Thomas Pichler, BSc

Development of a novel highly scalable RFID reader/writer firmware architecture

MASTER'S THESIS

to achieve the university degree of

Diplom-Ingenieur

Master's degree programme: Information and Computer Engineering

submitted to

Graz University of Technology

Supervisor

Ass. Prof. Dipl.-Ing. Dr. techn. Bernd Eichberger

Institute of Electronic Sensor Systems

Graz, February 2017

Acknowledgment

First I want to thank Ass. Prof. Dipl.-Ing. Dr. techn. Bernd Eichberger for supporting and helping me with this work. I also want to thank Dipl.-Ing. Dr. techn. Manfred Pauritsch for working together for more than seven years and the gained experience. Dipl.-Ing. Michael Ganzera has to be highlighted, which helped me a lot with this work. I also thank my family and friends for enabling the study and supporting me all the time.

Abstract

Radio Frequency Identification (RFID) becomes more and more important in our everyday life. Most Smartphones and credit cards as well as electronic passports or ID cards support this technology. This opens doors for lots of different applications. As multiple RFID standards are available on the market, which are quite different, a broad knowledge is required to bring this technology into new products and applications. This means that many customer specific modules are needed in order to simplify their development, to shorten the time to market and to reduce the software development costs. This requires a flexible and highly integrated firmware architecture and command set.

The goal of this master thesis is the creation and implementation of such a highly flexible and scalable firmware architecture. The software shall support market leading RFID High Frequency (HF; 13.56 MHz) front-ends and tags. It has to be easily portable for different hardware platforms and compilers. A simple to use and powerful command set enables the fast integration of this technology without the need of intense data sheet studies. Common host interfaces based on the Universal Serial Bus (USB) are also included to comply with customer demands. This includes the Chip Card Interface Device (CCID) and the Communication Device Class (CDC) protocols as well as the implementation of the Human Interface Device (HID) class. USB-HID emulates a keyboard and provides the possibility to read data from a tag automatically and send it as ASCII (American Standard Code for Information Interchange) character keystrokes to a host.

Table of Content

1. Introduction to Radio Frequency Identification (RFID)

The Radio Frequency Identification (RFID) technology can be split up into different frequency ranges. Each of them reacts differently due to the influence of the different wavelength and coupling.

Antenna theory distinguishes between the near-field and the far-field. In terms of RFID this results in different coupling behavior which has high impact on the reading distance and robustness.

The correlation between the frequency and wavelength is given as followed:

$$\boldsymbol{c = \lambda \cdot f} \tag{1}$$

Where c is the speed of light (299 792 458 m/s).

The transition zone between near-field and far-field is approximately given by the following equation:

$$\boldsymbol{r = \frac{\lambda}{2\pi}} \tag{2}$$

Where λ is the wavelength of the signal, and r is the radian sphere, which can be considered as a border between near-field and far-field (The border is not exactly defined due to the fact that the electromagnetic field changes gradually). This equation is only valid for small antennas (D<<λ), which is the case for Low and High frequency RFID technologies.

In case that D>λ (the dimension of the antenna is much bigger than the wavelength of the signal) the following estimation can be used for calculating the radian sphere:

$$\boldsymbol{r = \frac{2D^2}{\lambda}} \tag{3}$$

This estimated radian sphere is used to distinguish between near-field and far-field.[1]

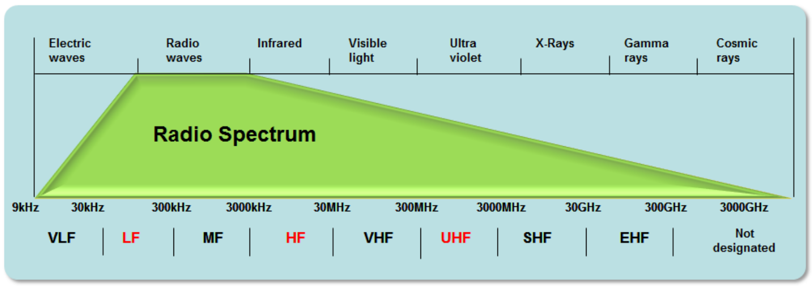
Antenna Near-Field

The Tag is powered by energy from a magnetic field. This magnetic field is the result of an inductive coupling between the Tag and the Reader. For the data transfer this field is modulated which can be detected by both, the reader and the Tag side.

Antenna Far-Field

In this domain capacitive coupling is used for supplying the Tag. The reader sends energy and data to the Tag, which uses backscattering for communication. One big advantage of this technology is, that read/write ranges are very huge.

[1] Compare Harvey Lehpamer Ef. D, RFID Design Principles, Second Edition, Artech House Books, 2012, ISBN-13: 978-1608074709, Chapter 5, p134-p135



VLF	Very Low Frequency	VHF	Very High Frequency
LF	Low Frequency	UHF	Ultra High Frequency
MF	Medium Frequency	SHF	Super High Frequency
HF	High Frequency	EHF	Extremely High Frequency

Figure 1: Overview of the different frequency ranges of RFID [2]

Low Frequency (LF) RFID

This quite common technology uses frequencies from 125 kHz up to 134 kHz. As the communication is based on magnetic coupling, metal and water have almost no influence. This is one of the main reasons why this technology is mainly used for animal tracking. The standard for this application is defined in ISO 14223 and ISO/ICE 18000-2. It is also used for access control, especially in the United States of America (USA). The read distance is typically smaller than 50 cm. The antennas are typical wire wound coils, which are more expensive in comparison to other technologies.

High Frequency (HF) RFID

This technology includes frequencies from 3 MHz to 30 MHz, but the typical frequency is 13.56 MHz. As the frequency is higher (but is still within the range of the nearfield) than for LF RFID, the surrounding, especially metal, influences the system much more. This also results in higher development effort and less reusability, as the antenna design and matching has to be done application specific. As described above the higher frequency decreases the robustness of the system, but the operating range increases to up to 1 m depending on the used standard and modulation. The data transfer rate is also much higher compared to LF RFID. As this is a good trade-off between ruggedness and distance, it is the most widespread RFID technology for consumer devices. This is the reason why this thesis focuses on the different standards based on the physical HF RFID definition at 13.56 Mhz. Typical applications are e-passports, banking cards, electronic ID credentials or even simple public transport tickets. Mainly a simple printed loop antenna is used on tag and reader side.

Ultra High Frequency (UHF) RFID

The frequency range of this technology is country-dependent, but most common is the range from 860 MHz to 960 MHz. A detailed description can be found in the EPCglobal Gen2 (ISO 18000-6C) standard. As this technology uses capacitive coupling and backscattering, the maximum read/write distance goes up to 15 m and the data rate is also much higher in comparison with HF RFID and LF RFID. As the ultra high frequency results in a very short wavelength (≈34.86 cm at 860 MHz), this means that the radiated signal gets absorbed and reflected. Because of the high range, this technology is mainly used for goods tracking and logistics. Different types of dipole antennas are used for the communication.

[2] Picture taken from Dipl.-Ing. Michael Ganzera, lecture “Identification and System Integration”, Campus 02, 2016

2. RFID System

Idea of Operation

A typical application consists of a sender (reader/writer) and a receiver (tag) part. These two components have different names according to the different standards.

The host communicates with the reader via different interfaces. The reader creates a radio frequency field. This field couples to the tag via inductive or capacitive coupling. The tag obtains energy and extracts data transmitted by the reader. The tag influences the field, by modulation for instance. The reader can detect and interpret this change according to a standard.

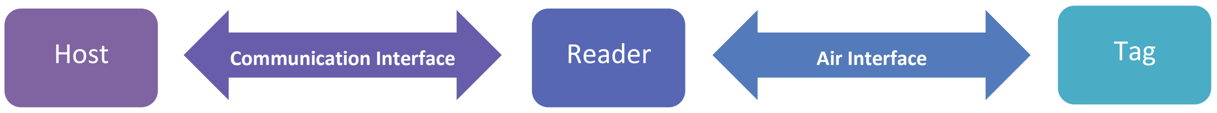


Figure 2: Topology of a typical RFID solution

As illustrated in Figure 2 multiple components have to work together to realize a RFID system. Each interface indicates the use of different standards according the given application.

As most of these components are application dependent and change quite often, a powerful architecture is needed to stay flexible and reduce the development time and cost. As can also be seen in Figure 2 the architecture has to define the communication interface. As most of the hardware has different communication interfaces, which are usually not compatible, an abstraction is required to enhance the portability. To implement the physical elements, a HF reader front end is used. As the number of different Air Interface standards is huge, this component is application specific. This results in a high coverage of different Integrated Circuits (IC) of different manufacturers. But the overall size of the framework is still limited due to the fact that low cost and low performance microcontrollers are mainly used. The Air interface standards define the used analog frontend on one hand, but also the suitable tags. As the memory structure, memory size, encryption and commands are various, most applications are limited to one kind of tag, or even disregard the capabilities of the tags. In recent years this has become even more important because of highly increased complexity of the tags. In addition to state of the art cryptography and the enhancement of the memory size, multiple applications can be stored on one tag. This results in a complex file system architecture, which is hard to implement and handle for consumer applications.

Many manufacturers want to make use of the new possibilities opened by RFID, but as this technology is changing quickly, manufacturers often are in need of special solutions. They are neither interested in deeply understanding the technology nor do they have the time to do so. As a result the need for dedicated add-on modules is high. It can be distinguished between customer specific automated solutions and generic modules. The former one handles all necessary steps to perform a customer specific action (like reading a sector of a tag for example), where the general module implements a set of (highly abstracted) commands, which can be used by the customer for enabling all tag features.

Host

This component is usually given and defined by the application. The properties can vary from high performance high power architectures to low performance low power systems. This means that the RFID system has to cover all the different aspects of the host and its capabilities.

Typical architectures of host systems are:

- Intel x86 / x64 (high power 32Bit or 64 Bit systems with a high amount of interfaces)
- ARM architecture (single core low power CPU to Quad core high performance)
- Microcontroller (limited interface options, computation power and power consumption)

Not only the hardware varies, but also its Operating System (OS). This has to be considered especially for the communication interface, which is described later. A list of common Operating Systems and their fields of use:

- Windows (x86/x64)
    - Commercial consumer and enterprise applications
    - Wide spread
- Embedded Windows
    - Especially for embedded applications
    - Commonly used within the industrial domain
- Linux
    - Standard and embedded applications
    - Scalable for specific requirements
- MAC
    - Mainly consumer applications
    - Only for specific vendor and hardware
- Real Time Operating Systems
    - Safety and security critical, mainly embedded applications
    - Low size and usually scalable

Application dependent power requirements are also important to be considered. Especially for applications like battery operated RFID door locks, the power consumption and therefore the overall lifetime is important. On the other hand standalone applications, like RFID-based time recording in companies, which are always connected to the power supply, have less stringent power consumption requirements.

Also the host language, has to be considered, as it influences USB-HID devices. This is necessary as languages and fonts differ. This can result in wrong interpretations of the transmitted data. As most semitic scripts are read from right to left, where Latin scripts are read from left to right, the languages have a huge influence on the data representation and overall communication process.

Communication Interface

The choice of the host system also influences the available communication interfaces. As even simple microcontrollers support interfaces like Universal Serial Bus (USB), the number of widespread interfaces is quite limited.

Most Common interfaces within the embedded domain are:

- Ethernet
- USB
- RS232 / Universal Asynchronous Receiver Transmitter (USART)
- IIC
- Serial Peripheral Interface (SPI)

In the domain of commercial Personal Computers (PCs) the first three interfaces are state of the art and nearly everywhere (OS independent) available. On the other hand IIC and SPI are commonly used in the embedded field. Most of the interfaces shown above do not cover the aspect of supplying the device. This leads to a higher effort for the hardware design and restricts the operating range.

For supporting most embedded hosts on one hand, but also to cover most PCs the implementation of USB is a good choice. For reasons of economy cheap microcontrollers (mainly working with the internal oscillator) do not support USB. For such cheap solutions the USART interface is quite widespread. As the same protocol, but other physical properties are used for the RS232 interface, which is today still widespread within embedded devices, this interface might be a good alternative for connecting cheap microcontrollers. Another advantage is that there are lots of different interface converters available on the market which allow a simple conversion between USART and different protocols like RS232, RS484 or USB. This increases the number of covered interfaces quickly, but keeps the firmware implementation simple.

Another important aspect for selecting a communication interface is the transfer speed, which depends on the application itself and the used components. The integrated USB interface (mainly version 2.0) of microcontrollers is a bottleneck, as the transfer rate from the microcontroller to the Tag is much slower. Interfaces like USART have an adjustable baud rate, which can be changed application dependent.

To cover not only the physical part of the host, but also the software related part, a driver is needed to communicate with the OS. As shown above, there are many drivers available. As the architecture is quite different and the driver is always hardware related, the support of different operating systems result in a huge amount of different drivers. As this is a well-known problem, standardized drivers, mainly for USB, are commonly available. These so called USB protocols are available on most platforms, even smartphones. This fact increases the need of implementing the standardized USB interface, as the effort for writing drivers can be avoided. As shown later, some of the on-board USB protocols allow to simplify everyday applications considerably and this results in an automated behavior of the module.

Universal Asynchronous Receiver Transmitter (USART)

The USART handles the signals of a serial port, which allows full duplex operation. The history of this type of connection goes back to the early days of modems. The standard defines the communication between a Data Terminal Equipment (DTE) and a Data Communication Equipment (DCE). The idea behind the protocol looks always the same but the physical layer differs. This means that multiple converters are available. These convert the USART signal, which has mostly 3.3V levels, into RS232, RS485 or other standards. Typically a DB9-Connector is used for connecting two devices using a serial port. For the basic communication only three physical lines are required: Transmit Data (TxD), Receive Data (RxD) and the common Ground (GND).[3] It is important to connect the TxD line of one device with the RxD line of the second to ensure the correct functionality. Two additional lines, Request To Send (RTS) and Clear To Send (CTS) can be added for hardware handshake. The support of this feature is manufacturer dependent and not available on all microcontrollers. Mainly in the industrial domain the use of hardware handshake is important to enhance the transmission reliability. The full serial port specification includes other signals, too. The types and their descriptions can be found within the related standards.

Most microcontrollers provide at least one USART interface. This peripheral unit converts byte oriented data into a serial bit stream, a process which is called framing.[3] As this interface is asynchronous, there is no clock line, and a time based synchronization is used. For detecting the start and end of a serial frame additional start and stop bits are added. The specification allows the following settings:

- 1 start bit (always fixed)
- 1 stop bit
- 1.5 stop bits (not common)
- 2 stop bits

For a simple bit flip detection a parity check is also part of the standard.[3] These settings are defined:

- **N**one, no parity bit is added
- **E**ven, if the number of bits with a logical '1' is even, '1' is added as parity bit, else '0'
- **O**dd, if the number of bits with a logical '1' is odd, '1' is added as parity bit, else '0'
- Mark, a bit which is equal to '1' is added to the frame
- Space, a bit which is equal to '0' is added to the frame

The receiver performs the same calculation depending on the settings and checks its correctness. In case of a failure the upper layer (application) is informed.[3]

The number of sent data bits can also be defined. From 4 to 8 data bits can be sent within one frame and the order within the frame is from the Least Significant Bit (LSB) to the Most Significant Bit (MSB).

A complete USART frame is illustrated below:

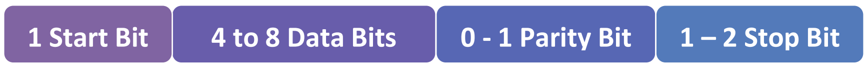


Figure 3: USART Frame

[3] Compare Lava Computer MFG Inc., RS232: Serial Ports, June 10, 2002, (Accessed: 27.1.2017): http://lpvo.fe.uni-lj.si/fileadmin/files/Izobrazevanje/OME/rs_232_serial_ports.pdf

The bitrate can also be configured and influences the maximum usable cable length. Typical Baud rates are 9600 bits/s or 115 200 bit/s. Modern high performance microcontroller even support bitrates up to 1 Mbit. The limiting factor is the cable length.

As the maximum cable length of the specification has been replaced by a maximum load capacitance of 2500 pF the maximum length strongly depends on the cable properties itself.[4] The higher the length of the cable, the mutual capacitance of the cable must be reduced.

As discussed above, additional signal lines, RTS and CTS can be used for a handshake. The standard allows either the use of this hardware handshake, software handshake (XON XOFF) or none. The main purpose for this is to reduce the risk of losing data at high transfer rates. Especially if the receiver needs more time for storing the data or even doing some pre-processing. If the sender cannot recognize such situations and continuously sends, data is lost. The extra lines are used to signal the sender that the receiver is not ready to process new data.[5]

The connection of two devices using hardware handshake is shown below:

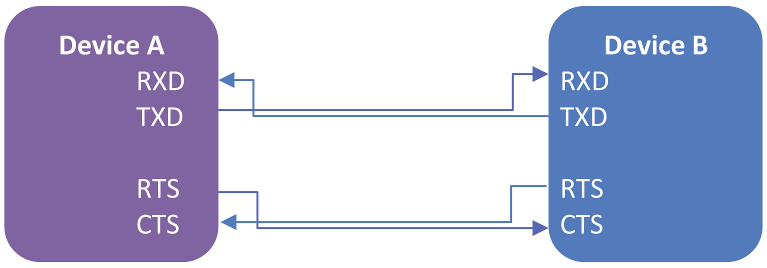


Figure 4: Two devices connected with hardware handshake [5]

Each device uses the input of the CTS signal to determine if the other device is ready to receive new data and sets the appropriate RTS signal.

There is another way of interpreting the RTS and CTS handshake. This legacy hardware flow control is illustrated below:

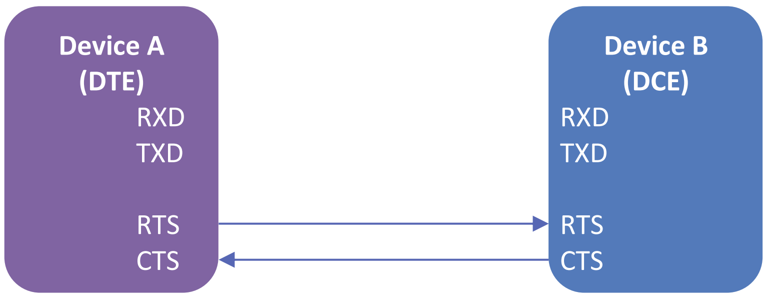


Figure 5: Legacy method for hardware handshake [5]

The names of the signal for these methods are the same but they are used in a different way. The DTE acts as a master and asserts the RTS signal, which results in change of the CTS signal, executed by the slave (DCE). The DCE can halt the communication by changing the level of the CTS line.[5]

[4] Compare Dallas Semiconductor, Application Note, 83, Fundamentals of RS–232 Serial Communications, March 9, 1998, p8

[5] Compare Silicon Laboratories Inc., Application Note, AN0059, UART Flow Control, September 16, 2013, p3-p4

As most microcontrollers support only one of these two methods, the hardware based handshake has to be implemented via software (setting the level of the signal manually).

In contrast the software handshake doesn't need additional lines and is quite easy to be realized based on the American Standard Code for Information Interchange (ASCII) control. The XON (ASCII 0x11) and XOFF (ASCII 0x13) symbols are some of this kind. The handshake works as follows: if a receiver needs a break, it sends the XOFF character. The Transmitter recognizes this special character and stops sending until a XON character is received.[6]

Consequently, it is important that both, the sender and receiver, are configured using the same parameters. A misconfiguration results in misinterpretation of the data.

There are techniques which allow Automatic Baud Rate (ABR) detection. This can be extremely useful for many applications. An example presented by Texas Instruments shows such an algorithm. It needs just one single Carriage Return (CR) ASCII symbol for detecting baud rates between 115 200 and 14 400 baud and a second one for detecting baud rates from 9 600 to 1 200 baud.[7]

The algorithm is based on the bit interpretation of the ASCII CR symbol. Depending on the bitrate the resulting/encoded value differs. The receiver gets configured for 115 200 baud and receives one CR symbol. This can be seen in the following figure:

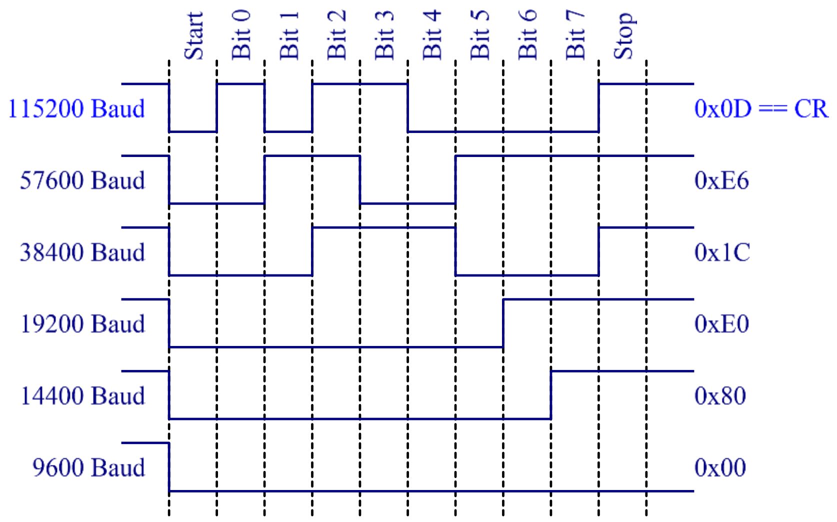


Figure 6: Bit patterns of CR for baud rates from 115 200 baud to 9 600 baud [7]

As only a comparison with these values is required, the implemtation of this method is easy. If a zero is received, the bitrate of the signal is lower and the receiver has to be reconfigured to 9600 baud and another CR has to be transmitted. More details about that can be found at 7. For selecting a suitable character for this algorithm, the variation of the lower bits is important.

[6] Compare Silicon Laboratories Inc., Application Note, AN0059, UART Flow Control, September 16, 2013, p4

[7] Compare Chuck Farrow, Texas Instruments, Application Report, SLAA215, Automatic Baud Rate Detection on the MSP430, October 2004, p4

Universal Serial Bus (USB)

The Universal Serial Bus (USB) is the most common peripheral interface these days. Especially in the computer domain, most accessories are connected via this interface. Over the years several versions with different power and speed capabilities were standardized. The latest versions of USB (3.0 or even 3.1) have data rates up to 4 Gbits/s. In the domain of microcontroller mainly USB 2.0 is supported. As all USB versions are backward compatible this is no problem.[8]

Within the USB standard different speed modes are defined:

- Super SpeedPlus (10 Gbit/s)
- Super Speed (5 Gbit/s)
- High Speed (470 Mbits/s)
- Full Speed (12 Mbit/s)
- Low Speed (1.5 Mbit/s)

For achieving such high data rates SuperSpeed and SuperSpeedPlus lines are reqired. These extra lines operate independently from the typical lines (Data + and Data -).[9]

Lower communication speeds do not always lead to poorer system performance. For data transfers in typical RFID systems, a communication speed of 1.5 Mbit/s is sufficient, because the data rate between tags and reader are 848 kBaud/s at most. Another reason for lower data rates is, that these modes are less susceptible to Electromagnetic Interference (EMI). This results in lower costs for ferrite beads and resonators as they can have a higher tolerance.[10] For high volume products, cost reduction is essential. Most microcontroller support Low and Full Speed mode. The former one usually without an external oscillator.[10]

The Bus itself is single host controlled. As there is no multi-master mode specified within the standard an extension called On-The-Go (OTG) introduces a host negotiation protocol. This allows the devices to negotiate their roles.[10]

The topology of the USB is a tiered star and therefore hubs are needed to extend the network. Such a topology has some advantages. In comparison of daisy chaining a tiered star topology allows to monitor the power consumption of each device. This additional information can be used for disabling malfunctioning appliances.[10]

The standard allows up to 127 devices per host.[8] As the bandwidth is split amongst these devices a typical computer features one host for each port. This leads to a higher overall performance.[10]

The physical connection of this serial bus is realized using 4 shielded wires. Two of them are used for powering the devices (+5V and GND). The other two (D+ and D-) are used for the differential communication and are twisted.[10] The need for twisted pair cables is given as the high data rate results in a higher EMI level. The typical cable length is up to 5 m but decreases with higher data rates. For USB 3.1 the cable length is reduced to about 1 m and two additional signal pairs are added (SSTX+ / SSTX- and SSRX+ / SSRX -).[8]

[8] Compare Craig Peacock, Beyond Logic, USB in a NutShell, 2007, Beyond Logic, Chapter 3: USB Protocols
[9] Compare Terry Moore, MCCI, USB 3.0 Technical Overview, October 8, 2009
[10] Compare Craig Peacock, Beyond Logic, USB in a NutShell, 2007, Beyond Logic, Chapter 1: Introduction

A Non Return to Zero Inverting (NRZ) scheme is used to encode the signal. To ensure adequate transitions bit stuffing is added. Additional information about the voltage levels and timings can be found within the USB standard.[11]

Plug and play are also supported by this bus. This means that the necessary drivers for the connected device are loaded dynamically. It also detects the removal of a device and unloads its driver. This is a user-friendly feature, which prevents the need for rebooting the host computer before using a new attached device.[11]

Within the last years multiple different connectors have become available. The naming scheme remains the same. Connectors from type A are mainly considered to be used for a host device, where type B sockets are used for devices. Various form factors are available to fit different application needs.

The physical connection is also used for selecting the device speed capabilities. As this is mainly highly integrated within the microcontroller the details can be found in the USB standard.

Any attached device can take power from the bus, but there are three classes of devices:

- Low power / bus powered
- High power / bus powered
- Self-powered

Each device defines its class and the amount of used current (in 2 mA steps) in its descriptor. Low power devices are bus powered and cannot draw more than 100 mA (1 unit load). Such devices also have to deal with a bus voltage from 4.4 V to 5.25 V. High power devices may use up to 500 mA (5 unit loads), but until they are configured only 100 mA can be used. For enumeration and configuration the device has to deal with a bus voltage down to 4.4 V. After that stage the voltage range is from 4.75 V to 5.25 V. Self-powered devices are externally supplied and are not allowed to draw more than 100 mA from the bus.[11] The values described above refer to USB 2.0. Newer standards allow higher currents. If the device consumes more power than defined, the host switches it off, independently from the used USB class.

The standard also defines power safe options such as the suspend mode.

For enhancing the robustness of the bus some common techniques are used. Differential drivers and receivers together with the shielding support the signal integrity. A Cyclic-Redundancy-Check (CRC) is generated and appended for detecting bit flips. It "gives 100% coverage on single- and double-bit errors"[12] Self-recovering protocols and flow control are also implemented for increased robustness.

[11] Compare Craig Peacock, Beyond Logic, USB in a NutShell, 2007, Beyond Logic, Chapter 2: Hardware
[12] Compare Universal Serial Bus Specification, Revision 2.0, April 27, 2000, p19

The protocol of USB is based on polling. Each transaction is initiated by the host and most transactions consist of three packets:[13]

- Token Packet
- Data Packet
- Status Packet

The host sends a token packet which describes the type, direction, device address and endpoint number. The addressed device decodes the token packet and depending on the direction it either receives a data packet or sends one. If there is no data to send, this is also indicated via the data packet. The destination typically confirms a successful transfer with a status packet.[14]

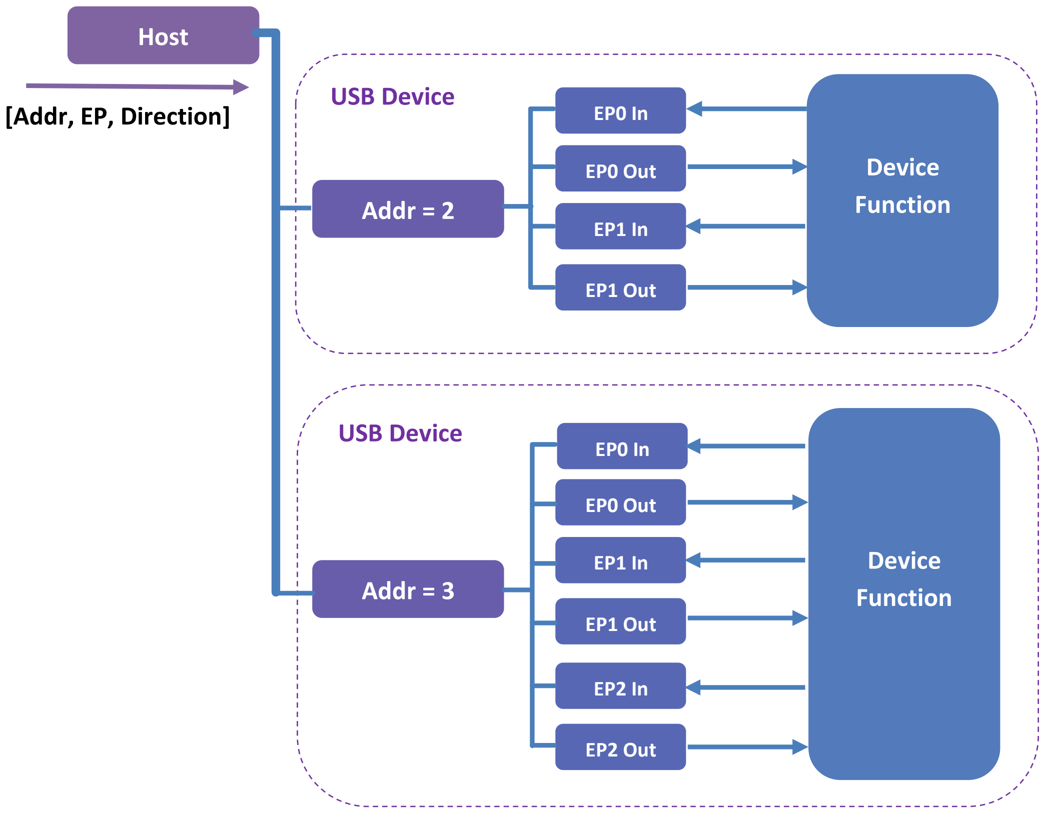


Figure 7: USB connection [14]

As shown above each device gets an address assigned during enumeration. Each unassigned device reacts to address zero, which results in an address from 1 to 127.[13] During this enumeration and configuration phase descriptors are shared between the host and the device. There are multiple types of descriptors like:

- Device Descriptor:
  Identifies the device by providing the Vendor and Product ID (VID and PID). It also defines the supported USB Version and the number of different configurations with the configuration descriptors.

[13] Compare Craig Peacock, Beyond Logic, USB in a NutShell, 2007, Beyond Logic, Chapter 3: USB Protocols
[14] Compare Universal Serial Bus Specification, Revision 2.0, April 27, 2000, p18-p19

- Configuration Descriptor:
  For each different configuration a configuration descriptor has to be provided (for the devices). The host can decide during enumeration which configuration to enable, but only one at a time. This means that if a High power and a Low power configuration are available, a host can decide based on its power management which configuration is used. Typical information within these descriptors are the power class, power consumption and number of interfaces. Most devices support just one configuration.
- Interface Descriptor:
  This type of descriptor mainly defines the features of the device. Typically each device class, like Human Interface Device (HID), which is described later, is one interface. Each interface consists of several endpoints and multiple interfaces can be featured at the same time. This allows the combination of multiple functions, like a keyboard and mouse, within one device. It is even possible to switch an interface via a host command. It also links to all class depended descriptors.
- Endpoint Descriptor:
  This descriptor defines the type of each endpoint and its capabilities. The different endpoint types are described later.
- String Descriptor:
  This optional type of descriptor is used for additional human readable information using Unicode strings for supporting multiple languages. They are typically used for naming the device, vendor or configuration.

[15] There are much more descriptor types defined within the different USB classes.

The topology of the different descriptors can be seen below:

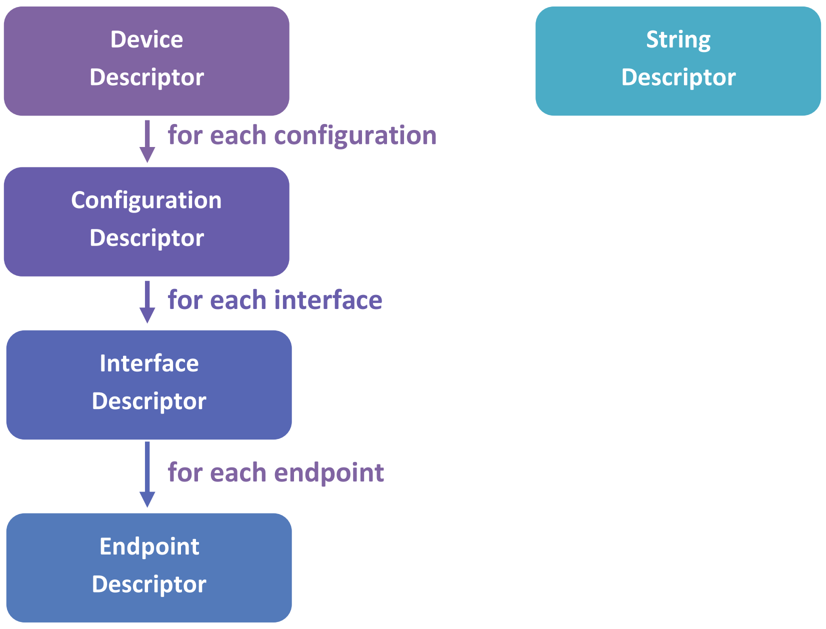


Figure 8: USB descriptor topology [15]

As can be seen in Figure 8 the number of descriptors needed by a device might be huge.

[15] Compare Craig Peacock, Beyond Logic, USB in a NutShell, 2007, Beyond Logic, Chapter 5: USB Descriptors

This has to be considered for chosing the storage capabilities of microcontroller based applications.

Each device uses the device address and its endpoints for communicating, but on higher levels so called pipes are used. These pipes represent a connection between the host and one or many endpoints. A distinction is made between Message- and Stream-pipes. The first has in contrary to the second, a defined USB format. They also define their capabilities related to bandwidth, direction, data flow and maximum packet/buffer size.[16] The control pipe is available for all devices and is responsible for exchanging the descriptors.

The USB standard defines four different types of endpoints / data transfers:

- Control Transfers:
  Mainly used for configuring the device, like sending the descriptors.
- Bulk Data Transfers:
  Is ideal for large non-time critical data transfers, as there is no guaranteed latency. Typical scenarios are data transfers of Mass Storage Devices. This type is not available for Low Speed devices. Additional CRC 16 is used for error detection.[17]
- Interrupt Data Transfers:
  Time critical data can be transferred with this type. As each transmission is initiated via the host, there is a (fix) latency for this transfer type.
- Isochronous Data Transfers:
  This type "occupies a prenegotiated amount of USB bandwidth with a prenegotiated delivery latency".[18] This transfer type can be used for streaming applications. Like the Bulk Data Transfer it is not available for Low Speed devices and also protected with a CRC.

The number of available endpoints is limited. Most microcontroller support about 5 logical endpoints, which is enough for most applications. It must be considered, that endpoint 0 is reserved for device configuration. The maximum number of endpoints per device is defined in the standard. As the bit size for the endpoint field within a typical USB frame is limited to 4 bit, up to 16 endpoints can be addressed.[16]

The USB standard also describes a number of predefined classes. Most of them are application specific like the Mass Storage device class. Each class has defined numbers and types of endpoints.

Some standards related to the application of a RFID reader are described below.

[16] Compare Craig Peacock, Beyond Logic, USB in a NutShell, 2007, Beyond Logic, Chapter 3: USB Protocols
[17] Compare Craig Peacock, Beyond Logic, USB in a NutShell, 2007, Beyond Logic, Chapter 4: Endpoint Types
[18] Compare Universal Serial Bus Specification, Revision 2.0, April 27, 2000, p20

Chip Card Interface (CCID)

This USB class is especially considered for the use of Integrated Circuit Cards (ICC) like Smart Cards. The specification considers all necessary aspects for exchanging frames. The advantage of this kind of implementation is, that the CCID-driver can be used that for the physical communication. The interpretation of the frames can be application specific, which enhances the capabilities. In general the Personal Computer / Smart Card (PC/SC) protocol is on top of the CCID class.

The CCID standard defines the class number, properties and endpoints. For enumeration and sharing the descriptors the control pipe is used, as well as for each USB class. Class specific are the use of the following pipes:

- Interrupt In
  This type is used for detecting the insertion or removal of Cards or reporting hardware errors. This endpoint is not mandatory as there are devices with integrated Smart Cards which cannot be removed.[19]
- Bulk In and Out
  These endpoints are used for sending the frames. In and Out always refer to the Host direction, so the Bulk In endpoint sends data from the device to the host. The Bulk out endpoint is used for sending frames to the device.

This results in a total number of two to three endpoints, as the interrupt endpoint is optional. As Bulk based endpoints are mandatory, this class cannot be implemented on USB Low Speed devices.

One CCID device can handle multiple Slots at the same time, but just one card per slot. The number of such slots is defined within the descriptor.

The data exchanged is mainly based on the common standard for contact Smart Cards (ISO 7816), which defines Application Protocol Data Units (APDUs) and Transport Protocol Data Unit (TPDUs). Details about this can be found within the ISO 7816 standard.

A so-called smart-card descriptor is also defined and part of this class. This specific descriptor defines physical properties of the device, like supported voltages or clock speed. This information is mainly used for checking ~~if~~ whether a technology is supported or not, but for customer specific information they can be ignored. This class defined descriptor is linked at the interface descriptor.

For the Interrupt In endpoint only two messages are defined: [20]

- RDR_to_PC_NotifySlotChange
  This message is always transmitted from the device to the host if a change of any slot is detected. 2 bits are used for each slot to signalize its current state and if the slot status has changed.
- RDR_to_PC_HardwareError
  If the device detects a hardware issue, like overcurrent on a specific slot, (the slot gets disabled) this frame is sent to inform the host.

[19] Compare Universal Serial Bus Device Class: Smart Card, CCID, Specification for Integrated Circuit(s) Cards Interface Devices, Revision 1.1, April 22, 2005, p11

[20] Compare Universal Serial Bus Device Class: Smart Card, CCID, Specification for Integrated Circuit(s) Cards Interface Devices, Revision 1.1, April 22, 2005, p56-p57

For contactless operation the implementation of this additional endpoint is a must as the number of tags within the field changes over time.

The following are the most important messages defined for the Bulk Out endpoint (PC to reader):[21]

- PC_to_RDR_IccPowerOn / PC_to_RDR_IccPowerOff
  This command can be used to change the voltage for a contacted slot. In general it is possible to use this for enabling or disabling a slot or even the whole Reader (if there is just one slot). In the domain of RFID it could be used for enabling or disabling the RF Field.
- PC_to_RDR_GetSlotStatus
  Forces the reader to return the status of one specific slot.
- PC_to_RDR_XfrBlock
  This command is the most important one as it is used for transferring the APDU or data in general.
- PC_to_RDR_Escape
  If for adding extended features for the CCID manufacturer.
- PC_to_RDR_IccClock
  With this command the reader can be forced to stop or start the clock for the connected card.
- PC_to_RDR_Secure
  This can be used for exchanging a Personal Identification Number (PIN).
- PC_to_RDR_Mechanical
  Some contact card reader support physical features for the card. This can be either the ejection or capturing of a card.
- PC_to_RDR_Abort
  This can be used to abort any current transfer.

The device is not committed to support all commands, but some of them are quite useful. There is an implementation from NXP Semiconductors, which also omits some messages. The implementation also shows a typical mapping from data frames to APDUs.[22]

Each bulk message consists of a 10 byte header and message specific data. This approach makes it easier to filter the different messages as a constant offset can be used. Each Bulk Out message causes at least one Bulk-In response.

Some of the commands, supported by the Bulk In endpoint are:

- RDR_to_PC_DataBlock
  Is used whenever the command sent by the PC contains data.
- RDR_to_PC_SlotStatus
  For slot related commands this answer is sent.
- RDR_to_PC_Escape
  Reader to PC response for the escape command.

[21] Compare Universal Serial Bus Device Class: Smart Card, CCID, Specification for Integrated Circuit(s) Cards Interface Devices, Revision 1.1, April 22, 2005, p26
[22] Compare NXP, User Manual, UM10915, Revision 1.0, March 9, 2016, p25

An example of a CCID communication can be seen in the following figure:

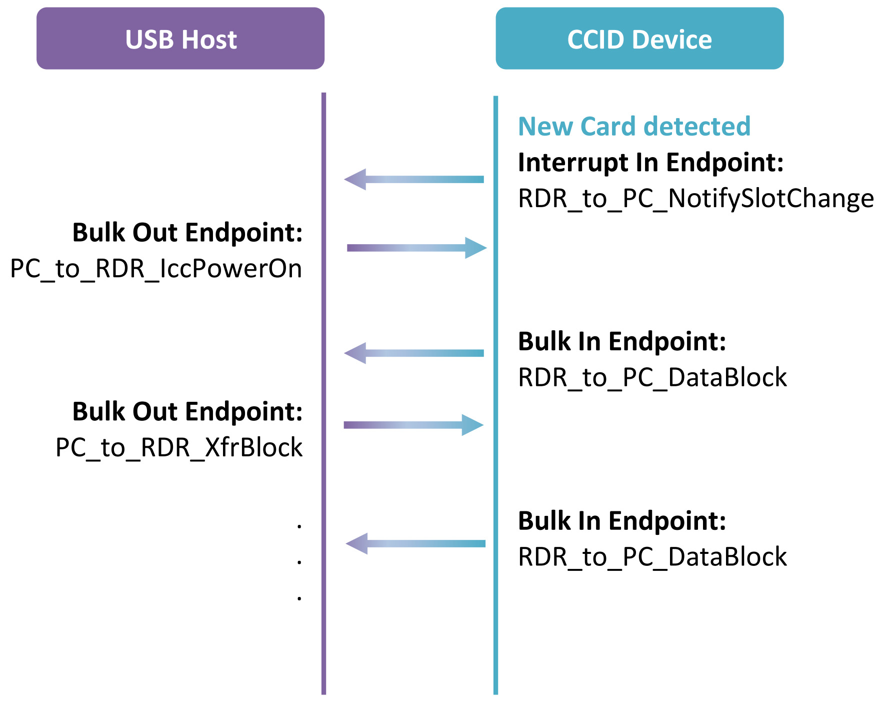


Figure 9: Simple example of a CCID application [23]

As illustrated in Figure 9 the communication flow is quite simple. The XfrBlock command can also be used to build an own protocol on top. The PC/SC standard covers this aspect, which results in a high number of devices supporting the PC/SC standard. Further details and implementation hints can be found either within the PC/SC standard or in the example from NXP, which was mentioned above.

[23] Compare Universal Serial Bus Device Class: Smart Card, CCID, Specification for Integrated Circuit(s) Cards Interface Devices, Revision 1.1, April 22, 2005, p59

Communications Device Class (CDC)

Since the early years lots of devices have featured the RS232/USART interface. Nowadays most modern computer do not support this interface by default, which leads to issues for developers who need a COM port. The solution is a subset of the USB Communication Device Class (CDC), called Abstract Control Model, which allows to emulate such a virtual COM port.

The main aspect for this subclass class is to connect legacy products, like old modems, to state of the art devices.[24] The Abstract Control Model supports the standard communication via the TxD and RxD pin and additional features like the break symbol (which can be used for synchronization).

For implementing this class three endpoints are required:[25]

- Interrupt
  Used for notification from the device to the host.
- Bulk In and Bulk Out
  Like for CCID these endpoints are used for the data transmission. As the Bulk endpoint mode is not supported by the USB Low speed standard, this class cannot be used for such devices.

This class features many request commands. The most important onces are:[26]

- SEND_ENCAPSULATED_COMMAND
  Sends Data via the transmission channel.
- GET_ENCAPSULATED_RESPONSE
  Reads Data received over the transmission channel.
- SET_LINE_CODING
  This command is for setting the bus specific settings like data rate, number of stop-bits, parity and data length for the DTE. More details about this settings can be found above where the USART is described.
- GET_LINE_CODING
  Requests the current settings of the communication channel.
- SET_CONTROL_LINE_STATE
  Can be used to tell the DCE that the DTE is present and ready.
- SEND_BREAK
  For synchronization purposes a special carrier modulation is defined within the RS232 standard. This command triggers this feature.

Some microcontrollers provide such device classes as a predefined Read Only Memory (ROM) Application Programming Interface (API). That speeds up the development time and reduces the code size. An implementation example can be found here [27].

As USB to USART converters are quite expensive, a firmware integration is very common. As it is an USB class, the driver comes preinstalled for each modern OS.

[24] Compare Universal Serial Bus Class Definition for Communication Devices, Version 1.1, January 19, 1999, p15
[25] Compare Silicon Laboratories Inc., Application Note, AN758, Implementing USB Communication Device Class (CDC) on SiM3U1xx MCUs, Revision 0.1, March 2013, p5
[26] Compare Universal Serial Bus Class Definition for Communication Devices, Version 1.1, January 19, 1999, p27
[27] Compare Silicon Laboratories Inc., Application Note, AN758, Implementing USB Communication Device Class (CDC) on SiM3U1xx MCUs, Revision 0.1, March 2013

Human Interface Device (HID)

The idea behind the Human Interface Device class is to create custom devices without any need of writing dedicated drivers. As it can be seen from its name the focus of this class is the interaction between humans and computer systems.[28] Typical devices using this class are:

- Keyboard
- Mouse
- Pointing devices
- Front panels
- Custom devices with low data communication

In general this class can be used for developing products that do not fit into other USB classes.[29] There are some requirements for using this class, for example, each transferred data has to be formatted as report. The structure of this report is shared with the host via a Report Descriptor (HID class defined descriptor). An optional feature of HID is a physical descriptor to provide information about the physical part of the human body to use the device.[30]

The link between the different descriptors is illustrated below:

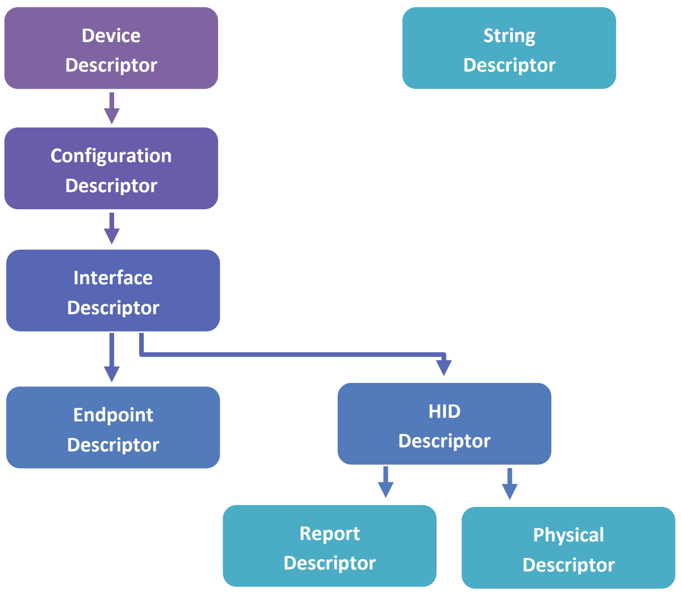


Figure 10: Topology of USB and HID descriptors [31]

In general the functionality of the device is mapped to the specific descriptors.

In general a HID device requires beside the Endpoint 0, which is always needed for configuration, only one Interrupt In endpoint. Optional the class supports an Interrupt Out endpoint for

[28] Compare Universal Serial Bus Device Class Definition for Human Interface Devices (HID), Version 1.11, June 27, 2001, p1

[29] Compare Silicon Laboratories Inc., Application Note, AN249, Human Interface Device Tutorial, Revision 0.5, March 2011, p2

[30] Compare Universal Serial Bus Device Class Definition for Human Interface Devices (HID), Version 1.11, June 27, 2001, p4-p5

[31] Compare Universal Serial Bus Device Class Definition for Human Interface Devices (HID), Version 1.11, June 27, 2001, p6

communication to the device.[32] This means that such a device can also be implemented using an USB Low Speed device.

In general the HID class is considered to transfer low data rates. Depending on the report setting data can be transmitted either via so-called short items, or long items. The former features up to 4 bytes of data. Long items can be used for exchanging up to 255 bytes of data.[33]

The direction of the data can also be described within the report. This allows also the request of data like the LED status for a keyboard.

Within the HID class some common devices are predefined. An example of such a device is a keyboard. Especially in the RFID and trading domain this kind of device can be much more than a simple keyboard.

Keyboard

In everyday life it is typically used as an input device for the PC. For several years this technology has also been used for barcode readers to input the read barcode as ASCII characters to the PC. This removes the need of installing drivers and enhances the portability. As it is not possible to send data to a keyboard (beside information for the LEDs for caps lock etc.) the scanner is used for configuring the device, too. So a special manufacturer defined barcode is used to define the interface and the encoding type.[34] As can be seen from this example new doors are opened with this idea. In the domain of RFID this could be used for automatically reading a Tag and writing its content to the PC.

The implementation of a HID keyboard is very simple. Just one Endpoint is needed for the communication. An example for the report descriptor can be found here [35]. It is also possible to report the change of multiple keys with one frame. If defined within the report descriptor the host sends the status of the LEDs (like caps lock etc.) whenever it changes. It is also possible to define the device as Boot Keyboard, which allows the usage even in the BIOS. This could be used for entering a BIOS password. Also the number of keys supported by the device can be defined to simplify the development.

This approach seems to be very generic and comfortable, but there are several pitfalls:

- Language
  As mentioned at the beginning a wrong language results in misinterpretation of the transferred data. On one hand it is possible to define a country code for the device, which can indicate the supported language, but on the other as it is not mandatory, it is very common to ignore this information.[36]

[32] Compare Universal Serial Bus Device Class Definition for Human Interface Devices (HID), Version 1.11, June 27, 2001, p20
[33] Compare Universal Serial Bus Device Class Definition for Human Interface Devices (HID), Version 1.11, June 27, 2001, p36-p37
[34] Compare IDAutomation, Programming Manual, SC7USB 2D, p12
[35] Compare Universal Serial Bus Device Class Definition for Human Interface Devices (HID), Version 1.11, June 27, 2001, p69
[36] Compare Universal Serial Bus Device Class Definition for Human Interface Devices (HID), Version 1.11, June 27, 2001, p53

It is more common not to define the mapping for each language on the device as the operating system does this in an easier way.[37]
This can lead to troubles if the attached scanner for instance has a different language than the normal keyboard and both are used.

- Speed
  The transfer speed of a HID keyboard is good enough for typing or normal usage. But for transferring the content of a whole Smart Card it might be too slow.
- Multiple keyboards
  As most OS support just one keyboard language, which is used for all attached keyboards, multiple keyboards with different languages lead to interpretation failures. This problem can only be solved by supporting all languages within the device.

Some operating systems like MAC force the user to press a special key combination to detect the language of the keyboard. The implementation of this can solve some of the described problems.

[37] Compare Universal Serial Bus HID Usage Tables, Version 1.12, October 28, 2004, p53

Reader

There are many different topologies for the reader which are strongly application dependent. In general a reader consists of at least one communication interface and a RF frontend, implementing at least one RFID standard. It typically converts the commands sent by the host into standardized RFID commands. Some solutions go far beyond by adding intelligent commands that perform multiple actions, which simplifies the application implementation.

Two approaches are mainly used within domain of RFID:

- Simple device
  For simple applications a RFID front end is directly connected to the host. This is mainly done through bus systems like SPI, UART or IIC. The advantage of this solution is that it is very cheap, simple, small and easy to change. The huge disadvantage is, that it is strongly platform dependent, as these bus systems are not available on all systems. The user application creation effort is higher, as everything has to be implemented there. This results in either using an API or implementing the whole workflow. As most APIs are strongly manufacturer dependent and not easily portable a generic solution is hard to achieve.
- Complex device
  For standalone applications or more complex solutions the reader consists of a microcontroller and a RFID frontend. This enhances the flexibility and reusability, as the communication interface to the host can always be the same, but the type of microcontroller can be easily changed. For typical applications inexpensive ARM Cortex M0 microcontrollers are sufficient. For high end solutions integrated computers such as a Raspberry Pi can be used instead. As this kind of topology is very common, some companies provide frameworks for such devices.[38] These frameworks are mainly limited by their support for different frontends, microcontrollers and tags. As they are considered for developers knowing the details of RFID and its standards, they do not ease the usage of standards or tags itself. Another limitation is the connection between the microcontroller and front end. Multiple interfaces are supported by the silicon but not within the APIs. As the time to market is often very high, integrated flexible modules, which provide more flexibility and capabilities than an API, are therefore needed. This helps to reduce the complexity of the application development and usability by integrating multiple interfaces into the devices and their firmware.

As the idea of such a complex device is no secret, some companies simply combined a microcontroller and a front end solution on one silicon chip.[39] Such a one-chip solution is described later.

Another approach is the use of CCID RFID reader ICs. They are fully compliant to the standard and are mainly a single chip solution, with a small form factor. The disadvantage on the reader side are the higher costs for the IC and a fixed USB interface. On the host side, no driver has to be developed, as they are part of many operating systems already, but the CCID stack must be implemented. For the application development detailed knowledge about the RFID standards is a must, as the CCID and

[38] Compare NXP, User Manual, UM10663, NXP Reader Library User Manual based on CLRC663 and PN512 Blueboard Reader projects, Revision 1.2, July 24, 2013, p3
[39] Compare NXP, Datasheet, PN7462, Revision 3.3, December 21, 2016, p1

upper layer only cover the communication and configuration but neither the protocol nor the tag handling.

Front End / Reader IC

Generally, it can be distinguished between front ends and so called reader ICs. The former one, often called booster, amplifies the given RF signal. This reduces the antenna size and enhances the read range.[40] As such a solution needs an additional digital part for signal generation it is not used in general designs.

Reader ICs fulfill typical system requirements like:

- Support of different RFID standards
- State of the art communication interface
- Small form factor

Multiple vendors offer different solution for this IC. The selection should be based on the application requirements, as the front end also defines the supported standards.

Each reader IC consists of an analog front end a digital part for the RFID protocol handling and at least one communication interface.

All in One Solution

Consist of RFID frontend and a freely programmable micro controller in one package, which allows reduced component costs and less Printed Circuit Board (PCB) space.

An example of such a device is the PN7462 from NXP. Its basic characteristic is:[41]

- 32 bit ARM Cortex M0 microcontroller
- High power RFID solution (HF RFID) (also available with a contacted Smartcard peripheral interface)
- Operating frequency up to 20 MHz
- Up to 160 kB Flash memory
- Several host interfaces like USB, USART, IIC and SPI
- Support of all common RFID Standards

In addition to its impressive feature set, the pitfall is within its details. The use of such an integrated solution reduces some flexibility, as the microcontroller and its features are given. Such a device may become a bottleneck in case of medium- to high-level applications, for example cryptographic operations have to be calculated within the microcontroller without a crypto unit, where the low internal operating frequency can become the bottleneck for fast transaction times.

For simple applications this approach is groundbreaking and brings many benefits. Future products of this type might comprise a more powerful microcontroller with a reader IC to remove the discussed issue.

[40] Compare Austriamicrosystems, FS_AS39230 (Accessed: 27.1.2017): http://ams.com/eng/content/view/download/382456
[41] Compare NXP, Datasheet, PN7462, Revision 3.3, December 21, 2016, p1

Antenna

For connecting the reader with the tag over the air, interface antennas are needed. Mainly loop antennas are used within the HF RFID domain. The transformer principle is used for energy transmission.[42] This is illustrated below.

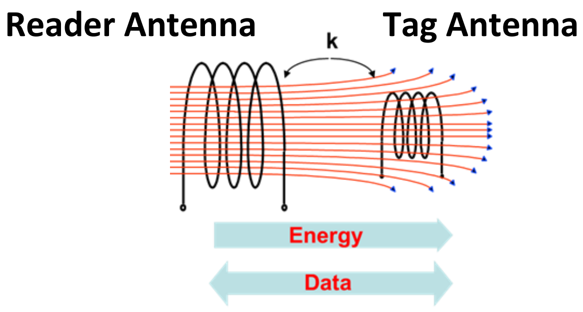


Figure 11: Principle of energy transmission in HF RFID [43]

A current through the antenna of the reader creates a magnetic flux ϕ. This flux induces a voltage V within the tag. Figure 11 shows the principle of an RFID system with two antennas of a different size.

For achieving power matching, a matching circuit is needed. This transforms the rectangular signal, generated by the reader, into a sine wave with the correct frequency. A typical example for a matching network is shown next:

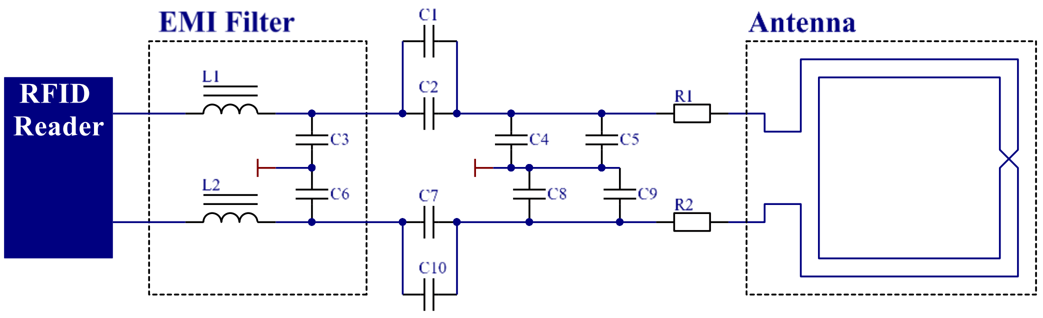


Figure 12: Example of a matching network commonly used [44]

As this technology operates within the nearfield, the surrounding has a high impact on the system. Therefore it is necessary to match each device application and environment specifically.

Some companies offer reader ICs with an integrated auto tune feature, which allows to compensate the influence of the environment within specified limits. An example of such a device is the AS3911 from Austriamicrosystems.

This principle of powering the tags is equal for all HF standards and is based on magnetic coupling.

[42] Compare NXP, Application Note, AN78010, Revision 1.0, November 2002, p6
[43] Picture taken from Dipl.-Ing. Michael Ganzera, lecture "Identification and System Integration", Campus 02, 2016
[44] Compare NXP, Application Note, AN11535, Measurement and tuning of a NFC and Reader IC antenna with a MiniVNA, Revision 1.1, November 3, 2014, p12, figure 9

Air Interface

This part defines the data transfer between the tag and the reader. For this purpose commonly the following techniques are used:

- **Amplitude Shift Keying (ASK)**
  The idea behind this digital modulation is that a bit stream is represented by different amplitudes of the signal. The simplest form of these techniques is called On-Off-Keying. This can be seen as an AND connection between the signal and the bit stream. This technique results in high data rates, but low noise immunity.[45]

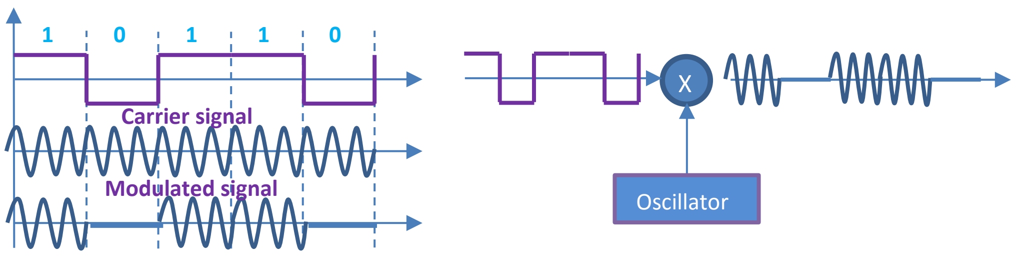


Figure 13: Functional principle of ASK [46]

As such a modulation of 100% (Signal on vs. Signal off) reduces the power, transmitted to the tag most standards use a lower modulation index. In general the modulation depth (M_D) can be calculated using the following equation:[47]

$$\boldsymbol{M_D} = \frac{\boldsymbol{modulated\ signal\ amplitude}}{unmodulated\ signal\ amplitude} \qquad (4)$$

Another specified value is the modulation index, which can be calculated using:

$$\boldsymbol{M_I} = \frac{\boldsymbol{peak\ signal\ amplitude - minumum\ signal\ amplitude}}{\boldsymbol{minumum\ signal\ amplitude + peak\ signal\ amplitude}} \qquad (5)$$

For enhancing the read range, a technology called active load modulation can be used. This technology enhances the range of RFID systems by not reducing the amplitude. This means that even for a system with small antennas (or in general a low coupling factor) good performance can be achieved. The IC manufacturer austriamicrosystems provides such a solution, which permits the use of a 100 times smaller antenna.[48] Such devices are used in the domain of smartphones, as their internal space for the antenna is limited. Even inconvenient materials, like metal, are used.

[45] Pete Sorrells, Microchip Technology Inc., Application Note, AN680, Passive RFID Basics, 1998, p4
[46] N. Vlajic, Analog Transmission of Digital Data: ASK, FSK, PSK, QAM, Fall 2010, (Accessed: 27.1.2017): https://web.stanford.edu/class/ee102b/contents/DigitalModulation.pdf , p6
[47] Atmel, Application Note, Requirements of ISO/IEC 14443 Type B Proximity Contactless Identification Cards, Rev. 2056B-RFID, November 2005, p5
[48] Nicolas Cordier, Austriamicrosystems, Technical Arcticle, How new 'boostedNFC' technology enables mobile phones and wearable devices to emulate contactless cards reliably, p3

- **Frequency Shift Keying (FSK)**
  This technique requires two signals with different frequencies. In general the carrier frequency and a second signal, derived from the carrier frequency (subcarrier) are used. In comparison the previous described ASK, the energy transmission to the tag is not being affected. On the other hand the demodulator is more complex, as it has to distinguish between two different signals. It is also more reliable than ASK, as the amplitude can be unintentionally influenced more easily than the frequency. The following figure illustrates this technique:

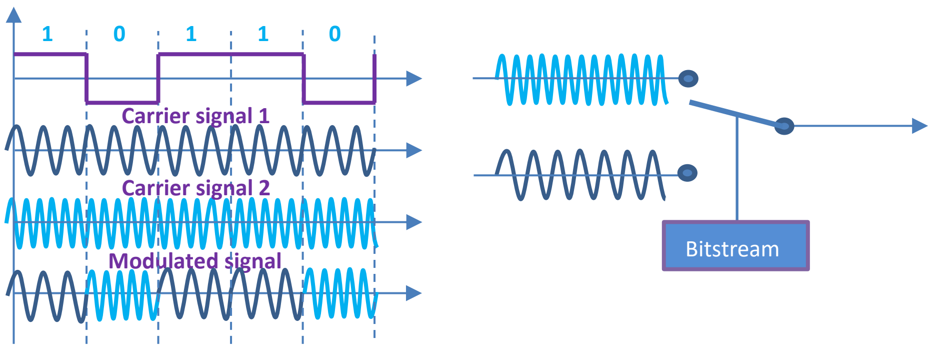


Figure 14: Principle functionality of FSK [49]

As can be seen in Figure 14, the data information is coded within the frequency. This also leads to lower data rates in comparison to other techniques.

- **Phase Shift Keying (PSK)**
  This technology is similar to FSK with one difference. Just one carrier is used. The data information is coded within the phase of the signal. Commonly, two approaches of this technique are used:[50]
  - The phase is changed for a ‘0’
  - The phase is changed for a transition (‘0’ to ‘1’ or ‘1’ to ‘0’)

In comparison to FSK higher data rates are possible and the reader design is moderate. Also the noise immunity is fairly good.[50] The topology of PSK is shown below:

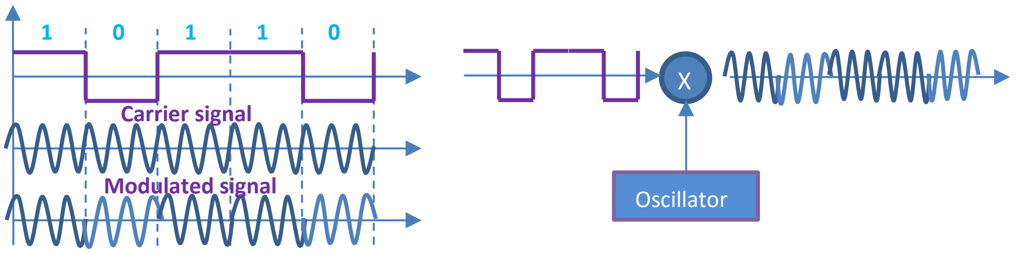


Figure 15: PSK system topology [51]

[49] N. Vlajic, Analog Transmission of Digital Data: ASK, FSK, PSK, QAM, Fall 2010, (Accessed: 27.1.2017): https://web.stanford.edu/class/ee102b/contents/DigitalModulation.pdf , p9
[50] Pete Sorrells, Microchip Technology Inc., Application Note, AN680, Passive RFID Basics, 1998, p4
[51] N. Vlajic, Analog Transmission of Digital Data: ASK, FSK, PSK, QAM, Fall 2010, (Accessed: 27.1.2017): https://web.stanford.edu/class/ee102b/contents/DigitalModulation.pdf , p12

These digital modulation techniques have the advantage that on top of this "physical" layer the data can be encoded.

Different encoding algorithms enhance the error recovery, bandwidth, synchronization capabilities and much more. Some of the most popular methods used within the domain of RFID are:[52]

- Non Return to Zero (NRZ)
- Differential Biphase
- Manchester

The basic principle is shown below:

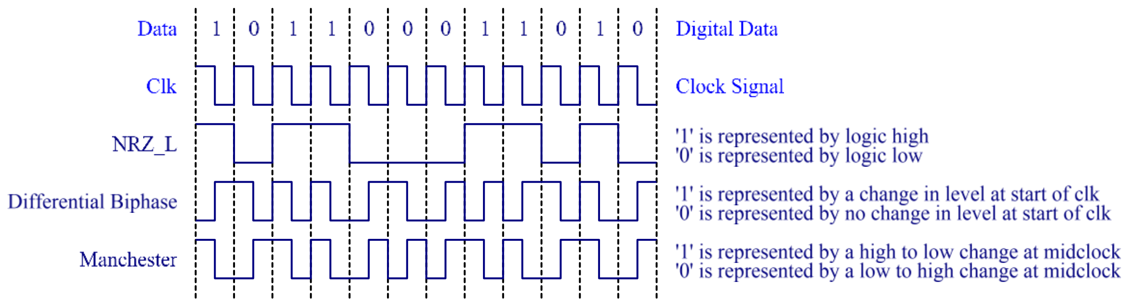

Data
1 0 1 1 0 0 0 1 1 0 1 0
Digital Data
Clk
Clock Signal
NRZ_L
'1' is represented by logic high
'0' is represented by logic low
Differential Biphase
'1' is represented by a change in level at start of clk
'0' is represented by no change in level at start of clk
Manchester
'1' is represented by a high to low change at midclock
'0' is represented by a low to high change at midclock

Figure 16: Overview of different coding used within RFID systems [52]

In general the encoding, modulation and bitrate are different for each standard. Even the communication from reader to tag and conversely can differ.

For supporting most tags within the HF RFID technology it is strongly necessary to cover the following standards. In general a RFID system does not support all of them together as the reader IC limits the number of standards.

Some manufacturer still deny any access to their own (not standardized) RFID technology. Some of them are reversed engineered by universities or security companies. The LEGIC Prime hack, is one popular example for that. It shows once again that security by obscurity is a bad approach.[53]

[52] Compare Pete Sorrells, Microchip Technology Inc., Application Note, AN680, Passive RFID Basics, 1998, p3
[53] Compare Henryk Plötz and Karsten Nohl, Legic Prime: Obscurity in Depth, December 28, 2009

ISO 14443 A/B

Like most standards, ISO14443 consist of several parts. Defining the physical characteristics, radio frequency, power and signal interface, initialization and anti-collision, and a transmission protocol.

This standard defines the reader as a Proximity Coupling Device (PCD) and the tag as Proximity Integrated Circuit Card (PICC). The physical characteristics like size of a tag can be found in part 1 of the standard. Part 2 is much more important for an implementation as it defines the carrier frequency, encoding modulation and all signal related properties for both directions (PCD->PICC and PICC->PCD).

In general two different signal interfaces are defined known as Type A and Type B. Both have a carrier frequency of 13.56 MHz, but the modulation mode and coding is different, as is illustrated in the following figure:

Direction	Type A	Type B
PCD to PICC	100% ASK Modified Miller 106 kbit/s	10% ASK Non Return to Zero (NRZ) 106 kbit/s
PICC to PCD	Load Modulation Subcarrier $f_c/16$ On-Off-Keying (OOK) Manchester 106 kbit/s	Load Modulation Subcarrier $f_c/16$ Binary Phase Shift Keying (BPSK) Non Return to Zero-Level (NRZ-L) 106 kbit/s

Figure 17: Signal details for the ISO 14443 A and B standard [54]

As most reader ICs have to be configured according to these values, this knowledge about each standard is a must. More specific details like the bit representation or timings can be skipped as this has to be fulfilled by the reader IC.

Both types support four different bitrates: 106 kBit/s, 212 kBit/s, 424 kBit/s and 848 kbit/s. It must be considered that for compatibility reason only the lowest bitrate can be used for the initialization and anti-collision.[55]

In general each communication is initiated by the reader, which leads to a master slave topology. The standard also defines a command set which has to be implemented for the reader and the tag. Each tag has a specific ID called Unique Identifier (UID). This ID was considered to be unique but within the recent years None Unique Identifiers (NUID) and even Random Identifiers (RID) are available on the market. The size of this ID is defined from 4 to 10 bytes.

One of the problems with RFID systems is that multiple tags can share the same medium (RF-Field) concurrently. Therefore an addressing scheme is important. This scheme based on the ID of the tag is called anti-collision and is different for Type A and B.

[54] Compare ISO/IEC 14443-2, Identification cards — Contactless integrated circuit(s) cards — Proximity cards — Part 2: Radio frequency power and signal interface, July 2001, p9

[55] Compare ISO/IEC 14443-3, Identification cards — Contactless integrated circuit(s) cards — Proximity cards — Part 3: Initialization and anticollision, Novembre 20, 2008

- **Bit oriented anti-collision used for Type A:**
  This procedure is split up in several steps. The principle technology is based on collision recognition. The PCD transmits a command which requests all tags within the field to reply with their complete ID. As some parts of this bit stream will be identical, collisions might occur. The reader has to detect them (bit position within the answer) and resolves the collision by sending a special frame. This frame indicates the number of valid bits (number of bits before the collision) and only PICCs that fulfill these requirements send an answer. This procedure can be performed up to 32 times.
  This means that two PICCs with the same ID, used within one RFID system, lead to problems, as they cannot be differentiated.
  Most reader ICs do not handle all these steps within the hardware, so this must be implemented in software. The standard also provides detailed additional information like a flowchart.[56]
- **Timeslot anti-collision used for Type B:**
  This anti-collision sequence uses a number of slots (N) defined by the first command (REQB) sent by the PCD. If the number is one, all PICCs response to the command else, a random number in the range of one to N. As the number of bits for the slot count is limited to 4, up to 16 slots are possible within one activation round. The reader uses a slot marker command to raise the current slot number. Each PICC notices this and is only allowed to response (transmit ATQB) within one slot (When slot number is equal to its random number). If multiple PICCs have the same random number, a collision will occur. The reader detects this collision and activates all other slots / tags. As activated PICCs do not react on the REQB command, all inactivated tags generate a new random number and the procedure is started again.
  If just one slot is available, the concurrent use of multiple PICCs is impossible. An example of this sequence is shown below:

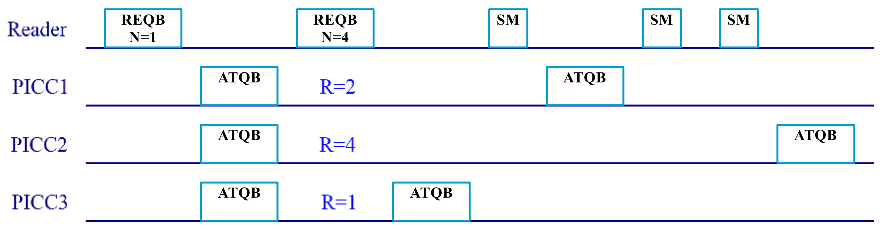


Figure 18: Timeslot anti-collision sequence with a different amount of slots [57]

As is illustrated above, the slot based anti-collision is straight forward as the reader must only detect a collision but not the bit position. This common method is also used for other standards. For an application using mainly one card, it can be a good approach to start with a slot counter of 1, as the overall time needed for processing all slots is avoided.

[56] Compare ISO/IEC 14443-3, Identification cards — Contactless integrated circuit(s) cards — Proximity cards — Part 3: Initialization and anticollision, Novembre 20, 2008, p9-p12
[57] Compare Atmel, Application Note, Requirements of ISO/IEC 14443 Type B Proximity Contactless Identification Cards, Rev. 2056B-RFID, November 2005, p12

- **Probalistic anti-collision used for Type B:**
  This approach is like the timeslot method, but the generated number is used in another way. This means that only the PICC with a generated random number ‘1’ is allowed to send its response. A higher number of available slots lead to a higher probability, that just one PICC has the randomly generated number ‘1’. An example of this anti-collision sequence is shown below:[58]

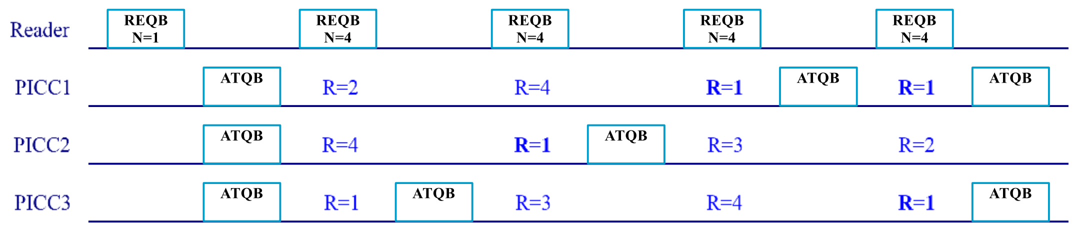


Figure 19: Functionality of the probalistic anti-collision sequence [58]

The meaning of the different commands and Tags can be found as additional information within the standard.

Part 4 of the standard describes a protocol extension including a transmission protocol.[59] It is important to know that this extension is optional and not implemented for all tags. This is the only possibility to enhance the data rate as all previous communication is fixed to 106 kbit/s.

[58] Compare Atmel, Application Note, Requirements of ISO/IEC 14443 Type B Proximity Contactless Identification Cards, Rev. 2056B-RFID, November 2005, p13
[59] Compare ISO/IEC 14443-4, Identification cards — Contactless integrated circuit(s) cards — Proximity cards — Part 4: Transmission protocol, March 19, 2007

ISO 15693

This standard names the reader Vicinity Coupling Device (VCD) and the tag Vicinity Integrated Circuit Card (VICC). As most standards it consists of several parts where part 1 describes properties like the maximum and minimum field strength or operating temperature.[60]

Within the second part the signal and transfer characteristic is defined. The operating frequency is 13.56 MHz and each transmission is initiated by the VCD.

The standard allows the PCD the use of either 10% or 100% ASK modulation.[61] As in general most VICCs support just one of those settings, both have to be implemented for full compatibility.

Pulse position modulation is defined for encoding the data. The VCD can decide either to use the 1 out of 256 or 1 out of 4 coding. The VICC on the other hand should support both, which is in general not the case.

The VICC detects the used modulation and bit encoding within the start of frame symbol, which is sent first by the VCD. Besides that the VCD can also select if the VICC shall either use one or two subcarriers and high or low data rates.[62]

Data Rate	Single Subcarrier	Dual Subcarrier
Low	6,62 kbit/s	6,67 kbit/s
High	26,48 kbit/s	26,69 kbit/s

Figure 20: Overview of the different configuration chosen by the VCD

As you can see the number of possible settings is huge. As most VICCs support just specific combinations a generic solution is hard to achieve. Figure 20 also shows the maximum bit-rates of about 26 kbit/s.

For communicating with multiple VICCs within the field, each tag has a UID. In comparison with the previous described standard the length is fixed to 8 bytes. It is also defined, that the MSB is set to 0xE0.

The anti-collision mechanism is also based on timeslots. Instead of a random numbers a mask and the UID is used to for selecting the appropriate slot. Further details can be found within part 3 of the standard.

In comparison to the previous standard ISO15693 also defines principles for additional VICC usage like reading, writing or locking. This leads to a defined limitation, as only a fixed number of bits is reserved for addressing. In general this standard limits the memory size to 8 kBytes. As this is not enough for modern applications some manufacturers build their own tag handling commands on top of this standard. This enhances the address range and capabilities.

[60] Compare ISO/IEC 15693-1, Identification cards — Contactless integrated circuit(s) cards — Vicinity Integrated Circuit(s) Card Part 1: Physical characteristics, p3-p5
[61] Compare ISO/IEC 15693-2, Identification cards — Contactless integrated circuit(s) cards — Vicinity cards Part 2: Air interface an initialization, p3
[62] Compare ISO/IEC 15693-2, Identification cards — Contactless integrated circuit(s) cards — Vicinity cards Part 2: Air interface an initialization, p8

FeliCa

This standard, defined by Sony cooperation, has become more relevant within the last years.

The communication properties are given with a carrier frequency of 13.56 MHz, ASK modulation and Manchester coding. The data rate can be either 212 kbit/s or 424 kbit/s. [63]

The standard also consists of a data link layer, which in contrast to the other standards, defines a fixed packet format. Such a frame consists of a preamble (6 bytes), sync code (2 bytes), data length (1 byte), packet data and CRC (2 bytes).[64] Which results in 11 byte additional data per frame.

On top of the datalink layer the application layer defines different packet types and a list of commands. As FeliCa tags are highly security related, even authentication commands are defined.

FeliCa also defines something similar to the UID. It is called Manufacture ID (IDm) and Manufacture Parameter (PMm).

For handling multiple tags simultaneously three different anti-collision methods are available:[65]

- **Time slot method**
  Is almost equal to the ISO 14443 Type B timeslot method. The goal of these methods is to reduce the probability of collisions by using multiple slots. In the end all tags within the field shall be activated after this procedure.
- **Identification of communication target by IDm**
  Using the IDm the reader can communicate mutually with this tag. For applications where always a fix card is used or the IDm is known, this might simplify the communication flow.
- **Identification of communication target in secure communication**
  During a secure communication each tag calculates a Message Authentication Code (MAC). As each tag is able to receive such a command, only the tag with the correct one can interpret the command and vice versa.

FeliCa also features a definition of a file system, which has not been mentioned in the previous standards. This system consists of four components:

- System
- Area
- Service
- Block Data

They are managed together as a “Block”. The filesystem itself is very complex and no reader available on the market offers a built-in support. In general, FeliCa is used in the domain of public transport systems and security related entry systems, especially in Japan.

[63] Compare SONY, FeliCa Card User's Manual Excerpted Edition, Version 2.01 No. M617-E02-01, p8
[64] Compare SONY, FeliCa Card User's Manual Excerpted Edition, Version 2.01 No. M617-E02-01, p9
[65] Compare SONY, FeliCa Card User's Manual Excerpted Edition, Version 2.01 No. M617-E02-01, p15-p19

ISO 18000-3M3

This standard, which is especially used for item management, consists of 6 parts, but only part 3 is related to HF RFID, as all other parts consider other frequencies. Within part 3 three different non-interfering modes are described. Each for a different purpose. MODE 1 is based on the ISO 15693 standard, MODE 2 defines a high speed interface and MODE 3 is based on the EPC (electronic product code) HF standard.

In general, these rather new technology is not widely common. One reason for that is a low number of tags supporting this standard. Within this domain the reader is named interrogator and same as for all RFID standards the interrogator initiates each transmission.

The reader communicates with the tag using a Double Sideband ASK (DSB-ASK) and Pulse Interval Encoding (PIE). This encoding technique guarantees that at least 50% of the overall power is delivered to the tag, even if a bit stream has only zeros. This is achieved by coding a zero with equal high and low pulses.[66]

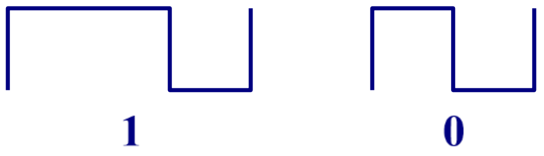


Figure 21: Pulse Interval Encoding bit definition

The communication from the tag to the interrogator is based on ASK and load modulation. Several methods for data encoding are defined like Miller – or Manchester with sub carrier. The reader also determines the frequency of the used subcarrier, which can be either 424 kHz or 848 kHz. [67]

This approach allows the use of more tags simultaneously. Therefore typical applications are document tracking or in general logistic solutions.

The standard also defines a very fast anti-collision sequence, which allows to scan about 800 tags/s. The data rate from the interrogator to the tag is typically between 26kbit/s and 100kbit/s. The tag can respond with 53 kbit/s up to 848 kbit/s.

The anti-collision sequence is based on the timeslot approach and an implementation is provided by NXP.[68]

The memory size of these tags are mainly defined in bits, which is enough for tracking applications. Especially for libraries this technology has become important within the last years.

[66] Compare ISO/IEC 18000-3, Information technology — Radio frequency identification for item management — Part 3: Parameters for air interface communications at 13,56 MHz, 2010, p52
[67] Compare NXP, Datasheet, SL2S1412; SL2S1512; SL2S1612 ICODE ILT-M, Rev. 3.2, October 8, 2013, p9
[68] Compare NXP, Application Note, AN11402, How to implement the ICODE ILT anti-collision, Rev. 1.0, October 23, 2013

Tag

The air interface is used to communicate with a counterpart to the reader, mainly called tag.

A tag consists of the RFID chip mounted on an antenna – this combination is called inlay. Further covering layers on top and bottom for artwork and / or a sticky bottom define an RFID tag. The features of such tags can vary a lot. In general the tag comes with a Unique Identifier (UID) and an application-dependent amount of memory and security features. Within the last years these capabilities have grown continuously, which enhanced the complexity of the tags. Several encryption standards, counters or even flexible file systems are available.

Tags not only differ with their standards or features, there are also two main technologies using active and passive tags. Active tags incorporate an internal power source (battery), which leads to a bigger and more costly form factor. Passive tags are powered by the reader's RF field. These topologies also differ concerning the operating range. Active powered tags have a much higher operating range as the signal sent back from the tag can be stronger than the signal from passive tags.

Different form factor and technologies allow an application specific choice of suitable tags. The form factors vary from credit card sized tags to very small round stickers with about 6mm diameter. Key rings, wristbands and watches are only some other form factors, which have become popular recently.

For selecting appropriate tags the used standard and its benefits and physical limitations must be considered. Another important parameter is the memory size. Depending on the technology the size varies from several bits up to multiple kBytes. In general tags with higher memory capabilities also offer more commands and features. This increases the development effort and this is one of the major reasons why in general, RFID modules only support simple low-level commands. This leads to features being unused.

As RFID tags are used in security related applications like e-passports or credit cards, several different encryption algorithms are supported. In the beginning NXP with their self-made MIFARE Crypto 1 algorithm invented the first encrypted tags. As the encryption was not public, it took a while until researchers hacked this algorithm.[69] It turns out that the system is based on security by obscurity which, in general, is a bad approach. Today the manufacturer added state of the art cryptography like AES, ECC or 3DES. These open standards enhance the security while complexity is increased.

As many tags are used in security critical applications, not every information is available for public use. This means that for implementing all security related and more complex features special datasheets and implementation guidelines are necessary. NXP for example offers these documents to registered and verified users via its so-called doc-store.[70] According to a given security level the user is allowed to download tag specific additional information. It is strictly forbidden to publish or share them as this information provided only after signing an NDA (Non-Disclosure Agreement). The same applies to other silicon manufacturers.

---

[69] Compare Márcio Almeida, Hacking Mifare ClassicCards, (Accessed: 28.1.2017): https://www.blackhat.com/docs/sp-14/materials/arsenal/sp-14-Almeida-Hacking-MIFARE-Classic-Cards-Slides.pdf

[70] (Accessed: 28.1.2017): https://www.docstore.nxp.com/flex/DocStoreApp.html

A special type of tag is quite new, they are called connected tags.[71] These special purpose tags are designed to add RFID technology easily to embedded systems and devices. Connected tags provide RFID tag functionality and a wired interface such as SPI or IIC. This can be used for bidirectional communication and data transfer. Another feature of such devices is that an internal circuitry allows energy harvesting for the supply of external electronic circuits such as sensors and microcontrollers. The energy for the supply is taken from the RF field.

Most RFID modules available on the market only implement the ISO standard specific commands. Which results in a complex implementation of each application. In general, all tags have different commands, which are based on the standard. Some manufacturers extend this functionality by own commands, which have different frames or timings.

A complete coverage of all available tags is nearly impossible. To enhance the capabilities of such a reader/writer module it might be a good solution to implement the commands of the most common tags.

Since Smartphones became RFID enabled, developers have been looking for new application fields. One solution is the simple paring process for a Bluetooth connection. To simplify this process, tags can be used. A special tag format called Near Field Data Exchange Format (NDEF) is used.[72] This standardized format allows each reader to interpret the content of a tag. For using this feature the tag needs a specified memory format. The format itself supports a long list of different actions like:

- Calling a telephone number
- Sending a SMS
- Creating a contact
- Pairing with a Bluetooth device
- Pairing with a Wi-Fi device

This enhances the usability and is a good alternative for QR codes. A typical application for this is a visiting card (vCard). This allows a customer to call the person by simply scanning the card instead of typing its number.

Such an encoding included in a module is barely available. As this format is quite complex, an implementation of such a feature increases the time to market significantly.

As the price for such tags is very low, the number of RFID enabled devices increases strongly.

[71] Compare NXP, Connected Tag Solutions, (Accessed: 28.1.2017): http://www.nxp.com/products/identification-and-security/nfc-and-reader-ics/connected-tag-solutions:MC_1429877262080

[72] Compare NFCForum, NFC Data Exchange Format (NDEF) Technical Specification NFC Forum, NDEF 1.0 NFCForum-TS-NDEF_1.0, July 24, 2006

3. Firmware Architecture

For covering all the described aspects the firmware architecture consists of multiple layers. This topology has multiple advantages like:

- Easily extendable, as only the specific layer has to be modified.
- Easily portable, only the hardware related layer has to be changed.
- Same structure for each project, allows to upgrade former projects with new features by simply replacing the necessary files.
- Revision history per file and not per project.
- Only the used components have to be added to the project.

The following figure shows the layers of the implemented architecture:

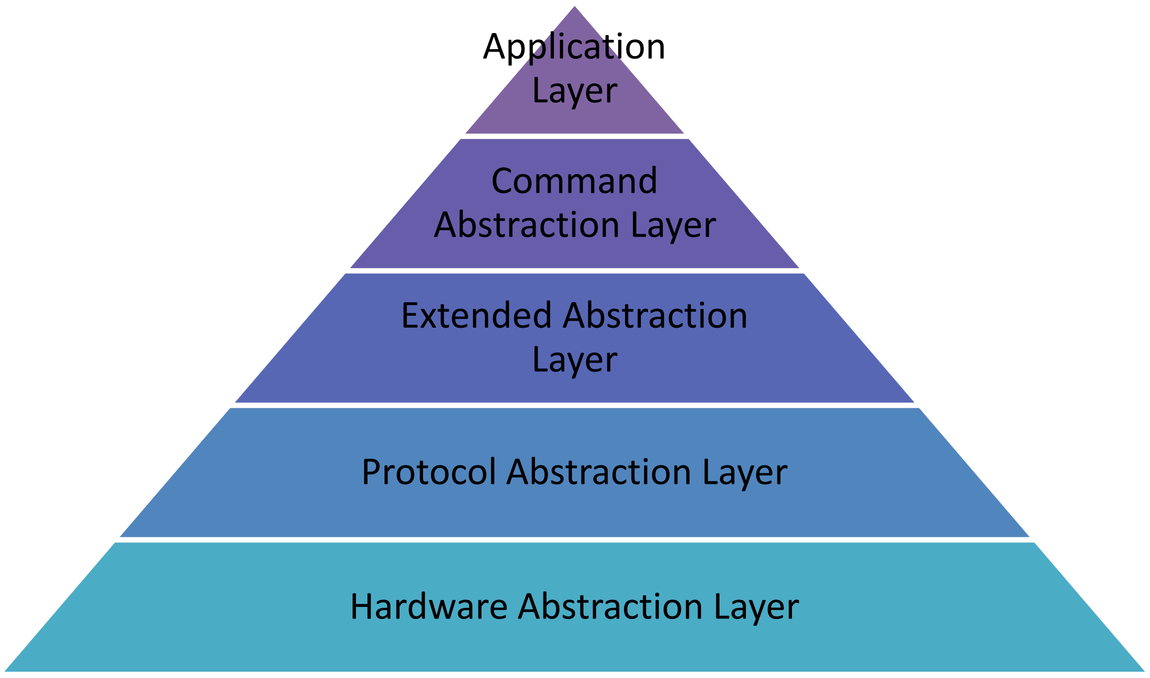

Application Layer
Command Abstraction Layer
Extended Abstraction Layer
Protocol Abstraction Layer
Hardware Abstraction Layer

Figure 22: Overview of the implemented abstraction layer

Each layer implements at least one functionality. The interaction between these layers is realized by using a function pointer concept, which will be described later in this document. In comparison to other state of the art libraries like the NXP Reader library, this implementation has two additional levels. The command and extended level are used to simplify the use of the underlying plains. As the solution, provided by NXP, is confusing and not easily extendable, it was decided to start from scratch. Another reason for this decision is the fact that the Hardware Abstraction Layer (HAL) of the available reader libraries is mainly considering the communication between the Reader IC and the Host controller. This means that it does neither allow to change the microcontroller family nor the communication interface. The implemented approach covers this by including an abstraction for the microcontroller and the reader IC.

Compared to other solutions and to enhance the usability the number of necessary files is also strongly reduced. This allows new users to get a fast overview of the structure and features of the framework.

Programming Language and Integrated Development environment (IDE)

Especially within the last years C++ compiler have become more widespread in the domain of ARM microcontrollers. But as there are still mainly C compiler available for most microcontrollers, C was chosen as programing language.

For developing microcontroller based applications multiple IDEs are available, which differ in their compilers and feature set. Even code completion or automatic code generation features are available. Some of them are for free, others can only be used for non-commercial use or for a license fee. After some research the LPC Xpresso IDE from NXP has been chosen.

This IDE is available with different licensing options. It can be used for commercial use as well as for private use. For activating the free version an account on LPCWare is necessary. This free account can be easily created. After installation this guide shows the steps to activate the free license: https://www.lpcware.com/content/faq/lpcxpresso/activating-lpcxpresso-free-edition (Accessed: 28.1.2017).

In contrast to the PRO-license, which costs around 500€ per PC, the free version has a debug code limit of 256 kB. The IDE allows to compile projects with a bigger code size, but it is not possible to load them to the target device using this IDE. The PRO version also features technical support and enhanced trace and profiling features.

As this limitation is not affecting most projects the free version of the LPC Xpresso has been used. The latest version of the IDE can be downloaded here: https://www.lpcware.com/lpcxpresso/home (Accessed: 28.1.2017).

It comes with a simple installer and multiple example projects for microcontroller from NXP and Freescale (now a part of NXP). The IDE itself is based on Eclipse, which also allows the user to add lots of helpful tools via the Eclipse Market place. In general this IDE comes with the GCC compiler, which supports C and C++ as well was ASM commands.

For enhancing the portability and reusability even for other platforms, the use of American National Standards Institute (ANSI) C is a must. It is also important not to use compiler specific commands. “Defines” shall also not be used for enabling or disabling code segments, as this leads to a higher level of confusion. For compatibility reasons the latest C standard (C11) should not be used.

This multiplatform IDE supports all common operating systems like MAC OS, Windows (7 and later) and Linux. This allows to support a wide range of development platforms.

Another advantage for the use of this IDE is the fact, that most microcontroller development boards from NXP feature a hardware debugger. This allows to flash firmware and debug the device. The necessary drivers for this tool are installed automatically during the installation process of the IDE.

During the development process a GIT based subversion system was used. This allows to track the changes and development process. It also reduces the effort of releasing a version of the framework. Such kind of submission projects can be either hosted on a local GIT server or on platforms like GITLab. Such tools are also very helpful for parallel software development as it automatically detects the changes and merges it together. It can also be used to track known bugs.

Hardware Abstraction Layer (HAL)

This layer is mainly hardware dependent and has to be implemented for any new hardware. In contrast to typical RFID libraries, this solution also covers the aspect of using different microcontrollers. Therefore this layer consists of a host and reader related part.

Minimum Hardware requirements

As not all microcontrollers have the same peripherals and capabilities, some minimum requirements for the used host controller have to be defined:

- One SPI Interface must be available for communicating with the reader IC. This can be realized using a Hardware peripheral or via Software. The Speed of this Interface shall be typically 10 MHz, but not more. The Chip select line must be implemented according the SPI standard. The signal levels have to be 3.3 V.
- At least one USART or USB connection must be available. This is needed for interfacing with the external host application. The supported bitrates for the USART communication may be platform dependent. The USB Peripheral must support at least 3 Endpoints and must feature USB full speed.
- At least one timer is necessary for implementing a common time base. It must be capable to count with a frequency of 1 MHz.
- For the code protection a manufacturer programmed unique serial number is needed. This is used to limit program updates only for specific microcontrollers and permits the use within non-authorized hardware.
- A volatile memory of at least 256 bytes is needed to permanently store configuration data. This can be either an Electrically Erasable Programmable Read-Only Memory (EEPROM) or the internal flash.
- A Code Read Protection (CRP) mechanism must be available for protecting the code.
- The size of the flash and Static Random-Access Memory (SRAM) has to be defined application specific. In general 32 kB flash and 4 kB SRAM fulfill the requirements for a standard application.

For the Reader side the following minimal requirements have to be fulfilled:

- A SPI interface to the host controller shall be used. This interface is chosen, because of the high data rate.
- There is no upper limit for the number of supported standards, but at least one RFID Standard must be supported.
- The signal level of the interface signals shall be 3.3 V.
- Distinct product ID. This is used for auto detecting the attached reader IC and enhances the usability as the reader IC can be changed dynamically during production.

For typical applications the following additional features are supported by the library but are not required:

- Three independent Pulse Width Modulation (PWM) channels for driving a RGB LED.
  Or: two GPIOs for driving a red and a green LED.
- One independent PWM channel for driving a Buzzer.
- Four GPIOs for driving Antenna switches for supporting multiple antennas at the same time.

- The possibility to enter the Flash Mode via the program.

These additional features allow to easily enhance the user experience and are mainly used on the upper most layer.

Host

This part of the HAL only considers the microcontroller. Most part manufacturers offer libraries for their microcontrollers. These are quite useful for fast prototyping and reduce the development effort. A huge disadvantage of this approach is, that these libraries often increase the latency dramatically. The size of these libraries can be enormous. To overcome these problems such libraries should not be used. The implemented framework only uses the given startup files and register definitions. This reduces the latency dramatically, but enhances the development effort, as every functionality has to be implemented in low level.

A general problem of a layer based architecture is the interaction between the layers. As all other layers are based on the HAL, a standardized interface is needed. This is achieved by defining and describing all functions that have to be implemented for each hardware. For reducing the development effort the number of these functions shall be kept at a minimum.

The following functions are used to connect the HAL with all upper layers. If the functionality is not available, the function has to be defined but not implemented. This file (uc_hal) looks the same ~~for~~ to all hardware.

- UC_HAL_SystemInit
  This function configures the microcontroller and peripheral like its watchdog and clock settings.
- UC_HAL_SPI_Init
  To configure the SPI interface to the reader, this function must be called.
- UC_HAL_SPI_Exchange
  The Reader part of the HAL and all other upper layer use this function for transferring data to the Reader IC.
- UC_HAL_UART_Init / UC_HAL_UART_DeInit / UC_HAL_UART_Send_String / UC_HAL_UART_Receive_String
  These UART related functions are used by the application to communicate with the external host system, if the USART is used as communication interface. An automatic baud rate detection, according to the principle shown in the first part, is implemented.
- UC_HAL_USB_Init / UC_HAL_USB_DeInit / UC_HAL_USB_Keyboard_SendString / UC_HAL_USB_Keyboard_SendChar / UC_HAL_USB_Send_String / UC_HAL_USB_Receive_String / UC_HAL_USB_CCID_Task
  If the USB interface is used, the following functions provide the whole USB functionality. The application can specify the mode of operation with the USB_Init function. Abstracted functions allow the application to easily send a String via the VCOM interface or Keyboard emulation. Even the CCID-Handling is abstracted. As the Endpoint management is different for most microcontrollers this has to be done controller specific.
- UC_HAL_WAIT_Init / UC_HAL_WAIT_MilliSeconds / UC_HAL_WAIT_MicroSeconds
  All time related functionality is based on this functions.

- UC_HAL_GREEN_LED / UC_HAL_TOGGLE_GREEN_LED / UC_HAL_RED_LED / UC_HAL_TOGGLE_RED_LED / UC_HAL_SET_ANTENNA / …
  For user interaction these functions are used. They provide simple access to the LEDs, buzzer and other peripherals.
- UC_HAL_RESET / UC_HAL_ReadSerialNumber / UC_HAL_ReadStorage / UC_HAL_WriteStorage / …
  These kinds of functions are used on upper layers for hardware related operations like storing data, or requesting the system to perform a reset.

As the function name is equal for each platform, the main program can also be easily changed. It also simplifies the change of the hardware platform as only the uc_hal and the linked register definition files have to be swapped.

For checking the correct behavior of the HAL a specific main program is written. This calls each of the provided functions and easily allows the verification of each function. Therefore an oscilloscope is needed to perform the timing checks.

As the names are always the same, this test case can be easily used for verifying each new hardware architecture. Verification at this level is very important as all upper layers rely on the correctness of this functionality.

As the hardware configuration and connection is not always the same, the PIN and PORT number can be easily adapted using defines. As these defines are only used locally within the uc_hal files, it is not confusing. It is a good approach to define a connection diagram for each microcontroller. This means that the connection for each possible peripheral is defined and will not change afterwards.

As can be seen above such an abstraction is always a tradeoff. On one hand it allows to easily swap the target technology but on the other special function might be left unused. This is the case for crypto engines. As in the segment of low power, low cost microcontrollers such as peripherals are still not widespread. Therefore the cryptographic functionality has to be implemented in software. This leads to a performance gap as a software implementation cannot be as efficient as a hardware module.

The current version of the library offers the support of different microcontrollers mainly manufactured by NXP. The low cost and low performance microcontroller series LPC11xx is supported as well as the mid power LPC11u6x series, which features a USB interface and enhanced capabilities.

The performance of these microcontrollers is sufficient for applications like a USB reader. As there was no application which required a higher capabilities high end microcontrollers are not implemented by now.

Reader

The reader abstraction is served by the HAL. In comparison to the host abstraction, this part is not compiler related. This means that the interaction can be realized much simpler and more generically. Therefore function pointers are used. The idea behind this approach is to dynamically change the used reader IC without changing any line of code. This can be useful for applications where only a preprogrammed microcontroller is sold. The disadvantage of this solution is, that the abstraction of each reader IC has to be included to the binary, which increases the code size. This can be omitted by directly defining the related functions.

In general this approach allows to change the used reader IC easyily with minimum effort. In contrary to the host abstraction an additional file is needed to connect the different layers. This interaction is shown below.

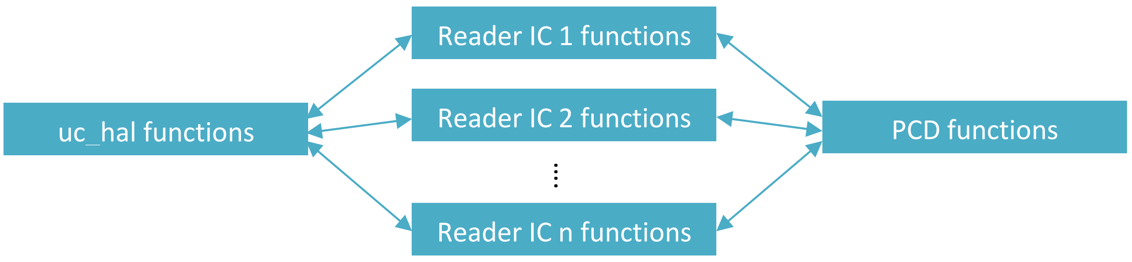


Figure 23: Interaction between the different parts of the HAL

The reader IC functions are implemented within a specific file for each device. As the name of the uc_hal functions are always the same, no specific link is required. The reader IC functions only use the SPI_Exchange and WAIT commands, provided by the uc_hal.

The PCD file is responsible to link the upper layers to the corresponding reader functions. This means that this file has to be modified for each new added reader IC.

The following functions are included in the PCD file and are used by all upper layers:

- PCD_Reset
  This function is used to reset the reader IC.
- PCD_InitTypeA / PCD_InitTypeB / PCD_InitTypeF / PCD_InitISO15693 / PCD_InitISO180003
  The upper layer calls these functions to initialize the reader for a specific protocol.
- PCD_FieldReset / PCD_Field
  The status of the RF field can be modified using these functions.
- PCD_GetVersion
  For detecting the connected reader IC this function returns the ID of the IC.
- PCD_Exchange_TypeA / PCD_Exchange_TypeB / PCD_Exchange_TypeF / PCD_Exchange_ISO15693 / PCD_Exchange_ISO180003
  These functions are called to exchange a frame according to the given standard.

A special function, called PCD_GetReaderIC, allows the upper application to either automatically detect or define the connected reader IC. This is realized by reading the reader chip ID and setting the function pointer accordingly. All functions are preset to dummy functions which return an error if called. This ensures that a call for an unsupported technology can be detected.

The overhead of this approach is limited. About only 15 function pointers are (globally) needed with no significant size impact. If the function pointers are set using the auto detect feature, the code size is increased as the compiler adds all functions according to the type of the function pointer.

The complexity of the reader related abstraction file strongly depends on the capabilities of the used reader IC. In general it has to set all register settings according to the provided standard. It is also responsible for interacting with the tag.

The reader abstraction requires detailed knowledge about the different RFID standards, as the registers have to be set correctly. Most ICs do not feature any mechanism to autoload the correct values. It must be also considered that some settings are antenna specific and might need changes according to the application and hardware design.

As the reader ICs mainly differs in the number of supported standards, the created test script verifies all of them. As described above an unavailable standard is represented by a dummy function and is skipped during this process. This half automated verification requires at least two tags for each standard. This is needed, as otherwise the anti-collision sequence cannot be verified.

The following NXP Semiconductor reader ICs are supported and implemented by the framework:

- CLRC663
- CLRC663 plus
- CLRC631
- MFRC630
- SLRC610
- PN512
- MFRC522
- MFRC523

They cover most of the state of the art standards and fulfill all requirements for most RFID applications. Most of them also feature the MIFARE crypto, which allows to read the widespread MIFARE Classic tags.

For the communication between the protocol and this layer, a structure is used. This struct defines not only exchange parameters like the data stream, but also settings like CRC or number of transmitted bits can be defined.

Protocol Abstraction Layer (PAL)

This layer consists of all implemented standards. The interconnection to the previous described HAL is done by using the pcd functions. All upper layers use the functions provided by the specific protocol.

The following figure shows the topology of this layer:

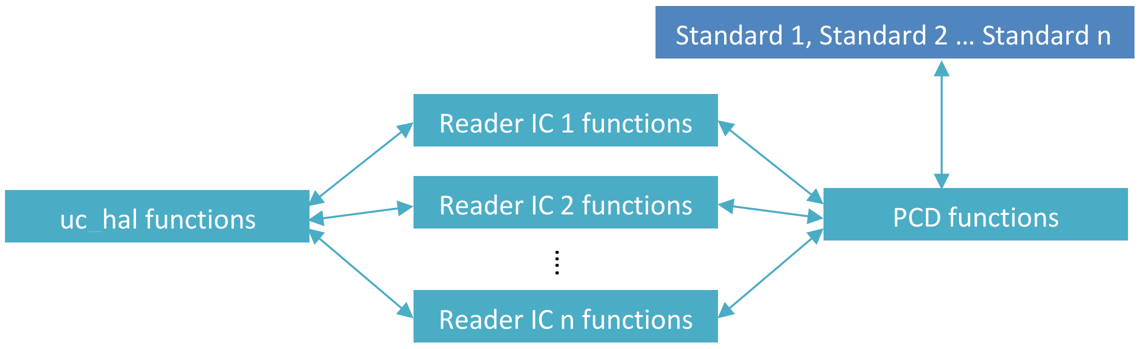


Figure 24: Topology of the lower two layers

Standards

At the moment the following RFID standards and features are implemented:

- ISO 14443 A / B:
  The current implementation allows the exchange of all different frame types for all different data rates. It also covers all Commands defined within the standard like:
  - REQA / B
  - HALT A / B
  - Select
  - Anti-collision
  - …

  The implementation of the anti-collision sequence also allows the use of multiple tags concurrently. For type B this sequence is implemented using the time slot approach. The anti-collision for type A is implemented as described in part two. For high sophisticated users this functionality is enough to cover these standards.

- ISO15693:
  The implementation of this standard does not only cover the defined commands but also manufacturer related commands, like from NXP and ST Microelectronics to enhance capabilities.
  It also supports all different kinds of option flags and bitrates. This allows to support all available ISO15693 tags.
  Some of the most important functions are:
  - Inventory
  - Select
  - Read Single Block
  - Write Single Block
  - Authenticate
  - …

The implementation also features the whole slot based anti-collision sequence. The details can be found within the standard or at part two.

- FeliCa:
  The implementation of this standard also covers the anti-collision sequence and all standard related functions. As this tag type is not that common and features lots of complex features, the functions are provided directly for the users.
- ISO180003:
  The most important thing for this implementation is the anti-collision sequence, as this standard provides the capabilities to read the ID of many tags at the same time. The implementation has been tested and verified using NXP Icode ILT Tags. Depending on the used tags (antenna and coupling factor) up to 65 tags can be read simultaneously. This can be enhanced by tuning the antenna and varying some register settings.

It must be considered that these components only work for reader ICs with the appropriate capabilities. It is not possible to send ISO 15693 frames if the reader IC does not support this standard.

As the interaction with the specific tags is also protocol dependent, the implementation of the different tags is also handled within this layer.

For the implementation all commands supported by the tags are implemented using the timing definitions within the datasheet, user manuals and standards. This results in a complete coverage of a high number of available tags.

Tags

The following ISO14443 Tags are completely implemented:

- Entire NTAG family including NTAG I2C connected tags
- MIFARE Ultralight including all generations and sizes
- MIFARE Ultralight C
- MIFARE Classic including all sizes and generations including EV1
- MIFARE Plus
- TOPAZ family
- Infineon MY-d series
- MIFARE DESFire all generations including EV2

The following ISO15693 tags are completely implemented:

- Entire ICODE family
- Austriamicrosystems Sensor Tags
- ST Microelectronics Sensor Tags

The following ISO180003 Tags are also completely implemented:

- ICODE ILT family

To support most other tags, special exchange commands are available for all implemented standards. These commands allow to exchange all commands to a tag according the selected standard.

As the number of available tags grew over time, it became more complicated to distinguish the exact type or family member of the read tag. Most companies did not care about this fact for years. This leads nowadays to a high effort for detecting the exact tag type. Newer tags support special *getversion* commands to achieve this goal. It took a while until all different tag generations and variants could be distinguished based on a complex decision tree.

The development procedure of this layer will probably never stop, as it will have to be modified for each new tag family. On one hand this will lead to a lower number of files, but on the other hand the version counting will become very important.

Based on the large list above it can be surmised, that this layer requires most of the effort, needed for the whole framework. Not only the implementation of all standards is done here, also the different types of tags have to be analyzed and implemented.

For using the properties and information of a tag in different layers, a special structure is used. It collects all features, advantages and information and reduces the number of parameters passed to each function. The size of this struct is about 50 bytes and has to be reserved for the number of concurrently used tags. This results in a limitation of concurrently used tag for microcontrollers with a low amount of SRAM.

Most RFID libraries available on the marked only include the different standards, which leads to a high effort supporting all the different standards. Manufacturer related libraries like the NXP reader library support only standards used by their manufactured tags. Which can lead to problems for applications using tags from other companies.

Extended Abstraction Layer (EAL)

In comparison to other libraries this implementation offers additional features which simplify the usage of most common features and allow an easy integration without any additional information.

Most but not all of these extensions are directly related to underlying layers. For example the whole cryptography is needed for using the encryption of a tag, but is not directly related to the protocol- or hardware abstraction layer.

The following figure shows the overall topology:

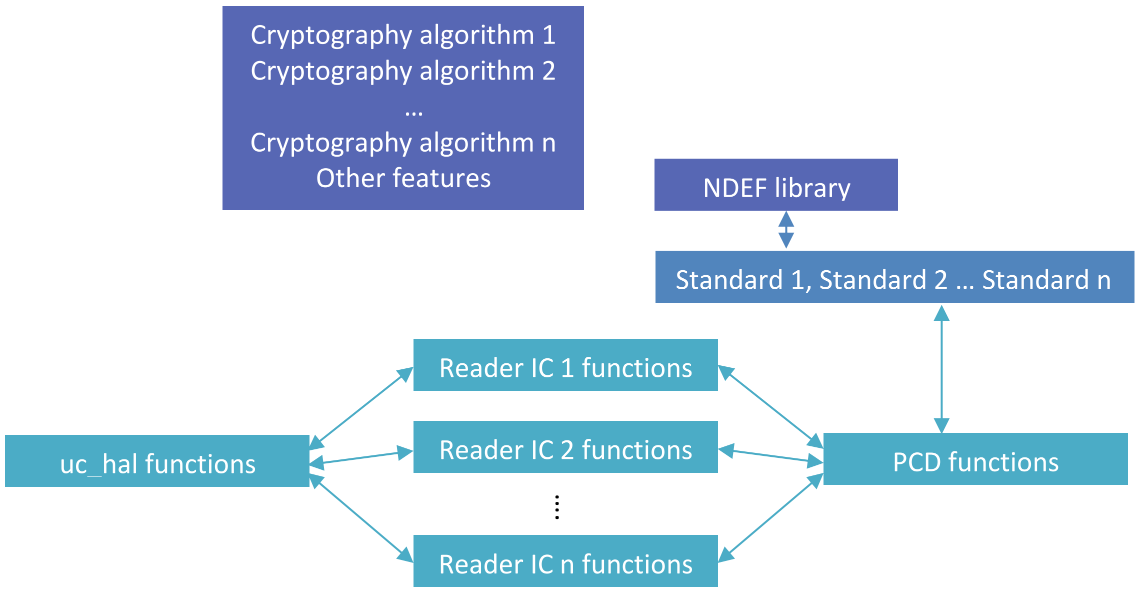


Figure 25: System topology from HAL to Extended Abstraction layer

As illustrated in Figure 25 this layer mainly implements cryptographic algorithms and the NDEF library.

Cryptography

As most low cost microcontroller do not feature an appropriate crypto unit, this part is totally implemented in software.

In general symmetric and asymmetric cryptography are distinguised. Symmetric cryptography uses the same key for en-, and de-cryption, while asymmetric cryptography requires two different private and public keys. Most common algorithms here are RSA and ECC.

Asymmetric cryptography cannot only be used for en- and de-crypting it is also possible to create digital signatures. Within the last years this approach has even taken place for tags.

The following figure shows a comparison between these two methods:

	Advantages	Disadvantages
Symmetric cryptography	Fast One key	Key derivation One key No generation of signatures
Asymmetric cryptography	Simple key derivation Two keys / increased security Signature generation possible	Slow Two keys

Figure 26: Comparison of asymmetric and symmetric cryptography

It is also possible to combine the advantages of both methods to achieve an easy exchange of symmetric keys via an asymmetric encrypted channel.

To cover most state of the art tags with their cryptographic features, the following algorithms are necessary:

- **MIFARE Classic Crypto 1**
  This algorithm is only necessary for MIFARE Classic tags. It is implemented in many reader ICs, but requires a license fee. Therefore at an early point in time the decision has to be made, if this kind of tag is used within the target application.
  As some researchers hacked this algorithm some years ago, it should not be used in security critical environment.
- **eXtended Tiny Encryption Algorithm (XTEA)**
  This is an extension to the Tiny Encryption Algorithm (TEA), which eliminates two common weaknesses.[73]
  In general this algorithm is not broken, if correctly used and the number of rounds is chosen wisely. This algorithm is provided for fast and simple encryption of the configuration parameters, if the AES algorithm does not fit into the flash memory because of the size of its pre-calculated tables.
- **Data Encryption Standard (DES)**
  This standard is mainly used within old MIFARE DESFire tags. Modern computers allow to find the used DES key with a brute force attack within hours.[74] This is one of the reasons why it is not used for modern tags.
- **Triple Data Encryption Standard (3DES)**
  The main problem of the DES is the short key length of 56 Bits.[74] To overcome this weakness, this algorithm uses the DES three times. This results in a higher computation time, which also leads to a higher power consumption needed by the tag.

[73] Compare Roger M.Needham and David J.Wheeler, Tea extensions, October 1996, (Accessed: 2.2.2017) http://www.cix.co.uk/~klockstone/xtea.pdf , p1
[74] Compare Karthik .S, Muruganandam .A, Data Encryption and Decryption by Using Triple DES and Performance Analysis of Crypto System, November 11 2014, ISSN (Online): 2347-3878 Volume 2 Issue 11, November 2014, International Journal of Scientific Engineering and Research

For encryption the DES algorithm is used as follows:

$$\boldsymbol{c} = \boldsymbol{E}(\boldsymbol{D}(\boldsymbol{E}(\boldsymbol{m}))) \tag{6}$$

The following equation shows the decryption procedure:

$$\boldsymbol{m} = \boldsymbol{D}(\boldsymbol{E}(\boldsymbol{D}(\boldsymbol{c}))) \tag{7}$$

Where **E** means encryption and **D** decryption, **m** is the message and **c** the cipher text. As can be seen in the equations above, the DES algorithm is used three times.
To overcome the problem with the short key length, a different key can be used for each calculation. This leads to a theoretical key length of 168 bit, but 112 bit in practical due the reason of the meet in the middle attack. It is also possible to select the same key for the first and third time, which decreases the key length to 112 bits. The use of the same key for all three times is not common, as this leads to the same key length as for DES.
This cryptographic algorithm is used for MIFARE DESFire and Ultralight C tags for example.
The power consumption of these tags is quite high as the cryptographic procedure requires lots of energy. This results in shorter read/write ranges.
Generally this method is still used in a huge number of products and applications.

- **Advanced Encryption Standard (AES)**
  This very new algorithm is much faster and more secure in comparison to the previous described methods. It does not only reduce-calculation time, but also power consumption and therefore it is widely used in mobile applications. Another advantage is its easy implementation.
  Three different key lengths are defined namely: 128 bit, 192 bit and 256 bit.[75] In the domain of RFID Tags mainly the 128 bit version is used.
  This round-based algorithm requires about 2 kByte of flash for storing additional lookup tables.

The algorithms mentioned above are all symmetric cyphers, as these are used for a secure data transmission to the tag. The user must know the corresponding key used by the tag. Depending on the tag this leads to the following problem:

The user wants to write data to an encrypted tag using a set key. For authentication reasons, this key must be sent to the tag. Depending on the implementation of the tag, this transaction is either done in plaintext or by using specific methods.

It is not a good idea to transmit the password of a tag in plain text, as the transmission medium-air is not secure. It is quite easy to log the transaction from a tag to the reader. Even an oscilloscope is good enough to reveal this information.

To overcome this problem, the authentication is implemented using a handshake mechanism. These differ for most tags, but the principle is always the same.

The figure below shows the implementation of this handshake used by the MIFARE Ultralight C tag.

[75] Compare Uli Kretzschmar, Texas Instruments, Application Report, SLAA397A, AES128 – A C Implementation for Encryption and Decryption, July 2009, p1

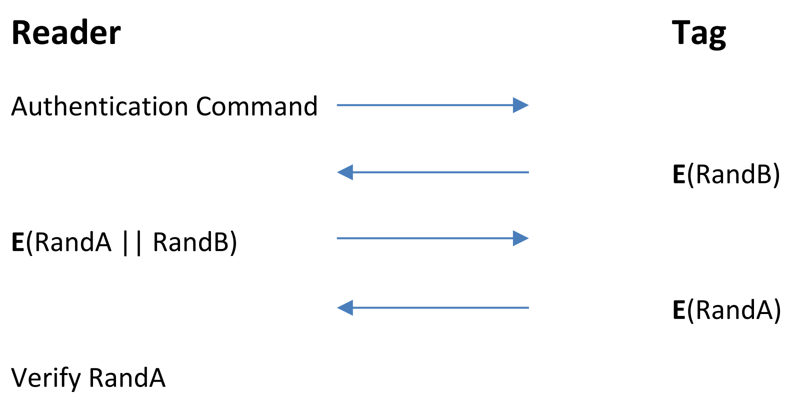


Figure 27: Basic principle of a key derivation for RFID Tags

As usual in the RFID domain, the reader initiates the authentication procedure. The tag generates a random number called RandB and encrypts it using the secret key. The reader receives this information and generates another random number called RandA. The received encrypted RandB and RandA are concatenated and encrypted by the reader using the same secret key. This result is sent to the tag. It is now able to decrypt the received data packet and verifies its previous generated RandB. This ensures that both parts know the secret key. As a last step, the RandA, which has been successfully decoded, is encoded using the secret key, and sent back to the reader. There it is compared to the previous generated RandA.

This method requires the use of random numbers, which sounds much simpler than it is. A deterministic environment is not ideal for non-deterministic behavior. Most microcontrollers do not feature a true random number generator. Therefore a pseudo random number generator has to be implemented. There are lots of different approaches for creating them, but most of them are neither good nor easy to implement. The typical approach of reading random values using an Analog to Digital Converter (ADC) leads to problems if the hardware is not safe from being modified, for example soldering the ADC pin to ground. It is also possible that a low-end microcontroller does not feature an ADC or that the ADC is reserved for the application.

In general three simple rules have to be followed:[76]

- Never use system generators like rand.
  On one hand these generators are not always available for microcontroller applications, but on the other they are flawed.[76] This results in the need of a self-made random generator.
- Add a good random generator to your code.
  Randomness cannot be achieved by an algorithm as it is always deterministic. There are tools available that verify the randomness of a random generator.[77]
  The simplest generator that passes these tests is the Keep It Simple Stupid (KISS) generator developed by G. Marsaglia. The basic implementation has a period of about 10^{37}.[78] An analysis of this algorithm showed that it shall not be used for cryptographic related

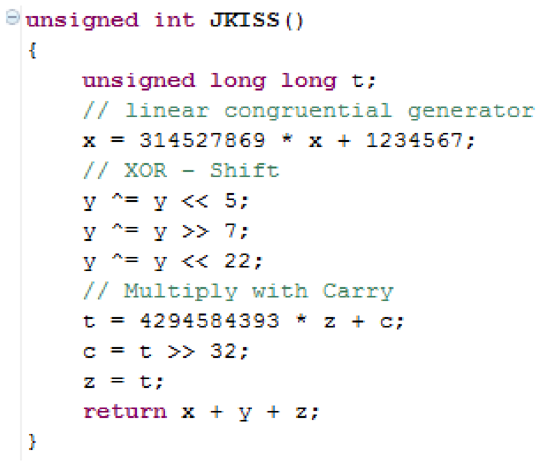
[76] Compare David Jones, UCL Bioinformatics Group, Good Practice in (Pseudo) Random Number Generation for Bioinformatics Applications, May 7, 2010, p1
[77] Compare PIERRE L'ECUYER and RICHARD SIMARD, TestU01: A C Library for Empirical Testing of Random Number Generators, August 2007, p3
[78] Compare David Jones, UCL Bioinformatics Group, Good Practice in (Pseudo) Random Number Generation for Bioinformatics Applications, May 7, 2010, p2

applications.[79] Modified versions of this algorithm with a much higher period length are much more suitable.
For cryptographic related applications like the generation of keys more complex techniques must be used. These are cryptographically secure pseudo-random number generators (CSPRNG) and not pseudo-random number generators (PRNG). As these are much more complex, they are not suitable for microcontroller applications. An analysis has shown that simple microcontrollers cannot be used for seeding a PRNG.[79]
To overcome these problems a microcontroller with a true random generator has to be used for each really security related reader like used for payment transaction.
For applications like a time recording system, a good implemented PRNG is sufficient.

- Seed your generator properly.
  This complex topic is one of the main problems in the domain of microcontrollers. Mainly the system time is used for seeding the generator, but this is predictable and not appropriate for multiple concurrent transactions.
  Some libraries like the standard C++ library offer functions like rand_s(), which can be used for seeding.

As a conclusion it can be said that for high security applications the use of an internal true random number generator is a must. For all other applications a PRNG is added to the framework. This is based on the KISS approach, which will be discussed below.

The KISS generator consists of three independent parts, which only generate pseudo random numbers in this combination, the linear congruential generator, the Xor – Shift operations and Multiply with Carry. David Jones modified the KISS generator and verified its implementation. The so-called JKISS is also used within this framework. It is easy to implement and to use.



```
unsigned int JKISS()
{
    unsigned long long t;
    // linear congruential generator
    x = 314527869 * x + 1234567;
    // XOR - Shift
    y ^= y << 5;
    y ^= y >> 7;
    y ^= y << 22;
    // Multiply with Carry
    t = 4294584393 * z + c;
    c = t >> 32;
    z = t;
    return x + y + z;
}
```

Figure 28: JKISS generator used as PRNG [80]

As can be seen above, this PRNG does not require much memory to be fast. It is very important to set the Seed values (x,y,z,c) dynamically. This is done by using a systick counter, which is set at 1 MHz. This aggravates the predication of the random number.

[79] Compare Greg Rose, KISS: A Bit Too Simple, 2011
[80] Compare David Jones, UCL Bioinformatics Group, Good Practice in (Pseudo) Random Number Generation for Bioinformatics Applications, May 7, 2010, p3

Recently several manufacturers have also begun to implement asymmetric cryptography in their tag ICs. One reason for that is the growing number of Chinese tag IC copies. Therefore this technique is mainly used to verify the originality of the tag. For this approach, a special command must be sent to the tag. The response is the UID encrypted with the private key. NXP offers the public key of this system, which allows the decryption of this data stream. If it matches the UID read during the activation process, the originality is verified.

Only the following asymmetric algorithm is used:

- Elliptic Curve Digital Signature Algorithm (ECDSA)
  This algorithm was developed as a counterpart to a handwritten signature.[81] It is based on the fact that everybody can use the same algorithm and the public key to decode a secret. Afterwards the correctness can be proven by a comparison of the decryption result and a known property. In the domain of document verification, this can be done by protecting the hash value of a document. If the encrypted hash does not match the calculated hash, the has been changed.
  The security relies on the difficulty to retrieve one key by knowing the other. Therefore the key generation is very important. For the tag related applications this is already done and only the verification has to be performed.
  The Elliptic Curve Discrete Logarithm Problem (ECDLP) is the reason for this difficulty. It also depends on the chosen curve. This means that the following properties must be known to verify such a signature:
    - Public key
    - Curve parameter
    - Which information was encrypted

  NXP for instance offers this information only for registered members of their Doc-store.

Newer tags like the NXP ICODE DNA allow to program a customer specific signature. This could be used to brand all tags used for a special application or customer.

Most of these cryptographic algorithms provide at least a pseudo code in their definitions, or are available as open source implementation. In general, it is not a good idea to start developing these from scratch. Implementation issues might lead to a total security loss. There are also lots of optimized license free implementations available.

Therefore the implementation for this framework is realized by using verified implementations. To enhance the calculation times these algorithms were optimized for microcontrollers.

Even with the optimized ECDSA algorithm the verification of a tag signature takes about 400 ms. It cannot be compared to other libraries, as this feature is only supported by this framework.

It is important to know that lots of tags support a so-called password protection. This sounds quite good but in reality, the password is exchanged in plain. So for security related applications the security supported by the tag, does not have to be considered.

[81] Compare Don Johnson and Alfred Menezes, The Elliptic Curve Digital Signature Algorithm (ECDSA), July 27, 2001, Springer-Verlag 2001, DOI: 10.1007/s102070100002, p1

NDEF Library

Another part of this layer is the NDEF library. It allows an easy use of the advantages given by the NDEF (NFC data exchange format) definition.

The idea behind this format is to trigger actions by reading a tag. Therefore the data has to be written in a special format and procedure to a tag.

It must be considered, that most tags differ in their memory layout, arrangement and size. Therefore this library is split into two parts.

- Data stream generation
- Data mapping

The first part can be used for all tags. After this data stream is generated, the mapping has to be done related to the specifics of the destination tag.

The technical documentation of the NDEF standard describes lots of different actions. The created library supports the following:

- Text
- URI
- Smart Poster
- vCard
- vCalender

All other NDEF messages can be sent to a tag using the normal write data command, which is described later.

The structure of an NDEF message is shown next:

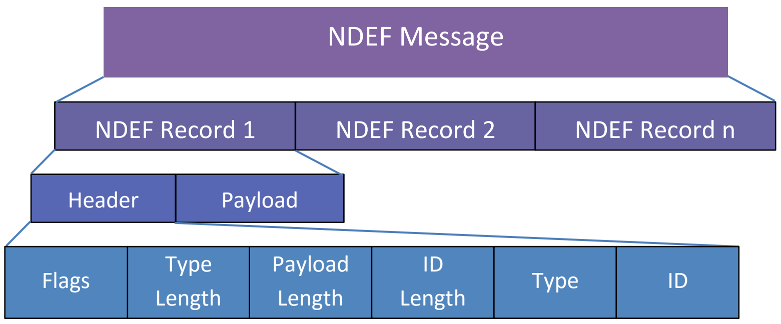


Figure 29: Structure of a NDEF message

As it can be seen in Figure 29 each message can consist of several records and each record consists of a header and the payload. The following flags are defined and used in the first byte of the header:

- Message Begin (MB) is only set for the first record within a NDEF Message.
- Message End (ME) is the counterpart to the MB and is only set for the last record within a NDEF Message.
- Chunk Flag (CF) this is used to indicate if it's the middle or first part of a chunked record.
- Short Record (SR) if this bit is set, the payload length field is limited to 8 bits, otherwise it consists of 32 bits.

- ID Length (IL) indicated that the ID Length field is available. Can be unused for typical applications.
- Type Name Format (TNF) this three bit long field defines the interpretation of the type field. Several different types are defined. Mainly used are “well known” and “URI”.

For correct functionality these flags as well as the following bytes have to be set correctly. Additional information can be found in the specification.

The implemented framework simplifies this process by offering high level functions for the most common NDEF messages. To get an overview of this process, an NDEF vCard is created step by step next.

Data stream generation

On a higher level the user only has to call the provided function like:

WritevCard(“Forename,Surname,Bday,Company,Email,Homepage,TelPriv,TelFirma”);

After passing and processing the given data the result can be seen below:

7	6	5	4	3	2	1	0
MB = 1	ME = 1	CF = 0	SR = 1	IL = 0	TNF = 0x02		
Type Length = 0x0C							
Payload Length = Len(Payload)							
ID Length is not present							
Type = “text/x-vCard”							
ID is not present							
Payload = “BEGIN:VCARD\r\n							
VERSION:3.0\r\n							
N: Surname; Forename;;;\r\n							
FN:Vorname\r\n							
ORG: Company \r\n							
URL:http://Homepage\r\n							
BDAY:Bday\r\n							
EMAIL;PREF;INTERNET:Email\r\n							
TEL;HOME;VOICE: TelPriv\r\n							
TEL;CELL;VOICE: TelFirma\r\n							
END:VCARD\r\n”							

Figure 30: Example of creating a vCard using the implemented NDEF library

As shown in the figure above, the NDEF format enhances the length of the data dramatically. The vCard record even defines the possibility to add a picture. This is not really applicable as the memory size of RFID tag is quite limited.

The provided framework function stores the whole NDEF message in a temporary buffer. This is necessary because the mapping looks different for most tags. On one hand this additionally required memory limits the capabilities of the framework, but on the other hand it is sufficient for most applications. There are multiple NDEF libraries available for platforms like android or windows, which are ideal for creating more complex NDEF messages.

Independent from the source of the data stream the NDEF message has to be written to the tag as a final step.

Data mapping

This part brings the different kind of tags and the previously generated data stream together. In general not all types of tags can be used for NDEF messages. Four different tag types are defined by the NFC Forum:[82]

- NFC Forum Type 1 Tag (Innovision Topaz)
- NFC Forum Type 2 Tag (NXP MIFARE Ultralight NXP MIFARE Ultralight C)
- NFC Forum Type 3 Tag (Sony FeliCa)
- NFC Forum Type 4 Tag (NXP DESFire, NXP SmartMX,..)
- NFC Forum Type 5 Tag (based on ISO15693 tag ICs)[83]

As the memory conditions differ, the mapping is especially defined for all of them. Therefore the framework checks if a tag is available in the field. If this is the case, the exact type has to be verified. In case of multiple tags, the process is aborted as the framework cannot know which tag to use. If the memory size of the verified tag is sufficient for storing the NDEF message the mapping and programming is performed.

An example how this mapping can be performed is shown below. It considers only NFC Forum type 2 tags, but is used to get an overview.

Type 2 Tags are general structured as follows:

Byte Number	0	1	2	3	Page / Block
UID	UID0	UID1	UID2	UID3	0
UID / Internal	UID4/Internal	UID5/Internal	UID6/Internal	Internal	1
Internal / Lock	Internal	Internal	Lock0	Lock1	2
OTP	OTP0	OTP1	OTP2	OTP3	3
Data	Data0	Data1	Data2	Data3	4
…	…	…	…	…	…
Data	Data44	Data45	Data46	Data47	15

Figure 31: Basic structure of a NFC Forum Type 2 tag

As shown above, the memory layout of a type 2 tag features 48 bytes of user data. Four additional One Time Programmable (OTP) bytes fulfill the changeable memory space. One Time Programmable does not mean that the whole byte can only be changed once. It only prevents a set bit of being reset. It also must be considered that some manufacturers include the OTPs to their user memory calculations.

Before the data stream is written to the tag, it must be formatted. The idea behind this procedure is that it can be easily decided where the tag includes a NDEF message or not. For this purpose the OTPs are used. This also means that if a tag is formatted once, it cannot be reset. Depending on the used tag it is also possible that the OTPs are already pre-set as required for an NDEF application.

[82] Compare NXP, NFC Forum Type Tags, White Paper V1.0, April 1, 2009, p21
[83] Compare NFC Forum, New NFC Forum Technical Specifications Broaden Tag Support and Enhance Interoperability, October 14, 2015, (Accessed: 28.1.2017) http://nfc-forum.org/newsroom/new-nfc-forum-technical-specifications-broaden-tag-support-and-enhance-interoperability/

The standard defines that the lowest byte of the OTP has to be set to 0xE1h, which is also called magic word. The second byte defines the version of the NFC Forum Type 2 Tag operation specification (upper four bits for the major and lower four bits for the minor version). The next and most important byte defines the memory size of the tag. Therefore this value multiplied by 8 shall be equal to the number of available memory (excluding the OTPs). For the example above this value is set to (48/8) 0x06h. The last byte of the OTPs defines the read and write capabilities of the OTPs and data segment (the upper 4 bits indicate the read access and the lower 4 bits the write access). This value is initially set to 0 (neither the read nor the write access is prohibited)

It must be considered that this read and write protection has no physical influence. It is just a definition, that according to the last byte of the OTP, the content is read or write protected. Most tags offer additional features for read and write protection, which takes place on the physical level.

As a next step the data stream has to be packed in a Tag Length Value (TLV) block. It starts with a specific NDEF Message TLV (0x03). Then the length of the whole NDEF Message is coded into either one or three bytes. After that the data stream, which was generated before, is inserted. These TLV block is completed with a terminated TLV (0xFE).

This means that the data mapping procedure increases the size of data stream by 7 to 9 bytes. This must be taken into account before the writing procedure starts. As seen in the previous vCard example, this requires more memory than available on some type 2 tags.

The last step is the writing procedure. In general the command for this also depends on the used tag, which increases implementation complexity.

For other tag types the formatting procedure is different. For example, tags with encryption have to be encrypted with a defined key to ensure consistency.

The higher the complexity of the tag the more effort is needed to map a NDEF message onto it. Tags like the MIFARE DESFire, which offers a flexible file system, require the generation of specific files with special access rights and format and contain NDEF messages. Detailed information about this can be found in the specification.

Once created, a counterpart has to perform the following actions for detecting and parsing an NDEF enabled tag:

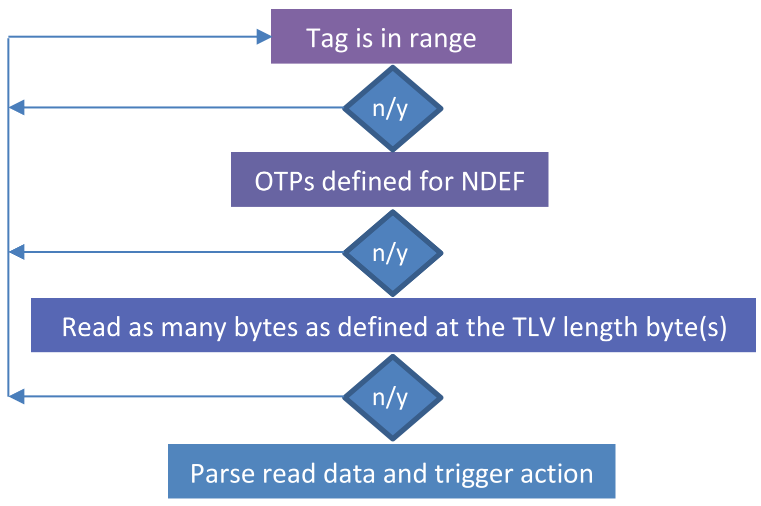

Tag is in range
n/y
OTPs defined for NDEF
n/y
Read as many bytes as defined at the TLV length byte(s)
n/y
Parse read data and trigger action

Figure 32: Procedure for detecting and parsing a NDEF enabled tag

In general the implementation of NDEF messages requires knowledge about the data stream format on one hand but also detailed information about the tag and its memory structure. The implemented simple library overcomes these aspects by providing an easy to use interface for the most common types of NDEF messages and tag types.

Command Abstraction Layer (CAL)

To allow an easy access to all the functionality, provided by the previous described layers, simple commands are added to the library. Before they are described, a general overview can be seen below:

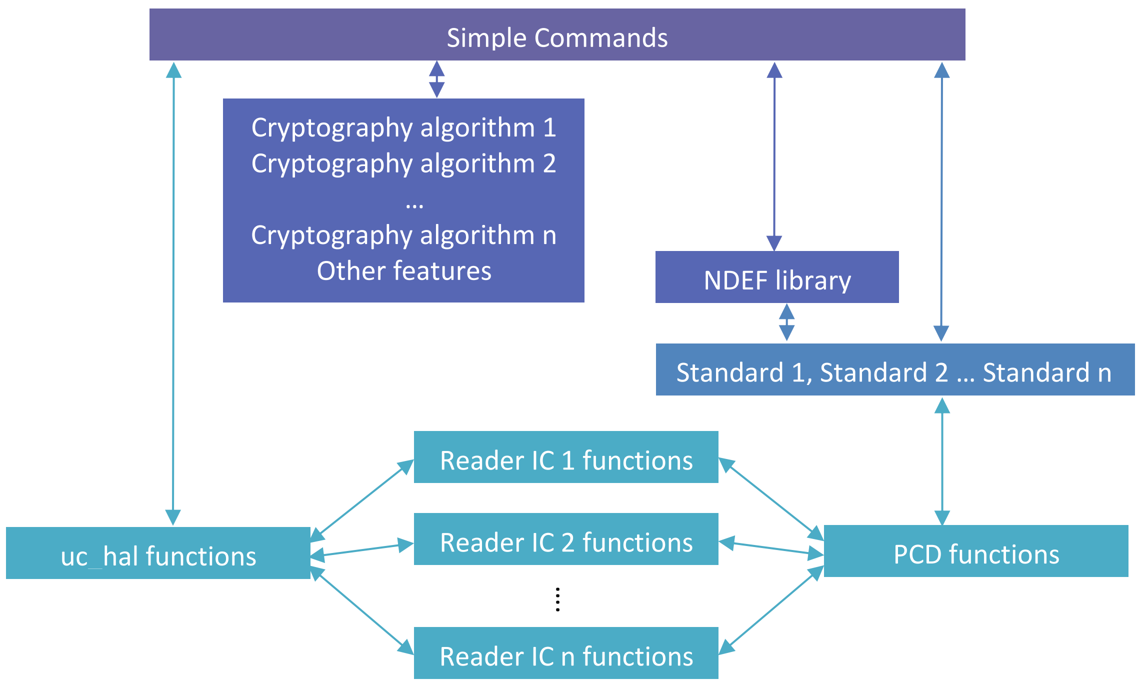


Figure 33: Firmware topology from HAL to the simple commands

As this layer has to interact at least with parts of the underlying layers, it is important that the interaction is clearly defined. As mentioned before, this is mainly done by strictly defined function names and definitions.

Before the command set can be defined an overview of other simple RFID solutions is made:

- Identive NFC Reader/Writer Module Family
  This very common solution offers a lot of different features. Not only multiple form factors are available, but also the number of supported host interfaces is enormous. Also industry related interfaces like Wiegand are available.
  It can be used in two different modes to cover most application scenarios:
  - NXP protocol
    As these modules use more integrated RFID controllers, the command set provided by the manufacturer (NXP) can be used to communicate directly with the module.
    It must be considered that these low level commands require special knowledge about the overall RFID technology.
  - EasyCommunication protocol
    This feature requires a module with equipped microcontroller. It simplifies the use of the RFID technology as all commands related to the supported commands are implemented. This is supplemented by the support of different tags and cryptographic algorithms.
    The commands also allow the configuration of the host interface, controlling two LEDs, a buzzer and storing data at the internal EEPROM.
    As it was one of the first modules with such an integrated intelligence, it opened doors for lots of applications. The development effort can be reduced as the implementation of the standards and tag related commands are provided. But as the number of tags and their possibilities grow fast, only the basic commands are implemented via this simple interface. For any advanced features, like the locking of a tag, everything has to be implemented via the low level commands.
    It is also important to know, that the commands provided for the tag handling, still require the knowledge of the basic concept, described within the standard. This leads to the problem, that lots of customers use the commands exactly as provided in the user manual, including the default crypto keys, which is terrifying.
    As each tag type has its special read/write commands, the user needs to use the correct commands related to the attached tag.

  Unfortunately Identiv discontinues the development of RFID modules. Therefore lots of former customers are in need to replace products to overcome this gap.
- Sonmicro SM130 series[84]
  These low-cost modules offer the basic functionality for the most common ISO 14443 A tags. This includes only the MIFARE Classic and MIFARE Ultralight tags. This restriction is one of the most common problems of these modules. For a new application, where no tags are currently in the field, this might be a good and cheap solution. But keep in mind, that neither the MIFARE Classic nor the MIFARE Ultralight offer real security.
  This module also offers a windows based software, which allows to investigate the memory content of an attached tag and the control of the module.

[84] Compare SonMicro, User Manual, SM130, Revision A.2, June 2006

The features of the command set can be compared to the identive approach, which means that it reduces the implementation effort, but still requires special knowledge. In comparison to the previous described module, only IIC and UART can be used as host interface. It also does not feature any cryptography as it is not required for the supported tags. There is also only one form factor available which limits the field of application.

- Jinmuyu JMYseries like the JMY504C[85]
  This product supports the ISO14443 A and ISO14443 B protocol and in comparison to the previous module the number of supported tags is increased. Also more secure tags like the MIFARE DESFire are supported. It also provides UART and IIC interface for communicating with the device. But ~~also~~ these modules only implement the basic functionality and no extended features. Another disadvantage is the use of an external antenna module, which might lead to problems for mounting this module in an application.
  The type B functionality is just given with two commands, card request and card halt. This only allows the reading of the UID which is not really sufficient for typical applications.

- Ubisys 13.56 MHz RFID USB READER[86]
  In comparison to the previously described modules, this one is based on a modern microcontroller supporting the USB standard. This enabled new possibilities like, CDC, HID and CCID.
  All these features are currently implemented in their module. The CDC functionality can be used for AT-like commands. A closer investigation shows that these commands, only support the default commands, defined in the standard. Beside FeliCa, all common standards are supported. But an abstraction of the commands to simplify the usage is missing.
  The HID feature allows to read the UID and write the content using the emulated keyboard automatically. This is a very useful feature, but as it just reads the UID it cannot be used for security related applications.
  The included CCID feature supports most common tags. But its use requires a host application supporting the PC/SC standard.
  Unfortunately the CDC based commands do not support state of the art tags like MIFARE DESFire or the NTAG family.
  It can be said that this module covers most aspects of a simplified RFID module, but there is still some room for improvement.

The summary above shows that there are multiple different approaches available on the market, but most of them are not very intuitive and therefore not really easy to use. To overcome this, the command set of the implemented firmware architecture, performs combined multiple actions. This allows to use the tags as wireless memory cards supporting most of the features. Another missing aspect of the modules mentioned before is the support of the connected tags. In the last years its field of application has grown. Therefore also this sector has to be included. It can also be seen that none of the modules above feature any NDEF support.

[85] Compare Jinmuyu Electronics Co. LTD, User Manual, JMY504C User's Manual, Revision 3.42, June 28, 2011
[86] Compare ubisys, 13.56 MHz RFID USB READER REFERENCE MANUAL

For cost reasons the use of integrated RFID controllers, like done for the identive module, is not feasible. The support of different, much cheaper, reader IC makes much more sense, as the appropriate and cheapest IC can be selected for the target application.

For changing, adapting or disabling commands, a good program structure is essential. Therefore I decided to use the following architecture:

```
typedef unsigned char (*CMD_Function) (char* Buffer, int Lenght, TYPEATAGStruct* Tag);

typedef struct
{
    char* Name;
    CMD_Function Cmd;
} CMDStruct;
```

Figure 34: Architecture for the commands

Each command is implemented using the same transfer parameters. Namely the receive buffer, the length of the received data and a pointer to the structure where all tag related information are stored. The return parameter is used to return an error code.

This approach is useful as they can be selected or deselected very easily. A struct, including a function pointer and a character pointer for the name, is used to manage the commands.

Therefore an array of this structure has to be implemented in the main program. Each used command is defined afterwards. As the name of the command can be set here, an application specific command set can be easily created and modifies.

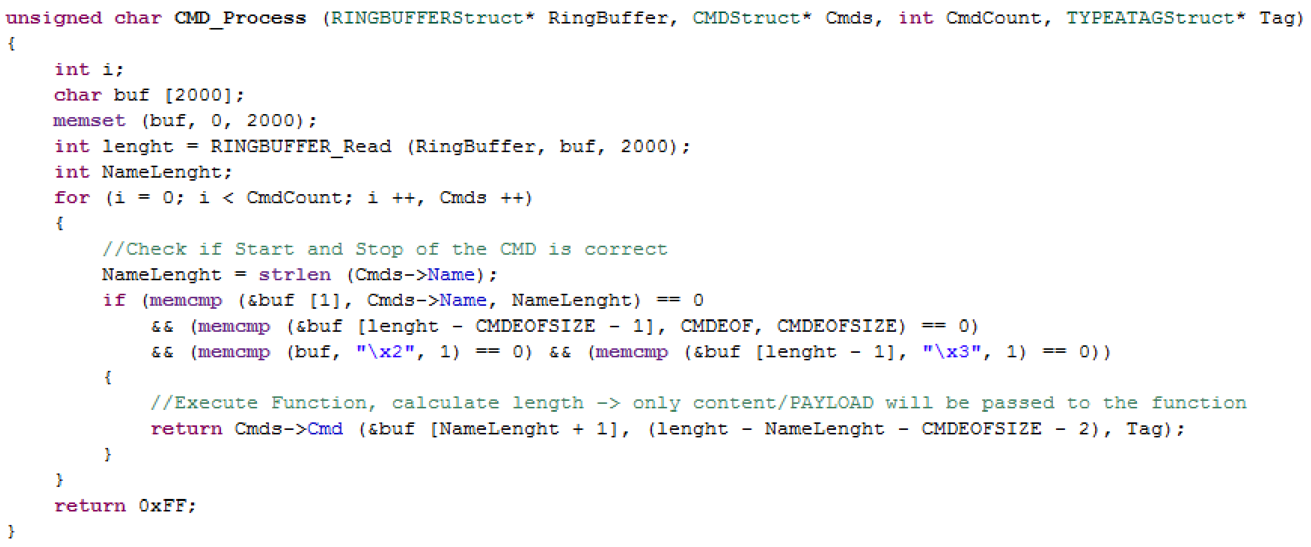
This approach is shown next:

```
CMDStruct CMDs [17];
int pos=0;
CMDs [pos].Name = "GI";
CMDs [pos].Cmd = CMD_GI;
pos ++;
```

Figure 35: Command structure used in the main program

As described in Figure 35 the link to a command is very simple. Only the function pointer and the name have to be set. The required space for the command struct is negligible.

This created pattern is also very comfortable for parsing the input and calling the appropriate command. The responsible function is shown next:



```
unsigned char CMD_Process (RINGBUFFERStruct* RingBuffer, CMDStruct* Cmds, int CmdCount, TYPEATAGStruct* Tag)
{
    int i;
    char buf [2000];
    memset (buf, 0, 2000);
    int lenght = RINGBUFFER_Read (RingBuffer, buf, 2000);
    int NameLenght;
    for (i = 0; i < CmdCount; i ++, Cmds ++)
    {
        //Check if Start and Stop of the CMD is correct
        NameLenght = strlen (Cmds->Name);
        if (memcmp (&buf [1], Cmds->Name, NameLenght) == 0
            && (memcmp (&buf [lenght - CMDEOFSIZE - 1], CMDEOF, CMDEOFSIZE) == 0)
            && (memcmp (buf, "\x2", 1) == 0) && (memcmp (&buf [lenght - 1], "\x3", 1) == 0))
        {
            //Execute Function, calculate length -> only content/PAYLOAD will be passed to the function
            return Cmds->Cmd (&buf [NameLenght + 1], (lenght - NameLenght - CMDEOFSIZE - 2), Tag);
        }
    }
    return 0xFF;
}
```

Figure 36: Function for processing and calling the appropriate command

As can be seen above the function requires the use of ring buffers. This functionality is given by the uc_hal, which depending on the selected interface, either fills the ring buffer with data received via USB or USART. Depending on the memory capabilities of the microcontroller, the size of this buffer has to be adapted. Additional transfer parameters are the command struct, its number of commands and a pointer to the tag structure.

After the ring buffer is read, a loop over the number of available commands is processed. One typical problem is that the start and stop characters are also used within the message. To overcome this, a sanity check is performed on the entire received frame. Therefore the prefix, start of the message and postfix is compared with predefined values.

If this matches, it is ensured that the structure and name of the command are correct. Else, the same is performed for the next command.

The call of the appropriate command function is done via the function pointer, which is stored in the command struct. It also allows to only transfer the payload to the command. This elegant way is easy to use and quickly adaptable. It also separates the used interface and command.

The structure and an example of such a command can be seen below:

Start of Frame	Command Name	Payload	End of Frame
\x2	WT	1,Test123\r\n\x2\x3	\r\n\x3

Figure 37: Structure of a command used for the firmware architecture

The example provided above might lead to a typical parsing problem. As the End of Frame is checked without touching the payload, it is not a problem for the used architecture.

The size of the available SRAM can be seen as a limitation. This can be overcome by using shorter commands.

As most commands have to send a response, send functions are provided. These are automatically selected according to the used interface using function pointers. The response format is defined as follows:

Start of Frame	Status code	End of Frame
\x2	OK	\r\n\x3

Figure 38: Response format for the implemented commands

As can be seen above, each frame starts with the STX (0x02) ASCII character and ends with the ETX (0x03).

Some of the implemented response codes are listed next:

Status code	Description
OK	Command executed properly
IP	Invalid Parameters are transmitted
PE	Protocol Error
MT	Multiple Tags are in the field – command not executed
NS	Not supported
AE	Authentication Error – probably the used key is wrong
…	…

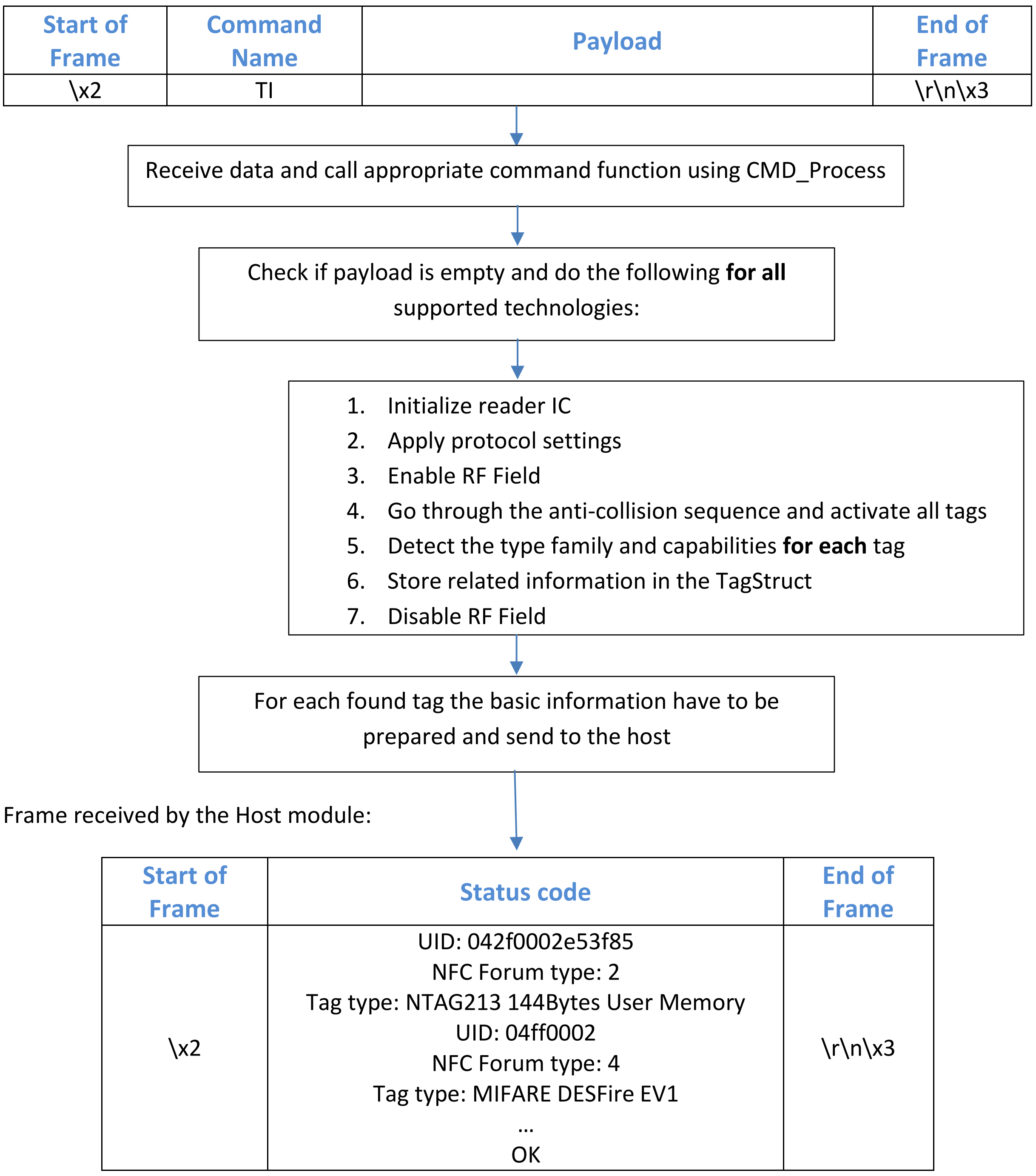
Figure 39: Overview of the response codes

These codes can be easily modified and extended. The handling of multiple tags is supported, but to ensure that the command is executed for the right tag, it either has to be specified, or just one tag is allowed to be used concurrently for that command.

As mentioned above, the goal of this architecture are commands that are simple to use. This can only be achieved by executing multiple tag dependent steps for one command. How complex this can be is shown by the example of the “Tag Info” command.

This command can be used to verify the type and name of all tags currently available in the RF field, including all supported technologies. Therefore the following simplified actions have to be performed.

Frame from Host to the module:



Start of Frame	Command Name	Payload	End of Frame
\x2	TI		\r\n\x3

Receive data and call appropriate command function using CMD_Process

Check if payload is empty and do the following **for all** supported technologies:

1. Initialize reader IC
2. Apply protocol settings
3. Enable RF Field
4. Go through the anti-collision sequence and activate all tags
5. Detect the type family and capabilities **for each** tag
6. Store related information in the TagStruct
7. Disable RF Field

For each found tag the basic information have to be prepared and send to the host

Frame received by the Host module:

Start of Frame	Status code	End of Frame
\x2	UID: 042f0002e53f85 NFC Forum type: 2 Tag type: NTAG213 144Bytes User Memory UID: 04ff0002 NFC Forum type: 4 Tag type: MIFARE DESFire EV1 … OK	\r\n\x3

Figure 40: Overview of the Tag Info command

The picture above gives an idea about the overall complexity and interaction. It can also be seen that the connection between the different layers is important. The involved layers differ concerning the used command.

Other commands like for reading or writing the tags are way more complex as also the different tag related commands and memory layouts must be considered. This enhanced complexity is the main reason why most available modules do not offer such a support.

The following variations and problems must be considered for more complex commands like reading or writing tags:

- Are multiple tags available and if they are how to deal with them
    - Cancel the command
    - Write to all
    - Write to one (first detected wins)
    - Select which tag shall be written to
- Technology used by the tag (which standard)
    - Switch through all standards
    - Let the user define which standard to use
- Type of the Tag (Sensor Tag, Connected Tag, Memory Tag)
- Memory layout (what are the sector and overall sizes, where are data regions, …)
    - Different sector size
    - Different memory regions
- Authentication (does the tag support or even require any authentication, which encryption algorithm is used)
    - To make this process more user-friendly, it is possible to specify the key within the command parameters.
    - As an alternative it is possible to preset a list of keys.
    - How to handle partially locked tags? Therefore the authentication must be redone for all sectors depending on the used tag.
    - Some tags features a counter which can be used to destroy itself if a wrong authentication key is used too often. So the number of retries must be configurable.
- Permission rights (are the memory regions accessible)
    - In case of NDEF the none physical locking behavior must be satisfied
- Which commands can/must be used (each tag defines other commands)
    - For increasing the read/write speed some tags provide fastread/fastwrite options which should be used if available
- In case of NDEF usage check if the tag is formatted properly / enough size available
- (optional) Verification (read back the written data to ensure correct behavior)
    - What shall be done if the tag was removed during the writing procedure

As it can be seen above a simple write command for example has to deal with a lot of different possibilities. Therefore the user has lots of possibilities to control the behavior. The command also maps the whole user memory (which can consist of multiple separate memory regions) to one linear accessible memory. It also allows to specify the start position of these commands with a byte offset. This makes it possible to use the memory of a tag like a file, which is not possible with any other known module. All the user has to do is sending a simple command like shown below:

Start of Frame	Command Name	Payload	End of Frame
\x2	RT	1,10	\r\n\x3

Figure 41: Example for reading from a memory tag

The user can easily specify the byte offset to start the reading procedure and the number of bytes to read.

A list of the most important simple commands is shown next:

Command	Description
RT	Reads the memory of a tag using the given offset and optional parameters like an authentication key.
WT	Writes the provided data to a tag. Options for setting an authentication key or byte offset are available.
WV	The Write Verify command automatically performs a read back and comparison of the written data
ET	Encrypts the given tag with a given key.
LT	The Lock Tag command enables the read and or write protection for the tag
OC	This command performs an originality check based on the ECDSA algorithm or a predefined list of Chinese clones.
TI	The Tag Info command returns the available information for the detected tags. This includes the name, tag type, UID and memory size.
TT	As for some applications only the name of the tag is used this command returns this information.
WO	This command allows to write data to the OTP memory of a tag. This is separated to avoid unintentional use.
…	…

Figure 42: Summary of important commands

The command set covers the most important aspects. In comparison to the commands of other modules, the implementation supports also the use of different encrypted tags. This can be used to overcome one of the most common security problems in the domain of RFID. The use of the chip serial number as a unique identifier. Lots of RFID based systems rely on the uniqueness of this identification number. As some tags provide the possibility of changing its UID, this approach is totally outdated. The simple use of encrypted tags can lead to a higher level of security without enhancing the effort.

In general, most tags provide a physical locking option, but the implementations are very different. One reason for that are the various memory layouts. This results in different positions of the bytes used for holding the locking information. Depending on the tag the locking can be performed for the whole tag, per sector or per byte.

The implemented originality check is also unique. It not only uses the described ECDSA algorithm with special read operations and a comparison with a list of known Chinese clones, it is also possible to differentiate them from original tags. Some of these clones that can be detected are:

- Fudan FM310
- Fudan FM284
- Giantec GT5640AM1D
- HuaHong HHIC2405

These tags are unlicensed copies of the famous MIFARE Classic tag. Which means that even the broken Crypto1, which is not public, has been copied.

Another part of the simple command set is related to the hardware and for example allows to control the LEDs or the buzzer. But also interface related settings can be defined here. An overview of these commands is shown next:

Command	Description
SB	This command allows to change the baud rate for the serial interface (USART or CDC).
SL	The sleep command allows the module to enter power saving mode. As this feature is not available for all microcontrollers it might not be supported for all platforms
SK	This command can be used to store a given key in the internal storage.
…	…

Figure 43: Hardware related commands implemented by the firmware architecture

The standardized HAL allows a simple implementation and extension of these commands. It can also be seen that the latency of this function is very low as there are no additional layers inbetween.

The limitation of these commands is given by the capability of the used hardware. Therefore the available features have to be selected for each application especially. This is provided by the use of the previous described CMDStruct.

Another aspect is the testing procedure of such a library. The implemented command pattern eased this, as test cases can be easily en- or disabled. For fast debugging and error detection each layer has a special value range for their error codes.

In recent years RFID was added in many devices. Some of them need the capability to use tags at different positions. In general this approach requires the use of multiple RFID modules. This increases the system cost and space and is not accessible in many applications.

A more cost efficient way is to reuse the microcontroller and attach multiple RFID reader ICs, which is shown next:

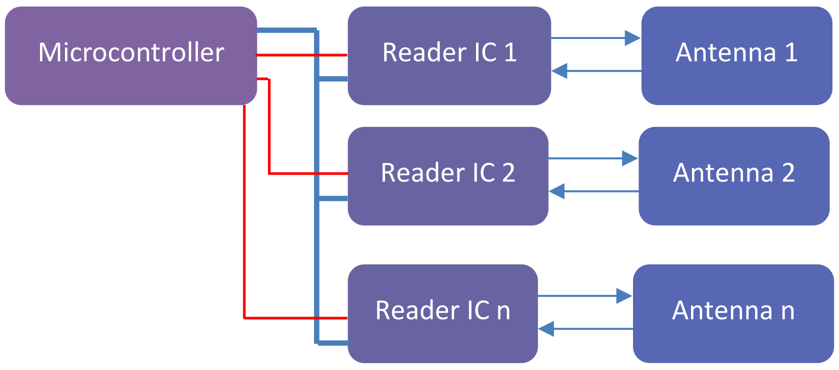


Figure 44: System topology using multiple reader ICs

The approach, shown in Figure 44, needs an adapted version of the firmware architecture as multiple reader ICs must be handled concurrently. In case of a SPI connection, additional chip select connections, which are marked red are required. The use of multiple reader ICs, usually the most costly component also increases the form factor size and costs of such a module.

Therefore the idea of a switchable multiple antenna RFID module was born.
It reduces the costs and can be easily integrated. An overview of this approach is shown below.

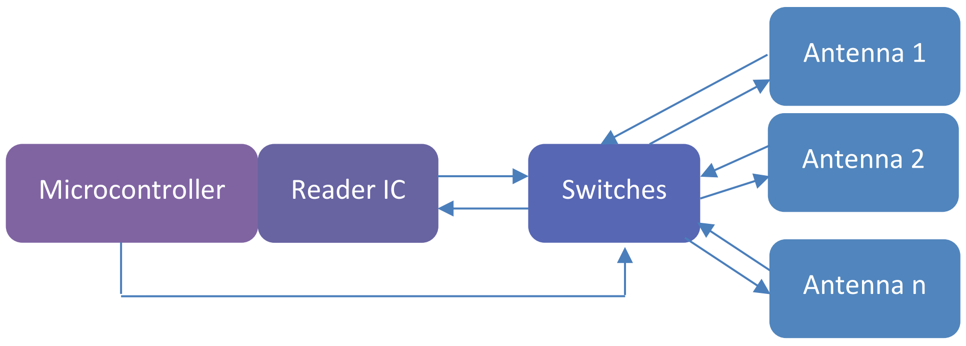


Figure 45: Topology of a multi antenna RFID module

Only a few modifications are needed to support this feature:

- Antenna switches are needed, which can be controlled by the host
- Each command has to be extended to either
  - Specify which antenna shall be used
  - Or performed the same command for all antennas consecutively

The simple structure of the commands makes this modification very easy. The only disadvantage of this solution is, that the switching frequency is limited by:

- The number of antennas
- The number of used standards
- The data transfer size per tag

To achieve a good performance the number of attached antennas is limited to four. This approach is a trade-off between performance, cost and design flexibility. Even the form factor is smaller as the antenna switches typically requires less space than additional reader ICs.

Such a solution allows the integration in even more applications. An example of a device using this feature is shown later in the thesis.

In addition to these simple commands it is also possible to exchange standard frames via this command set. Therefore the exchange command is implemented as follows.

Start of Frame	Command Name	Payload	End of Frame
\x2	EX	1,\x30\x22\x02	\r\n\x3

Figure 46: Example for exchanging an ISO14443 A frame

The first parameter of the example shown above, defines the used standard. The data is sent to tag afterwards. It must be considered that most standards provide different framing and CRC options. As the number of variations is huge, the advanced user has to prepare the frames correctly. With this feature almost all tags are covered by this framework.

Application Layer (AL)

This topmost layer covers application related features and fulfills the aspect of this framework. The correlation between all described layers is shown in the following figure:

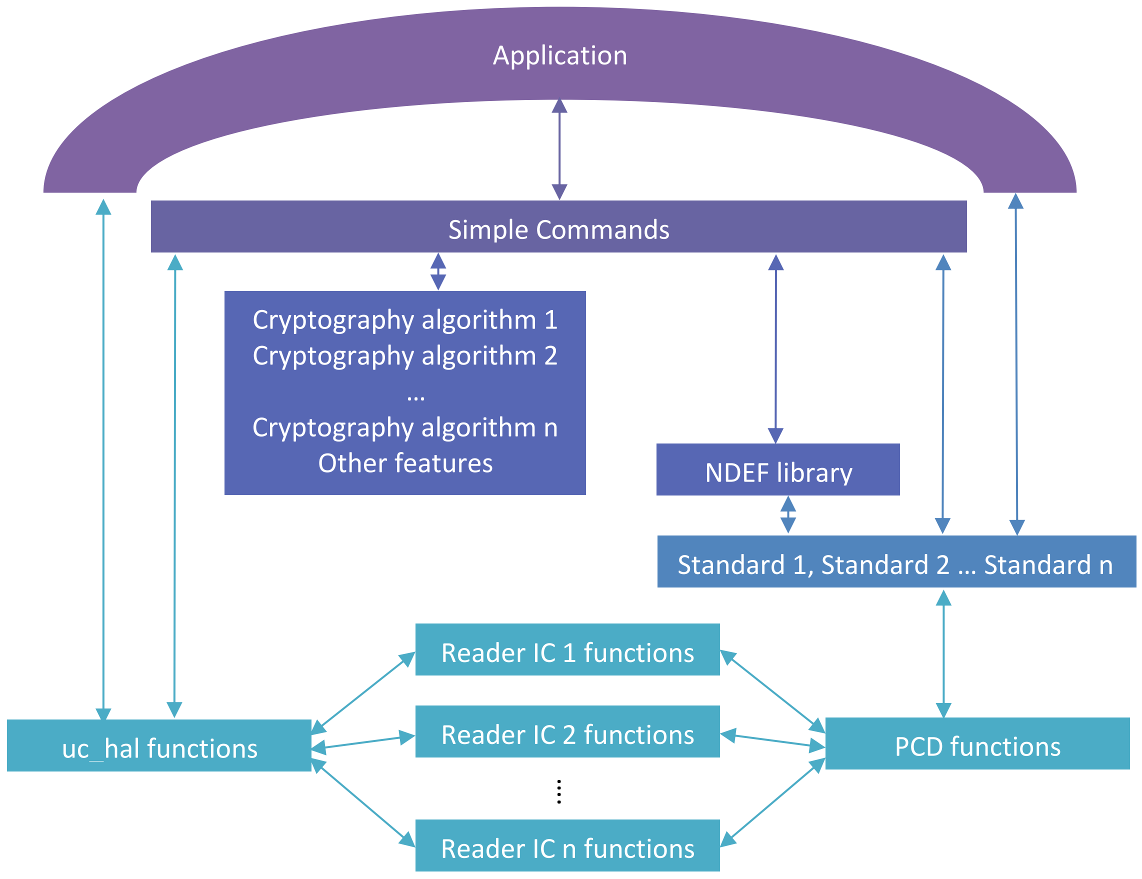


Figure 47: Overview of the overall firmware architecture

As shown above, the whole functionality of the underlying layers is available here. The main parts are the implemented command set, the different standards and the hardware related functions.

In general complex microcontroller related firmware is hard to debug. Because multiple peripherals like a watchdog or even the USB communication are strongly time-dependent. This layer is implemented for providing and coding snippets for target applications. This reduces the development effort, as lots of typically application related functions can be reused. Another advantage of these snippets is, that the verification process has to be done once and not for each application.

The defined interface structure also allows to use the same application code for different platforms. Therefore only the HAL (and maybe the IDE or compiler) has to be changed. This means that an existing application, can be reused easily.

This layer also defines the size of the compiled application, because most compilers only include the used functions in the resulting binary. Different compiling options which can be used for optimizing the resulting code are also very common. Some of these options exacerbate the debugging procedure, which results in the use of a debug and a release build.

One example for such an implemented and verified code snippet is the firmware protection mechanism. In state of the art applications a firmware upgrade via USB or another interface is a must. This leads to the problem that a compiled version of the binary has to be given to the customer.

To prevent other companies from reusing this firmware for their self-made/copy hardware, a solution had to be found. As the capabilities of microcontroller vary a lot, it is hard to find a generic way. A typical approach is the use of a bootloader. As this strongly hardware related software cannot be reused easily for multiple platforms, another way must be found.

A more generic solution is the use of a signature. Therefore a special application which generates an ECDSA signature was written. This signature is stored in the internal storage at a specific location where it will not be overwritten during the flash programing procedure. Only if this signature exists and is valid, the firmware executes.

The signature generation and verification procedure is shown below:

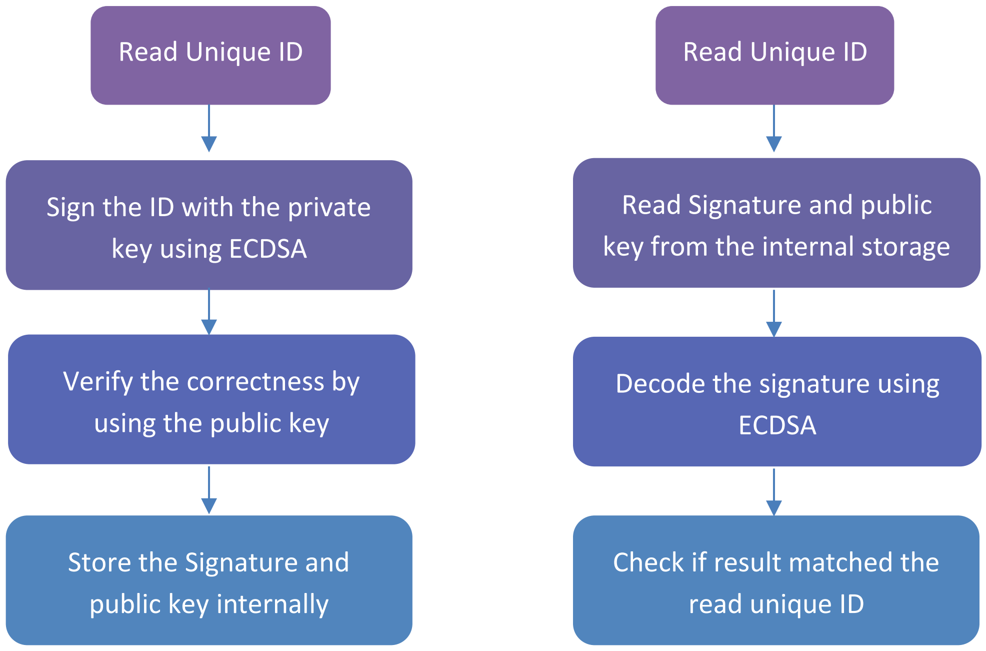

Read Unique ID
Sign the ID with the private key using ECDSA
Verify the correctness by using the public key
Store the Signature and public key internally
Read Unique ID
Read Signature and public key from the internal storage
Decode the signature using ECDSA
Check if result matched the read unique ID

Figure 48: Procedure for generating and verifying the signature

The advantage of the idea shown above, is the easy implementation. All required functions, like the generation of a signature, are available on underlying layers. But for using these features the used microcontroller has to feature a unique serial number, which is used for the signature generation. Without this uniqueness the generated signature could be copied and the protection mechanism would fail.

Additionally, this approach requires a special software, which is used during production, for creating and writing this signature. This additional effort is also needed for the bootloader alternative.

It must also be considered that the verification procedure during startup takes some time. For the hardware module originality check, about 200 ms are needed.

The application layer also provides USB related snippets. These allow to easily use the following features:

- Descriptor
  In general each USB device consists of unique descriptors. These define the name, capabilities and much more. As they are independent from the hardware, easy to modify descriptors are implemented. For changing the product, only some adaptions like changing the product name and ID are required.
  The implemented descriptors allows the use of the following configurations
  - HID Keyboard only
  - HID Keyboard and CDC (composite device)
  - HID Keyboard and CCID (composite device)
  - CDC only
  - CCID only

  A combination of CCID and CDC is not available, as the required number of endpoints is higher than available on most microcontroller.
- CDC
  The snippet provides simple read and write functions, which use a ring buffer. These non-blocking functions allow a fast and easy use of this interface. As the same technique is also implemented for the USART functions, the change of the interface is simple.
- HID Keyboard
  The implementation allows to write a string using the keyboard emulation. Therefore a default character mapping is defined. As mentioned in the beginning, the language of a USB keyboard cannot be defined in general and it does not make sense to store all different language mappings internally. But the implemented English layout should fit for most common applications.
- CCID
  The basic CCID stack is also included and provided. This allows the communication with the host system via the CCID protocol.
  In general it makes no sense to feature simple commands and CCID at the same device. The use of CCID needs lots of special knowledge and therefore simple commands make no sense. Either simple commands or CCID are used depending on the field of application.

In general, this layer is, as can be extracted from its name, application specific. But the implemented framework offers additional features for reducing the effort of the whole application development.

The implementation and verification is not possible without the use of real hardware. Therefore the used components are described next.

4. Hardware

Two different kinds of hardware were used for this implementation, the development platform and final products. The former one is needed for the developing and debugging process. Therefore this architecture has to be very flexible and easy to use with different kinds of microcontrollers and reader ICs.

After implementing and testing the framework, it was used in many different products which are described later.

Development Platform

For implementing and verifying the framework, hardware is needed. NXP offers with the LPC Xpresso Boards a flexible platform for a huge product range. One advantage is the common LPC Xpresso expansion connector, which allows an easy integration of external hardware like the reader ICs.

The development platform in general consists of a microcontroller and a reader part. As the demo boards for the RFID reader also feature this LPC Xpresso connector, an easy way for combining different systems was found.

Microcontroller

At the moment two different microcontroller are supported:

- LPC111x[87]
  - Low cost low power
  - 32-Bit ARM Cortex M0
  - Up to 64k flash
  - Up to 8k SRAM
  - 12MHz internal oscillator
  - Up to 50MHz
  - IIC
  - 2x SPI
  - USART
- LPC11u6x[88] with the LPC Xpresso boards OM13035/ OM13087
  - Med power
  - 32-Bit ARM Cortex M0+
  - Up to 256 k flash
  - Up to 32 k SRAM
  - 4 k EEPROM
  - 12 MHz internal oscillator
  - Up to 50 MHz
  - IIC
  - 2x SPI
  - USART
  - USB with several predefined ROM driver

[87] Compare NXP, Datasheet, LPC1110/11/12/13/14/15 Datasheet, Rev. 9.2, March 26, 2014
[88] Compare NXP, Datasheet, LPC11u6x Datasheet, Rev. 1.3, September 7, 2016

The LPC Xpresso boards OM13035 and OM13087 feature the LPC1115, which is the high-end version of the LPC11x family. As the whole family is pin compatible, different members of the same family could be used with the same hardware.

The following figure shows the used OM13035 demo board:

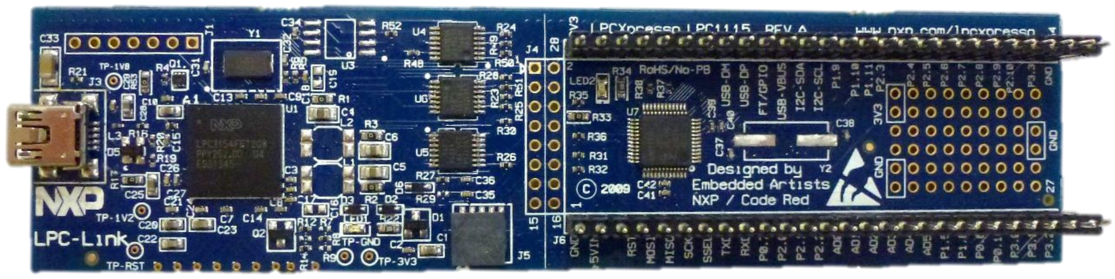


Figure 49: OM13035 demo board

As can be seen, this LPC Xpresso board not only consists of the microcontroller itself, but also covers the aspect of an easy debugging interface. Therefore the LPC-Link, which is a hardware debugger, is also part of the demo board.

The LPC11u6x family is covered by the OM13058 LPC Xpresso board. It features the LPC11u68, which is also the high-end version of the family.

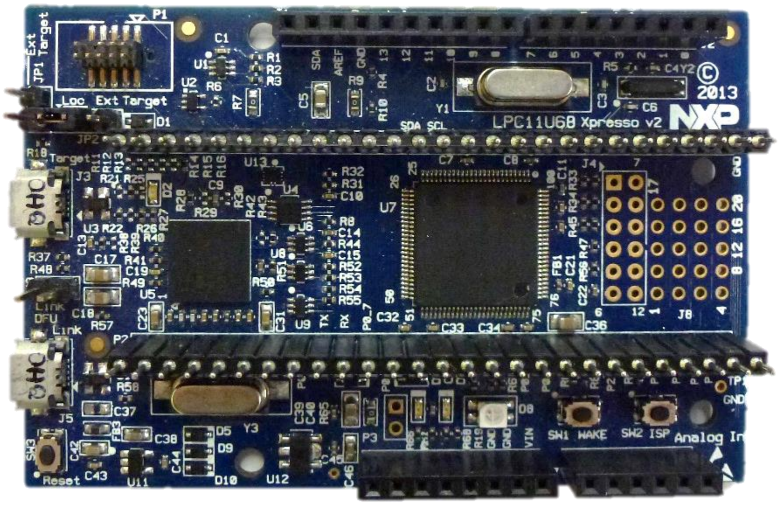


Figure 50: OM13058 demo board

As can be seen above, this demo board also features the LPC-Link hardware debugger. The required external wiring, for using the USB peripheral, is also available.

As the LPC Xpresso extension connector is the same for all LPC Xpresso Boards, this development platform can be easily extended with newer or more specific microcontroller. NXP also offers lots of additional material, like the schematics, the layout or even code examples for their demo boards.

Reader ICs

Several reader ICs platforms are offered by NXP. In this case, the PN512 and RC663 platform and its derivatives are used. Each platform consists of multiple family members. These differ in their integrated features and capabilities.

The PNEV512 demo board from NXP features the PN512 and is shown next

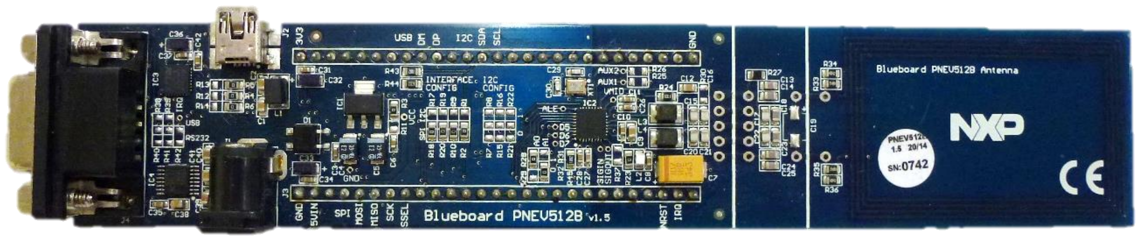


Figure 51: PNEV512 demo board

This demo board features the LPC Xpresso extension connector and allows to easily mount and unmount this board compared to other microcontroller boards.

For supporting all derivatives of this family, multiple boards, assembled with different reader ICs, are used. This allows the verification of the implemented auto detect algorithm and other features.

The CLEV663B demo board is used to cover the RC663 family. As described before, multiple boards assembled with different reader ICs are used.

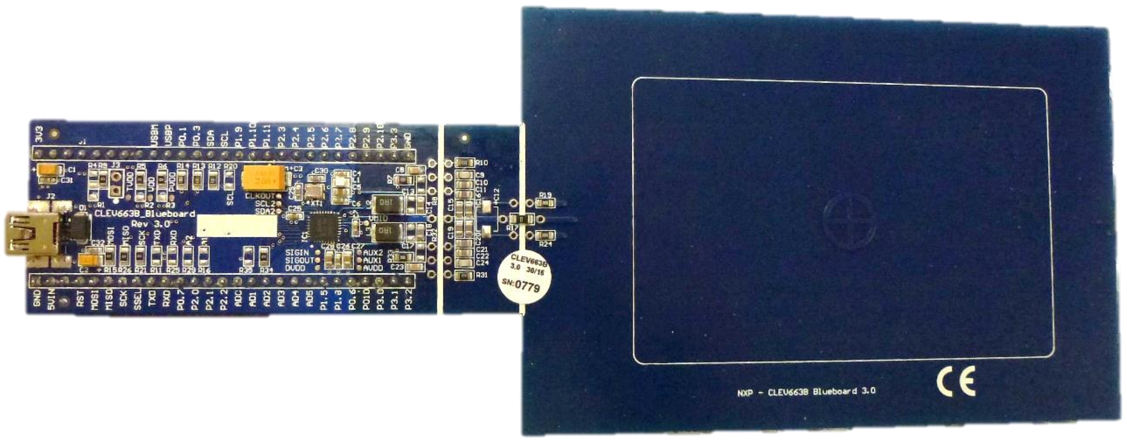


Figure 52: CLEV663B demo board

The CLEV663B board also features the LPC Xpresso extension connector.

Both boards support the use of different interfaces like IIC, SPI or even USART. As described at the beginning, SPI is used.

It must be considered, that the pinning of both boards is different. This leads to the problem, that either the use of the PNEV512 or the CLEV663B is possible. To overcome this issue, all CLEV663B boards are modified. This modification maps all pins to the same positions, as defined for the PNEV512 boards.

This modification is shown below:

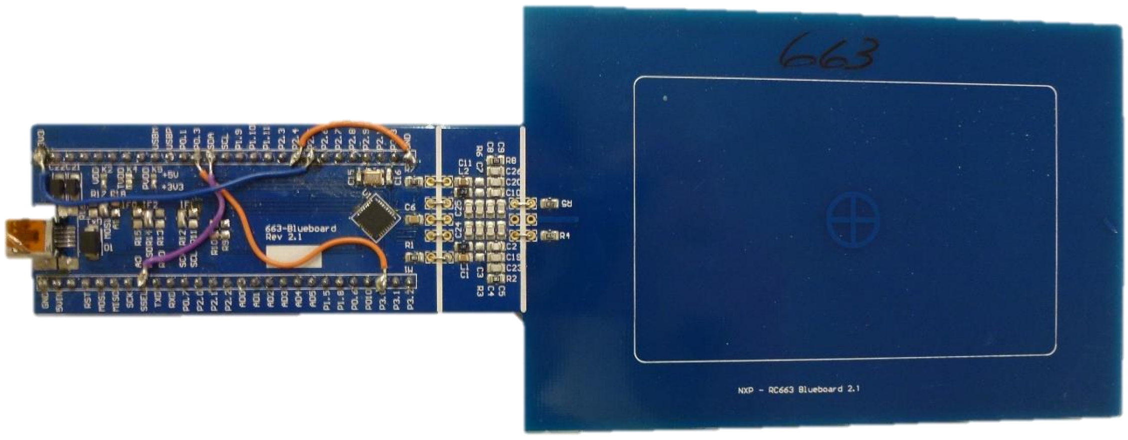


Figure 53: Modified version of the CLEV663B

The modification, shown above, ensures the correct mapping of the following pins:

- +5V
- GND
- CS
- MISO
- MOSI
- SCK
- nRESET/PDOWN

This completes the used hardware platform. In general also parts from other manufacturers can be easily used, as only the mentioned Pins have to be connected properly. It should be considered, that the cable length has to be kept as short as possible, due the fact that the SPI frequency is 10 MHz.

For enhancing the debug functionality the LPC-Link2[89] debug probe in combination with the LAPTOOL[90] from Embedded Artists is used. This not only allows to program and debug the devices, also the following features are included:

- 11 channel digital signal generator
- 11 channel logic analyzer
- 2 channel oscilloscope
- 2 channel analog signal generator

The included windows based software also allows to easily decode common communication standards like IIC, SPI or USART.

[89] Compare NXP, OM13054: LPC-Link2, (Accessed: 28.1.2017): http://www.nxp.com/products/microcontrollers-and-processors/arm-processors/lpc-cortex-m-mcus/lpc3000-arm9-mpus/lpc-link2:OM13054
[90] Compare Embedded Artists, LABTOOL, (Accessed: 28.1.2017): http://www.embeddedartists.com/products/app/labtool.php

The whole development platform with all components is shown below:

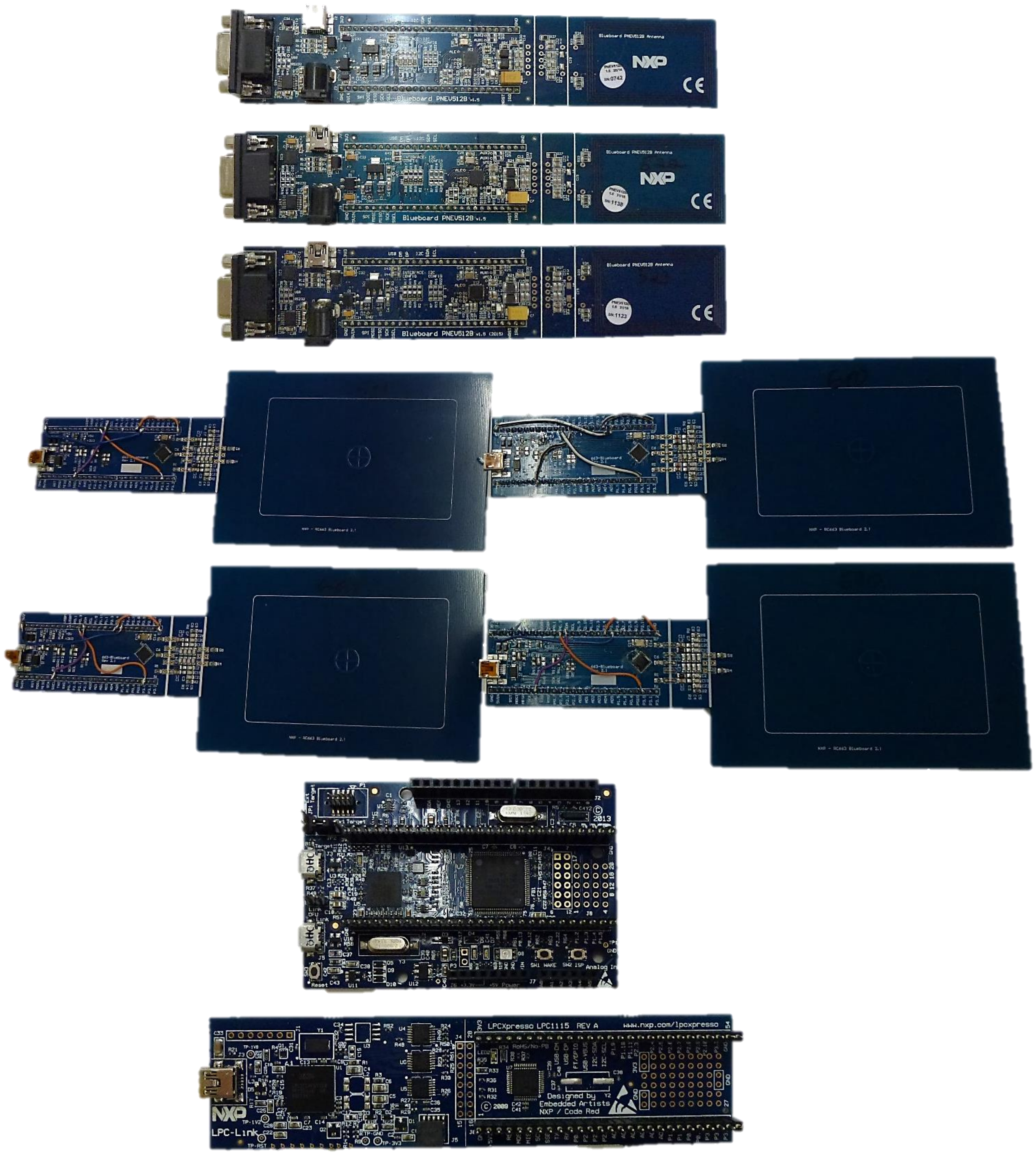


Figure 54: Used development platform

It consists of several PNEV512 and CLEV663B boards as well as the OM13058 and OM13035 LPC Xpresso Boards. Also the described combination of LPC-Link2 and LABTOOL is shown.

Implemented Products

The implemented firmware architecture is not only used on the development platform. The following products are also realized using this implementation.

Sangoma MSMA Multi Standard Multi -Antenna

This product is realized using the already described approach for using multiple antennas with only one reader IC. The product implements the LPC11u68 microcontroller and the RC663 reader IC. Antenna switches allow the use of up to four antennas. Multiple different antennas are available and can be used with this product. This allows new applications where tags must be used on different locations.

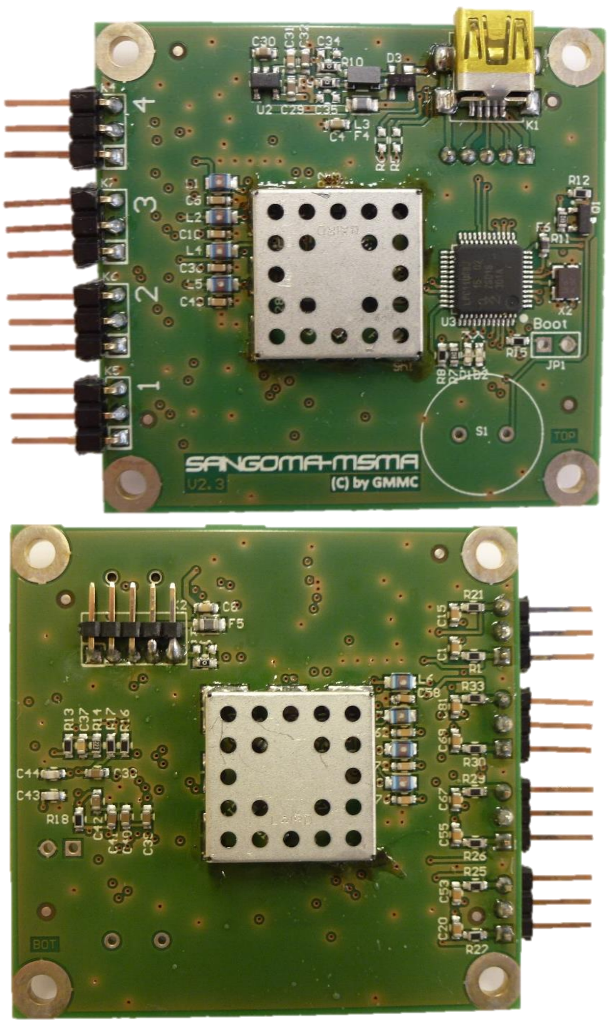


Figure 55: Top and bottom view of the Sangoma MultiAntenna module

As can be seen in Figure 55, the antennas can be easily mounted or unmounted. Twisted pair cables are used to connect the module with different antennas:

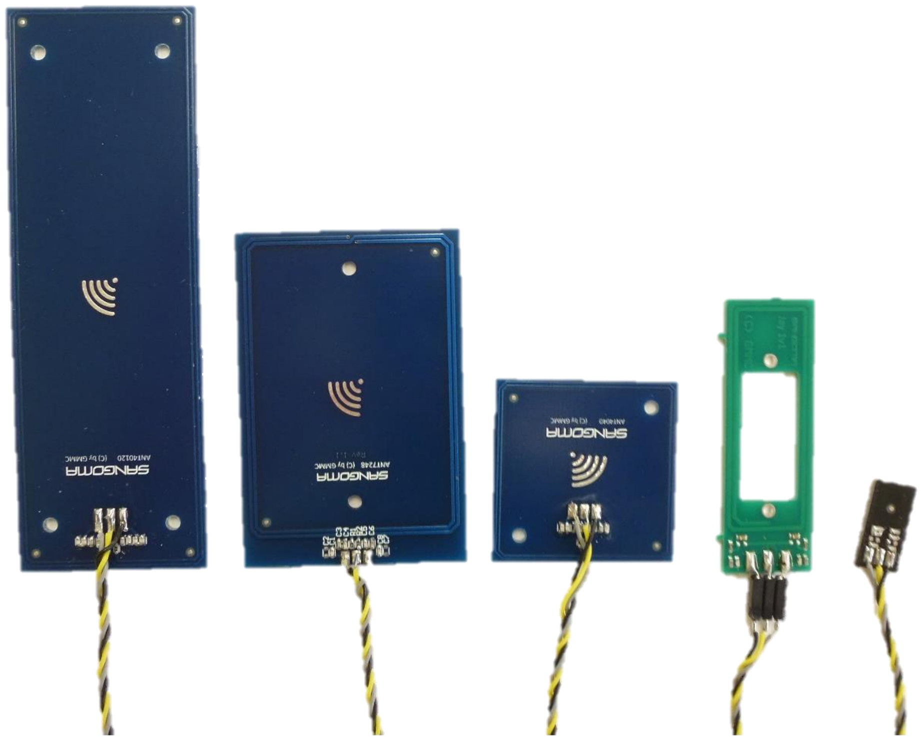


Figure 56: Different available antennas for the MutiAntenna module

These different form factors enhance the field of application.

The module is powered and connected to the host via USB. This allows an easy integration. The commands are sent to the module over the USB CDC interface. They also allow to specify the antenna number for each command.

Such a module is unique on the market because it combines the easy to use approach with a groundbreaking concept for using multiple antennas.

This CE and FCC certified module is already in use in an industrial application.

Sangoma NFC-Wizard

This easy to use desktop reader offers all features of this framework. Different form factors and configuration levels are available. In general, it is based on the LPC1115 low cost microcontroller in combination with the PN512 reader IC. For covering different host interfaces the following interface options are available:

- USB
- RS232
- USART (3.3 V level)

It also features LEDs and a Buzzer. The following figure shows the Reader:

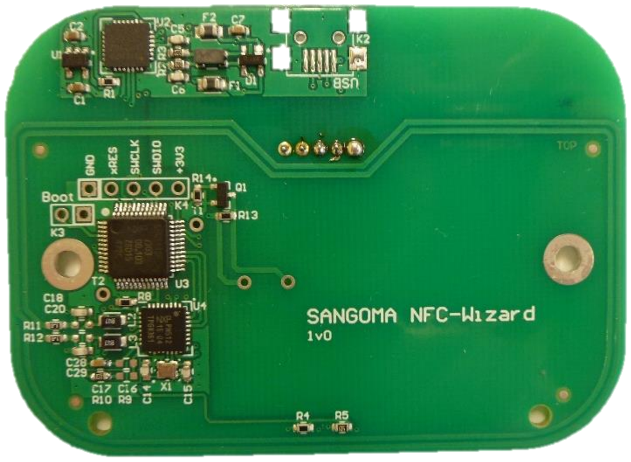


Figure 57: Topview of the Sangoma NFC-Wizard

As can be seen above, a CP2102 is used for converting the USART signal of the LPC1115 to USB. A host device can be either connected via the Mini-USB connector or over a specific USB cable.

This reader is also available with a case, which is shown below:

Figure 58: Sangoma NFC-Wizard with case

Beside this configuration also the following OEM versions are available.

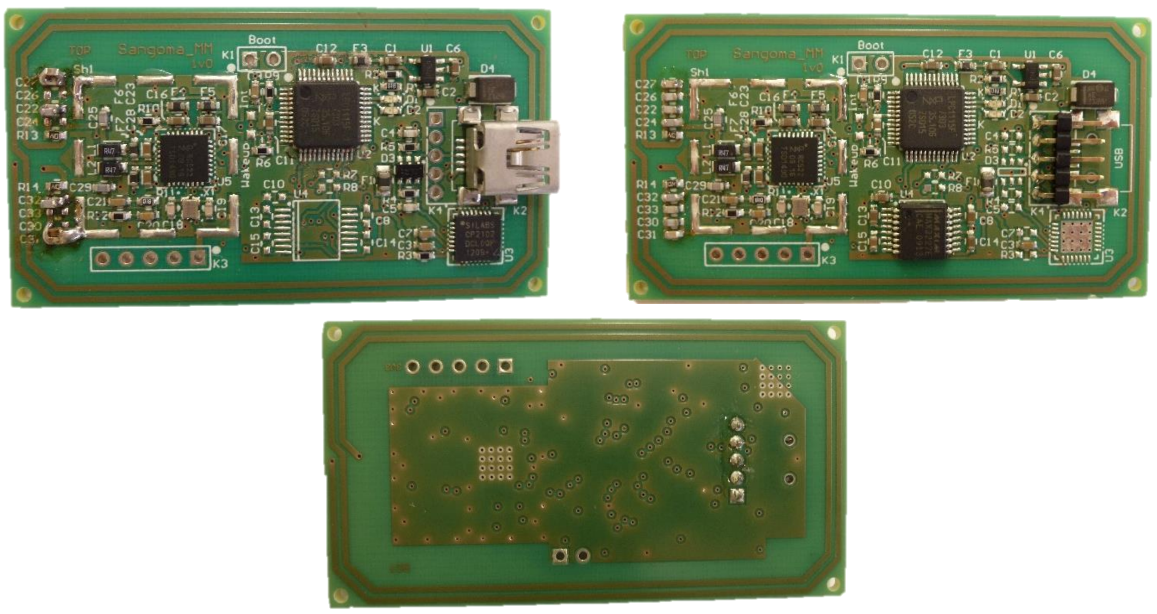


Figure 59: Sangoma Mini with USB and RS232 option

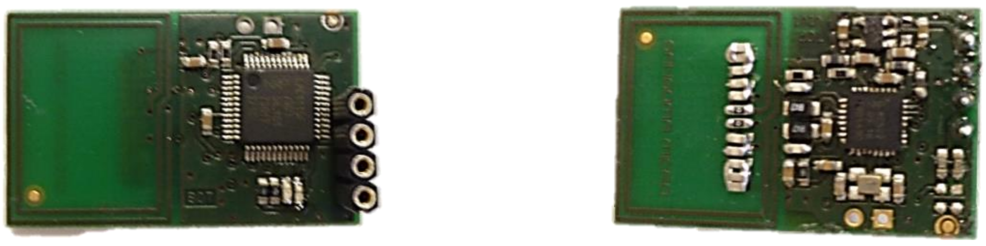


Figure 60: Top and Bottom view of the Sangoma Micro

As can be seen above, these products can be easily integrated into most existing applications.

Beside this, also a LPC11u68 version is available. This provides additional features like a RGB LED.

All modules feature the simple command set with different capabilities. As the SANGOMA Mini features the same dimension as former Identiv modules, they can be used as a replacement.

Sangoma DocTrack

For applications like document tracking in libraries, this reader can be used. It provides a huge antenna which has a read/write range of up to 25 cm. LEDs around the reader allow to signalize current behavior. The LPC11u68 in combination with the RC663 is used within this product.

This reader is also ideal for RFID based document scanning. Its implemented HID-Keyboard emulation can be used for typing, the UID or special parts from the user memory.

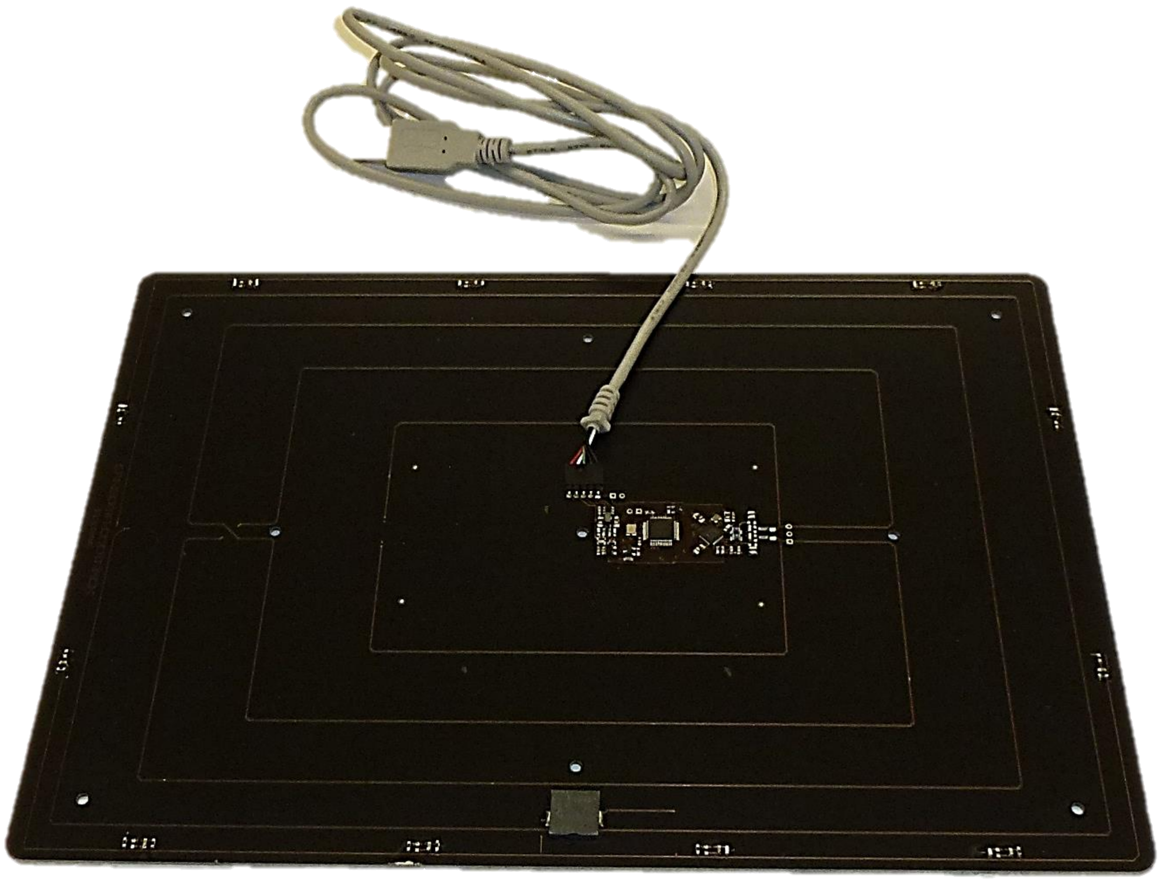


Figure 61: Sangoma DocTrack

The implemented firmware offers lots of possibilities. The implemented anti-collision allows to easily use multiple tags concurrently. The simple commands also feature a fast support to configure the used tags.

Bicycle Demo

This application was especially implemented for NXP. The goal is to show the capabilities of the NTAG-I2C connected tag family in a difficult environment. Therefore the position as well as the force on the pedals have to be monitored on the chain wheel. This data is transferred to the reader using the NTAG I2C connected tag. The reader itself is mounted on the bicycle frame, processes these data and sends them to a tablet via Bluetooth.

The difficulties for that application are the reaction time of the system as well as the power consumption of the monitoring module. As the bicycle is made of steel, the antenna matching is very important. It must be considered that the measurement of the force requires strain gauge strips connected to a bridge and a high performance ADC. This all requires much energy which has to be provided by the NTAG I2C, which is powered by the HF field.

The whole system consists of three parts:

Tag

This part of the system is mounted on the chain wheel and includes the low power microcontroller LPC812, a high precision ADC AD7780, an accelerometer LIS2DH12 and the NTAG I2C.

To minimize noise on the power supply lines, lots of capacitors are used. The PCB is shown next:

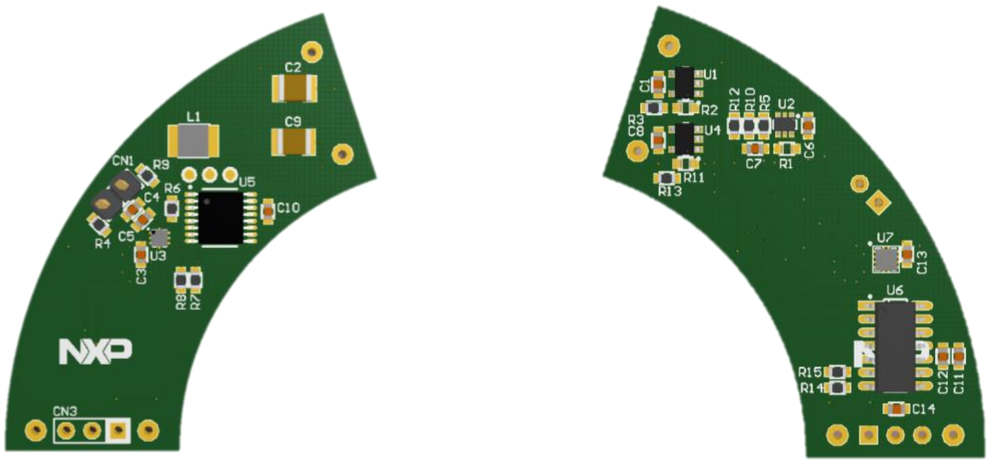


Figure 62: Top and bottom view of the Tag part

The Antenna is realized putting a wire on a Plexiglas circle. The strain gauge strips are glues on the crank using a special glue. The connection is done using thin wires. The assembled chain wheel is shown next:

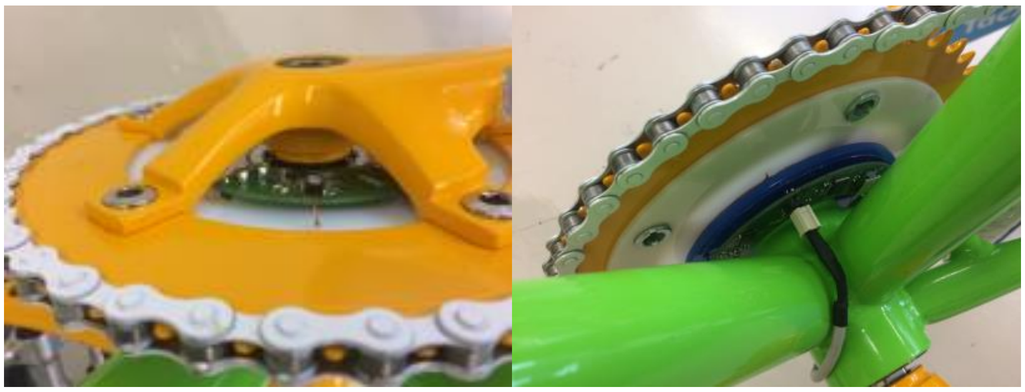


Figure 63: Chain wheel with antenna and tag PCB / reader mounted on frame

Reader

The reader itself consist of an LPC11u68 microcontroller a RC663 reader IC and a Bluetooth module. A power bank, which is mounted under the saddle, is used for powering the device. The module is shown next:

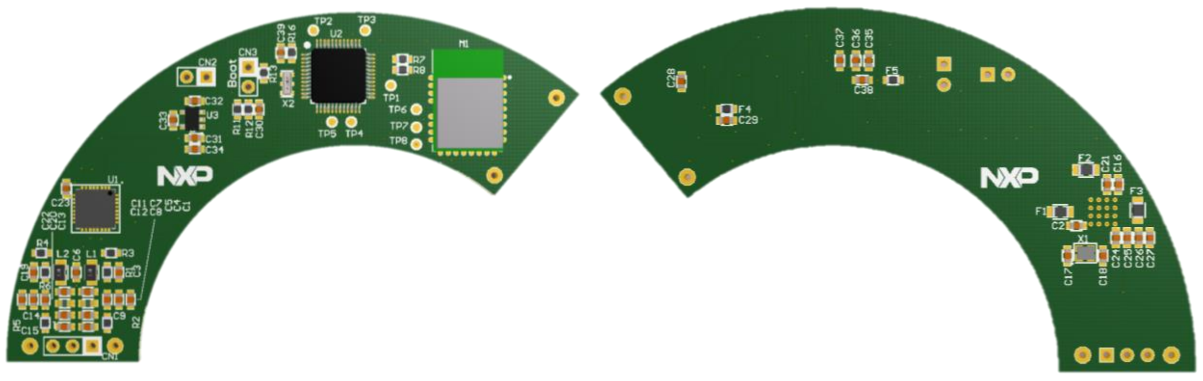


Figure 64: Top and bottom view of the reader part

The antenna, like on the tag side, is implemented using Plexiglas and wires. The whole reader part is shown next:

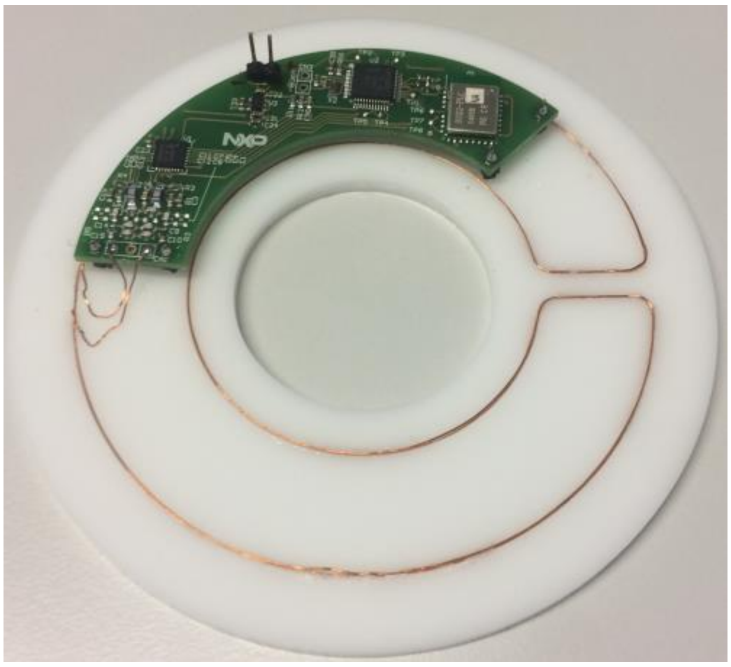


Figure 65: Reader PCB with used antenna

As can be seen, this module can easily be mounted on the frame of the bicycle.

The firmware is based on this architecture, which allows a fast and easy implementation. USART is used for the host interface, as the Bluetooth module acts as a transparent converter.

Tablet

For illustrating the monitored data an android application was used.

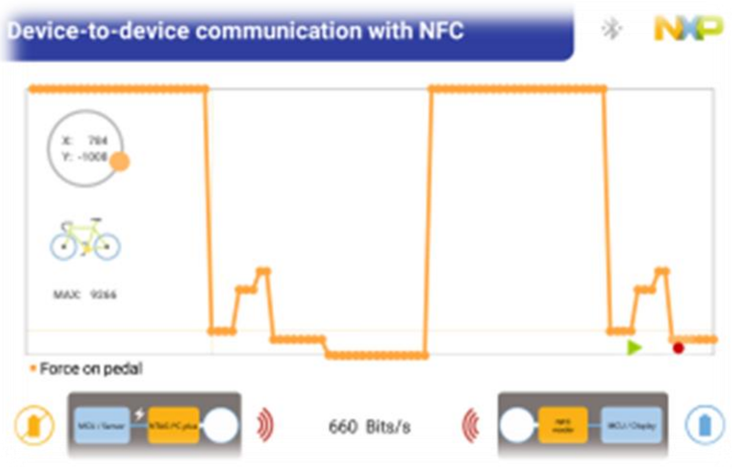


Figure 66: Screenshot of the Android App

As can be seen in the figure above, a chart is used to display the force. The position of the pedals is shown in a circle.

This android based application was provided by Martin Hollerweger from NXP.

A magnetic bicycle trainer is added to the whole system to simulate different resistances. The whole application is shown next:

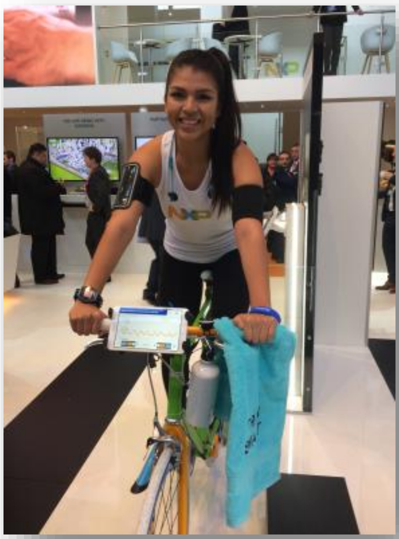


Figure 67: Picture of the whole application at the Electronica 2016

This demo was also shown on the Electronica 2016 in Munich.

5. Conclusion

This work shows the different types of RFID and their fields of applications. The topic is very complex, which aggravates the easy use in applications.

Therefore state of the art approaches of RFID libraries were analyzed. Based on this information a firmware architecture was created to cover and extend these typical approaches.

Additional implemented features are:

- Support for different microcontroller architectures
- Adding cryptographic algorithms
- Covering the NDEF format
- Implementing most common available tags
- Originality check

In contrast to available libraries the available command set not only implements the commands defined in the standards. The highly abstracted command set performs multiple commands consecutively to enhance the usability.

The overall size of the framework is strongly application specific. This makes a comparison very hard. The reader library from NXP needs about 12 kBytes of flash memory in a scaled version.[91] The implemented firmware takes about 15 kBytes (compiled with optimization for size). Additional features like the cryptographic architecture need more memory.

This results in the fact that even simple microcontroller can be used for this architecture.

The used development platform offers a simple changeable development platform, which covers lots of different components that can be combined very easily.

The products based on the created architecture feature the easy to use approach and are available in different form factors and capabilities.

This work shows the need for simple modules to overcome the complexity of the RFID technology and its different standards. It also provides an easy to use solution which covers these aspects.

In the future the UHF and the LF RFID standards could be included in this library to also cover these RFID technologies.

[91] Compare NXP, Application Note, AN11342, How to Scale Down the NXP Reader Library, Rev. 1.0, March 11, 2013

6. Bibliography

[1]	Harvey Lehpamer Ef. D, RFID Design Principles, Second Edition, Artech House Books, 2012, ISBN-13: 978-1608074709
[2, 43]	Dipl.-Ing. Michael Ganzera, lecture "Identification and System Integration", Campus 02, 2016
[3]	Lava Computer MFG Inc., RS232: Serial Ports, June 10, 2002, (Accessed: 27.1.2017): http://lpvo.fe.uni-lj.si/fileadmin/files/Izobrazevanje/OME/rs_232_serial_ports.pdf
[4]	Dallas Semiconductor, Application Note, 83, Fundamentals of RS–232 Serial Communications, March 9, 1998
[5, 6]	Silicon Laboratories Inc., Application Note, AN0059, UART Flow Control, September 16, 2013
[7]	Chuck Farrow, Texas Instruments, Application Report, SLAA215, Automatic Baud Rate Detection on the MSP430, October 2004
[8, 10, 11, 13, 15, 16, 17]	Craig Peacock, Beyond Logic, USB in a NutShell, 2007, Beyond Logic
[9]	Terry Moore, MCCI, USB 3.0 Technical Overview, October 8, 2009
[12, 14, 18]	Universal Serial Bus Specification, Revision 2.0, April 27, 2000
[19, 20, 21, 23]	Universal Serial Bus Device Class: Smart Card, CCID, Specification for Integrated Circuit(s) Cards Interface Devices, Revision 1.1, April 22, 2005
[22]	NXP, User Manual, UM10915, Revision 1.0, March 9, 2016
[24, 26]	Universal Serial Bus Class Definition for Communication Devices, Version 1.1, January 19, 1999
[25, 27]	Silicon Laboratories Inc., Application Note, AN758, Implementing USB Communication Device Class (CDC) on SiM3U1xx MCUs, Revision 0.1, March 2013
[28, 30, 31, 32, 33, 35, 36]	Universal Serial Bus Device Class Definition for Human Interface Devices (HID), Version 1.11, June 27, 2001
[29]	Silicon Laboratories Inc., Application Note, AN249, Human Interface Device Tutorial, Revision 0.5, March 2011
[34]	IDAutomation, Programming Manual, SC7USB 2D
[37]	Universal Serial Bus HID Usage Tables, Version 1.12, October 28, 2004
[39, 41]	NXP, Datasheet, PN7462, Revision 3.3, December 21, 2016

[38] NXP, User Manual, UM10663, NXP Reader Library User Manual based on CLRC663 and PN512 Blueboard Reader projects, Revision 1.2, July 24, 2013

[40] Austriamicrosystems, FS_AS39230 (Accessed: 27.1.2017): http://ams.com/eng/content/view/download/382456

[42] NXP, Application Note, AN78010, Revision 1.0, November 2002

[44] NXP, Application Note, AN11535, Measurement and tuning of a NFC and Reader IC antenna with a MiniVNA, Revision 1.1, November 3, 2014

[45, 50, 52] Pete Sorrells, Microchip Technology Inc., Application Note, AN680, Passive RFID Basics, 1998

[46, 49, 51] N. Vlajic, Analog Transmission of Digital Data: ASK, FSK, PSK, QAM, Fall 2010, Accessed: 27.1.2017): https://web.stanford.edu/class/ee102b/contents/DigitalModulation.pdf

[47, 57, 58] Atmel, Application Note, Requirements of ISO/IEC 14443 Type B Proximity Contactless Identification Cards, Rev. 2056B-RFID, November 2005

[48] Nicolas Cordier, Austriamicrosystems, Technical Arcticle, How new 'boostedNFC' technology enables mobile phones and wearable devices to emulate contactless cards reliably

[53] Henryk Plötz and Karsten Nohl, Legic Prime: Obscurity in Depth, December 28, 2009

[54] ISO/IEC 14443-2, Identification cards — Contactless integrated circuit(s) cards — Proximity cards — Part 2: Radio frequency power and signal interface, July 2001

[55, 56] ISO/IEC 14443-3, Identification cards — Contactless integrated circuit(s) cards — Proximity cards — Part 3: Initialization and anticollision, Novembre 20, 2008

[59] ISO/IEC 14443-4, Identification cards — Contactless integrated circuit(s) cards — Proximity cards — Part 4: Transmission protocol, March 19, 2007

[60] ISO/IEC 15693-1, Identification cards — Contactless integrated circuit(s) cards — Vicinity Integrated Circuit(s) Card Part 1: Physical characteristics

[61, 62] ISO/IEC 15693-2, Identification cards — Contactless integrated circuit(s) cards — Vicinity cards Part 2: Air interface an initialization

[63, 64, 65] SONY, FeliCa Card User's Manual Excerpted Edition, Version 2.01 No. M617-E02-01

[66] ISO/IEC 18000-3, Information technology — Radio frequency identification for item management — Part 3: Parameters for air interface communications at 13,56 MHz, 2010

[67] NXP, Datasheet, SL2S1412; SL2S1512; SL2S1612 ICODE ILT-M, Rev. 3.2, October 8, 2013

[68] NXP, Application Note, AN11402, How to implement the ICODE ILT anti-collision, Rev. 1.0, October 23, 2013

[69] Márcio Almeida, Hacking Mifare ClassicCards, (Accessed: 28.1.2017): https://www.blackhat.com/docs/sp-14/materials/arsenal/sp-14-Almeida-Hacking-MIFARE-Classic-Cards-Slides.pdf

[71] NXP, Connected Tag Solutions, (Accessed: 28.1.2017): http://www.nxp.com/products/identification-and-security/nfc-and-reader-ics/connected-tag-solutions:MC_1429877262080

[72] NFCForum, NFC Data Exchange Format (NDEF) Technical Specification NFC Forum, NDEF 1.0 NFCForum-TS-NDEF_1.0, July 24, 2006

[73] Roger M.Needham and David J.Wheeler, Tea extensions, October 1996, (Accessed: 2.2.2017)
http://www.cix.co.uk/~klockstone/xtea.pdf

[74] Karthik .S, Muruganandam .A, Data Encryption and Decryption by Using Triple DES and Performance Analysis of Crypto System, November 11 2014, ISSN (Online): 2347-3878 Volume 2 Issue 11, November 2014, International Journal of Scientific Engineering and Research

[75] Uli Kretzschmar, Texas Instruments, Application Report, SLAA397A, AES128 – A C Implementation for Encryption and Decryption, July 2009

[76, 78, 80] David Jones, UCL Bioinformatics Group, Good Practice in (Pseudo) Random Number Generation for Bioinformatics Applications, May 7, 2010

[77] PIERRE L'ECUYER and RICHARD SIMARD, TestU01: A C Library for Empirical Testing of Random Number Generators, August 2007

[79] Greg Rose, KISS: A Bit Too Simple, 2011

[81] Don Johnson and Alfred Menezes, The Elliptic Curve Digital Signature Algorithm (ECDSA), July 27, 2001, Springer-Verlag 2001, DOI: 10.1007/s102070100002

[82] NXP, NFC Forum Type Tags, White Paper V1.0, April 1, 2009

[83] NFC Forum, New NFC Forum Technical Specifications Broaden Tag Support and Enhance Interoperability, October 14, 2015, (Accessed: 28.1.2017)
http://nfc-forum.org/newsroom/new-nfc-forum-technical-specifications-broaden-tag-support-and-enhance-interoperability/

[84] SonMicro, User Manual, SM130, Revision A.2, June 2006

[85] Jinmuyu Electronics Co. LTD, User Manual, JMY504C User's Manual, Revision 3.42, June 28, 2011

[86] ubisys, Refernce Manual, 13.56 MHz RFID USB READER REFERENCE MANUAL

[87] NXP, Datasheet, LPC1110/11/12/13/14/15 Datasheet, Rev. 9.2, March 26, 2014

[88] NXP, Datasheet, LPC11u6x Datasheet, Rev. 1.3, September 7, 2016

[90] Embedded Artists, LABTOOL, (Accessed: 28.1.2017): http://www.embeddedartists.com/products/app/labtool.php

[89] NXP, OM13054: LPC-Link2, (Accessed: 28.1.2017): http://www.nxp.com/products/microcontrollers-and-processors/arm-processors/lpc-cortex-m-mcus/lpc3000-arm9-mpus/lpc-link2:OM13054

[91] NXP, Application Note, AN11342, How to Scale Down the NXP Reader Library, Rev. 1.0, March 11, 2013

7. Shortcuts

RFID	Radio Frequency Identification
LF	Low Frequency
HF	High Frequency
UHF	Ultra High Frequency
USB	Universal Serial Bus
CCID	Chip Card Interface Device
CDC	Communication Device Class
HID	Human Interface Device
ASCII	American Standard Code for Information Interchange
USA	United States of America
IC	Integrated Circuit
OS	Operating System
PC	Personal Computer
PCB	Printed Circuit Board
USART	Universal Asynchronous Receiver Transmitter
SPI	Serial Peripheral Interface
DTE	Data Terminal Equipment
DCE	Data Communication Equipment
TxD	Transmit Data
RxD	Receive Data
GND	Ground
RTS	Request To Send
CTS	Clear To Send
LSB	Least Significant Bit
MSB	Most Significant Bit
ABR	Automatic Baud Rate

CR	Carriage Return
EMI	Electromagnetic Interference
OTG	On-The-Go
NRZI	Non Return to Zero Inverting
CRC	Cyclic-Redundancy-Check
ICC	Integrated Circuit Cards
APDUs	Application Protocol Data Units
TPDUs	Transport Protocol Data Units
PIN	Personal Identification Number
API	Application Programming Interface
ROM	Read Only Memory
ASK	Amplitude Shift Keying
FSK	Frequency Shift Keying
PSK	Phase Shift Keying
NRZ	Non Return to Zero
PCD	Proximity Coupling Device
PICC	Integrated Circuit Card
OOK	On-Off-Keying
BPSK	Binary Phase Shift Keying
NRZ-L	Non Return to Zero-Level
UID	Unique Identifier
NUID	None Unique Identifier
RID	Random Identifier
VCD	Vicinity Coupling Device
VICC	Vicinity Integrated Circuit Card
NFC	Near Field Communication
MAC	Message Authentication Code
NDEF	Near-Field Data Exchange Format

PIE	Pulse Interval Encoding
DSB	Double Sideband
NDA	Non-Disclosure Agreement
IDE	Integrated Development Environment
ANSI	American National Standards Institute
SRAM	Static Random-Access Memory
EEPROM	Electrically Erasable Programmable Read-Only Memory
CRP	Code Read Protection
PWM	Pulse Width Modulation
AES	Advanced Encryption Standard
DES	Data Encryption Standard
RNG	Random Number Generator
PRNG	Pseudo Random Number Generator
CSPRNG	Secure Pseudo Random Number Generator
ADC	Analog to Digital Converter
ECDSA	Elliptic Curve Digital Signature Algorithm
ECDLP	Elliptic Curve Discrete Logarithm Problem
OTP	One Time Programmable

8. List of Figures