash up designs where different requirements are fulfilled with different functional parts (computing MCU, separate ADC, transceiver module) might have great performance, but the increased size and power consumption render them unusable. Thus this platform is one of the candidates for SHM applications, providing building links – wireless data transmission, low power and on board peripherals.

The process of this work lasted more than a year. During this time author learned the MSP430 platform thoroughly and did detailed research on CC430 radio. The output software was not the only design the author worked on during the research, but because of the strategic decisions, based on time and volume constraints, the Chronos based approach was chosen and presented in this work.

1.2 Objectives

The work aims to put on test the following goals using MSP430 radio devices:

- Data rate of 100 kbps or more per one node
- Live transmission of sensor measurements
- Network of multiple sensor devices
- Configurability from the PC
- Months of operating time on the dedicated power source
- Small size

There are many problems needed to be resolved to achieve these goals. First must be determined that if the technology have the capacity to support these requirements. Main concern is the wireless radio interface and its capabilities. If one needs to increase the data rate, it must take into account decreased sensitivity and therefore increased error rate. Increased transmission means increased duty cycle, which will result in increased energy consumption and higher noise level in the spectrum of the channel. Additional will fill the spectrum with the noise even more, the shared timeframes for transmission must be taken account. The sensor measurements must be sampled and processed as asynchronously as possible to use the resources effectively and use computing power on network related tasks. Also some buffering is needed to be done, since immediate transmission cannot be guaranteed. Size is most directly related to the frequency and sensor interfaces. The lower the frequency the bigger the size of the antenna.

In the end major part of the problems are associated with the wireless data transmission. Alleviating the problems and enabling more efficient operation of the radio is the key point to achieve the objectives.

1.3 Outline

Chapter 2 (Overview of the technology) first describes different wireless SHM systems. The studies covered are chosen so that different approaches to the problems are included, giving a spectrum of possible solutions. Then analyzes the transceiver products by different semiconductor manufacturers. The state of the art review covers the IC devices and their properties, the ecosystem, documentation and the overall ease of development of the products of certain platforms.

Chapter 3 (Development of MSP430 sensor network) contains the information of the development of the prototype of this work. Gives the description of different tools, hardware and software used to develop the wireless sensor node. Thorough details of the process of the development of CC430 transceiver modules is included. Then the work is summed up by evaluating the properties of the system. Several tests on the proposed sensor network setup is performed. Evaluated properties include energy consumption, range, data rate and performance.

Chapter 4 (Conclusions and future work) gives a final assessment of the MSP430 platform. Proposes the enhancements of the system and lists the architectural pitfalls that should be completely changed in order to enhance the performance.

2 Overview of the technology

List of state of the art of structural health monitoring applications and transceiver devices give a starting point of the technological capabilities. Different properties impact the functionality and usability of the designs. The influence of size, power consumption and wireless data transmission are taken into focus. The chapters conclude the review with the detailed properties of the chip and prototype systems' designs.

2.1 Prototype SHM systems

Good example of progress of SHM systems are developed prototypes during academic researches. Designs have considered the latest state of the art devices of that time. Methods and network logic retain the high level of experimentation and innovation. This results in various systems having totally different setup and working principles. The elaborated works are covering the whole spectrum of problems of SHM systems. Applications need to take into account multiple aspects: size, environment conditions, mobility, power consumption, sensor interfaces, sensor placement on the object, etc. Covering such a wide variety of requirements is nearly impossible even with the present technology. Although continuing progress in the chip technologies implicitly support sensor network solutions to be developed using low cost and low power microcontrollers.

This review is not an exhaustive list of implementations, but more a quick overview of different solutions during the timeframe of approximately 10 years. The oldest work analyzed is from around 1995 and the latest is from 2008. The focus is on the data transmission as this is the object of research of this work. So finally one should have a firm overview of the evolution of wireless sensor solutions. [1, 2]

2.1.1 RQEM

Remotely-Queried Embedded Microsensors Program is funded by the US Office of Naval Research. Its mission is to develop sensornodes, which can be embedded into the vessels and planes firsthand, so that they could be used monitoring cracks and deformations.

RQEM mote is a tiny sensornode (under 1 inch^2), which has strain sensors, transponder and radio antenna coil for powering the device and transmitting/receiving data. They can be used to monitor solid structures by placing the reader close to nodes. Reader transmitted energy powers up the node and then transmits back the data from sensors along with the node ID. Basically it makes the node act like a passive RFID tag, with the extra of sensor measuring circuitry. During the time when reader reads the data from the node, it is powered up and gets all its energy from that, thus it is active only when read. Transponder uses half of the frequency of reader, which makes the transmitting frequency 64/128 kHz respectively.

For measuring strain special rosette of micromachined curved-beam capacitive strain sensors are used. So-called microsensor also incorporates a temperature sensor. They were developed to replace fiber optic embedded sensors. The microsensors must be extremely low power and small size. This is important because the sensor device is only powered when the reader is close and it must be able to measure sensors and send back the data with the power it obtains from the emitted energy of the reader. Also keeping sensors extremely small can help to reduce power consumption. Besides the power consumption the size is important to make it fit into the structure without causing it to crack or change its properties (total size area of under 1 square inch and thickness of maximum of 1 composite layer or 0.005"). Developing a sensor satisfying these requirements was the main goal of the project. Combined with a small electronics (or small list of requirements for electronics), these properties make RQEM suitable to be embedded into the composite structures for the lifetime of the structure. [3]

The size and robustness definitely make this design a considerable approach. With the widespread adaptation of NFC standard lately, this design could be modernized and successfully used. However it would have only small range of applications where it could be used successfully (most likely static objects). But the price, size and the energy consumption make this one of the standout designs.

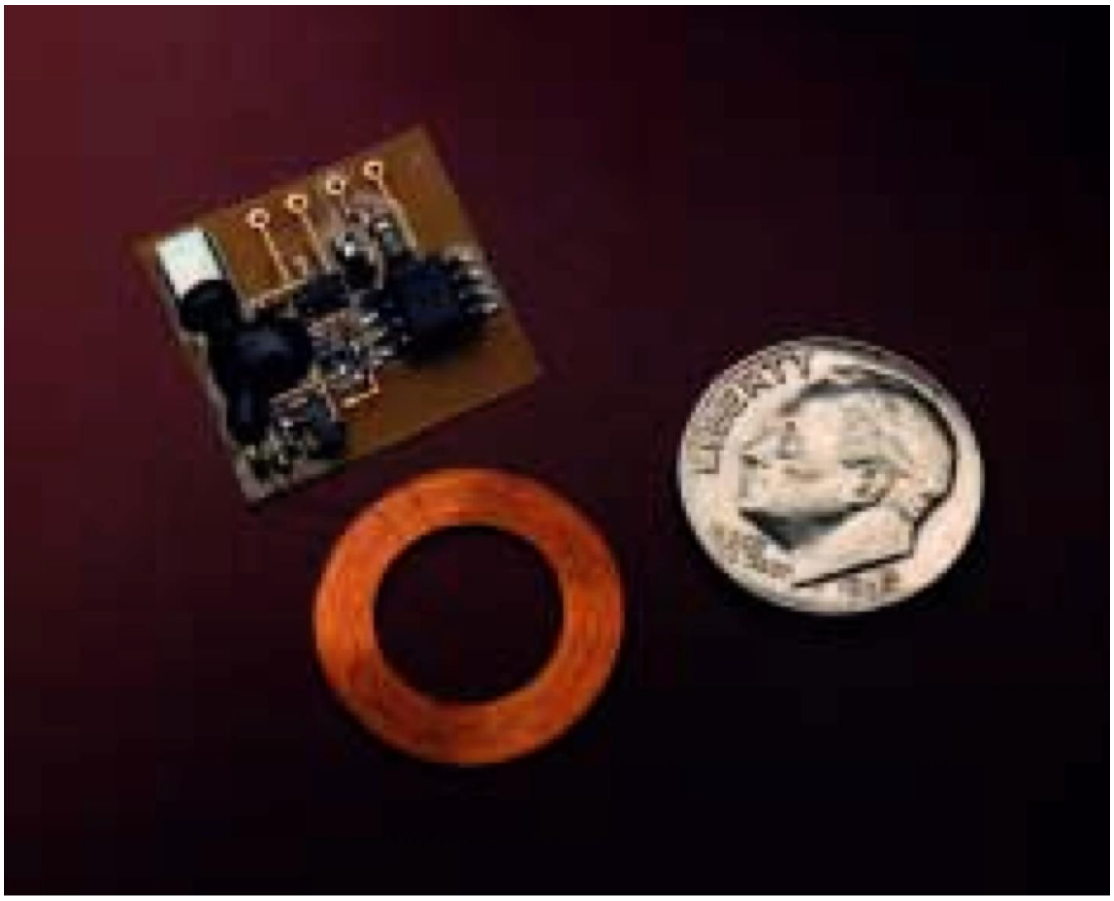


Figure 1: RQEM prototype along with the test circuitry on a common PCB [3]

2.1.2 WiMMS

Wireless Modular Monitoring System (WiMMS) is a great example how the early developments of wireless SHM systems were done. Through the multiple iterations of research the initial platform, which was developed by the 1997, evolved step by step into a sophisticated and feature rich low power SHM system by the 2004. [4, 5] The prototype evolution phase covers main problems take into the focus of this work – data rate, power consumption, synchronization.

2.1.2.1 Initial prototype – 1997

First iteration was done by E. G. Straserand and A. S. Kiremidjian at Stanford University. Their main goal was to have a wireless sensor unit, which could alleviate or totally eliminate the problems of the conventional cabled SHM systems. Problems as large installation time, cable breaking and corrosion, large distances between instrumentations points and therefore difficult routing could all be alleviated with a wireless system. [4]

Best components of that time (1995) were carefully chosen to fulfil the requirements. The core of the system is a Motorola 68HC11 MCU, which enables developing the software in higher level language C (good for abstraction of the modules, especially communicating part – reuse of the code), exposes peripherals (SPI, SCI) to connect with other components like RF modems and has multiple power saving modes. At the same time it packs enough memory (64K address space) to buffer the ADC measurements and parse the network packets. Communication part of the module was chosen to meet the requirements of the low power and noise immunity. The Proxim Proxlink MSU2 module is used for the data communication. It operates in the 902-928 MHz band allocated for ISM systems. It utilizes the DSSS modulation which enables high immunity to noise and interference and enables radio waves to penetrate reasonably well through the civil engineering materials. The maximum data rate of the module is 19.2 Kbps with the line of sight range up to 300 meters. It consumes about 700 mW of energy while transmitting and receiving, but consumes only 5 mW in standby. Another, complementing, RF module was also added to the unit for the synchronization purposes. The Radiometrix's TXM-418-F-5 transmitter and its compliment, the SILRX-418-F receiver, were added to the design of the base station and sensor unit respectively. They operate at 418 MHz and support a data rate up to 10 Kbps. Sensor units have a receiver which are listening to the synchronization pulse. Synchronization sequence is passed to the 68HC11 which time calibrates the internal structures for periodic monitoring.

Tests showed that the WiMMS system was able to send and receive commands and sensor data. The modular units were installed fast and did manage to alleviate or remove the problems of the cabling. The project fulfilled most of the requirements especially modularity and fast installing time. Only the accuracy of the measurements and the operating time were the only issues which were not solved.

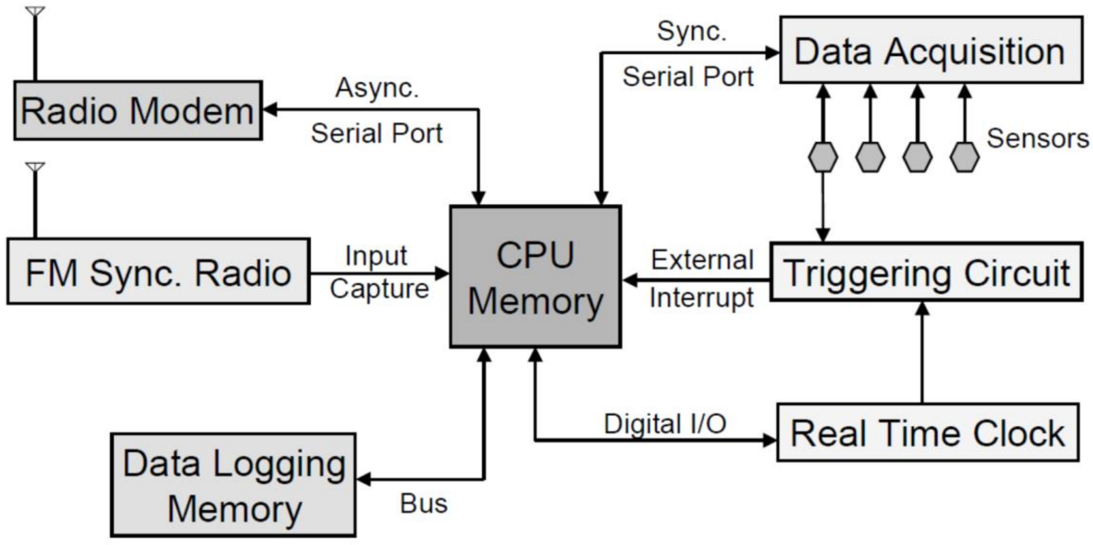


Figure 2: First WiMMS prototype functional schematic [4]

2.1.2.2 MCU improvement – 2001

An Atmel AVR 8-bit microcontroller with a RISC architecture was a replacement to the Motorola's 68HC11. Like the predecessor it is designed to support higher level languages like C. It packs a lot of functionality to the silicon like internal oscillators, serial communication, timers, PWM modulators and up to 4 full 8-bit I/O ports. AVR architecture supports high code density and most of the instructions are executed with one cycle, which means high MIPS and therefore efficient use of power resources without sacrificing the computational power. These properties enabled the SHM algorithms to be moved from the base station PC to the sensor unit itself, which is one of the main features and enhancements of the prototype. [6, 7]

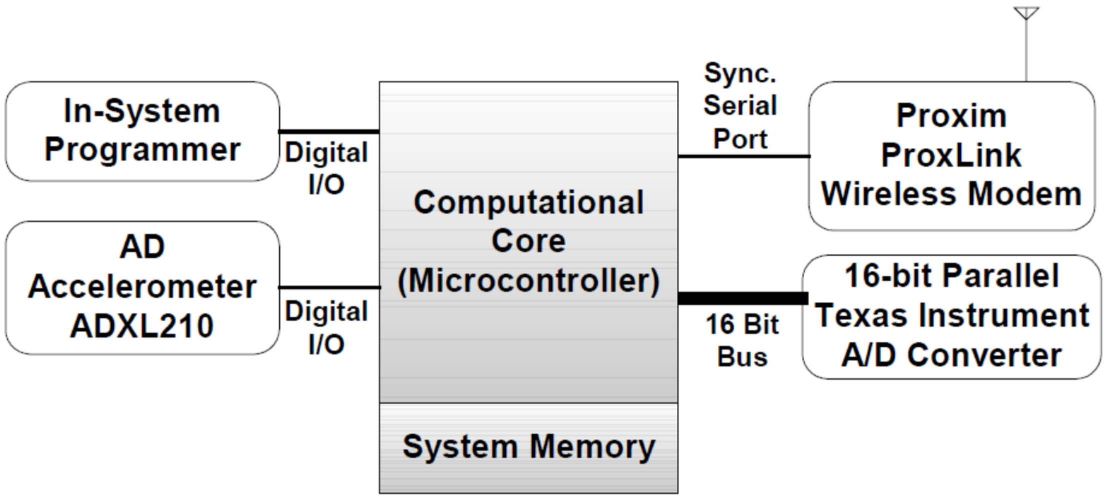

In-System Programmer
Digital I/O
AD Accelerometer ADXL210
Digital I/O
Computational Core (Microcontroller)
System Memory
Sync. Serial Port
Proxim ProxLink Wireless Modem
16 Bit Bus
16-bit Parallel Texas Instrument A/D Converter

Figure 3: Second WiMMS prototype functional schematic [6]

2.1.2.3 Redesign of the radio and computing techniques – 2004

This prototype was built to explicitly address the problems and bottlenecks of the previous specimens. First another MCU was added to the design in order to have more computation power for the accelerometer's signal processing algorithms, which includes floating point arithmetic. For that purpose a true 32-bit Motorola MPC555 processor with PowerPC architecture was chosen. It features 448 Kbyte flash ROM and 26 Kbytes of RAM, which can serve the data buffer for the measurements and fast execution of the code of computational algorithms. Another performance tweak is a completely different modem. A Proxim RangeLAN2 modem uses FHSS technique which enables bigger data rate (1.6 Mbps), more reliable communication link and up to 350 meters LoS range. It operates on 2.4 GHz unregulated FCC ISM band. [5]

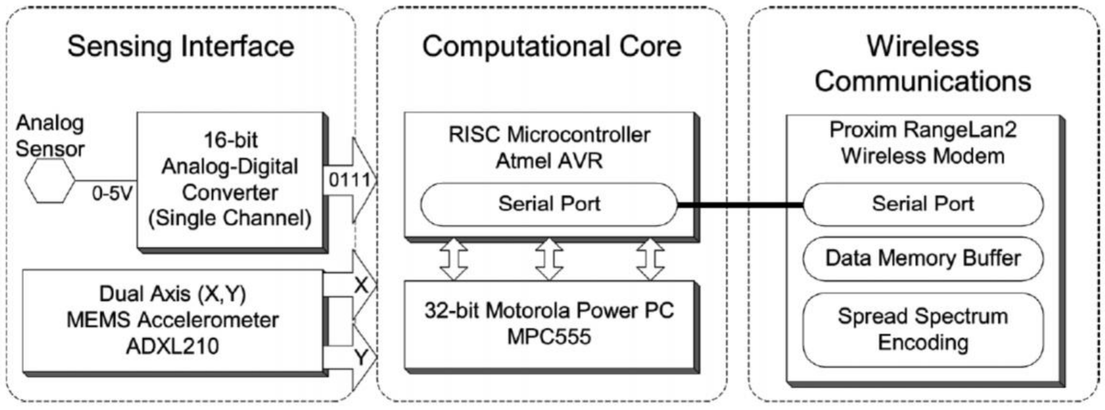

Sensing Interface
Analog Sensor
0-5V
16-bit Analog-Digital Converter (Single Channel)
0111
Dual Axis (X,Y) MEMS Accelerometer ADXL210
X
Y
Computational Core
RISC Microcontroller Atmel AVR
Serial Port
32-bit Motorola Power PC MPC555
Wireless Communications
Proxim RangeLan2 Wireless Modem
Serial Port
Data Memory Buffer
Spread Spectrum Encoding

Figure 4: Third WiMMS prototype functional schematic [5]

2.1.2.4 Conclusion

The WiMMS prototypes showcased different technologies and approaches to the sensor applications. The fine balance between local data processing and distributed computation was tested. One of the interesting solution was the synchronization of the nodes, where different radio interface and protocol was used. Besides the promising ideas for distributed data computing and synchronization, the solutions are not usable in the context of today.

2.1.3 RIMS

Goal of the project was to develop a small sensor node utilizing as much as possible latest state of art (2003) of microcontroller and wireless technology. Main objects were bridges where monitoring acceleration sensor data the faulty conditions in the structures would be determined and accidents would be avoided.

There are three essential parts of the system: H8/4069F MCU by Renesas, analog piezo resistance type accelerometer MA-3 by Microstone and RTL-8019AS wireless Ethernet module by Realtek. Extraordinary is the TCP/IP protocol stack support by the device. The aim for the widely used WLAN support was the simplicity of adaptation (due to the integrated chip), widely spread usage and high bandwidth. That enables anyone with the compatible device to monitor the data via web browser utilizing HTTP. In turn the data is processed asynchronously by the H8 MCU while the measurements are active. In order to achieve that ring buffer is used to store the results temporarily. That is external 2MB memory connected via DRAM interface of the processor. This decreases the data needed to be sent wirelessly. Data sent to the client device is in the form of histogram for a certain time period calculated by the H8 MCU.

One issue with the system is its power consumption. In the testing batteries were used. But those could power the system only for up to a month, depending on the amount of data exchanged. Suggested solution is to use different power source - either generating one or one with the bigger capacity. [8]

The RIMS prototype network provides the best usability and control from the standpoint of the user. The WLAN support with the powerful hardware design could enable to use these devices at a large scale objects like big buildings or bridges. The rate of the efficiency, due to the multiple chip design and external components and therefore requirement of mains power, render it unusable at small scale sensor networks.

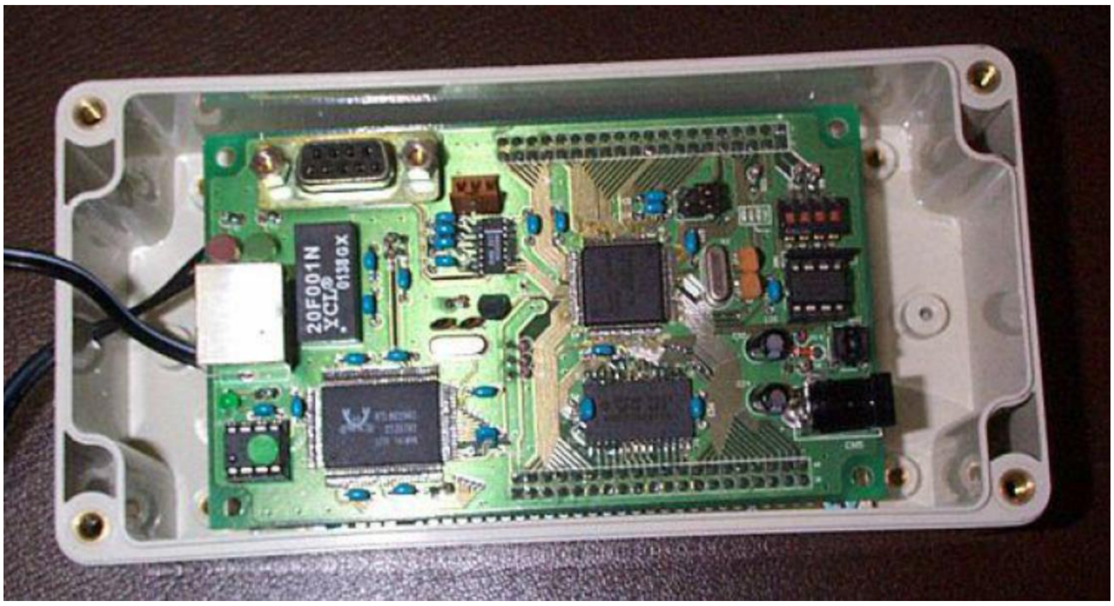


Figure 5: RIMS device in the box [8]

2.1.4 Reconfigurable 6LoWPAN based sensor network

One of the latest projects published (2009), which successfully can monitor and determine the faulty conditions of the structure. In the prototype tests it was used to monitor bridge like conditions, but with having humidity and temperature sensor additional to acceleration one, it can be used for other purposes as well.

It uses the Sensinode U100 Micro.2420 platform. [9] Since the hardware manufacturer is a developer of the 6LoWPAN software for the embedded devices, the wireless protocol is 6LoWPAN, which distinguishes it from most of the other projects. Although the future might seem promising for the IPv6 networks for embedded systems, in reality there is still much progress left to do. TCP/IP based networks require a lot more memory and processing power than other more lightweight protocols like Bluetooth LE for example. Also the library is not that mature yet to support all the functionality needed for sensor networks like sleeping nodes. That means the IPv6 networking is up and functional, some embedded systems specifics cannot be supported (routing protocols, synchronization of clocks). So the reduction of the functionality is done. The setup uses simple star topology with preconfigured unique addresses where sensor nodes are all having common sink node.

Micro.2420 node consists of MSP430 MCU and separate CC2420 radio transceiver. They are connected via SPI. The IEEE 802.15.4 compatible CC2420 radio is operating on 2.4 GHz. On top of it with the conjunction of MSP430 MCU the 6LoWPAN logic is implemented. Data is transmitted via UDP channels. Although UDP does not contain logic for packet transmission errors, the aid for that is implemented on packet level, where 3 bytes before each measurement contains 1 byte for data type, and 2 bytes for measurement sequence number, where final corrections are being made at the sink node. Sensor node also incorporates 500 KB flash where the measurements are stored.

The core of the sink node is the same Micro.2420 device. The sink node is connected to a PC via UART bridge. Through the PC one can configure the network. Fine tune options include the possibility to choose which node is sending the info, what is the sampling period and frequency (up to 1 kHz), accelerometer sensitivity (±2g or ±6g) and axis. On the PC, MATLAB software receives the measurements, reorders them, processes and visualizes them on the time chart. One of the biggest constraints of the system is the UART link between the PC and the sink node (due to the extra overhead from sink, not the actual sensor data itself). The application data transmission rate up to 6 kbps can be achieved (not including the TCP/IP overhead). [10]

The integration of 6LoWPAN is the future approach for many low power wireless solutions. This prototype is the most modern and the closest to suitable solution for sensor networks. As with the other designs the energy consumption problems are still present, but the functionality included with the IP protocol is endless.

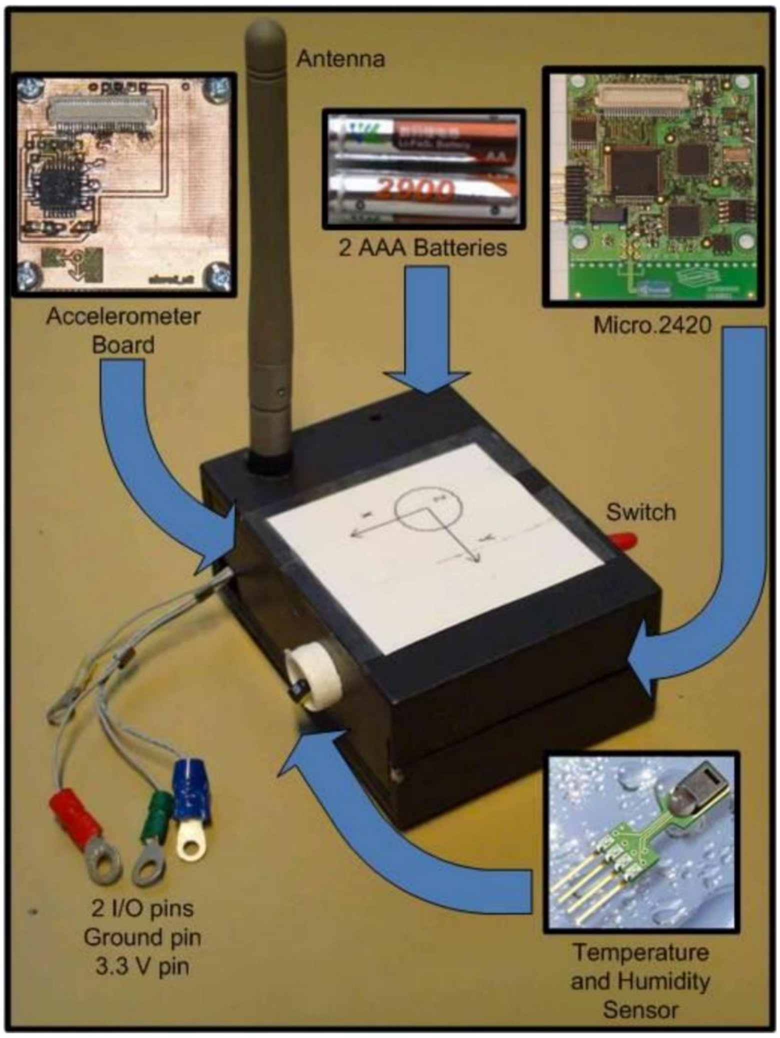


Figure 6: A reconfigurable 6LoWPAN wireless sensor node [11]

2.1.5 Mobile agent SHM network

Completely different to the other covered sensor node designs, this exact sensor network design aims for high computing power sensor nodes that are able to update the tasks and damage detection algorithms and finally be able to update the software it runs. It has 3 different boards which are doing different tasks. Main board is a Gumstix miniature embedded computer, an Intel based platform. It has an Intel PXA 225 processor with 64 MB RAM and 16 MB Flash memory. With 400 MHz frequency, it runs Linux OS on which the damage detection algorithms, which require FFT, are being run. The wireless communication board is connected via parallel port to the Gumstix computer and is responsible for WLAN communication. At the core it has an Atmega 128L CPU, which communicates with the Gumstix board, handles communication with sensor interfaces and measured data. It uses SPI and I^2C interfaces to communicate with different sensors, actuators, A/D interfaces and low power ZigBee communication module.

Due to the fact, that the Gumstix board runs a complete Linux OS, a lot of the focus was put on the software the system runs. Many open source libraries were used in the system like Ch (C/C++ interpreter), CLAPACK (mathematics library) and Numerical Recipes in C (mathematics library). [12] These packages enable to do SHM analyses and port the functionalities and software from other environments (e.g. MATLAB) to the Gumstix board. Besides these the most important functionality is done with the Mobile-C mobile agent platform. This library supports portable and secure execution of mobile agents which are C/C++ scripts. [13] Having mobile scripts being handled between the nodes, this enables the sensor network to become dynamically programmable and taskable. Software is layered between the two boards. The Gumstix runs all the high level data processing algorithms, Mobile-C and wireless communication logic. The sensor acquisition boards runs the low level tasks like sensors' data acquisition, power management, and ZigBee radio module. [12]

From the hardware standpoint the design does not provide much value. The problems with energy consumption is not solved. Also Zigbee based network is not future proof and powerful. However the idea of mobile agents, which carry embedded code, could be a technique of solving the problems of data processing and movement and network configurability. Author of the work believes that the C/C++ scripts should be replaced by some interpreted language like Javascript or Python. When compared with the IP network based solutions like the Micro.2420 based 6LoWPAN implementation (see Section 2.1.4), the power of mobile scripts could be used more effectively.

2.1.6 Summary and comparisons

Different research work of SHM systems were presented. Designs varied from energy harvesting to steady battery powered devices. Different radio protocols can be seen to be applied for the wireless data transmission and the trend shows that TCP/IP based networks are steadily applied on the embedded platforms.

Almost all the prototypes failed one basic requirement – power consumption. This indicates that there are no suitable methods and technology for wireless data transmission to enable ultra-low power energy consumption or enable very high data rates which would decrease the time where the radio circuit needs to be active.

The fine balance between the responsiveness with the data latency and power consumption is an aspect that was tried to be solved by the different systems. When one has a high responsive system, like the mobile agent SHM network (Section 2.1.5), that in turn means higher power consumption than less responsive poll based systems, like RQEM (Section 2.1.1). Low latency can be also connected to the configurability, since configuration changes require link with the low lag factor, to apply the settings.

With the current technology, the results are telling one thing – more sophisticated system equals exponentially higher energy consumption. All this leads to the fact that the radio part of the system is the most cumbersome and power hungry. Also it can be a bottle neck for the data transmission of the whole network. But the efficient usage of bandwidth and radio technology must be desired in order to have a usable sensor network design.

	Size	Wireless protocol	Data rate	Power consumption (sleep/idle/Rx/Tx)	Power source	Up time	Sensors	Features
RQEM	< 1 inch^2	Trovan RFID	< 64Kbps	active only when reader emanates the electromagnetic wave energy	Electromagnetic wave	powered only when read	micromachined strain sensors, temperature	size, robustness
Reconfigurable 6LoWPAN sensor node	88mm x 64mm x 35mm	6LoWPAN	6 Kbps	<1mA/x/25 mA/35 mA	2 x AAA	x	Humidity, temperature, xyz acceleration,	Configurability from PC
RIMS	300mm x 60mm x 80mm	WLAN	> 1 Mbps	x	Battery pack with unknown type	~ 1 month	3 axis acceleration	Direct connection from any Wi-Fi enabled device
WiMMS 1st	100mm x 13mm x 190mm	PPX (Proxim Packet Exchange) + Custom	19.2 Kbps	5mA/> 50mA />225mA/>225mA	18 x AA batteries	11 hours	2 axis acceleration	Measurements synchronization

		application data						
WiMMS 2nd	130mm x 100mm x 25mm	PPX + custom application data	19.2 Kbps	x	9V alkaline battery	x	2 axis acceleration	SHM algorithms processed by sensor node
WiMMS 3rd	100mm x 100mm x 25mm	PPX + MCP (Modem control packet) + application data	1.6 Mbps	110mA/220mA/320 mA	5 x AA	Up to 30 hours	2 axis acceleration	Separate MCU for computation intensive tasks
Mobile agent sense node	100mm x 60mm x 17mm	Zigbee	250 kbps	x	mains	not applicable	3-axis acceleration, strain gauge, humidity, temperature, PZT sensor actuator	Dynamic change of SHM algorithms, Linux OS

Table 1: Comparison of different SHM systems

2.2 Low power transceiver devices

This section discusses different manufacturers and their off the shelf integrated circuits supporting wireless data transmission. All the producers that are mentioned here can be considered as a design candidates for one's wireless sensor network. Different aspects are described that apply for the wireless sensor nodes' design. Different specimens and evaluation kits and other tools which help to develop on the devices of the platform are also discussed.

When common embedded design issues are put aside one can start to digest through the questions like power consumption and radio. [14] The key to resolving the problem is the whole dynamic functionality of the SoC. Questions like variety of configurations (I/O, clock system), mechanisms that support energy efficiency (DMA, different ADC modes, interrupt mechanisms, sleep modes) [15], analog digital interface characteristics (resolution, samples per second) and size (the SHM applications often require the sensor node to deeply embedded into the structure without causing the structural grid to alter much). [16] Depending on the application the sensor devices need to operate on a dedicated power source from weeks up to couple of years. The specification of regular microcontroller sometimes can state that many years of operation can be achieved, but one must note that the radio must be added into the power consumption calculations too. Since there is no common way to universally benchmark the energy efficiency of the microcontrollers right now [17, 18] one must either benchmark or rely on the information that is written in the datasheet. For example one of the main standpoints are the sleep and idle state ratios, how much current is drawn in each state, how fast the circuit can switch from one state to another. [19] The same apply on different modules in the MCU architecture. The studies clearly indicate that most of the energy is spent on the radio. [20, 21]

Product lines of ISM bands compatible transducers are described shortly. Specifically important is the conformance to the IEEE 802.15.4 standard which is the basis for 6LoWPAN and ZigBee standards. The ubiquitous IEEE 802.11 based (WLAN, Bluetooth) protocols are not elaborated, because of the higher power consumption and computing power needed. The IEEE 802.15.4 standard specifies physical layer and media access control for low-rate wireless personal area networks. [22] That includes regulations about modulation techniques, frequency bands, duty cycle, output power, etc. Closely related with the standard are the unlicensed ISM frequency bands. These are free to use publicly available frequency bands that are allocated to be used for industrial, medical and scientific (ISM) purpose, but virtually can be used for anything except the telecommunications. There are different regulations applied by the International Telecommunication Union (ITU) at the different regions of the world. 2.4GHz is available worldwide and in the region of the current work, Europe, the 868 MHz and 434 MHz bands can be used. [23] Using devices compatible to ITU regulations can save a lot of extra work. First one do not need to deal with the certifications. Using widely available band results in bigger choice of the transceivers that can be bought directly off the shelf. Last, when the widely available technology is used, the improvements and compatible technology can be developed by other parties as well. Thus the radio devices and producers described have SoC capable of working in one of the previously mentioned frequency bands. The focus is shifted more on towards the sub 1 GHz band. In general 2.4 GHz band can be a better choice because of the higher data rates and faster transmission times [24], but due to the higher noise in those channels, the 868 MHz band can be more robust. [25] The noise free environment and lower frequency can increase communication reliability and provide longer transmission ranges. [26, 27] However analyzing both in conjunction gives a better picture of the manufacturer's catalog, future improvements and technological capabilities. In the final comparison table only devices able to operate on 868 MHz band are listed.

Most of the manufacturers have not designed a completely new radio MCU architecture for SoC transceiver, but rather have embedded and expanded the functionalities of the radio transceiver to already mature architecture (TI MSP430, Freescale HCS08). This way it preserves the familiarity with the series of microcontrollers for the developers and enables the code reuse. Some of the designs use the approach where the transceiver module is externally interfaced via I/O (usually via SPI) to MCU. In general the embedded approach is more suitable for low power wireless sensor networks. Packing all the functionality into one chip package extends battery life and transmission range, delivers more robust performance (in the presence of interference) and minimizes the use of external components. [28]

2.2.1 Microchip

One of the most known MCU among the hobbyists and professionals is the Microchip's PIC series. Microchip has wide range of different radio receivers and transmitters for different frequency bands including 802.15.4 compatible devices. Some of them are PIC based and some of them are plain radio modules. They have 5 different modules for 2.4 GHz and 4 modules plus 7 MCUs for sub 1GHz frequency bands. Disadvantage of the Microchip product line is the lack of true transceivers. The only true transceivers are the radio modules which are not programmable and must be controlled externally. PIC based radio MCUs have only transmitting capability and they support only sub 1 GHz bands. On the positive note, Microchip have developed libraries for developing wireless network applications on their products. They have fully supported ZigBee RF4CE and Pro/Smart Energy profile library stacks. Additionally they have developed proprietary protocol called MiWi, which is intended to ease the development of wireless applications and portability of the code across the different Microchip devices. The MiWi supports different topologies and different types of network devices. It has dedicated development environment with fully integrated tools. Developing is supported by multiple development kits with preprogrammed modules ready to go right out of the box. Debugging the over the air communication can be done with the ZENA wireless USB adapters. Because of the lack of true transceiver microcontroller and therefore increased small form factor and power consumption, the author of this work do not seem much value including Microchip's microcontrollers to one's wireless sensor node design.

Information from the main and subpages of Microchip's personal area networks product page. [29]

2.2.2 Nordic Semiconductor

Nordic Semiconductor is focused only on low power wireless integrated circuits which operate on ISM bands. Technologies they are most focused on appears to be protocols operating on 2.4 GHz like Bluetooth Smart or ANT+.In total they have 4 different series (2 for each protocol) plus 3 series for universal 2.4 GHz solutions. This makes a total of 16 different chips with different configurations. They also provide sub 1GHz products, though the variety is tiny. Only one transceiver module and one complete SoC with integrated transceiver (appears in the table) which has the aforementioned module embedded to the 8051 core. For the developers they have dedicated forum on the homepage, a lot of different evaluation kits covering all the wireless products, white papers on technical topics and recommended design schematics and guidelines. Software development can be done with any toolchain that supports 8051 architecture. Example software and binaries are built using the KEIL toolchain. From the mainstream toolchains IAR also supports Nordic Semiconductor devices, with complete toolchain and header files. Overall the approach and product development of the Nordic Semiconductor microcontrollers is promising. They focus on specific needs of the product by producing the controller in multiple configurations. In their 2.4 GHz series they have controllers with ARM core. If the sub 1 GHz portfolio can progress to where their 2.4 GHz devices are, then their devices are comparable to the competitors'.

Information from the main and subpages of Nordic Semiconductor's ultra low power wireless solutions page. [30]

2.2.3 STMicroelectronics

STMicroelectronics has only one true SoC transceiver. It is ARM based IEEE 802.15.4 compatible transceiver working on the 2.4 GHz band. But they have a lot of experience with ARM based MCUs and they are producing much more than just microcontrollers. The wide catalog in electronic parts and diverse set of transceiver modules can contribute to a sub 1 GHz ready radio SoC. Specialized products for low power WLAN, Bluetooth and other ISM frequencies are available. The SPIRIT1 module has 4 selectable frequency ranges, multiple modulation options, and very high sensitivity and power efficiency. It is also compatible with the Wireless M-BUS standard. For the current development there are more than 10 evaluation boards (including SPIRIT1) to start getting familiar with the STMicroelectronics sub 1 GHz portfolio. Since their MCUs are based on (and therefore controller of the transceiver module's evaluation boards) ARM, there are plenty of toolchains, starting from the free GCC to paid licensed IAR. STMicroelectronics has some aid software for the developers to calculate the energy consumption and debug the wireless data links. This includes also tutorials, application notes and reference design. Their portfolio is thin and while heavily implementing ARM design, it is reasonable to reach out to other producers, who also have integrated radio IC and ARM core on one chip, but have broader list of devices to choose from.

Information from the main and subpages of STMicroelectronics' low-power RF solutions page. [31]

2.2.4 NXP Semiconductors

Portfolio boasts assets specialized on wireless networks. That includes products from aerospace to automotive radio link solutions, but also the ISM unlicensed band products. When talking about the spectrum in focus - the ISM wireless modules and microcontrollers - are made with a high specialization on 802.15.4 based protocols, that includes special packet features for different protocols and sleeping modes. For that purpose proprietary 6LoWPAN based JenNet-IP library, RF4CE and different Zigbee protocol stacks have been developed. For 2.4 GHz 3 different chips with 32-bit RISC architecture (JN516x series) and 8 transceiver modules are in catalog. For lower frequency ISM bands only one transceiver module (OL2381, listed in the Table 2) is present. This is traditional SPI controlled module with FSK and ASK modulations. It has relatively good sensitivity, but the maximum data rate is meagre and therefore the current consumption is high. Fortunately the products are backed by demo kits. The OL2381 has a throrough demo kit with multiple functions and GUI for the PC, which enables to configure the device in detail and trigger the data transmission. Overall there are plethora of developer tools for NXP products, even the Android and iPhone platforms have an app where one can calculate the RF parameters for the devices. The developer tools are mostly free, that includes the wireless toolset (IDE, compiler, programmer) for JN516x series. Drawback is the price of the evaluation kits. For example JN516x demo kit costs over 1000 USD (quote from Digikey). Adding the fact that the products are having proprietary design (JN516x custom core), toolchain and evaluation kits therefore small community, the start of the development can be cumbersome, which renders the platform not suitable for the initial development of sensor networks. [32, 33, 34]

2.2.5 Freescale

Freescale is another high profile electronics manufacturer presenting wide array of products in their catalog. Having various 2.4GHz modules and microcontrollers with either HCS08 or different flavors of ARM core, they do not have smaller choice under 1 GHz band products. Besides 3 either only transmitting or receiving modules they have two true SoC transceiver families based on different cores. The first device, MC12311 with HCS08 (listed in the Table 2), has the radio with the same parameters as the Kinetis KW0x family devices which are based on ARM core. The Kinetis series has more features and 32-bit architecture, while the Motorola's 68HC05 compatible core do not pack such features, but for small applications that can lead to smaller power consumption. Aforementioned MCUs are backed by the evaluation kits with the price from 79 to 149 American dollars. Wireless libraries for under 1GHz applications are two. The AMIHO Technology developed Wireless M-BUS library can be used out of the box for one's M-BUS product. For other type of protocol implementations Freescale provides SMAC (Simple Media Access Controller) stack which is available as C source files. This supports all their radio modules and enables to do the low level routines and configuration, without having any protocol specific packet layer. Can be used as an underlying library for ones protocol stack. Development tools for HCS08 platform are developed by open source communities and also commercial providers like IAR Systems, Cosmic Software, P&E Microcomputer Systems and many others. [35] Also Freescale itself is providing a CodeWarrior based development studio. Plenty of tools and documentation is available on home site. With a wide range of portfolio for different technologies the co-operating partners can be easily found on the same place as their own tools and documentation. Software like BeeKit, PopNet, VLAB that can greatly reduce the time of development, can be easily accessed. For the development of new devices for under 1GHz. Freescale products can be strongly recommended, because the parameters of the radio devices are more flexible and powerful than the competitors' (except Silicon Labs' Si10xx series – refer to the Table 2). Only drawback can be the availability of more sophisticated and protocol specific libraries, which might increase the development time for the first prototype. [36, 37]

2.2.6 Silicon Labs

In total Silicon Labs offer 6 families of highly integrated SoC devices. Products cover the 2.4 GHz band and the spectrum up to 1 GHz. Some of them are 8051 and some are ARM based. 8051 based devices have so called EZRadioPRO (as is the case with the Si1080 listed in the Table 2) transceiver, which is Silicon Labs' developed high RF performance transceiver family. Their focus with their products is energy consumption and radio module efficiency. They claim the industry's lowest consumption in active and sleep mode. Combined with the RF performance, the Si1080 is easily the MCU with the best characteristics among those listed in the Table 2. Starting developing on the platform is made very easy. All the transceiver modules and MCUs come with a specialized development kit (price ranging from 300 – 800 USD). Boards come with a simple software demo, where one can do basic things: settings, data transmission, etc. Silicon Labs IDE is the officially supported environment for all the Silicon labs' microcontrollers. Separate Keil compiler kit must be bought, which is listed on the webpage along the IDE. Overall info for development is easily accessed with one click on one root webpage. [38] Developer aid software includes among many a Simplicity Studio (MCU tools, documentation, code libraries, etc.), EZRadio (toolset to create and deploy wireless applications), Clock software (clock customization and calculation of various parameters) and many more. Only ZigBee library stack is provided via Ember ZigBee software, which includes debugging and other features. Other examples and libraries about radio can be accessed via Simplicity Studio. The ease of finding the resources, the high performance radio and the separate energy efficient ARM processor line (probability for the future products having integrated the ARM core) means that the Silicon Labs' products cannot be ablated from the list of potential platforms to be used to develop one's wireless product.

Where not cited, the information is from the main page and subpages of Silicon Labs' microcontroller page. [39]

2.2.7 Atmel

Atmel provides a lot of products for the 2.4 GHz band, either with AVR or ARM CPU core. Unfortunately there are only one product for under 1 GHz band and it is plain transceiver module. Different with the other manufacturers is the full compatibility to 802.15.4 specification, which means that it supports spread spectrum modulation techniques specified for ZigBee and 6LoWPAN. That means no FSK, ASK or OOK modulations are integrated into chip. The purpose of this is just to concentrate on the mainstream consumer electronic appliances, because this technology is most produced. Like Microchip's PIC, Atmel's AVR microcontrollers are widely know. With one of the biggest community, finding information for development is not a problem. Atmel has developed their own development envrionment called AVR Studio, which is one of the easiet to use toolset on the market and it is completely free. Toolchain is based on open source GNU GCC software. Filled with a lot of plugins/software in and out of the IDE, will render the development process to be one of the smoothest and fastest. Starting Atmel RF development is fast by using their ZigBit modules. Modules have complete certified RF design on a ready made PCB which makes them easy to integrate into one's application. They are based on the available products in the portfolio so one can take them as a demo boards for radio transceiver controllers or use ZigBit evaluation kits where the whole system is implemented. They come in different flavours and packages including the convenient USB dongles. To complement the ease of the development, Atmel has made available multiple radio networking stacks. The simplest and smallest is Lightweight Mesh which provides the most basic functionality to access the radio hardware and to develop proprietary porotocols on top of it. They provide full 802.15.4 MAC library, which is the base for ZigBee and 6LoWPAN implementations. On top of it are built ZigBee RF4CE and ZigBee Pro (called BitCloud) stacks, which are also available for free to download on the website. The ease of development and the size of the community definetly make the Atmel products one of the best to start developing wireless sensor networks. Right now they are heavily relied on 2.4 GHz, spread spectrum, so making a simple sub 1 GHz solution is not possible currently.

Information from the main page and subpages of Atmel's 802.15.4 microcontrollers page. [40]

2.2.8 Texas Instruments

Texas Instruments is producing one of the biggest portfolio of electronic devices in the industry. That includes high end computer processors, digital signal processors and microcontrollers. For sensor applications they have dedicated microcontroller family with 16-bit RISC architecture called MSP430, which is focused on ultra low power applications while providing low cost. Besides the dedicated computing, they have 10 different modules of CC (stands for Chipcon) series transceivers, where some are based on 8051 core. [41] Former Chipcon which was acquired by Texas Instruments in 2005 [42] gave the knowledge and integrated circuit designs that made possible to include SoC transceivers in their catalog. Besides the 8051 based transceivers, they have CC430 family of transceivers. CC430 series is a fusion of MSP430 microcontroller and Chipcon radio transceiver circuit. Therefore Texas Instruments provides two different series of transceivers with full capabilities of modern MCU, both of them are having principally the same transceiver core, where the CC series have more different configuration devices (sub 1GHz and 2.4 GHz), while CC430 family devices all have sub 1GHz Chipcon transceiver core..

The series in the focus of this work is the CC430 family. It packs the features of the modern and low power MSP430 architecture. The radio core present on all the CC430 family microcontrollers is principally the same as in the Chipcon CC1101 MCU, with the minor modifications because of the nature of the embedment into the different architecture. [43] Since CC1101 is designed to operate only on frequencies under 1 GHz, there are no 2.4 GHz devices for MSP430 based transceivers. If one needs more different configurations for the radio, the CC series covers all the frequencies and requirements of IEEE 802.15.4 for the modulation schemes and frequencies. So if one requires a broader grip for the application portfolio, the 8051 based Chipcon transceivers should be used.

Since the CC430 and CC series are connected closely, there are plenty of tools helping to start the development. Documents about radio communications which are dedicated to Chipcon microcontrollers do also apply for the CC430 family. The same applies to the development kits and other hardware tools. This makes the ecosystem quite powerful, where the migration from one family to another is quite painless. There are multiple development kits based on both families. The radio antenna reference circuit designs are available for all the devices. Only difference that exist between the two (besides the fundamental CPU architecture) is the flash programmers and development environments. The Texas Instruments developed CCS (see Section 3.1.1.2) do not support 8051 architecture, while IAR Systems workbench can be used for both. Different libraries like Z-Stack for ZigBee, SimpliciTI (see section 3.1.5), Wireless M-Bus and Bluetooth are supplied with the product portfolio with the examples and development kits. Code samples with the low level initialization and interfacing of the radio in different architectures and external SPI connection is provided.

Due to the large range of development articles aimed directly to help to start the development with different products and different documents where one can find useful information for one’s product design and development tools, the Texas Instruments radio MCU platforms coax to develop on this platform.

2.2.9 Table of summary

	Frequency range (MHz)	Modulation	Sensitivity (dBm)	Maximum TX power (dBm)	Maximum data rate (kbps)	Power consumption – RX/TX(at max output power)/sleeps	Price (USD)	CPU core
Microchip MRF49XA*	433/868/915	FSK	-110	7	256	11mA/15mA/0.3µA	1.79	x
Nordic Semiconductor nRF9E5	433/868/915	GFSK	-100	10	100	12.5mA/30mA/2.5 µA	2.66	8051
STMicroelectronics SPIRIT1*	150-174/300-348/387-470/779-956	FSK, GFSK, MSK, GMSK, OOK, ASK	-118	16	500	9.8mA/49.3mA/0.85µA	2.15	x
NXP Semiconductors OL2381*	315/434/868/915	ASK, FSK, GFSK	-118	10	112	16.5mA/22mA/0.5 µA	2.38	x

Freescale MC12311	315/433/470/868/915/928/955	FSK, GFSK, MSK, GMSK, OOK	-120	17	300	16mA/95mA/0.1µA	3.9	HCS08
Silicon Labs Si1080	142-1050	OOK, FSK, GFSK	-126	20	512	13mA/85mA/0.05µA	3.58	8051
Atmel AT86RF212B*	769-935	DSSS, BPSK, O-QPSK,	-110	10	1000	9.2mA/28mA/0.2µA	3.16	x
Texas Instruments CC430F5137	300-348/389-464/779-928	ASK, FSK, GFSK, MSK, OOK	-117	12	500	18.5mA/36mA/1µA	3.65	MSP430

Table 2: Comparison of different 868 MHz transceivers

* *Not a complete SoC, must be interfaced to external MCU.*
Prices were recorded at May 10th 2014. Quotes from www.digikey.com web-shop with the cheapest items packaging type where applicable and at the bulk price of 1000 pieces.

Detailed products' information from the following references: [44, 45, 46, 47, 48, 49, 50, 51].

The void of the embedded 868 MHz transceivers on the market is eminent. Still most of the focus is put on the 2.4 GHz band devices and therefore the devices for sub-1GHz bands are the minority. With the ITU regulations and more wide adaptation of those frequencies in different countries will have an effect on sensor networks.

Author believes that the most future proof products will be the ARM core integrated transceivers. With the powerful and continuously developing architecture and a true 32-bit design introduces a powerful computing capabilities, while providing the small energy consumption. The best products at that field are the Freescale and Silicon Labs products, but right now only 2.4 GHz is supported. However Texas Instruments with the low power, feature rich MSP430 platform and a true transceivers for the sub-1GHz bands seem to be the best choice on the market right now. With the addition of rich set of documents and development aid, the platform used also in this work, is strongly suggested by the author.

3 Sensor network development on MSP430

This project was limited by the time and volume constraints, so the aim was to reuse as much code as possible from the libraries, code snippets, kits, etc. By doing this it enabled to concentrate more on the essence of the work – the wireless data transmission – and how to make existing functionality better, but not thinking how to create one. During the development different set of hardware and software tools were extensively used. The most important ones and integral for the successful development of MSP430 based sensor network devices are described in this chapter. The functionality, price, adaptability is discussed. The assessment of the practice of using these tools is given.

In order to assay the developed wireless network performance and reliability, different types of tests were conducted. Tests contained many different scenarios and for that purpose, software was developed. The summary of the results tell if the goals were achieved and what are the main improving areas. Also they showcase the physical limits of the hardware and software.

3.1 Development

First concept used different microcontroller and development kit for the sensor node (MSP430–EXP430FR5739 board [52]). Since the initial design (wired communication from sensor nodes to radio concentrator node) finally grew into very complicated and fragile, the plan was dropped. Finally the transceiver node, which was the radio concentrator intially, was chosen to do also simple sensing functionality. It perfectly suited for the development ideology as well, since existing libraries and code examples were also present. The mock sensor development was finally done on Olimex MSP430 RF development board (see Section 3.1.2.1). It has the CC430F5137 MCU mounted which is mostly back-compatible with the Chronos development kit's (see Section 3.1.2.2) CC430F6137 (only large difference being the LCD_B module availability). Based on this, porting the software from one MCU to another was relatively easy task. This was also possible because of the the hardware schematics, where there were no major differences between the two. This enabled to run same code on both of the kits with no modifications.

3.1.1 Development environments

There are 2 widely known and officially supported development environments for MSP430 platform: IAR Embedded Workbench (developed by IAR Systems) and Code Composer Studio (developed by Texas Instruments). This is not an exhaustive list of existing solutions since there are other options as well like Crossworks by Rowley Associates, Energia an open source development environment initiated by Robert Wessels, Imagecraft by Imagecraft Inc., Quadravox by Quadravox Inc., and command line GCC toolchain called MSPGCC (open software, part of the GNU Tools). But those tools are either obsolete or not supported by Texas Instruments software, drivers and reference papers. So IAR workbench (from now the acronym IAR EW is used) and Code Composer Studio (from now the acronym CCS is used) are the only options which are being supported and will be supported in the future also. Both IDEs were used and evaluated throughout this project.

3.1.1.1 IAR Embedded Workbench

Prior to this work, the author of this paper have been mostly using IAR EW for embedded development (although not for MSP430 toolchain). So in the first prototype design development (MSP430FR5739 + CC430F5137) the IAR EW version 5.5 was used. The free code size limited version was used, because no license was available for the university work group. Second the code to be developed was thin enough not to break code size limit (1.5KB). Evaluation mode did not prohibit any features which were needed for the development anyway.

Three projects were setup and maintained inside one workspace in the IAR. Each project resembled different node in the architecture (sensor node on MSP430FR5739, radio concentrator for the sensor nodes on CC430F5137 and access point for PC also on CC430F5137). This was the basic setup of the usage of the IDE. Every project can be set up using different settings and also common settings can be made. For example libraries for radio, FIFO buffers, common typedefs and protocol specific C-files were shared by the projects. Environment automatically handles the compiling of files even when the files are from different locations on the hard disk. Grouping them together under the project tree is enough. Groups are shown as folders in the graphical interface.

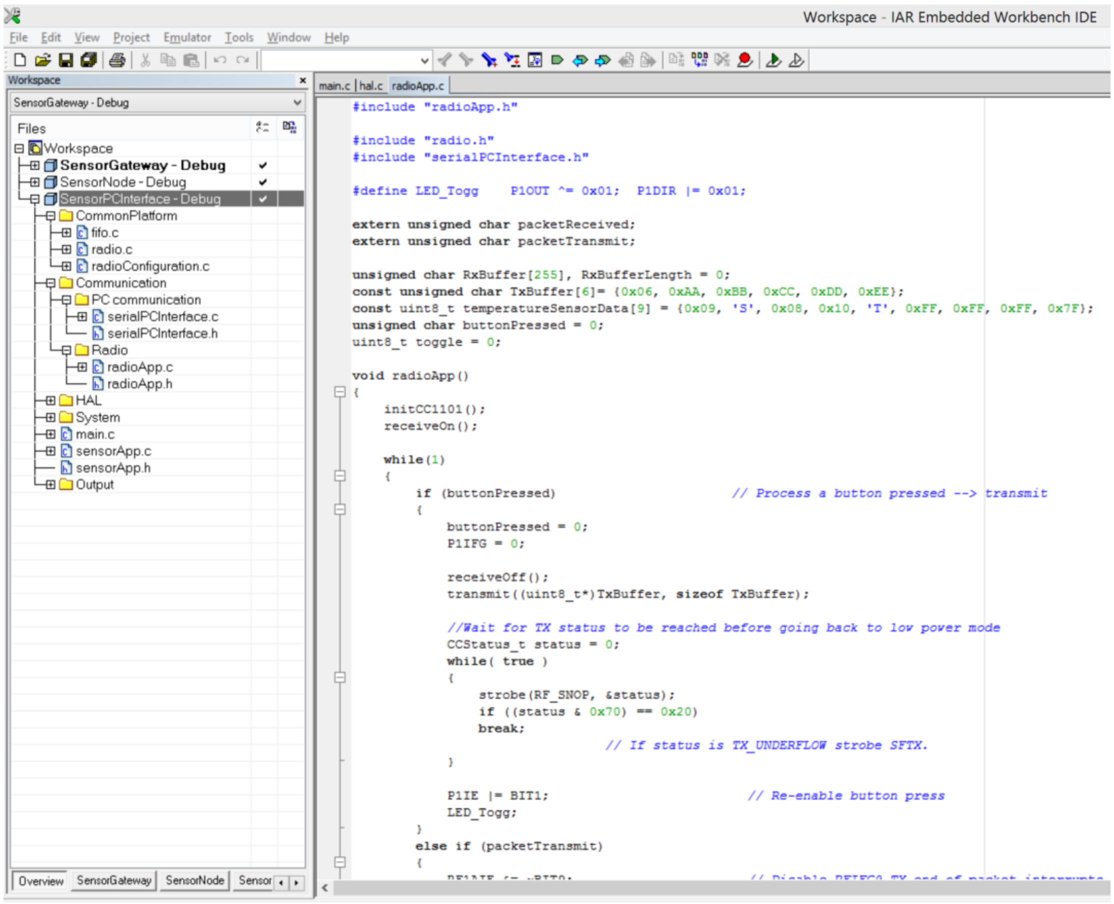
Compiler and linker settings can be easily set in the graphical interface under the project options. This includes choice of language and dialect (different C and C++ standards are supported), preprocessor and linker definitions, pre-build, post-build commands, debugging and debugger settings, binary file format, etc. The toolchain already includes linker and header files for different MCUs. Also the runtime standard C/C++ libraries are present with flexible configuration options (`printf()`, `malloc()`, etc.). One of the biggest advantages using IAR is its intrinsic headers library, which can be used effectively to reduce the size of the code and make it more readable. [53]

When the functional part of the toolchain is concerned, the IAR produces efficient code and features powerful debugging tools. [54, 55, 56] MSP430 toolchain linker, debugger and compiler are completely proprietarily developed and maintained. However when this project is concerned, the author cannot see any superiority of the IAR proprietary compiling algorithms and debugging features.

When esthetics are concerned, largely different is the graphical interface. It is solely developed by IAR Systems. It is not based on any other known IDE framework. It follows rather minimalistic approach, with simple menus and icons. Old school Windows 98/XP like design. While this is a great approach for firmware programmers, the lack of functionality is concerning. For example the auto completion, syntax style, text and icon hovering is not working well if at all.

Syntax style is very basic and not giving intuitive feel of the code quickly. Since it is proprietary software the plugin ecosystem is also missing so there are no tools or they are hard to be found. Also it appears that it does not support Windows 8 well, as it was the OS used by the author. The text editor area automatically zoomed out the text too much. There were other inconsistency with the UI as well like overlapping panels, hidden buttons, etc.

For the price schemas there are multiple flexible options. From USB dongle limited versions to floating licenses based on the local or global network. In turn the application pricing schema is split by the functionality provided between different versions – standard, limited and baseline.



```c
#include "radioApp.h"

#include "radio.h"
#include "serialPCInterface.h"

#define LED_Togg    P1OUT ^= 0x01;  P1DIR |= 0x01;

extern unsigned char packetReceived;
extern unsigned char packetTransmit;

unsigned char RxBuffer[255], RxBufferLength = 0;
const unsigned char TxBuffer[6]= {0x06, 0xAA, 0xBB, 0xCC, 0xDD, 0xEE};
const uint8_t temperatureSensorData[9] = {0x09, 'S', 0x08, 0x10, 'T', 0xFF, 0xFF, 0xFF, 0x7F};
unsigned char buttonPressed = 0;
uint8_t toggle = 0;

void radioApp()
{
    initCC1101();
    receiveOn();

    while(1)
    {
        if (buttonPressed)                          // Process a button pressed --> transmit
        {
            buttonPressed = 0;
            P1IFG = 0;

            receiveOff();
            transmit((uint8_t*)TxBuffer, sizeof TxBuffer);

            //Wait for TX status to be reached before going back to low power mode
            CCStatus_t status = 0;
            while( true )
            {
                strobe(RF_SNOP, &status);
                if ((status & 0x70) == 0x20)
                break;
                                    // If status is TX_UNDERFLOW strobe SFTX.
            }

            P1IE |= BIT1;                            // Re-enable button press
            LED_Togg;
        }
        else if (packetTransmit)
        {
```

Figure 7: Snapshot of the IAR EW IDE

3.1.1.2 Code Composer Studio

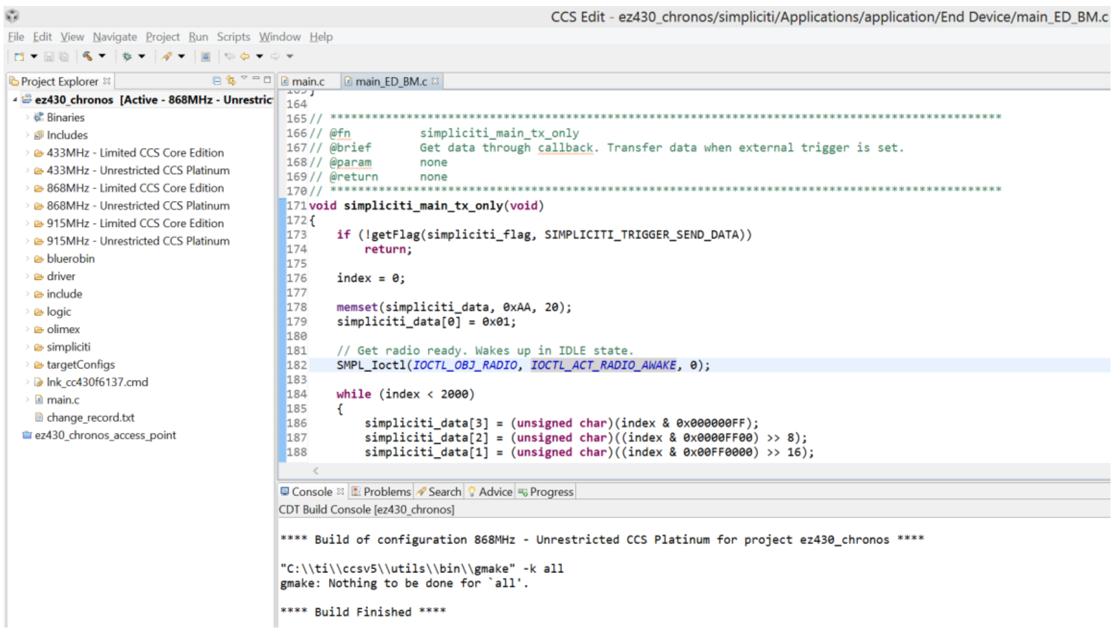
CCS version 5.5 was a new experience for the author of this work. Previously older version (version 4) of CCS has been used. So there were still a lot of familiarity, because since version 4 CCS is based on widespread Eclipse IDE. Fully supported version of the toolchain was used, because licensing server with 100 floating licenses was set up (thanks to the Texas Instruments European University Program) at the University. This was also required, since Chronos projects' compiled code would have not fit into the limits.

Chronos development software (see 3.1.3) projects were used right from the start of CCS experience. Workspace consisted of Chronos watch and USB access point projects. Configuring project's settings, pre-build, post-build and linker actions can be made in the graphical interface. Different menus, buttons and views can be completely customized as in any other Eclipse based IDE, which makes the graphical interface very comfort and easy to use. CCS adds new views like debugger windows (register, memory, variable watch windows), GRACE tool and ULP advisor windows.

The toolchain is based on GNU tools, therefore the POSIX environment is being used to invoke commands. Compiler supports wide variety of settings from GNU compiler language extensions as well. Developers who are familiar with the development with the GCC and make system, the adoption of CCS is seamless. IDE automatically manages makefiles and include paths, so the files included in the project and grouped into the folders do not need extra declarations and include paths. Linker and MCU specific header files are already included with the install of the toolchain. Same account true for the standard C/C++ runtime libraries.

Only hassle during the development was the support of some debugger/programmer drivers. For example Olimex-JTAG-Tiny-V2 programmer, which was used in the development, drivers for CCS version 5 were not officially supported. This is due to the fact that some subsystems of CCS has been changed between the major versions, so the 3rd party tool providers have not managed to keep up with the changes. But this problem is leveraged thanks to the CCS features like automatic firmware updating and drivers' updates of the programmer/debugger, like was the scenario with this case.

Licensing system is rich. From node locked single user to various amount of floating licenses. Licensing is based on the subscription where annual fees are applied. This means that every year subscription must be renewed. At the same time users are guaranteed to get all the updates and new versions with no extra cost. No functionality based cost scheme is applied. All the features of CCS are available with the subscription. This includes various DSP related features (like C6EZFlo, Image Analyzer), Android/Linux debugging even if you develop for only platform which do not use these features. Subscription includes all the Texas Instruments architectures so one can develop from ULP to DSP devices. Texas Instruments have developed even more features like GRACE (a visual peripheral configuration generator) and System Analyzer (analyzes and visualizes internals of the MCU like CPU load and memory usage) which makes it more powerful than the IAR EW and the cost of the toolchain is somewhat more flexible and less burdening than the IAR Systems provide.


CCS Edit - ez430_chronos/simpliciti/Applications/application/End Device/main_ED_BM.c
File Edit View Navigate Project Run Scripts Window Help
Project Explorer
ez430_chronos [Active - 868MHz - Unrestric
Binaries
Includes
433MHz - Limited CCS Core Edition
433MHz - Unrestricted CCS Platinum
868MHz - Limited CCS Core Edition
868MHz - Unrestricted CCS Platinum
915MHz - Limited CCS Core Edition
915MHz - Unrestricted CCS Platinum
bluerobin
driver
include
logic
olimex
simpliciti
targetConfigs
lnk_cc430f6137.cmd
main.c
change_record.txt
ez430_chronos_access_point
main.c
main_ED_BM.c
Console
Problems
Search
Advice
Progress
CDT Build Console [ez430_chronos]

```
// ******************************************************************************************************
// @fn          simpliciti_main_tx_only
// @brief       Get data through callback. Transfer data when external trigger is set.
// @param       none
// @return      none
// ******************************************************************************************************
void simpliciti_main_tx_only(void)
{
    if (!getFlag(simpliciti_flag, SIMPLICITI_TRIGGER_SEND_DATA))
        return;

    index = 0;

    memset(simpliciti_data, 0xAA, 20);
    simpliciti_data[0] = 0x01;

    // Get radio ready. Wakes up in IDLE state.
    SMPL_Ioctl(IOCTL_OBJ_RADIO, IOCTL_ACT_RADIO_AWAKE, 0);

    while (index < 2000)
    {
        simpliciti_data[3] = (unsigned char)(index & 0x000000FF);
        simpliciti_data[2] = (unsigned char)((index & 0x0000FF00) >> 8);
        simpliciti_data[1] = (unsigned char)((index & 0x00FF0000) >> 16);
```

```
**** Build of configuration 868MHz - Unrestricted CCS Platinum for project ez430_chronos ****

"C:\\ti\\ccsv5\\utils\\bin\\gmake" -k all
gmake: Nothing to be done for `all'.

**** Build Finished ****
```

Figure 8: Snapshot of the CCS IDE

3.1.1.3 Conclusion

Finally CCS was used throughout of the work that is presented here. Code Compose Studio provides IDE that is more functional and less erroneous, the support is more future proof and the Eclipse ecosystem provides many plugins and enhancements. Lately Texas Instruments have put more focus on supporting CCS than they do with IAR EW. Although the proprietary compiler of IAR Systems can generate more efficient code, the difference is dismal in the context of sensor network applications on MSP430.

	IAR Embedded Workbench	**Code Composer Studio**
Supported OS	Windows (XP, Vista, 7, 8)	Windows (XP, 7, 8), Linux
Cheapest license	~1200 USD (baseline package)	445 USD (node locked, single user, annual)
Trial version constraints	30 day evaluation or 4KB code size limit	30 day evaluation (can be extended) or 16 KB code size limit
Requirements for host machine	1GB RAM, 2GB disk space, at least Windows XP	1GB RAM, 300MB disk space at least

Languages supported	Assembly, C, C++	Assembly, C, C++
C standards supported	C89, C99, Embedded C++	C89, C99, C++03
Features	Librarian, Texas Instruments ULP advisor, C-Spy debugger, MISRA C, RTOS support	Librarian, GRACE, Texas Instruments ULP advisor, SYS/BIOS real-time OS, Linux/Android Debug, C6EZFlo, System Analyzer, Image Analyzer, Scripting, Optimizer assistant

Table 3: Comparison of IAR and CCS development environments [53, 57, 58, 59, 60, 61, 62]

** Quote for the IAR toolchain price was received via email.*

3.1.2 Wireless development kits

There were 2 development boards used throughout this work to test and run the solution developed. The basic differences were the size and the amount of included tools. While Olimex product is made to supply the user with a hardware, the Chronos kit and all its contents are made to showcase the capabilities of Texas Instruments low power solutions.

3.1.2.1 Olimex MSP430-CCRF

The board has been designed to jump start the development of CC430 radio devices. The development kit package includes one fully assembled PCB. The circuitry on the board includes voltage regulator with wide range input voltage (3-12V), external 32768 Hz and 26 MHz quartz crystal, JTAG connector, button, LED, I/O pin holes and antenna circuitry interface. The core of the system is CC430F5137 transceiver MCU. The size of the board is 78.49mm x 39.12mm x 1.00mm.The kit includes minimal amount of software examples for the CC430 family transceivers.

The simplicity and the size of the board is perfect for the development. One can easily access pins, while keeping the size of the board minimal. The I/O pin layout design with the Chronos kit watch (see next chapter) is compatible, which enables one to run the same code on both of the kits without any pragma switches at the code level. Only drawback of the kit is the need to purchase the programmer separately. For the development of this work the Olimex's MSP430-JYAG-TINY-V2 programmer was used. [63]

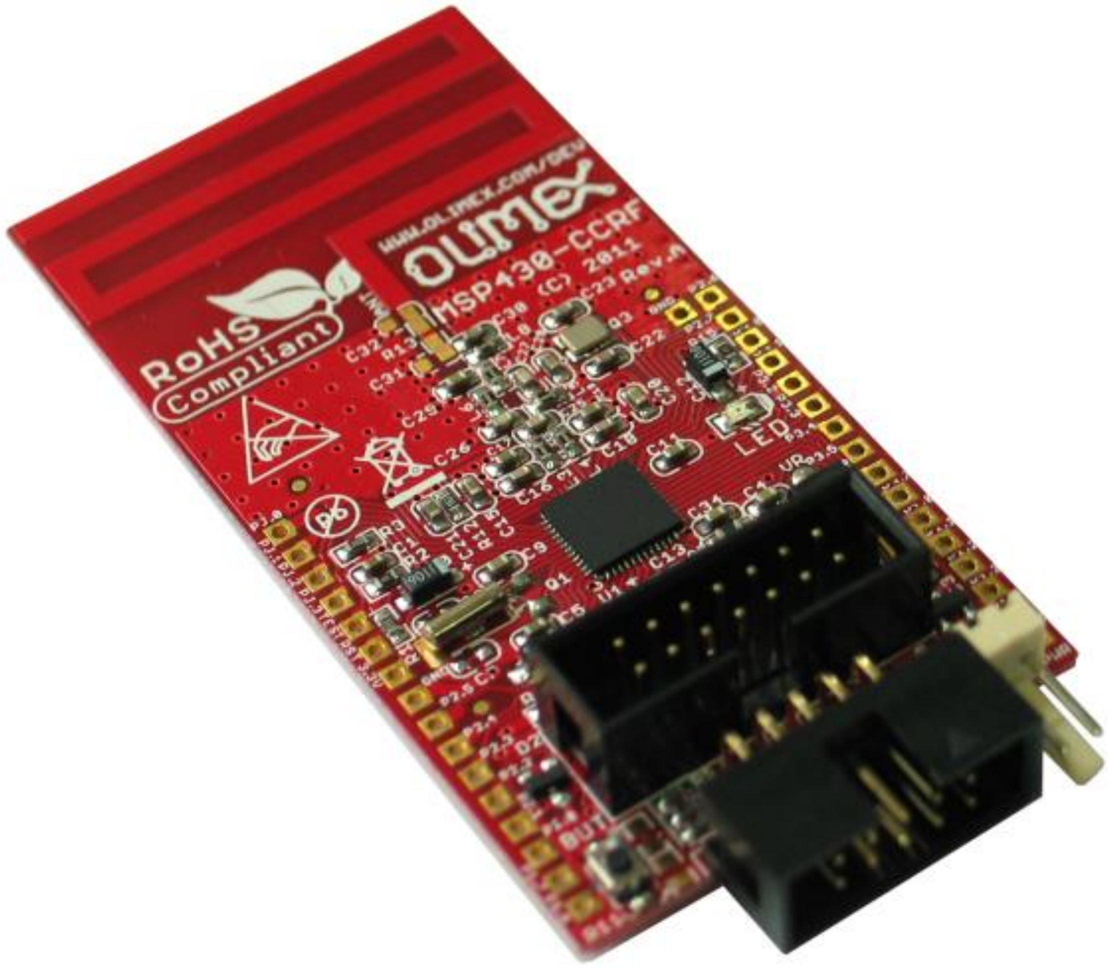


Figure 9: Olimex MSP430-CCRF development board [63]

3.1.2.2 eZ430-Chronos

Kit includes (Chronos WHITE):

- 1x eZ430-Chronos module with wristwatch housing
- 1x eZ430-RF USB debugging interface
- 1x MSP430F5509 + CC1101 USB RF access point
- 1x 4-pin solder-on debug connector for the USB RF access point

There are 2 different versions of this kit. In this work the "WHITE" version is used. Probably this kit has the best price to value ratio in the market (58 USD). Included devices provide full support for development and testing of the whole sensor network. Especially fancy is the sensor node, which by default is embedded in the wristwatch housing and the whole set is a complete wristwatch. The watch node incorporates touch and acceleration sensors manufactured by Bosch. Adding to that the example software is thorough, with multiple modes and the LCD screen on the watch much operations can be easily done with the stock software.

Out of the box settings already enable one to experiment a lot. User interface for the controlling (Chronos Control Center) end of the application is also powerful.

Support for all of the functionality can be done with the GUI application. Besides sensor data readings one can use the application to send commands to the PC. The support for keystrokes and mouse movement is already built in to the application. From the tabs on Chronos Control Center one can choose different modes of functionality: live acceleration data, PC control via watch, time synchronization, temperature and altitude logging and wireless firmware update. With another utility for PC – Chronos Datalogger – one can log the different sensor readings, over the long period of time.

Modifying the Chronos watch hardware is somewhat complicated due to the relatively tiny PCB. Available I/O is mostly occupied by the LCD. But the small size does not limit the kit to be able to operate at the low frequencies. All 3 different regions are represented: 433, 868 and 915MHz range. The example software can be downloaded and used as a basis for one's project. All parts of the software is left without obfuscation, so one can use it as a powerful application to modify it to ones needs.

This kit is the most suitable for the introduction of MSP430 RF devices. Other devices included (debugger/programmer, RF USB dongle) can be easily used for other MSP430 related projects as well, because they are compatible also with other MSP430 devices. Same for software, which is a great source of portable code for MSP430 RF devices. Software can be downloaded for the radio modules as well for PC applications. [64]

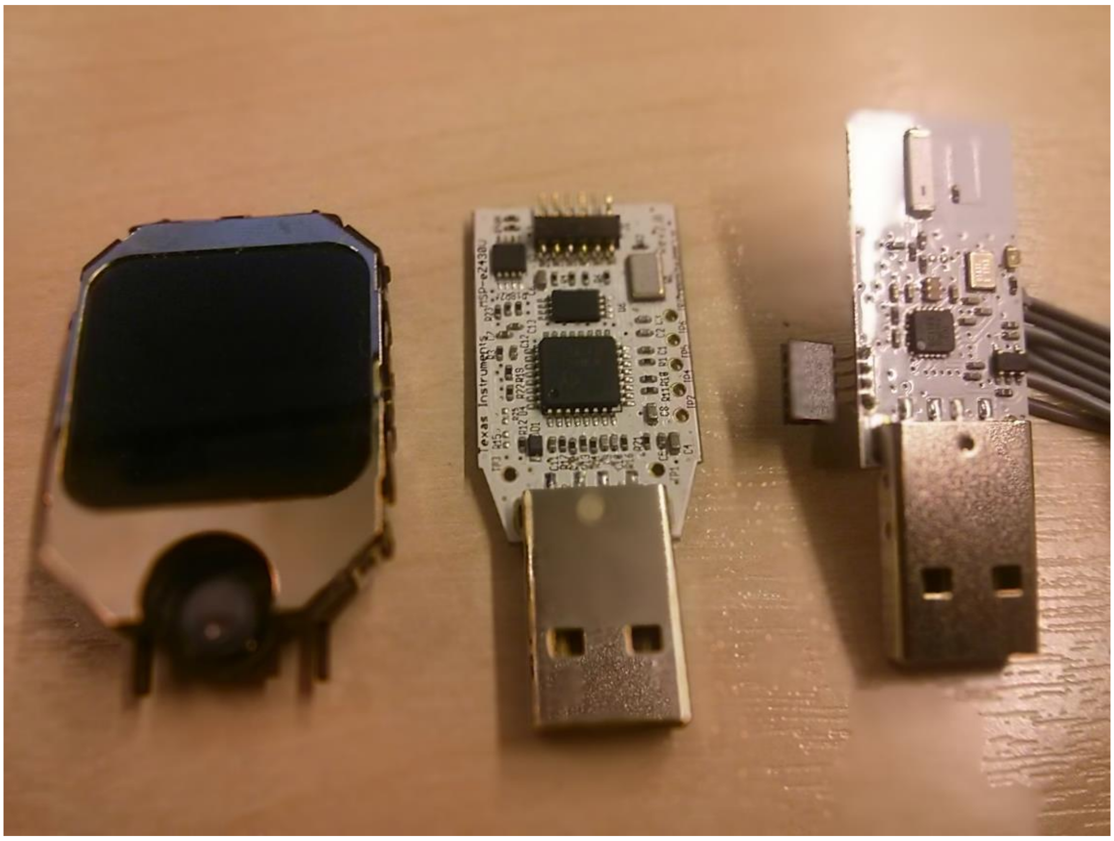


Figure 10: eZ430-Chronos kit devices – watch, programmer, access point

3.1.3 Texas Instruments Chronos software

The software library which was taken as a reference for the current work is a Texas Instruments Chronos wireless development system. It comes as supporting software package for the development kit of physical devices consisted of watch, access point and debugger/programmer (see Section 3.1.2.2). Besides the tools on PC to invoke commands and receive data on the devices it includes all the sources for the firmware of the devices and the graphical interface. Thus making it a perfect reference platform for a custom MSP430 radio design, enabling one to develop an application from top to bottom.

Software libraries used in this work are a Code Compose Studio projects for Chronos watch and USB access point. Firmwares for both of the devices are modified a lot since the logic includes a lot of Chronos application specific code. That means the communication layer and main logic flow is heavily modified for the Chronos watch features. If one wants to develop clean code (e.g. only code that is related to one's application) a lot of the Chronos specific code must be deleted and redesigned. That is valid for the project settings also. Nevertheless this approach is less time consuming than developing from the scratch.

Common layer for both of the projects is SimpliciTI radio library (Section 3.1.5). Since both of the radios, in order to communicate, must use the same parameters and protocol design. The same is valid for the Bluerobin radio module. Although just about anything else is different between two projects, since one is battery operated end device and the other one is USB connected radio access point.

3.1.3.1 Chronos watch software

Software modules are grouped into folders which makes the project easy to manage. Basic modules are SimpliciTI library for radio, driver modules (SPI, I^{2}C) for the logic of the on board peripheral modules (acceleration, pressure), logic module for the application menus (state variables, handling the shutdown states of the driver modules) and Bluerobin radio protocol specifics.

The top logic of the project is contained in the main.c file in the project root folder. Reading the code inside one can get a quick overview of the firmware, its states, modules and global variables. The `main()` function can be easily modified to meet the requirements of one's application.

As can be seen from the main logic, the essence of the application is quite thin. Application and peripherals specific initialization logic is implemented and no further modifications and conditional switching is not needed. The program continuous cycle is simple too. When the device has run the specific logic of one of the application features, the user interface (the LCD display on watch) is updated and the device goes back to low power mode. Waking up the device is done by the timers or buttons.

When the code gets analyzed and tested deeper the shortcomings arise in the context of this work. First, the dedicated timer is assigned to any of the actions including wireless data transmission, which means that it cannot be used for surveying some physical phenomenon over long period of time. It is present to limit the energy consumption when there is unintended triggering of the sequence. Second the frequency of the data packets is too sparse. Cause of this is the relatively heavy logic associated with the program flow of transmission and synchronous measurements of the sensor data. Last issue connected to the wireless data transmission is the link validation check. When the link has been first established, there will be a sequence of data packets transmitted, even when the addressee is not present anymore.

The benefits of the software stack is associated with the working wireless library and utilities for the CC430 modules. The integration of the SimpliciTI library, means that there is no need to supply and modify the library to one's project. A thin layer of board support package has been set up which supplies simple functionality to the library like I/O, timers, buttons etc. The MCU specific register related defines has been set up, so one do not have to adapt the library to the current radio chip. The utilities can be everything related to functionality like timers, LED lights, interrupts, sleep routines that can be reused as original or modified forms in one's application.

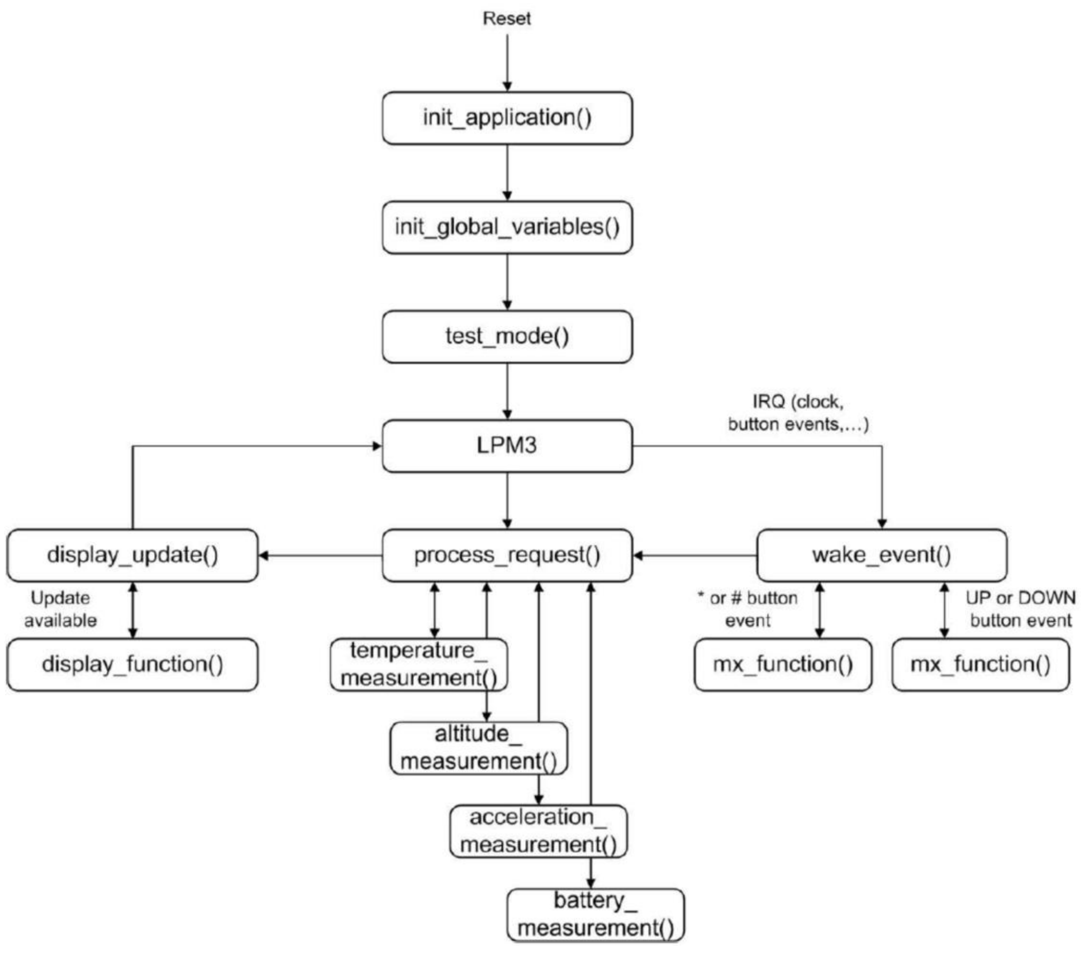


Figure 11: Chronos watch software flow [64]

3.1.3.2 Chronos access point software

The access point is essentially a driver for the backend, which in the context of Chronos application is a PC. It has two distinct components: USB logic and Chronos SimpliciTI radio network logic. Each part is being run asynchronously as much as possible where the Chronos SimpliciTI radio network logic is the application main flow. USB part is running mostly interrupt based since it has to move the data buffers from one end to another. The main principal moving information between the PC and the access point is command – data response logic. With every command and data transfer the PC must issue a command with length byte plus data, which will then trigger a response from access point. For example if wireless data is needed to be moved, the PC must first issue corresponding command to access point, which will transfer the data if available.

When command is invoked by the PC, first it gets decoded in the access point. If it is SimpliciTI specific, it is getting relayed to the corresponding wireless task. The PC does not have to handle the SimpliciTI logic, the wireless application specifics is implemented in the access point. The data, commands and modes of functionality of SimpliciTI application are all handled in the access point.

With the poll-receive logic with PC and thick layer of SimpliciTI logic for the access point link, the firmware is relatively slow. For example larger and more frequent data packets from the end device are getting lost, as was experienced during the tests. In the context of this work more data is needed to be put through, so the present logic must be changed. Another drawback is the support of the connection of one device only. This could be increased as it is supported by the SimpliciTI stack.

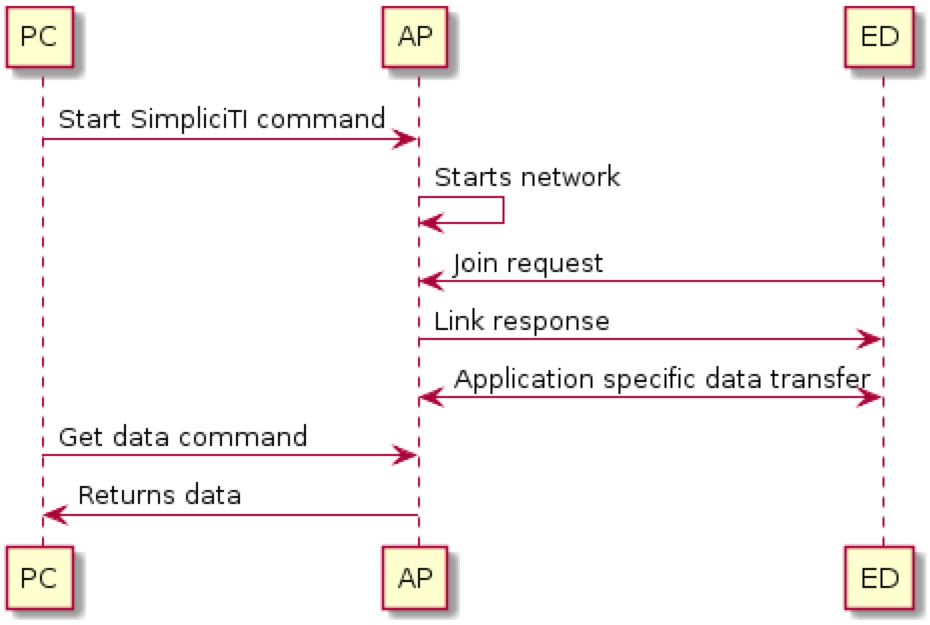


Figure 12: Data movement between the devices of Chronos development kit

3.1.4 CC1101 Chipcon radio transceiver

Besides the basic properties like the frequency ranges and supported modulation schemes that was covered in Section 2.2, more information is given about the internals of the transceiver. Features like packet handling, sensitivity and power consumption are described.

3.1.4.1 Differences between versions

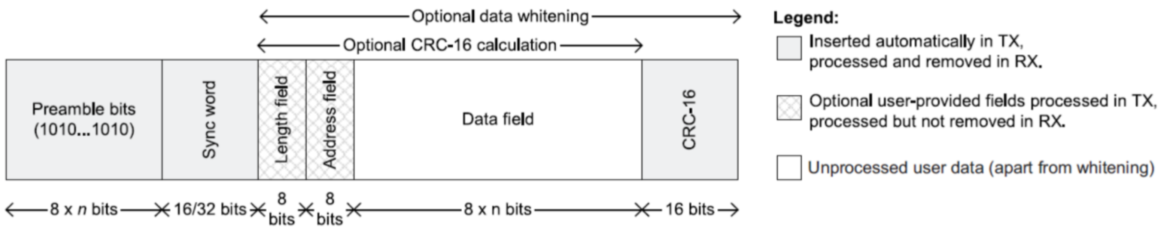
CC1101 radio is an upgrade from CC1100, which is not recommended for new designs anymore. While they remained compatible in code and register settings level, there were some fine tune enhancements. Main upgrades over the old version included the received signal attenuation option, receiving circuitry ADC settings retention, better noise immunity and more frequency bands. [65] Other than that the chips are interchangeable and can communicate to each other.

The differences between CC430 and the original CC1101 are also minor. Most of the changes are due to the fact that it is embedded into the MSP430. For example some signals are mapped differently (GDOx multiplexing) and the core can execute commands while in sleep state. Main differences that can be taken into consideration are the void of the forward error correction and interleaving and everything related to the states. Other than that no changes to the design that could affect the performance or command strobes execution. [*43*]

3.1.4.2 Design and features

Since the radio is designed focusing on proprietary solutions, it is not compatible with any of the major protocols under the IEEE 802.15.4 umbrella. Although the 802.15.4g standard have amendments, which included FSK modulation (supported by the circuitry) and more sub 1 GHz frequency bands. [66] But in practice the FSK modulation is not supported in PHY/MAC layer by any major protocols like ZigBee or 6LoWPAN. The FSK modulation can be used for Wireless M-Bus protocol for example.

The functionality has special features to support one's protocol design. For example hardware supported general packet format design that enables the application to optionally configure length and address bytes which are then automatically added and checked by hardware with each packet. For a robust performance and valid data, the CRC can be automatically added and checked also. All these fields can be used to do the automatic filtering of packets to speed up the packet processing. The packet length filtering is more varied. One can choose between fixed length, variable length or infinite length mode. Variable length mode enables to discard the packets which are not having preconfigured size. Receive and transmit buffers are 64 bytes in size, which are reserved only for application's data. Exception is when the receive buffer is configured to contain RSSI and LQI plus FCS bytes. Preamble and sync bits can be configured to have determined length and value. These mechanisms can contribute to the packet filtering and more reliable performance of radio link by enabling settling time for the receiving circuitry to detect the info bits correctly.



Optional data whitening
Optional CRC-16 calculation
Preamble bits (1010...1010)
Sync word
Length field
Address field
Data field
CRC-16
8 x n bits
16/32 bits
8 bits
8 bits
8 x n bits
16 bits
Legend:
Inserted automatically in TX, processed and removed in RX.
Optional user-provided fields processed in TX, processed but not removed in RX.
Unprocessed user data (apart from whitening)

Figure 13: The packet format of the CC1101 radio [43]

Chip has multiple energy and pass rate efficiency mechanisms. The simplest form of this is the CCA, the condition which must be satisfied before the radio will enter the transmit state. This helps to keep the noise low on the channel by not enabling to talk while there is some other transmission active at the same time. Whitening is a technique which applies some determined loops of code on the data. This can be used for encryption and data link reliability purposes. At the technical point of view it keeps the output bandwidth power distribution smooth. Using the chips applied on data, keeps it more random, since data can contain long sequence of zeroes and ones, this will decrease the error rate. Another technique that is for example applied on Bluetooth communication links, is the frequency hopping. When switching the frequency, the recalibration and settling delays must be taken into account. This is supported in the hardware where there is special registers to specify channel number and channel spacing.

Operation modes and sequence of states are programmable. That means that one can choose what happens after receiving or transmitting data or when the radio is woken up. Different timers are supported for the non-deterministic events. Receiver timeout and wake up on radio timers automatically change the state of the radio either to sleep or vice versa. All contributing to the energy efficiency and bigger data rate. There are four defined states: transmit, fast transmit on (initialized and ready to transmit, goes to transmit when data is supplied to FIFO buffer), receive, idle and sleep.

Finally the power amplifier, which greatly can reduce the power consumption and enhance the transmission range. At the runtime the radio can use 8 pre-determined values, which are described in the device PATABLE. The table is used automatically by the core for some modulation shaping techniques like ASK or OOK. The constants used to achieve certain output power at defined frequency are different and can be calculated from the datasheet. [51] Range of values for the output are from -30 to +13dBm. [51, 67, 68]

3.1.4.3 Programming model

Since Chipcon based radio cores are used in many different configurations in different designs (Section 3.1.2.1, Section 3.1.2.2 and Section 3.1.6.2) there are different ways to do the radio controlling and programming. Radio registers are like the microcontroller registers where everything system specific (from signals to threshold values to constants) can be controlled, read and configured. When one uses the SoC variant of the CC series microcontroller (like CC2530 or CC1010), one can access the registers in their code like every other register in the microcontroller memory. In the case of CC1101 the radio registers must be accessed externally. When used as a separate transceiver module the SPI interface is used and inside the CC430 the RF1A interface is used. Nevertheless same principals apply to both of the controlling modes. The RF1A interface is a collection of MSP430 internal registers that is used to interface the CC1101 radio core. RF1A interface can be used to control everything from data transfer, status feedback to interrupt vectors. The write byte/word, read byte/word approach is used for both the SPI and RF1A approach. There are different instructions defined that can be invoked via interface to set the state, read data or configure the registers. That means registers in the radio cannot be directly accessed. Some complexity is added with the interface models, where additional errors can be introduced. Nevertheless the interface model enables thoroughly to configure and control the radio core without any notable latency.

3.1.4.4 Detailed characteristics

Parameters based on the CC430 integrated core [51]:

- Supply voltage 2.0-3.6 V
- Programmable data rate from 0.6 kbps (26 kbps for MSK) to 500 kbps (MSK and 2-FSK) or 250 kbps (2-GFSK, OOK, ASK)
- RF crystal tolerance ± 40 ppm
- Sleep mode current consumption 100 µA, idle state 1.7 mA
- Receive current consumption 16 mA @ 250 kbps 868 MHz with -100 dBm input
- Transmit current consumption from 18mA to 36mA @ 868 MHz with output from -6 dBm to +11 dBm
- Receiver sensitivity -90 dBm @ 250 kbps 868 MHz, 2-GFSK modulation

3.1.5 SimpliciTI library

SimpliciTI is a wireless proprietary protocol aimed to be easy to use, lightweight and portable across the Texas Instruments' different transceiver controllers. It is targeted to CC and CC430 family of microcontrollers and can be ran on MSP430 microcontrollers too. Since it is not designed to be conforming to any other standard or protocol, it can be used by utilizing lightweight SimpliciTI API, to implement one's proprietary radio network and protocol on top of the stack. Source code is free to use and redistribute. Library can be downloaded in many configurations, since it explicitly does support different development kits and reference designs by Texas Instruments.

3.1.5.1 Architecture

The architecture consists of 3 different layers. The data link/physical layer also named as MRFI (minimal RF interface) contains everything specific to low level radio registers and radio hardware management. Second layer, network layer, deals with the logic of the SimpliciTI protocol, being the intermediate layer between the application and low level radio interface. The most abstract layer, the application layer, exhibits very basic up to 10 command API which can be used from the user's application to manage the wireless data transmission. One can also build up its application using only network layer (as was done partly in the current work), this means that the ports, addresses and contexts related to programming model must be manually set up. [69]

The 10 APIs with the quick explanation are as following [70, 71]:

- `SMPL_Init()` – initialize SimpliciTI stack.
- `SMPL_Link()` – initiate linking sequence with the devices that are listening.
- `SMPL_LinkListen()` – same as the previous, but will block until the ACK has been received.
- `SMPL_Send()` – transmit array of data.
- `SMPL_SendOpt()` – transmit array of data, with some predefined options like automatic ACK.
- `SMPL_Receive()` – receive the payload from the oldest frame in the buffer.
- `SMPL_Ioctl()` – enables to configure the network layer (radio mode and properties, address, link tokens).
- `SMPL_Ping()` – ping the device.
- `SMPL_Unlink()` – removes the link from the connections table.
- `SMPL_Commission()` – modify and access the connection table and different port assignments.

3.1.5.1.1 MRFI

Relatively thin layer, where all the CC radio core specifics are being implemented and defined. The most direct connection one usually have with this layer is the settings file generation. Settings are described in a C header file where the values of the registers are defined with the following pattern.

```
#define SMARTRF_SETTING_{REGISTER_NAME} {NUMERICAL_VALUE}
```

Values are read during the compile time by the preprocesser and placed into the array, which is used to set up the radio core when the MCU starts. The settings can be generated using the Texas Instruments' Smart RF Studio (see Section 3.1.6.3). The generated file simply can be placed into "smartrf" folder inside the "mrfi" layer folder tree.

There are total of 17 APIs which are exhibited in the *mrfi.h* header file. These include managing and reading the radio core state, transmit power, receive and transmit FIFO buffers and different features like address filtering.

One can easily use this layer as a software reference for its own Chipcon based radio core software implementation. Coupled with the Texas Instruments developer aid software, implementing only this layer is enough to send and receive data from point to point connections.

3.1.5.1.2 Protocol

Protocol part of the stack is more or less implemented in the network layer and partly in the application layer too (when hardware specific preamble, synchronization sequence and FCS is excluded). Data packets are relatively simple and with low overhead. Depending on the packet the SimpliciTI packet data overhead can take up to 12 bytes. That includes length byte, destination and source addresses (both 4 bytes), port and device info byte plus transaction ID byte. More overhead could be present depending on the exact radio core, but usually consists of atleast RSSI, LQI with FCS check bytes. Packet size in total is defined by the radio core fifo buffer sizes. In the case of CC1101 used in this work, the buffer can allocate 64 bytes. When the SimpliciTI overhead and frame info is subtracted, the maximum payload length of 50 bytes can be sent by the application. The maximum application payload varies but can be up to 52 or 113 (for example CC2420 [72]) bytes.

In the protocol headers, the basic functionality is utilized. The length byte indicates the length of the total packet including length byte itself. The port byte, indicates to which port the frame should go to (ping, link, join, user application, etc.). Two most significant bits specify encryption and forwarding options. The address bytes specify the source device and destination, based on which the received frame can be discarded, forwarded or saved. Device info specifies the device type, frame acknowledgment response or request status, receiving state, and frame hop count. Transaction ID byte is an incrementable indicator showing the sequence of the frame. An example flow of packets can be seen in the appendix A.

3.1.5.2 Configuration

Library features two simple configuration files, which can be easily edited to achieve the desired configuration. Configuration files can be found in the „Applications/configuration“ folder in the library folder tree. The „smpl_nwk_config.dat“ has all the definitions for the network, this must be ecquivalent on all of the devices across the network. One can set up the encryption, link and join tokens, maximum payload size and many more fundamental properties of the network. This configuration file do not change based on the device type. The „smpl_config.dat“ file is specific to a device type. Properties like device type, device address, input and output frame queue sizes and number of maximum connections are specified. This file most directly affects the memory usage and performane of the device. There can be 3 different types for SimpliciTI based networks: end device, access point and router. The configuration should be more lightweight on memory for end devices, while other device types should have reasonable size queue buffers and connection tables.

3.1.6 Tools

Hardware and software tools were extensively used throughout the development process. RF tools provided information that would have not been obtained any other way. All the articles (except one software tool) are Texas

Instruments products, designed to complement the development process of their low power radios.

3.1.6.1 CC1111 USB evaluation module kit

USB sniffer was used in conjunction with the SmartRF packet sniffer (see section 3.1.6.2) software. It is a simple to use device, which is controlled by the sniffing software. It contains CC1111 SoC, with the radio similar to CC1101, making it compatible (with slightly different register setup values) to communicate and receive packets. It can capture SimpliciTI and other proprietary format data that is compatible to the radio. . The 8051 based MCU contains radio and USB logic which is based on the libraries that are available and free to download. It serves the purpose of reference design and software demonstrator of how to use CC1111 based SoC devices. [73]

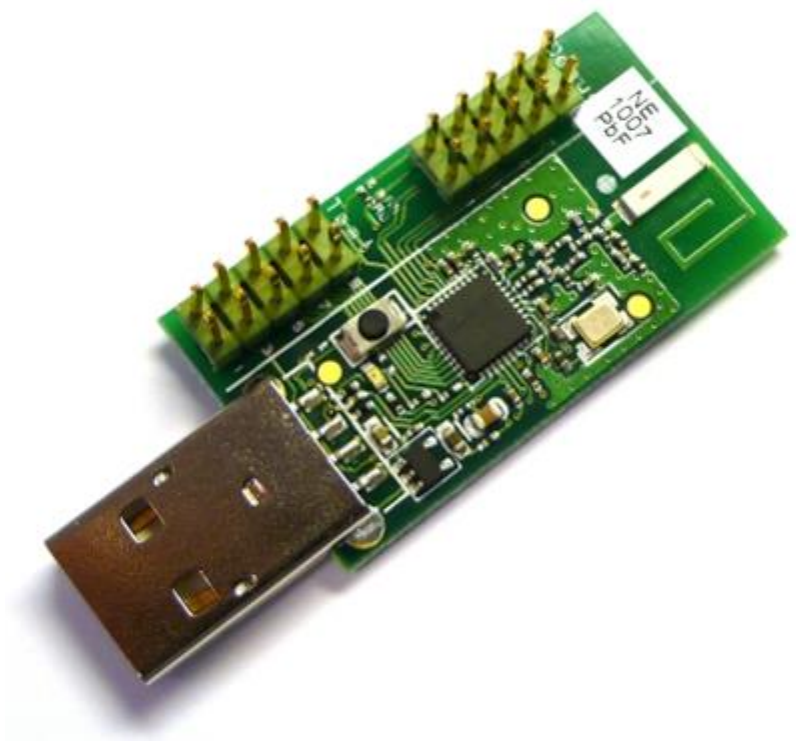


Figure 14: CC1111 USB sniffer device [73]

3.1.6.2 SmartRF Protocol Packet Sniffer

It is a Texas Instruments software to support the sniffering of the wireless data links. Supports multiple protocols and devices. From Bluetooth to ZigBee to proprietary format protocols (including SimpliciTI) can be parsed and visualized in the GUI. It has an extensive list of supported devices and development kits (over 10) that can be interfaced with the tool. The radio interface settings for the device can be loaded inside the tool, which means no reprogramming is needed for the device to start to capture radio traffic with different parameters.

The user interface enables to select different filters for the packets to be displayed and select the packet fields to be visible or hidden. Addresses of the devices are automatically recorded and included in the address book, where one can convienently filter devices. Interface has also the timeline where all the traffic is visualized in sequence per device activity. Captured data can be saved to binary format ".psd" file. Tool enables to load the data from the file also. PC application enables to forward all the data via sockets to UDP ports, so it can be highly customized and integrated to one's application. The example output of the wireless data is exhibited in the screenshot in the Appendix A. [74]

3.1.6.2.1 SmartRF Packet Sniffer binary parser

The large amounts of recorded data by the packet sniffer needed to be processed. Since the packet sniffer can save the recorded data only to a proprietary binary format the parser was developed. Parser reads the binary *psd* extension file and translates the data to a CSV file. It is written in C++ as a Visual Studio 2013 project, but the C++ standard library is heavily used and therefore should be able to be compiled with every standard toolchain.

The SimpliciTI packet binary format saved by the SmartRF packet sniffer contains length of 271 byte records. The record includes utility headers like timestamp and packet number as well a captured packet. The data that do not fill the entire 271 byte unit, is left empty, therefore the file size is packets captured times 271 bytes. An example of the parsing logic of the tool is included in the Appendix C.

3.1.6.3 SmartRF Studio

SmartRF studio is a graphical tool to support the development of the Texas Instruments low power radio devices. It can be used to evaluate the radio devices and generate register settings. The generated settings can be immediately tested on the connected devices – the radio link can be analyzed in detail. Tool supports more than 30 Texas Instruments radio devices, including the CC430 family. [75] Evaluation is supported with the features like packet handling and command strobe panel. That enables one to analyze and create custom packets. Also to invoke command inside the SoC devices.

In this work it was used to generate register settings for the following devices: CC430F5137, CC430F6137, CC1111, CC1101. The tool was used in „expert mode", where one can specify each register settings and radio link main properties like channel spacing and deviation frequency. An example snapshot of the register settings window and export can be seen in appendix D. The exported settings can have different formats. For one's software project the C header can be exported, for packet sniffer the „prs" extension and multiple versions of XML formats.

3.1.7 Software improvements

The development of more high performance sensor network required fundamental changes in the software of access point and end device. While the Chronos library projects were a great starting point, they were tightly bound to the example development kits and the features exhibited. For example one can't use the main transmit procedure of SimpliciTI (`simpliciti_main_tx()`) to send data at very high rates, because it includes the sampling of acceleration and other application level logic that renders the sequence slow. Same can be said for the access point, where the poll-response logic just did not stand up to the high frequency polling. While the USB is fast and can have high data rates, the encapsulation of small packets introduce big overhead and lag of the bus. The task of the software improvements is to find the bottlenecks and make the platform usable as a generic SHM library.

3.1.7.1 Sensor node

Restructuring needed a removal of the Chronos application logic flow first. Initialization and setup of the application specific modules (stopwatch, Bluerobin wireless stack, calorie counter, etc.) were removed. When the overhead of the special functionality was removed, the wireless data transfer routines had to be modified. Most of the work was spent on fine tuning the wireless data transmission, packet format, timestamp generation and link robustness, the problems described in Section 3.1.3.1 The requirements of the node is to have high data rate (penetrate radio limits), packets acknowledgment functionality, supported features for testing and easily modifiable software.

First the most minimalistic and economical sequence of commands needed to be found. Since the SimpliciTI APIs are synchronous, keeping the logic as small as possible enables to increase the transmission rate. Overall the additional functionality is not needed anyway and when SHM application is built upon the developed software it must be done easy and fast. The following snippet of code is used to send data in the developed software.

```
void joinNetworkAndSendData()
{
      open_radio();

      if (simpliciti_link())
      {
            // Get radio ready. Wakes up in IDLE state.
            SMPL_Ioctl(IOCTL_OBJ_RADIO, IOCTL_ACT_RADIO_AWAKE, 0);

            setFlag(simpliciti_flag, SIMPLICITI_TRIGGER_SEND_DATA);

            SMPL_Ioctl(IOCTL_OBJ_RADIO, IOCTL_ACT_RADIO_RXIDLE, 0);

            simpliciti_main_tx_only();

            clearFlag(simpliciti_flag, SIMPLICITI_TRIGGER_SEND_DATA);
      }

      SMPL_Ioctl(IOCTL_OBJ_RADIO, IOCTL_ACT_RADIO_SLEEP, 0);
      sInit_done = 0;

      close_radio();
}
```

The procedure `joinNetworkAndSendData()` can be called anytime and anywhere in the code. It powers up the radio core (`open_radio()`), joins the network (`simpliciti_link()`) and then using the SimpliciTI APIs (see

Section 3.1.5.1) triggers the data transmission. All the procedures called are default APIs of the Chronos or SimpliciTI stack, except the `simpliciti_main_tx_only()` which is modified according to this work requirements. One can easily insert the application specific functionality into this procedure or call it every time when the data is needed to be transmitted (increases overhead). For example in the case of demo application and tests, there is modified `simpliciti_main_sync()` before the transmit procedure, to get the synchronization timestamp from the access point.

For the robustness of the link the mechanisms provided by SimpliciTI APIs can be used. There are two different approaches: automatic acknowledgment and user application acknowledgment or replay.

The `SMPL_SendOpt()` API has the acknowledgment option (`SMPL_TXOPTION_ACKREQ` flag) as one of the argument. When the option is specified the automatic ACK flag is set in the packet header, which informs the receiver to send the confirmation. It provides the lowest overhead in the code, seamlessly returning status code with the procedure. However, when tested, it resulted that not all the ACK responses could not be determined by the stack. When analyzed, the logic is implemented so that between the transmission and ACK receiving the no-operation wait is invoked. When there is not the packet received within the wait duration, the confirmation is discarded. For alleviation one could increase wait time, but that will increase the total synchronous operation time, which in turn decreases the data throughput. With simple applications this approach is recommended, for more sophisticated requirements custom acknowledgment logic should be implemented. The user application can use the `SMPL_SendOpt()` and `SMPL_Receive()` APIs for the confirmation or use embedded logic inside the packet. With the latter approach there is more specialized functionality depending on the application and can increase the reliability while not hampering the throughput.

One of the tasks which was not directly connected to the radio network, but needed to be used as the transmission data, is the timestamp. Timestamp is the essential data in the sensor measurements. The MSP430 platform features RTC module, which asynchronously can count the time, without requiring direct processing power and can function in various low power modes. The drawback however is the second resolution which is not satisfactory for the high sample rate measurements. At least a millisecond resolution timestamp is needed, which would enable sample rates up to 1 kHz. The MCU Timer_A was utilized for that purpose. A capture-compare mode is used where the interrupt is generated approximately after each millisecond. The input frequency for the timer is the output of external low power 32768 Hz (±20ppm) quartz crystal. The base counter is the Unix `time_t` data type, which is synced from the access point. The 32-bit universal time structure and 16-bit millisecond is sent with the packets in the test and demo modes. The following code demonstrates how the timestamp is counted.

```
void timestampTick()
{
     if (++s_timestamp.milliseconds >= 1000)
```

```
    {
         s_timestamp.seconds++;
         s_timestamp.milliseconds = 0;
         millisecondCorrectionCounter = 1;

           TA0CCR1 += 50;

           return;
     }

     if (millisecondCorrectionCounter++ >= 4)
     {
          millisecondCorrectionCounter = 1;
          TA0CCR1 += 32;
     }
     else
     {
          TA0CCR1 += 33;
     }

}
```

The main problems with the stock Chronos application software got resolved while the Chronos functionality and project remain usable. The software can be run on all the CC430 family devices as the development used both Chronos watch (CC430F6137) and Olimex CCRF development kits (CC430F5137). The modified application gives a great basis for the wireless network application. It is used for the tests and demo of this work since it is easily expandable. The diversity of different setup applications can be seen in the Appendix F where the project's Git source control tree is visualized.

3.1.7.2 Access point

Besides the SimpliciTI library, that is included in the end device as well and is more or less the same, access point software runs USB stack too, which adds burden to the hardware. The software is being run on MSP430F5509 and the CC1101 radio transceiver is interfaced via SPI. The shortcomings exist at both functional libraries. The software modifications add the support for multiple devices in the wireless network, higher wireless data rate and faster data rates on USB communication.

The poll-receive logic got eliminated from the data transfer flow. The bottleneck already existed with the low data rate radio settings and needed to be changed. Data transfer was slow because, the access point needed to decode the message, forward it to the radio module, which supplied data and then the replay could be sent back. Also when there were packets that did not got polled from PC, the data would have been lost. Another drawback of the logic was the poll packet format, where the length byte should have been determined before the data packet was to be received, which in the context of this application is impossible. The flow got redesigned. When the data is received in the SimpliciTI module it is automatically appended to the USB buffers and sent to PC. So the burden of handling data could be switched to operating system COM port buffers. This approach reduced the overhead in the accompanying PC tool as well. The software can simply read the OS COM buffers at the time it is suitable for the code flow. With the OS buffers big enough for storing serial port data, there is automatic buffering of data by the platform too.

The connection support for more than one end device got implemented. For that purpose a SimpliciTI device specific configuration file „smpl_config.dat" was modified. The preprocessor definition `NUM_CONNECTIONS` specifies the maximum number of supported devices. In order to support features in the library runtime configurations the application logic needed to be changed too. The link id table needed to be set up and integrated into the application flow, where there should be divided processing window for each linked device.

Finally the timestamp synchronization process was implemented. The timestamp is supplied with the SimpliciTI start command. When one starts the end devices before the access point, that means the sensor nodes are sending join frames before the join window is opened in the access point, one could get synchronization error of 1 second (because end devices are sending join frames one per second). The opening of the join window in the access point will immediately trigger the linking frames to be exchanged between entities and synchronization error therefore is limited by the end device polling rate. No resynchronization of the timestamp was implemented.

The access point got the support of multiple end devices, fast wireless data relaying functionality and for the testing and demo utilities a timestamp synchronization flow. For the tests the access point software was again modified to encapsulate frame receiving information with the packet sent to the PC.

3.1.7.3 Access point control tool

To control the access point and receive data, a command line tool was written in C++. A Visual Studio 2013 project was set up with along the packet sniffer parser (Section 3.1.6.2.1). Goal for the tool is to have a lightweight client for the access point, which could be easily modified for different tasks.

Tool essentially consists of 1 module, where the communication with the access point is realized and the packets from the COM port are being read and sent. For the low level COM handle drivers the *BM_Comm.c* and *BM_Comm.h* source files are used, which are from the Chronos kit PC tools application. The tool opens the serial port, specified by the supplied numerical value on command line. Then sends the SimpliciTI start command to the access point and then starts to parse the received data. The tool has different versions depending on the output data format, but common output format is the CSV format. The parsing and the communication with the command line are running in two parallel threads in order to keep the tool responsive and fast.

The source code for the tool along with packet sniffer parser can be seen at the following link: https://github.com/svenKautlenbach/SmartRF-SimpliciTI-Chronos-PC-suite. For a snippet of the parsing logic see Appendix G.

3.1.8 Radio network

When the prototype system design was being discussed there were different configuration and setups discussed for the whole network data exchange. First ideas were concentrated on pure power consumption point of view. The idea was not to have the radio transceiver on every node. The data was designed to be transferred first over the common wired network to the concentrator. Aim was to have LIN like network where only one wire would have been needed for the data transmission between the sensing node and the transceiver master node. MSP430 peripherals also supported it, thanks to the various UART settings. Each micro network would have had one master transmitter node. The concentrator would have had the energy source with a capacity large enough to power the sensing nodes. This design had one big advantage – low power consumption. The power of radio transceiver would have been completely eliminated on the sensing nodes. Charging the nodes would have been more compact either, since only one device for each micro network would have had the need to be charged. Also there would have been some size reduction of the nodes, since the radio part with the PCB antenna would have been eliminated. Also there would have been downsides as well. Wires in the composites must be small enough to not to alter the properties of the composite itself. Also it would have added extra complexity in the production of the material and a new approach would have been needed to be developed. Network specific problems also raised. Synchronizing the measurements all over the wired network until they arrive to the end point via the concentrator would have added overhead. The concept was initially tested by the author of this work, but did not succeed due to the complicated and distributed data management.

Finally the wired nodes were cut from the network and the transceiver nodes were chosen to form the network. It was supported by the Chronos libraries. The limitations of the library was needed to be overcome though. The data rate settings for the network provided maximum data of around 70 kbps, although the hardware enabled more. Essentially the network was consisting of only 2 devices, making it a point to point between access point and one end device. The changes were needed on the radio hardware settings level and the SimpliciTI radio protocol level too.

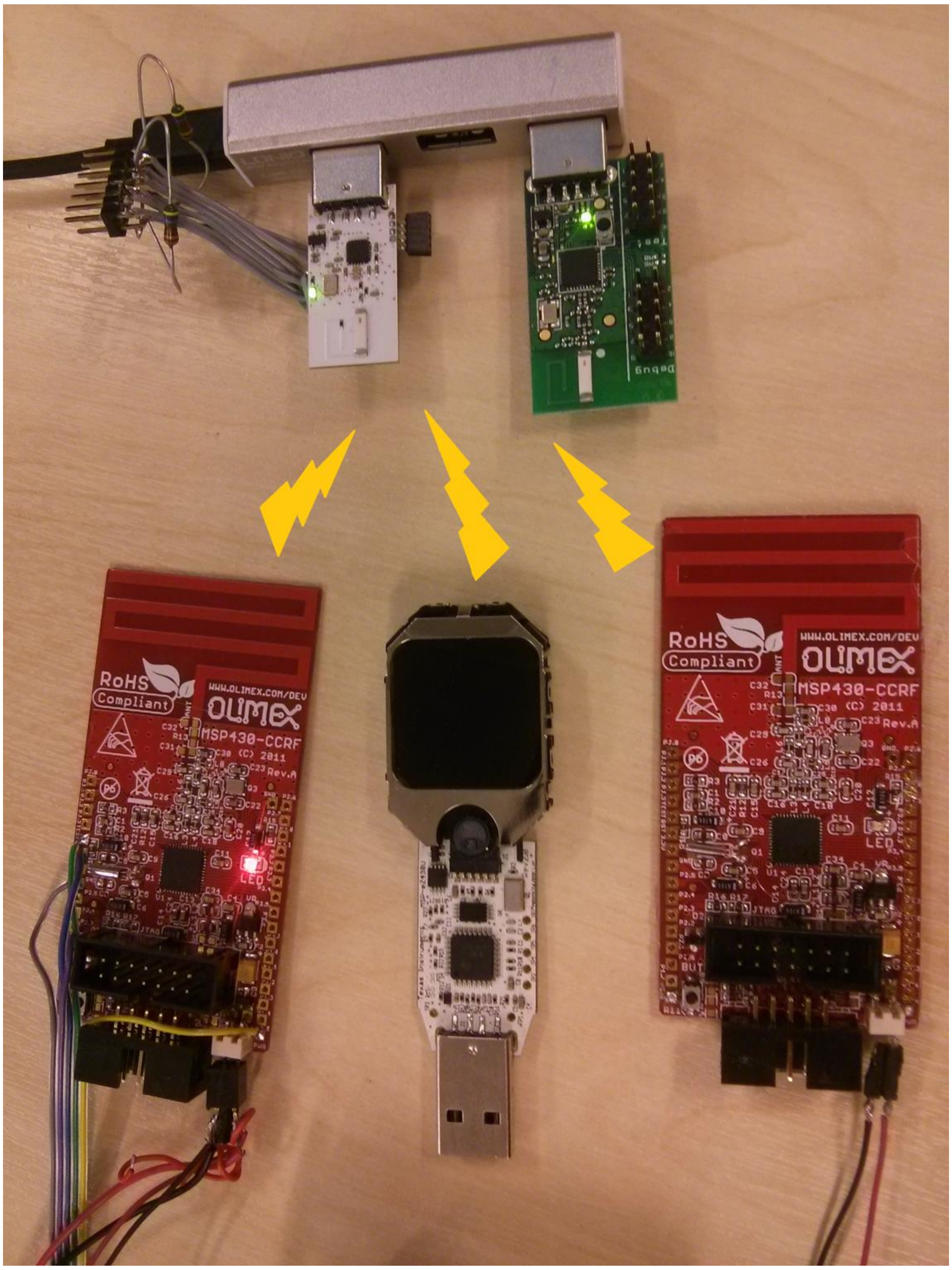


Figure 15: Complete set of devices used throughout development – access point, packet sniffer (upper row), Olimex boards with Chronos watch module in between (bottom row)

3.1.8.1 SimpliciTI design

The network is a simple collection of point-to-point connections – star network. The master of the network is a single access point and the end devices are all linked to it. Essentially SimpliciTI stack supports router devices and mesh network configuration, but this practice was not tested or supported in this work.

Important defined values in the SimpliciTI network configuration file:

- MAX_APP_PAYLOAD=50
- MAX_NWK_PAYLOAD=9
- APP_AUTO_ACK
- EXTENDED_API
- MAX_HOPS_FROM_AP=1

Important defines in the access point specific configuration file:

- NUM_CONNECTIONS=8
- ACCESS_POINT
- AP_IS_DATA_HUB
- STARTUP_JOINCONTEXT_ON

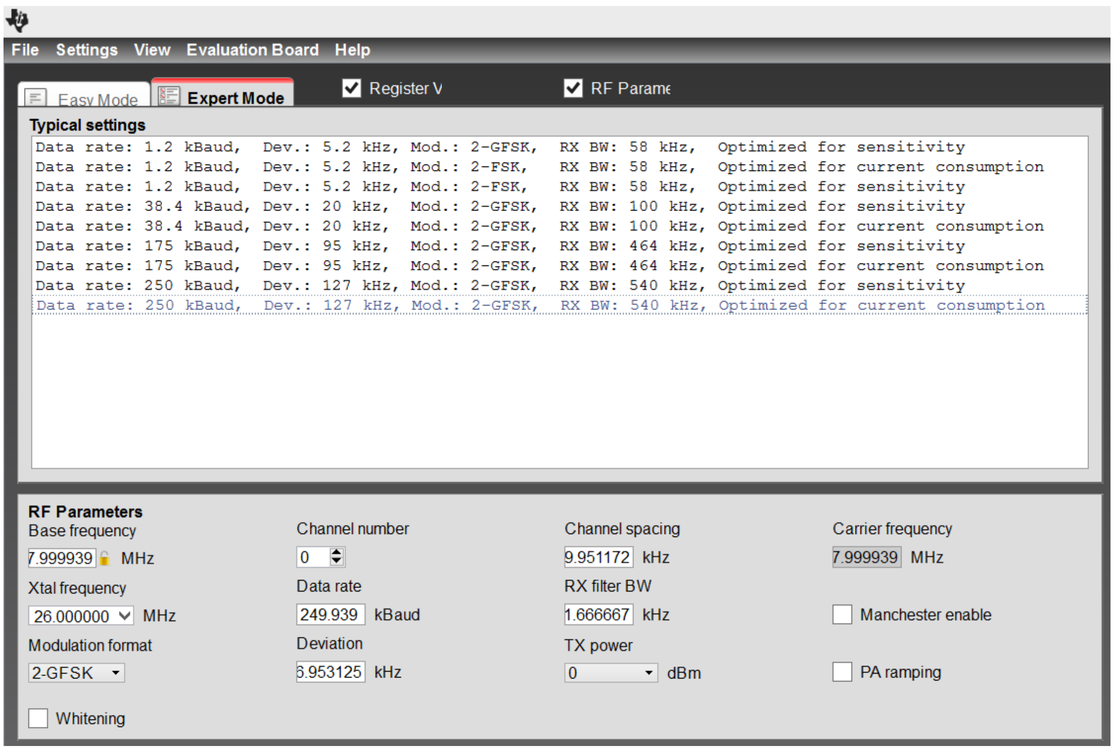
Important values for end device specific configuration file:

- END_DEVICE
- NUM_CONNECTIONS=1

The summary based on these defines tell that the maximum user application payload length can be 50 bytes, which is the maximum for this type of radio. The extended features of the SimpliciTI library can be used, like auto acknowledgment, which is used in the tests. The hops and connection count on end device hint that there is point to point connections. The data hub definition along with the startup join context will make the access point automatically listen for a new join frames from other end devices.

3.1.8.2 Register settings

To support the high data transfer the new radio register settings were generated. Essential change is the data rate, along with the channel spacing and deviation parameters. The modulation scheme was unchanged, although there could be higher data rates with other modulations schemes like MSK. The use of the Gaussian filtered FSK modulation is used because of the lower noise in the channel and conformance to the IEEE 802.15.4 g standard. The settings were exported as a C header file and put into the software project, where the values could be included in the compile time.



Typical settings

```
Data rate: 1.2 kBaud,  Dev.: 5.2 kHz, Mod.: 2-GFSK,  RX BW: 58 kHz,  Optimized for sensitivity
Data rate: 1.2 kBaud,  Dev.: 5.2 kHz, Mod.: 2-FSK,   RX BW: 58 kHz,  Optimized for current consumption
Data rate: 1.2 kBaud,  Dev.: 5.2 kHz, Mod.: 2-FSK,   RX BW: 58 kHz,  Optimized for sensitivity
Data rate: 38.4 kBaud, Dev.: 20 kHz,  Mod.: 2-GFSK,  RX BW: 100 kHz, Optimized for sensitivity
Data rate: 38.4 kBaud, Dev.: 20 kHz,  Mod.: 2-GFSK,  RX BW: 100 kHz, Optimized for current consumption
Data rate: 175 kBaud,  Dev.: 95 kHz,  Mod.: 2-GFSK,  RX BW: 464 kHz, Optimized for sensitivity
Data rate: 175 kBaud,  Dev.: 95 kHz,  Mod.: 2-GFSK,  RX BW: 464 kHz, Optimized for current consumption
Data rate: 250 kBaud,  Dev.: 127 kHz, Mod.: 2-GFSK,  RX BW: 540 kHz, Optimized for sensitivity
Data rate: 250 kBaud,  Dev.: 127 kHz, Mod.: 2-GFSK,  RX BW: 540 kHz, Optimized for current consumption
```

RF Parameters

Base frequency	Channel number	Channel spacing	Carrier frequency
7.999939 MHz	0	9.951172 kHz	7.999939 MHz
Xtal frequency	Data rate	RX filter BW	
26.000000 MHz	249.939 kBaud	1.666667 kHz	☐ Manchester enable
Modulation format	Deviation	TX power	
2-GFSK	6.953125 kHz	0 dBm	☐ PA ramping
☐ Whitening			

Figure 16: CC430 radio register settings

3.2 Evaluation of wireless nodes

Infrastructure used for the tests included a Windows 8.1 equipped laptop, Chronos access point (Section 3.1.2.2), Texas Instruments' CC1111 USB evaluation module kit (Section 3.1.6.1), Olimex MSP430-CCRF development boards (Section 3.1.2.1) and battery pack to power the sensor devices (see Appendix B).

For the tests the access point firmware was modified to meet the requirements of data analysis format. Therefore the access point parsing tool (Section 3.1.7.3) was modified too. Same is true for packet sniffer parsing tool (Section 3.1.6.2.1).CSV files generated by those tools were later imported to Microsoft Excel where they could be analyzed. The firmware of the end device was modified according to the test, but the common changes were the inclusion of timestamp and packet counter in the packet data.

The quantity like tests were carried through in the controlled conditions, which means no signal blocking and scattering objects between the end device and the access point and near proximity were present to influence the results.

3.2.1 Data rate tests

The goal for the data rate tests was to penetrate the limits of the radio hardware in conjunction with the Chronos and SimpliciTI software. The results of the tests give approximate limits of the developed network. Data would help to answer questions like "What are the real data rate limits?", "Where do the bottlenecks exist?" and "What can be done to alleviate or eliminate the problems?". Finally the statistical figures can be constructed and analyzed, based on the test results.

Devices are situated so that sensor devices are about 55 centimeters apart each other. The sniffer and access point are connected to common USB hub device, which is connected via cable to a laptop. They are situated between end devices (see Appendix A).

Software running in end devices is modified specifically for the tests, but no library specific parts are altered. Just the application flow and data packet specifics. Data packets have structure that all the important information could be analyzed with the tests. Data needs to contain information about the packets' sequence number and the time at the end device at the time of sending. Additional information is encapsulated by the access point to have more sophisticated statistics per link. The data that is counted here is the actual application payload (application useful data), without any of the SimpliciTI overhead and radio hardware specific bytes, which makes a total of maximum of 50 bytes per packet (section 3.1.5).

timestamp (time_t)	milliseconds	packet index	random data
4 bytes	2 bytes	4 bytes	40 bytes

Figure 17: End device packet structure for the tests

link ID	packet counter	RSSI	LQI	FCS	sensor node packet
1 byte	2 bytes	1 byte	7 bits	1 bit	50 bytes

Figure 18: Access point packet structure for the tests

3.2.1.1 Maximum payload without ACK

There were more than 10 test runs, each with 10000 packets sent from the end device. The idea was to burst out data at the maximum rate, without intervening the software flow with other tasks.

	Average value	**Delta**
RSSI (dBm)	-44.55	0.76
LQI	49.14	0.06
Pass rate	96.99%	8.49
Average bit rate (kbps)	105.55	4.31
Absolute maximum during 1 second period (kbps)	135.1	17.2

Table 4: Data rate test without acknowledgment

3.2.1.2 Maximum payload with ACK

Test setup is identical to the previous test, with the exception of the ACK frame response usage. As can be seen there are no losses, all the packets were received. At the same time can be seen that the throughput has reduced.

	Average value	Delta
RSSI (dBm)	-52.8	14.12
LQI	49.0	0.19
Pass rate	100%	0
Average bit rate (kbps)	50.75	38.4
Absolute maximum during 1 second period (kbps)	74.4	8.8

Table 5: Data rate test with the acknowledgment

3.2.2 Line of sight tests

The transmission range test can determine many parameters of the devices. First the actual distance that could be covered with the radio link. Second the chip capabilities can be determined. The sensitivity limit, which is one of the key parameters for transceivers, is put to the test. The limit of sensitivity level is the threshold where the data transmission errors start to occur.

The receive statistics were analyzed approximately after each 10 meters. According to the Olimex development board datasheet, range up to 85 meters could be achieved with the 250 kbps data rate at 868MHz. [63] The maximum range that finally was put to test was 100 meters, where still some packets could be captured. The sensitivity limit of -90 dBm was determined. This is the level where more than 95% of the packets were not received correctly. The datasheet [51] states -100dBm sensitivity, which could be true, but probably with very high error rate and practically not usable radio link. Since near the sensitivity limit the transmission error is fluctuating a lot, the exact ratio between distance and the packet loss cannot be determined. During the tests at 40 meters away from the transmitter 100% of packets could be recorded, farther away errors started to occur. The error rate between 60 – 90 meters was very volatile and the exact loss ratio was hard to determine. Thus only true metrics that can be associated with the data range and link error rate is the sensitivity limit. This can be different with different speed and register settings. Also this is heavily dependent on the environment. Tests were carried at the outside conditions on the flat field, with no major agitators.

3.2.3 Energy consumption tests

The supply voltage supplied to the circuitry is 3.3V due to the voltage regulator present on the development board. The current consumed by the circuitry was measured using the low ohm resistor (3 ohms) in series with the supply line. The voltage drop on the element was measured with an oscilloscope.

3.2.3.1 Transmit

Current is measured while the device is on the burst transmit mode, like in the data transmission tests (Section 3.2.1). Sending the maximum packet length of 50 bytes ensures the high duty cycle and correct mean peak values in the current consumption readings. The values of 20.13 to 24.13 mA was measured with the output power ranging from -6dBm to 0dBm.

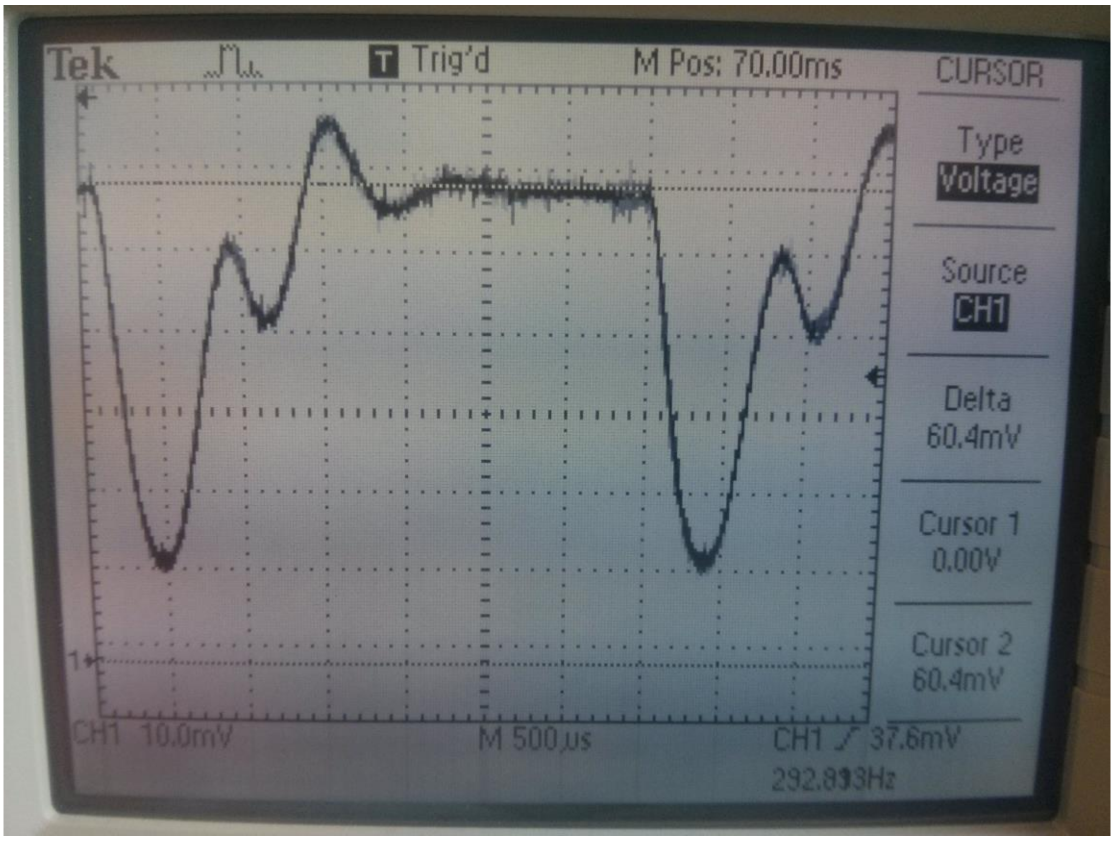


Figure 19: Measurement of the transmission current consumption

3.2.3.2 Receive

In this test the device is in the receive mode while second device is sending burst of data packets. Test setup is the same when measuring current consumption in transmit test, except the device is in the opposite mode.

The values measured were rather static. A rough 17 mA current consumption was recorded at around -40dBm sensitivity.

3.2.3.3 Idle

The device has been initialized and has done the same start up procedures as in the usual working condition. The current consumption is measured when the program is cycling in an endless loop while incrementing integer. Two measurements are done. First measurement without the radio core being shut down (in the idle mode) results in current consumption of 4.58mA. When the radio core is completely shut down the 2.87mA reading is obtained. This indicates that shutting down the radio core while not transmitting or receiving data can save around 1.80 mA.

3.2.3.4 Sleep

In this test, the peripherals and clock system is initialized as in other applications of this work. The MCU is put to sleep, while the radio core is powered down. The sleep mode *LPM3* is activated. A reading 4.2µA was obtained.

3.2.3.5 Results

Current consumption comparison between the experimental tests and datasheets at 3.3V supply.

	Transmit (0 dBm)	**Receive (near -40 dBm)**	**Idle / Idle with radio off**	**MCU sleep (LPM3)**
Tests	24.13 mA	17.06 mA	4.58mA/2.87mA	4.2uA
Datasheet	18 mA	15mA	x	2.2uA

Table 6: Current consumption

On the datasheet row the data is the combined information from Olimex datasheet [63] *and the CC430 family datasheet* [51]. *One must note that datasheets are referring to typical average values, so the comparison is not absolute.*

3.2.4 Demo application

For the graspable results of this work and a working live example, the demo application was designed. It is based on the same radio parameters and Chronos library modifications as already explained in previous chapters. Minor modifications were done on the packet parsing logic of the PC access point tool and on the end device itself to include the sensor measurements and data capsuling into a packet.

The sensor data objects were added to the payload. Sensor measurement objects are containing as less overhead as possible. One sample of physical phenomenon object size is 3 bytes, where the first byte marks the type plus 2 bytes for the calculated value. The type byte is an ASCII char, which is chosen to represent the first letter of the name of the sensor type. Since all the physical values measured can be fit into 16 bits variable, 2 bytes are allocated for the converted value.

Sensor data object.

type	measurement data
1 byte	2 bytes

Figure 20: Sensor data object structure

Where type can have the following values:

- 'x' – accelerometer x-axis
- 'y' – accelerometer y-axis
- 'z' – accelerometer z-axis
- 'b' – battery/operating voltage
- 't' – temperature

Complete packet format used in the demo, sent by the end device.

timestamp (time_t)	milliseconds	packet counter	sensor data objects
4 bytes	2 bytes	4 bytes	n x 3 bytes

Figure 21: Packet structure of the end device in the demo application

The output of the demo can be seen on the command line where the last values of all of the nodes connected are printed. Additionally the tool logs all the received packets to a CSV file.

The demo also exhibits the compatibility of the code between the CC430 family transceivers, where the same firmware is run on Chronos watch device and Olimex development board (Section 3.1.2). Both of the devices can be linked to the access point without any difference at the sequence of operation.

3.2.5 Results

With the results that are moderately worse than specified by the datasheets, the combination of software and hardware shortfall can be implicated. However in real life scenario the measured values are not awful either. Datasheets state the laboratory results, while the real life data that was recorded by the tests had the burden of software, timers and probably not the most fine-tuned power settings.

The wireless data link results indicate that the software performance probably is the bottleneck for the data rate. The period of roughly 3.4 milliseconds was measured of one `SMPL_Send()` call which results about 295 calls in one second. Now when the packet length is 50 bytes, the theoretical maximum rate of 117 kbps is achieved. That means the test results actually showed that with the average of 105 kbps indeed the close maximum rate was achieved. The maximum rate of 135 kbps in one second frame was recorded, which shows the data rate that can be achieved with the perfect conditions. The volatility of the data rate and also the `SMPL_Send()` duration is present. That is the result of the CCA mechanism which is measuring the noise in the channel and therefore is controlled by the environmental conditions. Now when the SimpliciTI overhead is added to the payload calculations too, which makes 64 bytes per packet, the average data rate of 150 kbps is the true value of the link. So 150 kbps out of the datasheet stated 256 kbps is pretty good result, because there are more hidden payload by the radio packet format and the radio driver used by SimpliciTI is not the most efficient. Details of enhancements will be covered in the next chapter.

Another rather positive outcome was the packet error rate of the transmission without the ACK mechanism. A roughly 97% pass rate is a very good result when the transmission was simply bursting out data. Although with the packet sniffer, all packets were captured successfully. Which simply indicates the burden of the software overhead for the access point, since the two devices were equally close to the transmitter. If one closely follows the sensitivity limits, the use of acknowledgment do not add extra value in the network because it simply decreases the data rate too much.

Another implication of the tests showed that when multiple devices are trying to transmit on the same channel the real throughput in the channel do not increase. So the data rate results should roughly be divided with the number of devices on the channel. Another issue that would be added with the multiple devices is the scheduling, which will decrease the throughput even more.

The current consumption tests showed that for large scale data transmission the solution is not sustainable. During one transmission period of 3.4 ms the current consumption is roughly 24mA. When the two standard AAA batteries, which could be used to power the device, are taken into account around 1000mAH capacity could be reached. With a power source like that one could roughly transmit continously 40 hours. With a scheduling algorithm this could be of course increased dramatically. When the amount of data is calculated the 1000mAH capacity source could be used to transmit around 1.9MB worth of data. With this energy consumption the solution is suitable for only infrequent monitoring of objects.

The demo application however showed that for low rate monitoring applications, even up to three devices could be successfully used in one common network. The devices do not appear to have any major problems of connecting and receiving data.

4 Summary and future work

The Chronos development kit based solution got largely upgraded network physical layer. On top of it new layer functionality was developed. With the radio devices the accompanying PC tools and access point software was enhanced too.

Author of this work reached a conclusion where it was found that with the usage of SimpliciTI library that is the essence of the network implementation, near limit performance was showcased. Following guidelines are the blueprints that the author suggests to follow for the future enhancements of similar work:

1) Access point logic should run in the PC. The dongle hardware should feed all the radio data directly to the PC. This way there will be much more processing power to do the network related task. At the same time dedicated hardware can effectively deal with the radio network only. This approach also gives better modularity and dynamics of the solution.
2) While SimpliciTI library is a great for custom sensor network solutions, because of the low overhead, it was designed having a low throughput networks in mind like home automation. The library already incorporates drivers for Chipcon radio family. On top of the "mrfi" layer new implementation should be written, which would use the resources more efficiently.
3) The radio chip FIFO buffers should be used in a different way, utilizing "infinite length" packet format. The support at the architectural level for that is present. The SimpliciTI simply fills the buffer then waits when all the data is sent and then going through the initialization procedures fills it again. With infinite packet length format the buffers will be filled whenever there is free slot in the memory. This is interrupt based and efficient.
4) MSK and GFSK modulation schemes comparison research. The CC430 family RF circuitry does not support the Gaussian filtered MSK modulation, which makes the use of MSK to pollute the spectrum with higher level of noise. The approach where the frequency agility is used with the MSK could be researched. Since MSK modulation is stated to double the data rate, the research comparing the data rates achieved with multiple device network using cleaner spectrum GFSK versus MSK could be done.

5 Acknowledgments

I am extremely thankful for all the people who did contribute in any form or level. My colleagues at Axinom, who managed to allocate me the vacation to finish the studies and cope with the reduced input to important projects. Directly influenced my progress the people who were participating at the “Smart Composite” working group at the university – thank you for your patience and tolerant attitude towards my delays and “could not find time” project updates. You all shared a lot of knowledge.

Instrumental to the whole process were the programs which funded the research. Archimedes funded project AR 12139 “Smart Composites – Design and Manufacturing” enabled to group all these awesome people and work together. Texas Instruments “European University Program” enabled to use full featured development tools for MSP430 platform for free, anywhere, anytime.

Thanks to the guy, who has chosen to be anonymous, with whom I had an opportunity to work side by side when I was doing laboratory work at the university. Thanks for the educational chats and thorough help whenever needed. Henrik Herranen, who was the leader of Smart Composite working group, always kept up the energy and through his efforts everything was moving all the time. Finally my supervisor Olev, who directly gave input of any form – development devices, code and experimentation results. Thanks for allocating the time and sharing the great ideas about the thesis content!

6 References

[1] J. P. Lynch and K. J. Loh, “A Summary Review of Wireless Sensors and Sensor Networks for Structural Health Monitoring”, *The Shock and Vibration Digest, Vol. 38, No. 2*, (2006)

[2] G.-D. Zhou and T.-H. Yi, “Recent Developments on Wireless Sensor Networks Technology for Bridge Health Monitoring”, *Mathematical Problems in Engineering Volume 2013, Article ID 947867*, (2013)

[3] D. Krantz, J. Belk, P.J. Biermann, J. Dubow, L.W. Gause, R. Harjani, S. Mantell, D. Polla, P. Troyk, “Project Update: Applied Research on Remotely-Queried Embedded Microsensor”, Proc. SPIE 3673, Smart Structures and Materials 1999: Smart Electronics and MEMS, 157, (1999)

[4] E.G. Straser, A.S. Kiremidjan, “A modular, wireless damage monitoring system for structures.”, (1998)

[5] J.P. Lynch, K.H. Law, A.S. Kiremidjian, E. Carryer, “Design and performance validation of a wireless sensing unit for structural monitoring applications”, *Structural Engineering and Mechanics, Vol. 17, No. 3-4*, (2004)

[6] J. P. Lynch, K. H. Law, A. S. Kiremidjian, T. W. Kenny, E. Carryer and A. Partridge, “The Design of a Wireless Sensing Unit for Structural Health Monitoring”, (2001)

[7] J. P. Lynch, K. H. Law, A. S. Kiremidjian, T. Kenny and E. Carryer, “A wireless modular monitoring system for civil structures”, (2002)

[8] S. Aoki, Y. Fujino, M. Abe, “Intelligent bridge maintenance system using MEMS and network technology”, *Smart Structures and Materials 2003: Smart Systems and Nondestructive Evaluation for Civil Infrastructures, Shih-Chi Liu, Editor, Proceedings of SPIE Vol. 5057*, (2003)

[9] *Micro.2420 U100 Datasheet* (2007, visited 03.04.2014). `http://www.ee.oulu.fi/~ikram/microseries/sensinode-datasheet-U100R2-20070923.pdf`

[10] M. Bocca, E. I. Cosar, J. Salminen, L. M. Eriksson, “A Reconfigurable Wireless Sensor Network for Structural Health Monitoring”, *4th International Conference on Structural Health Monitoring of Intelligent Infrastructure (SHMII-4)*, (2009)

[11] E. I. Cosar, “A Wireless Toolkit for Monitoring Applications”, Master’s thesis, Helsinki University of Technology (2009)

[12] S. G. Taylor, K. M. Farinholt, E. B. Flynn, E. Figueiredo, D. L. Mascarenas, E. A. Moro, G. Park, M. D. Todd, C. R. Farrar, “A Mobile-Agent Based Wireless Sensing Network for Structural Monitoring Applications”, (2009)

[13] *A Multi-Agent Platform for Mobile C/C++ Agents* (visited 02.05.2014). `http://www.mobilec.org/overview.php`

[14] D. Maurer and A. Descombes, “Selecting an embedded MCU: How to avoid evaluation trap?”, (visited 02.03.2014). `http://www.design-reuse.com/articles/22878/embedded-mcu-selection.html`

[15] J. Borgeson, S. Schauer, H. Diewald, “Benchmarking MCU power consumption for ultra-low-power applications”, (2012, visited

21.03.2014).
http://www.ti.com/lit/wp/slay023/slay023.pdf

[16] Y. Wang and K. H. Law, "Wireless Sensor Networks in Smart Structural Technologies", (visited 18.03.2014). http://eil.stanford.edu/publications/yang_wang/InTech_Wireless_Sensing_Control.pdf

[17] "ARM, Renesas,TI join move for low power MCU benchmark", (visited 11.03.2014). http://www.electronicsweekly.com/news/components/microprocessors-and-dsps/arm-renesasti-join-move-for-low-power-mcu-benchmark-2013-02/

[18] "ULPBench™ Benchmark Software", (visited 11.03.2014). http://www.eembc.org/benchmark/ulp_sl.php

[19] "Accurate power consumption estimation for low power MCUs", TA0342 Technical Article, 018552 Rev 1, (2011, visited 11.03.2014). http://www.digikey.com/web%20export/supplier%20content/stmicroelectronics_497/mkt/stm32/DM00024152.pdf?redirected=1

[20] H. Diewald, "Low-power MCU benchmarking: what datasheets don't tell you", (2013, visited 12.03.2014). http://www.embedded.com/design/power-optimization/4421443/1/Low-power-MCU-benchmarking--what-datasheets-don-t-tell-you

[21] R. Kale, N. Singh, H. Alasti, A. Nasipuri, R. Cox, J. M. Conrad, L. V.d. Zel, B. Rodriguez, R. McKosky and J. Graziano, "Design and Implementation of a Wireless Node for Advanced Sensor Processing and Network Integration", IEEE (2010)

[22] "IEEE 802.15.4", Wikipedia, (visited 05.05.2014). http://en.wikipedia.org/wiki/IEEE_802.15.4

[23] M. Loy, R. Karingattil, L. Williams, "ISM-Band and Short Range Device Regulatory Compliance Overview", Application Report, SWRA048, (2005, visited 13.03.2014). http://www.ti.com/lit/an/swra048/swra048.pdf

[24] T.-H. Lee, H.-S. Chiang, L.-H. Chang, M.-C. Hsieh, "Performance Analysis of IEEE 802.15.4 868MHz, 915MHz and 2.4GHz Physical Schemes in 6LoWPAN"

[25] M. Woehrle, M. Bor, K. Langendoen, "868 MHz: a noiseless environment, but no free lunch for protocol design"

[26] "Free-space path loss", Wikipedia, (visited 01.05.2014). http://en.wikipedia.org/wiki/Free-space_path_loss

[27] *"Designer's Guide to LPRF", TI Low Power RF*, (2010, visited 23.04.2014). http://www.ti.com/lit/sg/slya020a/slya020a.pdf

[28] *"RF Transceivers", (visited 25.04.2014)*. http://www.semtech.com/wireless-rf/rf-transceivers/

[29] "Personal Area Networks", (visited 25.04.2014). http://www.microchip.com/pagehandler/en-us/technology/personalareanetworks/home.html

[30] „Ultra low power wireless solutions", (visited 25.04.2014). http://www.nordicsemi.com/eng/Products

[31] „Low-Power RF Solutions", (visited 25.04.2014). http://www.st.com/web/en/catalog/sense_power/FM1968/CL1976

[32] *„32-bit MCU and IEEE802.15.4 transceiver for low-power wireless networks*", (visited 25.04.2014). http://www.nxp.com/products/rf/wireless_microcontrollers/#overview

[33] *„Application and design manual for High Performance RF products", NXP Semiconductors RF Manual 17th edition*, (2013, visited 25.04.2014). http://www.nxp.com/documents/selection_guide/75017428.pdf

[34] *„JN516x Wireless Microcontrollers",* (visited 25.04.2014). http://www.nxp.com/techzones/wireless-connectivity/jn51xx/jn516x.html

[35] *„HCS08 8-Bit Processor",* (visited 25.04.2014). http://www.ip-extreme.com/IP/hcs08.shtml

[36] *„Transceivers and Wireless MCUs",* (visited 25.04.2014). http://www.freescale.com/webapp/sps/site/taxonomy.jsp?nodeId=0106B9837F

[37] *„Sub-1 GHz Wireless",* (visited 25.04.2014). http://www.freescale.com/webapp/sps/site/application.jsp?code=APL1GHZWRLSS

[38] *„Wireless Development Tools",* (visited 27.04.2014). http://www.silabs.com/products/wireless/Pages/DevelopmentTools.aspx

[39] *„Wireless Microcontrollers (MCUs)",* (visited 27.04.2014). http://www.silabs.com/products/wireless/wirelessmcu/Pages/default.aspx

[40] *„802.15.4 Microcontrollers",* (visited 27.04.2014). http://www.atmel.com/products/wireless/802154-microcontrollers/transceivers.aspx

[41] *„Broadband RF/IF",* (visited 27.04.2014). http://www.ti.com/analog/docs/enggresdetail.tsp?familyId=367&genContentId=3573

[42] *„TI Completes Acquisition of Chipcon", Investor Relations*, (visited 27.04.2014). http://www.ti.com/corp/docs/investor/compinfo/PRarchive/sc06014.shtml

[43] *„CC430 Family User Guide", SLAU259E*, (2013, visited 12.02.2014). http://www.ti.com/general/docs/lit/getliterature.tsp?baseLiteratureNumber=SLAU259

[44] *nRF9E5, Multiband Sub 1-GHz RF System-on-Chip*, (visited 28.04.2014). http://www.nordicsemi.com/eng/Products/Sub-1-GHz-RF/nRF9E5

[45] *SPIRIT 1, Low data rate, low power sub-1GHz transceiver, datasheet*, (2013, visited 28.04.2014). http://www.st.com/st-web-ui/static/active/en/resource/technical/document/datasheet/DM00047607.pdf

[46] *OL2381, Highly integrated single-chip sub 1 GHz transceiver, product datasheet*, (2011, visited 28.04.2014). `http://www.nxp.com/documents/data_sheet/OL2381.pdf`

[47] *MC12311 Highly-integrated, cost-effective single-package solution for the sub-1 GHz, Wireless MBUS Standard*, (2011, visited 28.04.2014). `http://cache.freescale.com/files/rf_if/doc/data_sheet/MC12311.pdf`

[48] *Si106x/108x, Ultra-Low Power MCU with Integrated High-Performance Sub-1 GHz Transceiver*, (visited 07.05.2014). `http://www.silabs.com/Support%20Documents/TechnicalDocs/Si106x-8x.pdf`

[49] *Silicon Labs MCU Selector Guide*, (2014, visited 07.05.2014). `http://www.silabs.com/Marcom%20Documents/Resources/mcu-selector-guide.pdf`

[50] *AT86RF212B, Low Power, 700/800/900MHz Transceiver for ZigBee, IEEE 802.15.4, 6LoWPAN, and ISM Applications*, (2013, visited 07.05.2014). `http://www.atmel.com/images/Atmel-42002-MCU_Wireless-AT86RF212B_Datasheet.pdf`

[51] *MSP430™ SoC With RF Core*, (visited 03.03.2014). `http://www.ti.com/lit/ds/symlink/cc430f5137.pdf`

[52] *"MSP-EXP430FR5739 FRAM Experimenter Board", User's Guide, SLAU343B,* (2012, visited 15.11.2013). `http://www.ti.com/lit/ug/slau343b/slau343b.pdf`

[53] *IAR C/C++ Compiler, Reference Guide for Texas Instruments' MSP430 Microcontroller* family, (visited 15.03.2014). `ftp://ftp.iar.se/WWWfiles/msp430/webic/doc/EW430_CompilerReference.pdf`

[54] *"Is GCC a 'good' compiler?", Nigel Jones*, (2010, visited 15.03.2014). `http://embeddedgurus.com/stack-overflow/2010/02/is-gcc-a-good-compiler/`

[55] *"Debugging using the IAR C-SPY Debugger", IAR Systems*, (visited 17.03.2014). `http://supp.iar.com/FilesPublic/UPDINFO/005832/arm/doc/infocenter/tutor_debugging.ENU.html`

[56] *"IAR Systems ups performance for the ultra-low-power MSP430 microcontrollers from Texas Instruments", Product News*, (2013, visited 16.03.2014). `http://www.iar.com/Documents/pdf/prm/en/1483462.pdf`

[57] *Code Composer Studio (CCS) Integrated Development Environment (IDE)*, (visited 16.04.2014). `http://www.ti.com/tool/ccstudio`

[58] *Download CCS, Texas Instruments Wiki*, (visited 26.01.2014). `http://processors.wiki.ti.com/index.php/Download_CCS`

[59] *"IAR Embedded Workbench for TI MSP430", INTEGRATED DEVELOPMENT ENVIRONMENT AND OPTIMIZING C/C++ COMPILER FOR MSP430*, (visited 15.03.2014). `http://www.iar.com/Products/IAR-Embedded-Workbench/TI-MSP430/`

[60] *System requirements*, (17.03.2014). `http://www.iar.com/Products/IAR-Embedded-Workbench/Technology/System-requirements/`

[61] *System Requirements, Texas Instruments Wiki*, (visited 17.03.2014). http://processors.wiki.ti.com/index.php/System_Requirements

[62] *MSP430 Optimizing C/C++ Compiler v 4.3, User's Guide, SLAU132I*, (2013, visited 05.03.2014). http://www.ti.com/lit/ug/slau132h/slau132h.pdf

[63] *"MSP430-CCRF development board", User's Manual*, (2013, visited 25.09. 2013). https://www.olimex.com/Products/MSP430/Starter/MSP430-CCRF/resources/MSP430-CCRF.pdf

[64] *eZ430-Chronos Development Tool User's Guide, SLAU292F*, (2013, visited 19.02.2014). http://www.ti.com/lit/ug/slau292f/slau292f.pdf

[65] *„Upgrade from CC1100 to CC1101", Design Note DN009, SWRA145A, E. Simensen*, (visited 17.05.2014) . http://www.ti.com/lit/an/swra145a/swra145a.pdf

[66] *K. S. Panchal, "IMPLEMENTING PHYSICAL LAYER (PHY) OF IEEE 802.15.4G STANDARD WITH DIRECT SEQUENCE SPREAD SPECTRUM (DSSS) USING OFFSET QUADRATURE PHASE SHIFT KEYING (O-QPSK)", Master thesis, San Diego State University*. (2012, visited 11.05.2014)

[67] *"CC430 Vs CC1101",Texas Instruments, PowerPoint presentation*, (visited 10.05.2014). http://www.dowellcn.com/uploadfile/CC430%20Vs%20CC1101.pdf

[68] *"Low-Power Sub-1 GHz RF Transceiver", SWRS061l, Texas Instruments*, (visited 10.05.2014) .http://www.ti.com/lit/ds/swrs061i/swrs061i.pdf

[69] *„Introduction to SimpliciTI", PowerPoint presentation*, (visited 04.05.2014). http://www.ti.com/lit/ml/swru130b/swru130b.pdf

[70] *L. Friedman, SimpliciTI: Simple Modular RF Network Developers Notes, version 1.10*, (2007)

[71] *SimpliciTI Compliant Protocol Stack*, (visited 27.03.2014). http://www.ti.com/tool/simpliciti

[72] *CC2420 2.4 GHz IEEE 802.15.4 / ZigBee-ready RF Transceiver, Preliminary Datasheet*, (2004, visited 18.04.2014) . https://inst.eecs.berkeley.edu/~cs150/Documents/CC2420.pdf

[73] *„CC1111 USB Evaluation Module Kit 868/915 MHz"*, (visited 05.03.2014). http://www.ti.com/tool/cc1111emk868-915]

[74] *"SmartRF Protocol Packet Sniffer"*, (visited 02.03.2014). http://www.ti.com/tool/packet-sniffer

[75] *„SmartRF Studio"*, (visited 04.05.2014). http://www.ti.com/tool/smartrftm-studio

7 MSP430 platformil baseeruv sensorvõrk

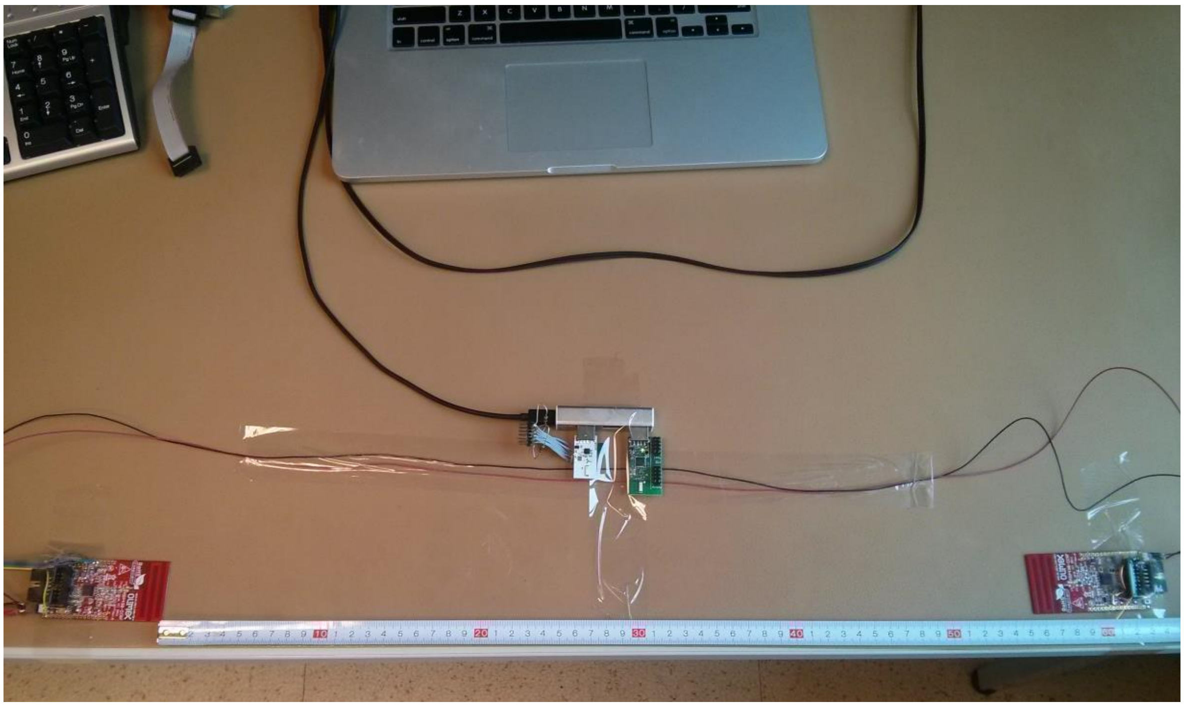
Materjalide struktuuri muutuste jälgimine on üha kasvav trend, mis areneb suuresti tänu pooljuhttehnoloogiate täienemisele. Meetodid on samuti arenguga kaasas käinud ning tänaseks on välja jõutud juhtmevabade sensorlahendusteni. Sensorsõlmed on võimelised täitma suuremal või vähemal määral kõiki vajalikke funktsioone – sensorite sampeldamine, andmete töötlemine, kommunikatsioon.

Aspekt, mis piirab funktsionaalsust on voolutarve. See probleem on eriti teravalt esile kerkinud juhtmevabade lahendustega, kus raadiolainete vastuvõtt ja saatmine tekitab kordades suurema energiatarbe võrreldes tavaolekuga. Peale selle kujutab raadiolüli endast ka pudelikaela, kus ühes kanalis saab korraga andmeid edastada üks seade. Samuti on piiratud läbilaskevõime. Sellest hoolimata on selliste seadmete uurimine populaarne uurimisvaldkond. Juhtmevabad sensorid võimaldavad väiksemaid seadmeid, paremat vastupidavust keskkonnatingimustele, modulaarsemat ja paindlikumat arhitektuuri ja seda kõike odavamalt kui juhtmega lahenduste puhul.

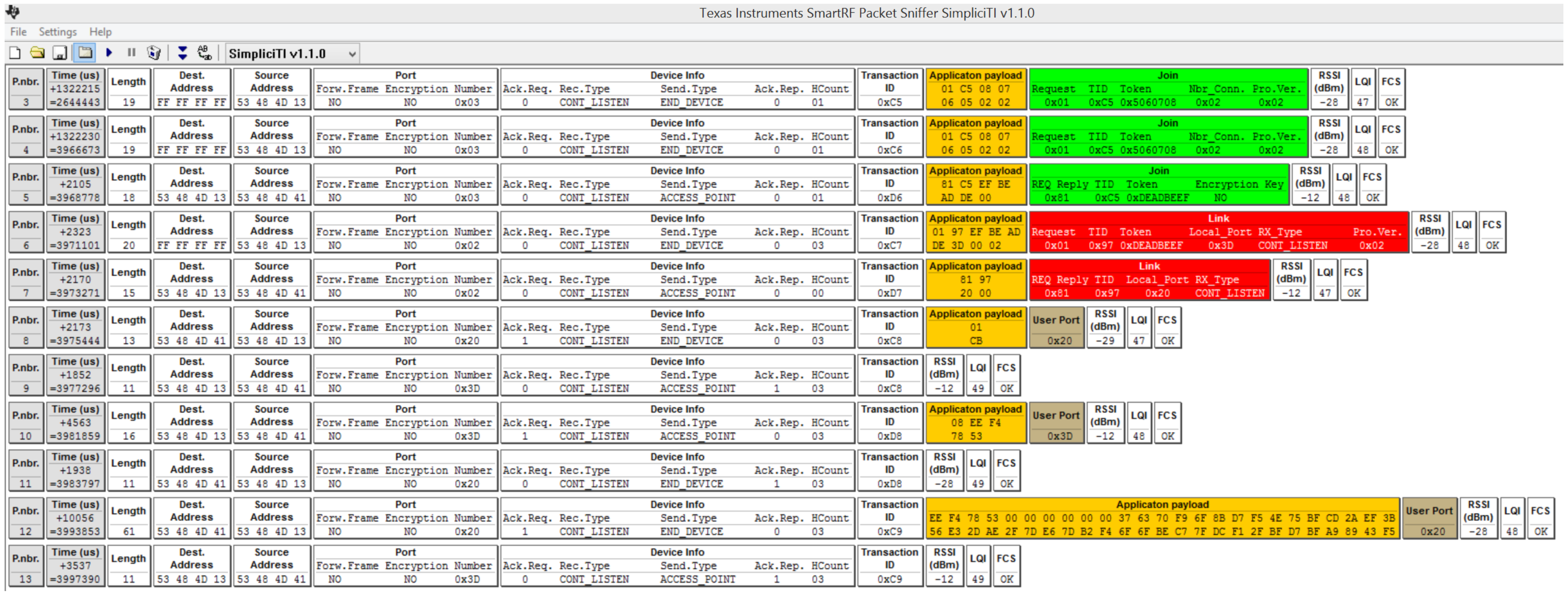
Käesolev töö uuris MSP430 platformil baseeruvat lahendust, kus võeti aluseks Texas Instrumentsi Chronose arendusplatvorm. Eksisteerivat lahendust täiendati ja mugandati vastavalt uurimitöö ülesannetele. Testid näitasid rahuldavaid tulemusi, milel põhjal jõuti järeldusele, et väiksemamahuliste andmete juures on süsteem kasutatav, kus ülempiiri seab voolutarve.

8 Appendices

Appendix A Radio link data rate test bed



Appendix B SmartRF Packet Sniffer records of the SimpliciTI traffic

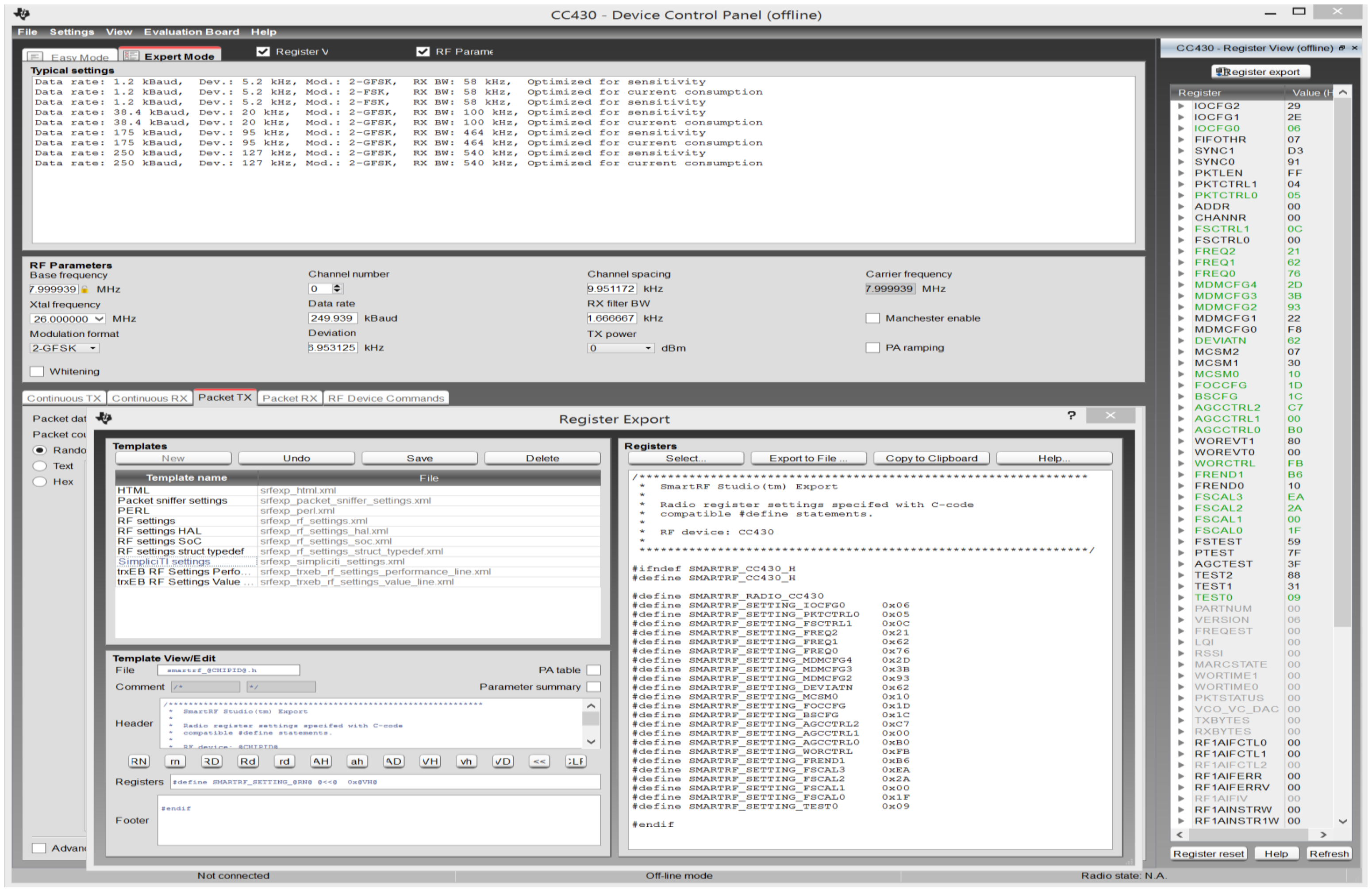


```
| P.nbr. | Time (us) | Length | Dest.       | Source      | Port                                | Device Info                                                  | Transaction | Applicaton payload |
|        | +1322215  |        | Address     | Address     | Forw.Frame Encryption Number        | Ack.Req. Rec.Type     Send.Type     Ack.Rep. HCount          | ID          | 01 C5 08 07        |
| 3      | =2644443  | 19     | FF FF FF FF | 53 48 4D 13 | NO         NO         0x03          | 0        CONT_LISTEN END_DEVICE    0        01              | 0xC5        | 06 05 02 02        |
  | Join                                                    | RSSI  | LQI | FCS |
  | Request TID  Token      Nbr_Conn. Pro.Ver.              | (dBm) |     |     |
  | 0x01    0xC5 0x5060708  0x02      0x02                  | -28   | 47  | OK  |

| P.nbr. | Time (us) | Length | Dest.       | Source      | Port                                | Device Info                                                  | Transaction | Applicaton payload |
|        | +1322230  |        | Address     | Address     | Forw.Frame Encryption Number        | Ack.Req. Rec.Type     Send.Type     Ack.Rep. HCount          | ID          | 01 C5 08 07        |
| 4      | =3966673  | 19     | FF FF FF FF | 53 48 4D 13 | NO         NO         0x03          | 0        CONT_LISTEN END_DEVICE    0        01              | 0xC6        | 06 05 02 02        |
  | Join                                                    | RSSI  | LQI | FCS |
  | Request TID  Token      Nbr_Conn. Pro.Ver.              | (dBm) |     |     |
  | 0x01    0xC5 0x5060708  0x02      0x02                  | -28   | 48  | OK  |

| P.nbr. | Time (us) | Length | Dest.       | Source      | Port                                | Device Info                                                  | Transaction | Applicaton payload |
|        | +2105     |        | Address     | Address     | Forw.Frame Encryption Number        | Ack.Req. Rec.Type     Send.Type     Ack.Rep. HCount          | ID          | 81 C5 EF BE        |
| 5      | =3968778  | 18     | 53 48 4D 13 | 53 48 4D 41 | NO         NO         0x03          | 0        CONT_LISTEN ACCESS_POINT  0        01              | 0xD6        | AD DE 00           |
  | Join                                                    | RSSI  | LQI | FCS |
  | REQ Reply TID  Token       Encryption Key               | (dBm) |     |     |
  | 0x81      0xC5 0xDEADBEEF  NO                           | -12   | 48  | OK  |

| P.nbr. | Time (us) | Length | Dest.       | Source      | Port                                | Device Info                                                  | Transaction | Applicaton payload |
|        | +2323     |        | Address     | Address     | Forw.Frame Encryption Number        | Ack.Req. Rec.Type     Send.Type     Ack.Rep. HCount          | ID          | 01 97 EF BE AD     |
| 6      | =3971101  | 20     | FF FF FF FF | 53 48 4D 13 | NO         NO         0x02          | 0        CONT_LISTEN END_DEVICE    0        03              | 0xC7        | DE 3D 00 02        |
  | Link                                                                    | RSSI  | LQI | FCS |
  | Request TID  Token       Local_Port RX_Type      Pro.Ver.               | (dBm) |     |     |
  | 0x01    0x97 0xDEADBEEF  0x3D       CONT_LISTEN  0x02                   | -28   | 48  | OK  |

| P.nbr. | Time (us) | Length | Dest.       | Source      | Port                                | Device Info                                                  | Transaction | Applicaton payload |
|        | +2170     |        | Address     | Address     | Forw.Frame Encryption Number        | Ack.Req. Rec.Type     Send.Type     Ack.Rep. HCount          | ID          | 81 97              |
| 7      | =3973271  | 15     | 53 48 4D 13 | 53 48 4D 41 | NO         NO         0x02          | 0        CONT_LISTEN ACCESS_POINT  0        00              | 0xD7        | 20 00              |
  | Link                                         | RSSI  | LQI | FCS |
  | REQ Reply TID  Local_Port RX_Type            | (dBm) |     |     |
  | 0x81      0x97 0x20       CONT_LISTEN        | -12   | 47  | OK  |

| P.nbr. | Time (us) | Length | Dest.       | Source      | Port                                | Device Info                                                  | Transaction | Applicaton payload |
|        | +2173     |        | Address     | Address     | Forw.Frame Encryption Number        | Ack.Req. Rec.Type     Send.Type     Ack.Rep. HCount          | ID          | 01                 |
| 8      | =3975444  | 13     | 53 48 4D 41 | 53 48 4D 13 | NO         NO         0x20          | 1        CONT_LISTEN END_DEVICE    0        03              | 0xC8        | CB                 |
  | User Port | RSSI  | LQI | FCS |
  |           | (dBm) |     |     |
  | 0x20      | -29   | 47  | OK  |

| P.nbr. | Time (us) | Length | Dest.       | Source      | Port                                | Device Info                                                  | Transaction | RSSI  | LQI | FCS |
|        | +1852     |        | Address     | Address     | Forw.Frame Encryption Number        | Ack.Req. Rec.Type     Send.Type     Ack.Rep. HCount          | ID          | (dBm) |     |     |
| 9      | =3977296  | 11     | 53 48 4D 13 | 53 48 4D 41 | NO         NO         0x3D          | 0        CONT_LISTEN ACCESS_POINT  1        03              | 0xC8        | -12   | 49  | OK  |

| P.nbr. | Time (us) | Length | Dest.       | Source      | Port                                | Device Info                                                  | Transaction | Applicaton payload |
|        | +4563     |        | Address     | Address     | Forw.Frame Encryption Number        | Ack.Req. Rec.Type     Send.Type     Ack.Rep. HCount          | ID          | 08 EE F4           |
| 10     | =3981859  | 16     | 53 48 4D 13 | 53 48 4D 41 | NO         NO         0x3D          | 1        CONT_LISTEN ACCESS_POINT  0        03              | 0xD8        | 78 53              |
  | User Port | RSSI  | LQI | FCS |
  |           | (dBm) |     |     |
  | 0x3D      | -12   | 48  | OK  |

| P.nbr. | Time (us) | Length | Dest.       | Source      | Port                                | Device Info                                                  | Transaction | RSSI  | LQI | FCS |
|        | +1938     |        | Address     | Address     | Forw.Frame Encryption Number        | Ack.Req. Rec.Type     Send.Type     Ack.Rep. HCount          | ID          | (dBm) |     |     |
| 11     | =3983797  | 11     | 53 48 4D 41 | 53 48 4D 13 | NO         NO         0x20          | 0        CONT_LISTEN END_DEVICE    1        03              | 0xD8        | -28   | 49  | OK  |

| P.nbr. | Time (us) | Length | Dest.       | Source      | Port                                | Device Info                                                  | Transaction |
|        | +10056    |        | Address     | Address     | Forw.Frame Encryption Number        | Ack.Req. Rec.Type     Send.Type     Ack.Rep. HCount          | ID          |
| 12     | =3993853  | 61     | 53 48 4D 41 | 53 48 4D 13 | NO         NO         0x20          | 1        CONT_LISTEN END_DEVICE    0        03              | 0xC9        |
  | Applicaton payload                                                                       | User Port | RSSI  | LQI | FCS |
  | EE F4 78 53 00 00 00 00 00 00 37 63 70 F9 6F 8B D7 F5 4E 75 BF CD 2A EF 3B               |           | (dBm) |     |     |
  | 56 E3 2D AE 2F 7D E6 7D B2 F4 6F 6F BE C7 7F DC F1 2F BF D7 BF A9 89 43 F5               | 0x20      | -28   | 48  | OK  |

| P.nbr. | Time (us) | Length | Dest.       | Source      | Port                                | Device Info                                                  | Transaction | RSSI  | LQI | FCS |
|        | +3537     |        | Address     | Address     | Forw.Frame Encryption Number        | Ack.Req. Rec.Type     Send.Type     Ack.Rep. HCount          | ID          | (dBm) |     |     |
| 13     | =3997390  | 11     | 53 48 4D 13 | 53 48 4D 41 | NO         NO         0x3D          | 0        CONT_LISTEN ACCESS_POINT  1        03              | 0xC9        | -12   | 49  | OK  |
```

Appendix C SmartRF Packet Sniffer „.psd“ file parser logic

```
struct packetData
{
	std::string destinationAddress;
	std::string sourceAddress;
	uint8_t port;
	uint8_t transactionId;
	std::string dataHex;
	int8_t rssi;
	uint8_t lqi;
	bool fcsOk;
};

static struct packetData parsePsd(const std::vector<uint8_t>& packetBinary)
{
	struct packetData packet = {};

	size_t dataLength = packetBinary.at(15);
	packet.destinationAddress = bufferToHex(std::vector<uint8_t>(packetBinary.begin() + 16, packetBinary.begin() + 20));
	packet.sourceAddress = bufferToHex(std::vector<uint8_t>(packetBinary.begin() + 20, packetBinary.begin() + 24));
	packet.port = packetBinary.at(24);
	packet.transactionId = packetBinary.at(26);
	packet.dataHex = "EMPTY";

	size_t applicationDataLength = dataLength - 11;
	if (applicationDataLength > 0 && applicationDataLength <= 50)
	{
		packet.dataHex = bufferToHex(std::vector<uint8_t>(packetBinary.begin() + 27, packetBinary.begin() + (27 + (dataLength -
11))));
	}

	// When there are erroneous packets logged.
	if (applicationDataLength > 50)
	{
			packet.fcsOk = false;
			return packet;
		}

		int8_t rawRssi = static_cast<int8_t>(packetBinary.at(27 + applicationDataLength));
		int16_t calculatedRssi = (rawRssi >= 128 ? ((rawRssi - 256) / 2 - 72) : (rawRssi / 2 - 72));
		packet.rssi = (calculatedRssi < -128 ? -128 : calculatedRssi);
		packet.fcsOk = ((packetBinary.at(27 + applicationDataLength + 1) & 0x80) > 0 ? true : false);
		packet.lqi = packetBinary.at(27 + applicationDataLength + 1) & 0x7F;

		return packet;

	}
```

Appendix D Snapshot of the SmartRF Studio tool



Typical settings

```
Data rate: 1.2 kBaud,  Dev.: 5.2 kHz, Mod.: 2-GFSK,  RX BW: 58 kHz,  Optimized for sensitivity
Data rate: 1.2 kBaud,  Dev.: 5.2 kHz, Mod.: 2-FSK,   RX BW: 58 kHz,  Optimized for current consumption
Data rate: 1.2 kBaud,  Dev.: 5.2 kHz, Mod.: 2-FSK,   RX BW: 58 kHz,  Optimized for sensitivity
Data rate: 38.4 kBaud, Dev.: 20 kHz,  Mod.: 2-GFSK,  RX BW: 100 kHz, Optimized for sensitivity
Data rate: 38.4 kBaud, Dev.: 20 kHz,  Mod.: 2-GFSK,  RX BW: 100 kHz, Optimized for current consumption
Data rate: 175 kBaud,  Dev.: 95 kHz,  Mod.: 2-GFSK,  RX BW: 464 kHz, Optimized for sensitivity
Data rate: 175 kBaud,  Dev.: 95 kHz,  Mod.: 2-GFSK,  RX BW: 464 kHz, Optimized for current consumption
Data rate: 250 kBaud,  Dev.: 127 kHz, Mod.: 2-GFSK,  RX BW: 540 kHz, Optimized for sensitivity
Data rate: 250 kBaud,  Dev.: 127 kHz, Mod.: 2-GFSK,  RX BW: 540 kHz, Optimized for current consumption
```

RF Parameters

Base frequency: 7.999939 MHz	Channel number: 0	Channel spacing: 9.951172 kHz	Carrier frequency: 7.999939 MHz
Xtal frequency: 26.000000 MHz	Data rate: 249.939 kBaud	RX filter BW: 1.666667 kHz	Manchester enable ☐
Modulation format: 2-GFSK	Deviation: 3.953125 kHz	TX power: 0 dBm	PA ramping ☐
Whitening ☐			

Templates

Template name	File
HTML	srfexp_html.xml
Packet sniffer settings	srfexp_packet_sniffer_settings.xml
PERL	srfexp_perl.xml
RF settings	srfexp_rf_settings.xml
RF settings HAL	srfexp_rf_settings_hal.xml
RF settings SoC	srfexp_rf_settings_soc.xml
RF settings struct typedef	srfexp_rf_settings_struct_typedef.xml
SimpliciTI settings	srfexp_simpliciti_settings.xml
trxEB RF Settings Perfo...	srfexp_trxeb_rf_settings_performance_line.xml
trxEB RF Settings Value ...	srfexp_trxeb_rf_settings_value_line.xml

Template View/Edit

File: smartrf_@CHIPID@.h PA table ☐
Comment: /* */ Parameter summary ☐

Header:

```
/****************************************************************
 *  SmartRF Studio(tm) Export
 *
 *  Radio register settings specifed with C-code
 *  compatible #define statements.
 *
 *  RF device: @CHIPID@
```

Registers: `#define SMARTRF_SETTING_@RN@ @<<@  0x@VH@`

Footer: `#endif`

Registers

```
/***************************************************************
 *  SmartRF Studio(tm) Export
 *
 *  Radio register settings specifed with C-code
 *  compatible #define statements.
 *
 *  RF device: CC430
 *
 ***************************************************************/

#ifndef SMARTRF_CC430_H
#define SMARTRF_CC430_H

#define SMARTRF_RADIO_CC430
#define SMARTRF_SETTING_IOCFG0      0x06
#define SMARTRF_SETTING_PKTCTRL0    0x05
#define SMARTRF_SETTING_FSCTRL1     0x0C
#define SMARTRF_SETTING_FREQ2       0x21
#define SMARTRF_SETTING_FREQ1       0x62
#define SMARTRF_SETTING_FREQ0       0x76
#define SMARTRF_SETTING_MDMCFG4     0x2D
#define SMARTRF_SETTING_MDMCFG3     0x3B
#define SMARTRF_SETTING_MDMCFG2     0x93
#define SMARTRF_SETTING_DEVIATN     0x62
#define SMARTRF_SETTING_MCSM0       0x10
#define SMARTRF_SETTING_FOCCFG      0x1D
#define SMARTRF_SETTING_BSCFG       0x1C
#define SMARTRF_SETTING_AGCCTRL2    0xC7
#define SMARTRF_SETTING_AGCCTRL1    0x00
#define SMARTRF_SETTING_AGCCTRL0    0xB0
#define SMARTRF_SETTING_WORCTRL     0xFB
#define SMARTRF_SETTING_FREND1      0xB6
#define SMARTRF_SETTING_FSCAL3      0xEA
#define SMARTRF_SETTING_FSCAL2      0x2A
#define SMARTRF_SETTING_FSCAL1      0x00
#define SMARTRF_SETTING_FSCAL0      0x1F
#define SMARTRF_SETTING_TEST0       0x09

#endif
```

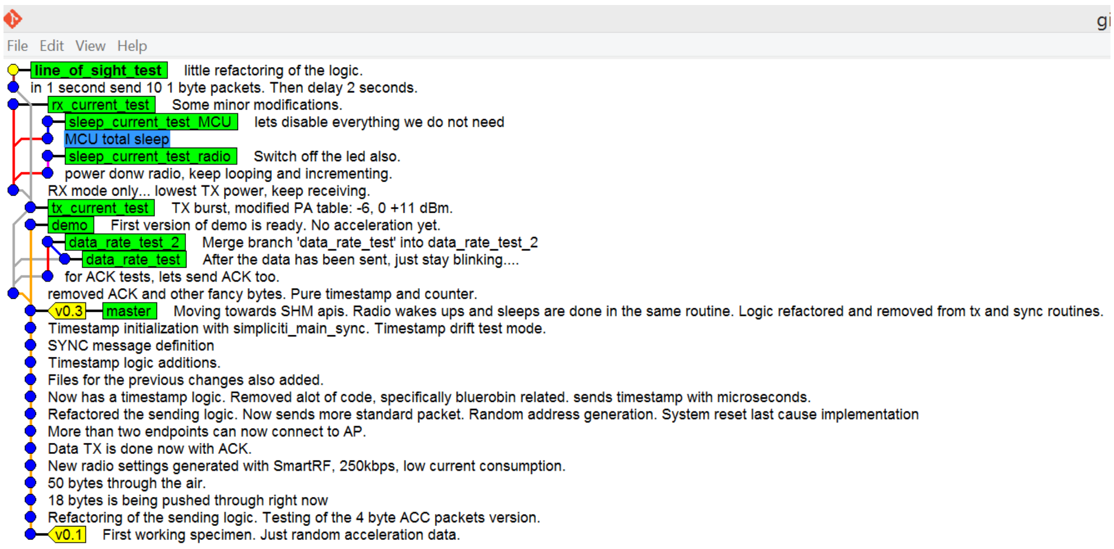
Register	Value (H
IOCFG2	29
IOCFG1	2E
IOCFG0	06
FIFOTHR	07
SYNC1	D3
SYNC0	91
PKTLEN	FF
PKTCTRL1	04
PKTCTRL0	05
ADDR	00
CHANNR	00
FSCTRL1	0C
FSCTRL0	00
FREQ2	21
FREQ1	62
FREQ0	76
MDMCFG4	2D
MDMCFG3	3B
MDMCFG2	93
MDMCFG1	22
MDMCFG0	F8
DEVIATN	62
MCSM2	07
MCSM1	30
MCSM0	10
FOCCFG	1D
BSCFG	1C
AGCCTRL2	C7
AGCCTRL1	00
AGCCTRL0	B0
WOREVT1	80
WOREVT0	00
WORCTRL	FB
FREND1	B6
FREND0	10
FSCAL3	EA
FSCAL2	2A
FSCAL1	00
FSCAL0	1F
FSTEST	59
PTEST	7F
AGCTEST	3F
TEST2	88
TEST1	31
TEST0	09
PARTNUM	00
VERSION	06
FREQEST	00
LQI	00
RSSI	00
MARCSTATE	00
WORTIME1	00
WORTIME0	00
PKTSTATUS	00
VCO_VC_DAC	00
TXBYTES	00
RXBYTES	00
RF1AIFCTL0	00
RF1AIFCTL1	00
RF1AIFCTL2	00
RF1AIFERR	00
RF1AIFERRV	00
RF1AIFIV	00
RF1AINSTRW	00
RF1AINSTR1W	00

Not connected Off-line mode Radio state: N.A.

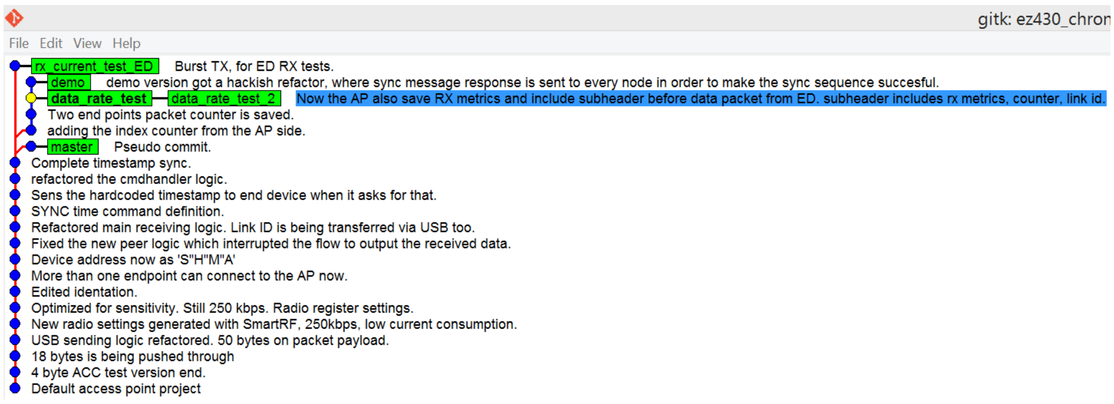
Appendix E Hyperlink to all the contents of the work

https://drive.google.com/folderview?id=0ByzSpgDwHSL7dUNEZ0Jja2RLVXM&usp=sharing

Appendix F Git snapshots of the access point and the end device software development history



```
line_of_sight_test  little refactoring of the logic.
in 1 second send 10 1 byte packets. Then delay 2 seconds.
rx_current_test  Some minor modifications.
sleep_current_test_MCU  lets disable everything we do not need
MCU total sleep
sleep_current_test_radio  Switch off the led also.
power donw radio, keep looping and incrementing.
RX mode only... lowest TX power, keep receiving.
tx_current_test  TX burst, modified PA table: -6, 0 +11 dBm.
demo  First version of demo is ready. No acceleration yet.
data_rate_test_2  Merge branch 'data_rate_test' into data_rate_test_2
data_rate_test  After the data has been sent, just stay blinking....
for ACK tests, lets send ACK too.
removed ACK and other fancy bytes. Pure timestamp and counter.
v0.3 master  Moving towards SHM apis. Radio wakes ups and sleeps are done in the same routine. Logic refactored and removed from tx and sync routines.
Timestamp initialization with simpliciti_main_sync. Timestamp drift test mode.
SYNC message definition
Timestamp logic additions.
Files for the previous changes also added.
Now has a timestamp logic. Removed alot of code, specifically bluerobin related. sends timestamp with microseconds.
Refactored the sending logic. Now sends more standard packet. Random address generation. System reset last cause implementation
More than two endpoints can now connect to AP.
Data TX is done now with ACK.
New radio settings generated with SmartRF, 250kbps, low current consumption.
50 bytes through the air.
18 bytes is being pushed through right now
Refactoring of the sending logic. Testing of the 4 byte ACC packets version.
v0.1  First working specimen. Just random acceleration data.
```



```
rx_current_test_ED  Burst TX, for ED RX tests.
demo  demo version got a hackish refactor, where sync message response is sent to every node in order to make the sync sequence succesful.
data_rate_test  data_rate_test_2  Now the AP also save RX metrics and include subheader before data packet from ED. subheader includes rx metrics, counter, link id.
Two end points packet counter is saved.
adding the index counter from the AP side.
master  Pseudo commit.
Complete timestamp sync.
refactored the cmdhandler logic.
Sens the hardcoded timestamp to end device when it asks for that.
SYNC time command definition.
Refactored main receiving logic. Link ID is being transferred via USB too.
Fixed the new peer logic which interrupted the flow to output the received data.
Device address now as 'S''H''M''A'
More than one endpoint can connect to the AP now.
Edited identation.
Optimized for sensitivity. Still 250 kbps. Radio register settings.
New radio settings generated with SmartRF, 250kbps, low current consumption.
USB sending logic refactored. 50 bytes on packet payload.
18 bytes is being pushed through
4 byte ACC test version end.
Default access point project
```

Appendix G Access point command line tool's packet parsing logic

```
void SimpliciTi::parseAndLogPackets()
{
	readData(false, 50);

	while (m_comDataBuffer.size() > 0)
	{
		// We do not know the new packet length and we must have atleast the complete header.
		if (m_currentPacketSize == 0)
		{
			// Not enough data to find the header though.
			if (m_comDataBuffer.size() < USB_PACKET_HEADER_LENGTH)
			{
				return;
			}

			// Searching for 0xFF, 0x06.
			auto newPacketBeginning = std::search(m_comDataBuffer.begin(), m_comDataBuffer.end(),
usbPacketStartSequence.begin(), usbPacketStartSequence.end());

			// Complete packet header not found.
			if (newPacketBeginning + USB_PACKET_LENGTH_BYTE_INDEX >= m_comDataBuffer.end())
			{
				std::cout << "Packet start not found, discarding " << m_comDataBuffer.size() << " bytes" << std::endl;
				m_comDataBuffer.erase(m_comDataBuffer.begin(), m_comDataBuffer.end());

				return;
			}

			// If this is somehow still zero, then next time new packet will be searched for anyway.
			m_currentPacketSize = *(newPacketBeginning + USB_PACKET_LENGTH_BYTE_INDEX) - USB_PACKET_HEADER_LENGTH;

			if (((newPacketBeginning + USB_PACKET_HEADER_LENGTH) - m_comDataBuffer.begin()) > USB_PACKET_HEADER_LENGTH)
```

```
                                        std::cout << "New packet header found, but discarding more bytes (" << (newPacketBeginning + 
USB_PACKET_HEADER_LENGTH) - m_comDataBuffer.begin()
                                        << ")." << std::endl;

                                // Erase all the not useful data and the header so later we could just cut the usable data out.
                                m_comDataBuffer.erase(m_comDataBuffer.begin(), newPacketBeginning + USB_PACKET_HEADER_LENGTH);
                        }

                        if (m_comDataBuffer.size() < m_currentPacketSize)
                        {
                                return;
                        }

                        // Lets extract the packet data out.
                        m_fileLogCallback(std::vector<uint8_t>(m_comDataBuffer.begin(), m_comDataBuffer.begin() + m_currentPacketSize));
                        s_packetsReceived++;

                        m_comDataBuffer.erase(m_comDataBuffer.begin(), m_comDataBuffer.begin() + m_currentPacketSize);

                        m_currentPacketSize = 0;
                }

}
```