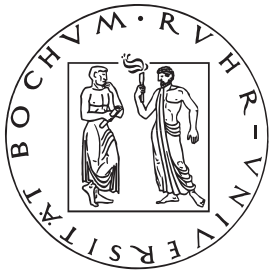
Embedded Security Analysis of RFID Devices

Timo Kasper

July 10, 2006

Diploma Thesis
Ruhr-University Bochum



Chair for Communication Security
Prof. Dr.-Ing. Christof Paar

Co-Advised by:
Dipl.-Ing. Dario Carluccio
Dipl.-Phys. Kerstin Lemke-Rust

Statement

I hereby declare, that the work presented in this thesis is my own work and that to the best of my knowledge it is original, except where indicated by references to other authors.

Hiermit versichere ich, dass ich meine Diplomarbeit selber verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt, sowie Zitate kenntlich gemacht habe.

Date / Datum	Timo Kasper

Contents

List of Figures

Nomenclature

CL_n Cascade Level n
AC Alternating Current
ADC Analog to Digital Converter
AES Advanced Encryption Standard
ASK Amplitude Shift Keying
ATQA Answer To Request, Type A
ATS Answer To Select
CMOS Complementary Metal-Oxide Semiconductor
DC Direct Current
DDR Data Direction Register
DEMA Differential ElectroMagnetic Analysis
DES Data Encryption Standard
DIP Dual In-line Package
DPA Differential Power Analysis
ECC Elliptic Curve Cryptography
EOC End Of Communication
FDT Frame Delay Time
FIFO First In First Out
HF High Frequency
HLTA Halt command, Type A
IC Integrated Circuit
IDE Integrated Development Environment
ISR Interrupt Service Routine
LED Light Emitting Diode
LF Low Frequency
MISO Master In Slave Out
MOSFET Metal-Oxide Semiconductor Field-Effect Transistor

MOSI Master Out Slave In
MRTD Machine Readable Travel Document
MSB Most Significant Bit
NDA Non Disclosure Agreement
NFC Near Field Communication
NOP No Operation (computer processor instruction)
NRZ Non Return to Zero
NVB Number of Valid Bits
OOK On Off Keying
PC Personal Computer
PCB Printed Circuit Board
RAM Random Access Memory
RATS Request Answer To Select
REQA Request command, Type A
RF Radio Frequency
RFID Radio Frequency IDentification
RISC Reduced Instruction Set Computer
ROM Read Only Memory
SAK Select AcKnowledge
SCK Slave Clock
SEL SELect code command
SMD Surface Mounted Device
SNR Signal to Noise Ratio
SOC Start Of Communication
TTL Transistor-Transistor-Logic
UART Universal Asynchronous Receiver-Transmitter
UHF Ultra High Frequency
UID Unique IDentifier
USB Universal Serial Bus
VCP Virtual Com Port
WUPA Wake-Up command, Type A

1 Introduction

1.1 Evolution of RFID

1.1.1 History

When the notion of Radio-Frequency Identification (RFID) arose in the 1940s, it was used for identification of objects, i.e., allied airplanes by the military forces [46]. The so-called active tags needed a power supply, had rather large dimensions and carried small amounts of data, e.g., a fixed unique number. As technology evolved, with modern silicon wafer manufacturing, chip sizes with an area as small as 0.15×0.15 mm^2 and a thickness of only 7.5 μm are possible [19], resulting in lower energy consumption. This enables passive tags, which draw the energy needed for operation completely from the RF[1] field that is generated by a reader device. At the same time, it is now possible to put much larger memories and even microcontrollers with crypto co-processors on the chip of the tag, so that applications like contactless, cryptographically enabled smartcards and their use as credit cards or digital passports are becoming widespread and RFID can be an ubiquitous technology.

1.1.2 Standards for Contactless Smartcards

Different standards are available for RFID technology, described in more detail in the RFID handbook [15], operating at frequencies from 135 kHz in the LF[2] range to 5.8 GHz in the UHF[3] range. The relevant ones for cryptographic applications, almost exclusively operated in the HF[4] range at 13.56 MHz, are mentioned briefly. Table 1.1 shows a comparison of the standards with regard to operating frequency, approximate operating range and maximum data rate.

The standard for closely coupled smartcards, namely the ISO 10536, was developed between 1992 and 1995 and never succeeded in the market, due to high manufacturing costs and only small advantages compared to contact-based cards.

The ISO 14443 standard for proximity coupling, described in Section 2, is often the choice for access control and ticketing purposes.

[1] Radio Frequency
[2] Low Frequency
[3] Ultra High Frequency
[4] High Frequency

Vicinity cards, as specified in the ISO 15693, can be read from a greater distance, compared to proximity cards, at the cost of a lower data rate. In addition, the energy consumption of an ISO 15693 compliant tag has to be lesser, due to a lower specified magnetic field strength being necessary for operation which, combined with the low data rate, very likely makes state-of-the-art cryptography impossible. Note that the maximum operating range, given in Table 1.1, is only achievable using the long distance mode of ISO 15693 compliant tags, for which a data rate of only 1.65 kBit/s is specified.

The NFC[5] standard has been pushed mainly by Philips and Sony, is compatible to the ISO 14443 A standard, and shall be used for short-range communication between electronic devices [43].

Standard	Card Type	Range	Frequency	Data Rate
ISO 10536	Close Coupling	$\leq$ 1 cm	4.9152 MHz	9600 Bit/s
ISO 14443	Proximity Coupling	8 ... 15 cm	13.56 MHz	847.5 kBit/s
ISO 15693	Vicinity Coupling	1 ... 1.5 m	13.56 MHz	26.48 kBit/s
ISO 18092	Near Field Communication	$\approx$ 10 cm	13.56 MHz	424 kBit/s

Table 1.1: Comparison of standards for contactless smartcards

1.1.3 Relevant Applications

The ISO 14443 standard [22] is employed by many leading chip manufacturers in various RFID applications, e.g., Mifare identification chips from Philips[6], which are used for ticketing, during the world championship 2006 in Germany [45] and for public transport in the London Underground [4], or Texas Instruments' chips being implanted in MasterCard's PayPass [3] and Visa Contactless RFID payment cards [2]. At the Ruhr University in Bochum, contact based smartcards have recently been upgraded with a contactless prepaid payment function, which is based on the ISO 14443 standard and enables, for example, the automatic recognition, if a discount is to be granted, due to the status (student, employee, pensioner, etc.) of the respective person. Another crucial application is the digital passport (e-passport), standarised by the International Civil Aviation Organization (ICAO)[7], in which an ISO 14443 compliant chip [5] stores biometric data [8], in addition to the personal particulars.

New inventions like wearable RFID wristbands or transponders implanted in shoes, and even tags injected under the skin of human beings, are nowadays used instead of a key to gain access to restricted areas. Identification and tracking purposes (e.g., of children, elderly people, patients in a hospital) might become pervasive in the near future. Tagged

[5] Near Field Communication
[6] http://www.semiconductors.philips.com
[7] http://www.icao.int

money is one more vision, with RFID chips in the paper, to make counterfeiting more difficult, or tagged airline baggage, to ease automatic transportation.

In general, a wide deployment of the ISO 14443 standard can currently be noticed for contactless applications demanding for privacy and security, with the resulting need for high computation power, which at the moment can only be achieved via inductive coupling (see Section 2.1.1) and a relatively short reading distance.

1.2 Motivation

1.2.1 New Risks

As with every new technology, new threats appeared with the deployment of RFID, beginning in the 1950s, when enemies airplanes pretended to be from the other party by replaying a previously recorded answer. This demanded for inventions like Feistel's two pass authentification challenge, which, in extended variations, is still often used to prevent such attacks in modern RFID systems [46]. Moreover, the interchanged data is often encrypted with common block ciphers [35] like AES[8] and (Triple-)DES[9], or sometimes even public-key algorithms like ECC[10], where security or privacy issues are relevant. Still, modern offenders get physical access to the chip or its field and perform so called side channel attacks [36] like a DPA[11] or a DEMA (see Section 1.3.1), which make it possible to obtain a secret key stored on the device by analysing the power consumption or electromagnetic emanation over the time and correlating it with a data hypothesis and the code being executed. Other implementations of attacks aim at introducing an error during computation of a device, which can ease cryptanalysis.

1.2.2 RF Impacts

The physical interface of contactless smartcards brings new opportunities for possible attackers, because the wireless transmission of data via the RF[12] field can easily be eavesdropped by an attacker, without the carrier of the tag taking note of it. So sniffing, i.e. acquiring and analysing the data transmitted between reader and tag to obtain a certain information, for example someone's photo or fingerprint, is possible over sometimes large ranges. Eavesdropping of communication between ISO 14443 compliant devices over a distance of several meters has been performed by Finke and Kelter [14]. The communication data can be recorded, collected and maybe decrypted later on. People also can be tracked, for example by a set of tagged items, which were recently bought and

[8] Advanced Encryption Standard
[9] Data Encryption Standard
[10] Elliptic Curve Cryptography
[11] Differential Power Analysis
[12] Radio Frequency

carried around by an individual, whose movings then can be monitored. A relay (passive man-in-the-middle) attack is also feasible, i.e., redirecting the data interchanged between reader and tag over a separate communication channel to pretend to be the owner of someone else's tag. The data could be manipulated in a way that gives some advantage to the attacker before relaying the data - an active man-in-the-middle attack. The number of possible threats is large and becoming larger, showing the necessity of well designed security schemes in the various systems.

1.2.3 Limitations

The energy consumption, i.e., the maximum number of switching transistors of a passive RFID tag is limited [27], whilst having the advantages of smaller size, lower weight and less cost. Typical implementations using a 0.35 μm process have 5000 gates and consume a current of 15 μA [46]. Furthermore, as the industry wants to keep the prices low, security measures and physical protection on the chip, demanding for much chip area, may be rarely implemented. Hence, certain mechanisms to protect devices against side channel- and other attacks will be very lightweight or won't be found at all on some RFID devices [34].

Some proprietary RFID systems have already been broken, for example the Digital Signature Transponder (DST), manufactured by Texas Instruments, employed in vehicle immobilisers that are used additionally to carry out payments. Bono et al.[7] reverse engineered the protocol, decrypted the communication, i.e. figured out the secret key, and, in addition, purchased gasoline and started an automobile by simulating DST devices.

1.2.4 Privacy Considerations

Civil Liberties groups and other organisations, e.g., the FoeBud in Germany with their "stop RFID" campaign[13], fear the abuse of RFID based applications and warn people not to ignore threats like universal surveillance and violations of the privacy of individuals. Medical information getting into the wrong hands might result in unemployment, and tracking of movements, for example by tagging employees at the workplace, in a significant loss of privacy.

It is important on the one hand not to exaggerate these problems and thus provoke fears in the population, and on the other hand not to underestimate these challenges and find solutions, to profit from the advantages of the modern technology and at the same time protect it from being misused.

[13] `http://www.foebud.org/rfid`

1.2.5 Towards More Security

In order to improve the security analysis of RFID systems, tools providing the contactless interface and being able to perform known attacks, as well as to analyse the capabilities and functionality of the hardware used in an RFID system, need to be developed. As the standards for contactless smartcards differ very much with regard to operating frequency, communication interface and transmission protocol [15], the hardware for a reasonable security analysis must be quite specialised and tailored to one certain standard.

The RFID tool, that is developed and built up as a part of this diploma thesis, is generally applicable to all devices compliant to part 2 (RF power and signal interface) and part 3 (initialisation and anticollision) of the ISO 14443(A) [22], no matter if a proprietary protocol, including cryptography, is implemented on a higher layer.

1.3 Related Work

1.3.1 DEMA

A DEMA[14] is a special form of electromagnetic side channel analysis of cryptographic ICs and, as shown by Carluccio et al.[10], can be applied to RFID smartcards. An antenna connected to an oscilloscope, placed as close as possible to the chip for obtaining a high Signal to Noise Ratio (SNR), is used to gather information about the secret key stored on the device, by measuring and evaluating the electromagnetic emanation during operation. To reduce the influence of the RF interface on the measurements and to further increase the SNR, the chip can be removed from the plastic packaging and the antenna separated from it, as depicted in Figure 1.1. Now, the communication between an RFID reader and the smartcard, via the antenna, which remains in the plastic of the card in the background of the picture, can take place spatially and electrically separated from the measurements with the chip, in the foreground of the picture.

As DEMA is based on a statistical test, for which subsequent measurements have to be synchronised and superimposed without too much jitter, it is helpful to have a reliable signal to trigger the scope.

The protocol of the Philips Mifare DESFire contactless smartcard, i.e., the applied mutual three pass authentification, has been reverse engineered [9] until to the point necessary for carrying out a DEMA to potentially achieve the secret key stored on the device. In the attack performed by Carluccio, so-called challenges, needed for the mentioned authentification protocol, were generated by a commercial RFID reader device and had to be extracted from the communication data afterwards, which was very time consuming. As the protocol used was readily implemented in the reader, the communication could not be aborted (and then restarted) at any moment, i.e., after willingly sending invalid data.

[14] Differential Electro Magnetic Analysis

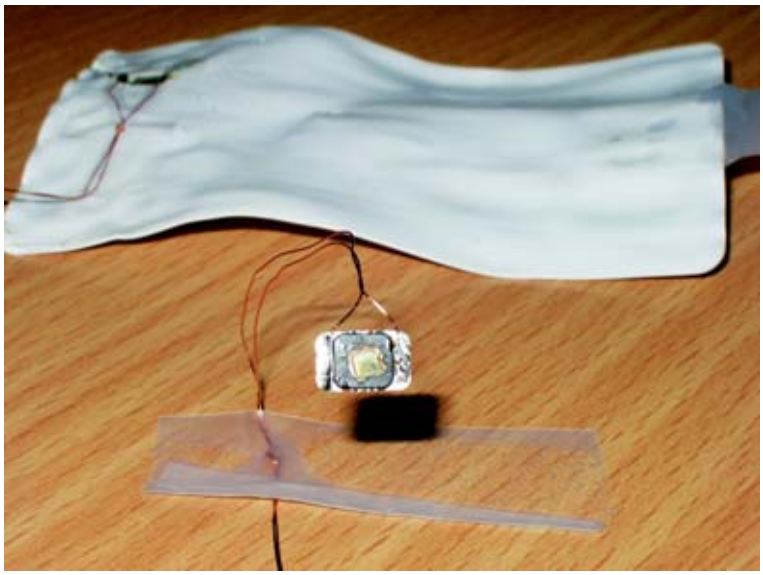


Figure 1.1: Separating the chip and the plastic packaging of a smartcard

1.3.2 Relay Attack

A relay attack, also called a passive man-in-the-middle attack, without being able to modify the data interchanged between reader and tag, as described by Kfir and Wool [23], was practically carried out by Hancke [17]. The special feature of this attack is, that it works on the physical layer and therefore can not be prevented by basic authentification protocols and encryption of the data interchanged. The antenna of a reader, possessed by the offender, has to be placed close enough to the contactless card or tag of a victim, while a second device emulating a tag is brought into the field of an RFID reader, e.g., at a cash desk possibly located at a distance from the owner of the card. The data being transferred by this reader is acquired and directly forwarded on the bit layer through a communication link to the reader of the attacker. There, the data is retransmitted to the card of the victim, which then answers to the request of the remote reader, without its owner noticing it. The answer is relayed back via the device emulating a tag to the cashpoint's reader again and so, as the attacker continues relaying the data, both reader and tag will be convinced, that they are in close vicinity to each other, share the same secret and carry out their task, e.g., authorise a payment.

Hancke and Kuhn [18] proposed a possible countermeasure against these kind of attacks, based on ultra-wideband pulse communication. This method is not being employed in devices currently available on the market, so still the most effective way to circumvent such an attack, for the devices currently in use, is to construct a Faraday's cage around the tag, e.g., by wrapping it with aluminum foil (investigated in Section 4.4.1).

1.3.3 Remote Power Analysis

Another power analysis attack requiring no physical contact to the device was performed by Oren and Shamir [34], with RFID tags operating in the UHF range, where so called backscattering is used for data transmission from tag to reader, instead of inductive coupling (see Section 2.1.1) in the HF range, as specified in the ISO 14443. Similar to the ISO 14443, the data is transferred from a reader to a tag by the use of gaps in the field of the reader (compare with Section 2.2.1), which at the same time has to provide the energy needed for operation of the tag. During the pauses, the tag draws the energy from a built in capacitor, which needs to be recharged when the field is turned on again. This leads to different shaped energy peaks occurring after the gaps, depending on the amount of power consumed by the tag during the pause, noticeable at the antenna of the reader. This behaviour was exploited to find an 8-bit password for the kill command of EPC Global tags. The described method may also be applicable to transponders compliant to the ISO 14443, which has to be further researched.

1.4 Possible Applications

The devices developed here shall ease the security analysis of cryptographically enabled RFID devices with an ISO 14443A compliant RF interface, and make it possible to perform the following tasks:

- use of a transparent and flexible contactless interface on the bit layer, i.e., an implementation of a low level reader,
- emulation of an RFID tag,
- replay attack,
- relay attack,
- active MITM (man-in-the-middle) attack, i.e. possibility to intervene in the communication,
- investigations of conformance to the ISO 14443[15],
- (remote) power analysis,
- DEMA,
- fault analysis,
- analysis of protocols, i.e., logging of the communication data,
- fast communication with a PC or other cryptographic hardware via USB,
- testing of different types of antennas and tuning methods in diverse environments.

[15] experiments with the tool developed in this thesis showed, that an RFID reader did not strictly obey timing requirements specified in the ISO 14443 and so eventually facilitates relay attacks

2 Technical Review of the ISO 14443A

This work focuses on devices compliant to the ISO 14443 A standard, using a data rate of $\frac{f_c}{128}$, where f_c denotes the carrier frequency of the reader, leading to $\frac{13.56\ MHz}{128} = 106\ \frac{kBit}{s}$ in both directions, as specified in part 2 of the standard [22]. In this thesis, the terms tag, card and transponder are used equivalently, and are therefore interchangeable.

2.1 RFID Operation Principle

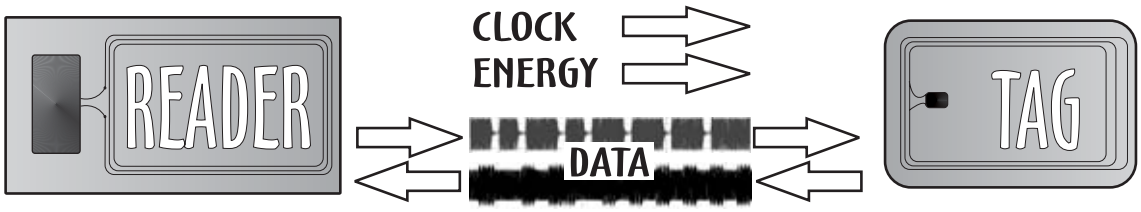


Figure 2.1: General RFID System

A minimum RFID system consists of two main components, namely a reader generating a field, i.e., a sine wave with a frequency of 13.56 MHz, which supplies the second component for the system, a so called tag or transponder, with energy[1] and often a clock signal for operation of its digital circuits [15]. A chip on the tag contains data, which may be fixed and stored in a ROM, or changeable and stored in a RAM, and furthermore must have the capability to en- and decode the information interchanged with the reader. For more sophisticated applications, microcontrollers and operating systems for comfortable access to the stored data, and cryptographic co-processors, to encipher the communication, are employed. Both transponder and reader are equipped with a coupling element, which in the case of the ISO 14443 is a coil with typically 3-10 windings, permitting data transfer in both directions. Note, that the term RFID reader is a rather misleading description for a device that does not only receive data from the tag, but of course also transmits data to it, while often being connected to another system, e.g., a PC (Personal Computer).

[1] in the case of passive tags

2.1.1 Inductive Coupling

The wavelength λ of an electromagnetic field is calculated following equation 2.1, where c denotes the speed of light and f the carrier frequency, which here is equal to 13.56 MHz, as defined in the standard.

$$\lambda = \frac{c}{f} = \frac{3 \cdot 10^8 \ \frac{m}{s}}{13.56 \ MHz} \approx 22.1 \ m \tag{2.1}$$

Obviously, the derived wavelength is several times greater than the typical operating distance between reader and tag, which is approximately 8-15 cm [15]. Accordingly, the field emitted from the coil of the reader may be treated as purely magnetic[2], leading to the term inductive coupling being used to describe the communication- and energy link between reader and tag.

2.2 Communication Details

According to the ISO 14443, a reader transmits data to a tag by means of switching the field temporarily off, i.e., create short gaps in the field, which are detected and decoded by the tag. The tag answers employing load modulation as described below in Section 2.2.2, which in turn is sensed and decoded on the side of the reader.

The communication is based on a master-slave principle, where the reader is always the master, and the tag is the slave. The reader talks first, and then listens to the answer of the tag[3], while keeping the field alive to supply it with energy.

1. reader sends data to the tag (termed downlink)
2. waiting time until to the answer of the tag
3. tag answers (termed uplink)
4. waiting time until to the next request from the reader

...proceed with 1 until finished.

2.2.1 Reader → Transponder

For the downlink, modified (pulsed) Miller coding is used, where the data is represented as follows.

[2] similar to the common transformer principle
[3] a so called half duplex system

Modified Miller Coding

The correlation between NRZ[4], Miller code and the modified variant (at the bottom) is depicted in Figure 2.2.

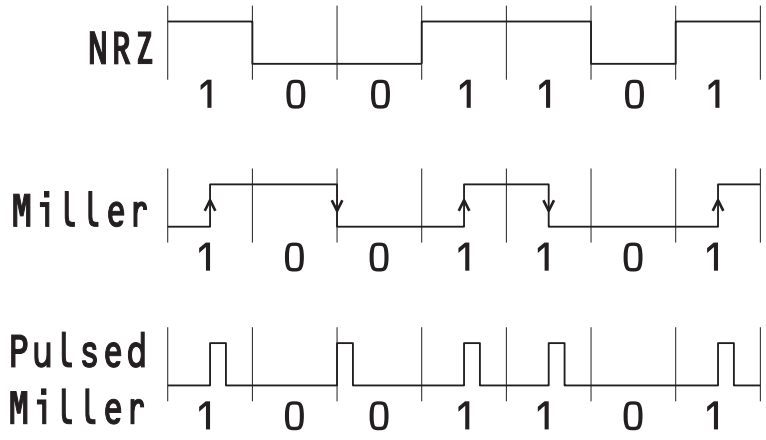


Figure 2.2: (Pulsed) Miller Coding

- *Logic 1*: Pause in the middle of the bit period, i.e. after $\frac{64}{f_c} \approx 4.72\ \mu$s
- *Logic 0*

  α) previously *0* or *SOC*[5]: Pause at the beginning of the bit period

  β) previously *1*: No modulation for the full bit duration.
- *SOC*: Pause at the beginning of a bit period (equals *0* after *0*)
- *EOC*[6]: *Logic 0* followed by no modulation for a full bit period

Pauses have to be created with a duration of approximately 2.5 μs[7], with 100% ASK[8], i.e., the field has to be completely switched off and on by the reader.

2.2.2 Transponder → Reader

Load Modulation

As explained in Section 2.1.1, the energy consumed by a tag is supplied by the reader via the two transformer-like coupled coils of the RFID system. The resulting feedback of

[4]Non Return to Zero
[5]Start Of Communication
[6]End Of Communication
[7]more precise between 2 and 3 μs
[8]Amplitude Shift Keying

the transponder, drawing more or less energy from the field, can be sensed by a varying amplitude at the antenna of the reader. By switching on and off an additional load resistor and thereby deliberately taking more energy from the field than during normal operation, the tag transmits its data to the reader, sometimes referred to as OOK[9] in the literature. As the coupling between tag and reader is weak and the resulting effect on the field almost not noticeable, a subcarrier of the reader's carrier frequency is generated by the tag and used to switch the resistor, leading to the transmitted information being placed in sidebands of the carrier and making the detection of the achieved 10 mV change of useful signal at a carrier amplitude of 100 V[10] possible [15].

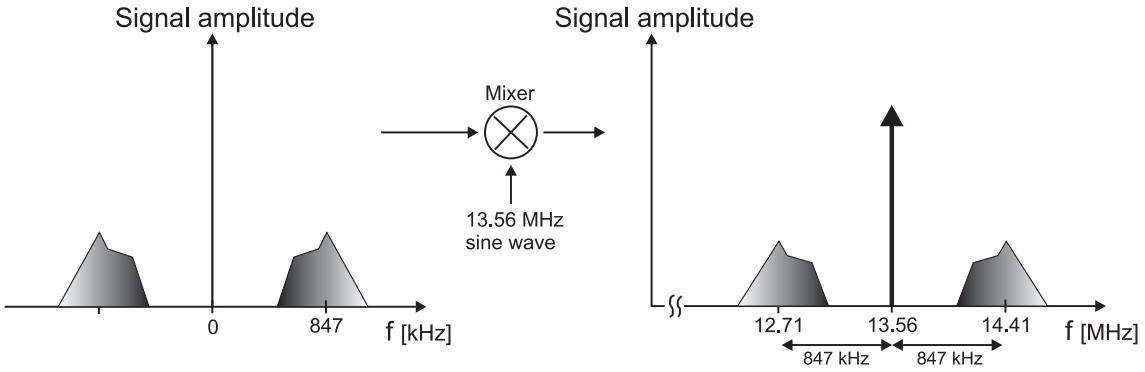


Figure 2.3: Modulation Principle

Figure 2.3 illustrates the described process: On the left side, a low pass filtered signal containing the information to be transmitted, e.g. a 106 kBit/s data stream, has been modulated with a 847 kHz subcarrier, as described in Section 3.1.5, resulting in the depicted symmetric frequency spectrum[11], which can be obtained by performing a Fourier transform (see [13] for details). Modulating this signal again with a 13.56 MHz sine wave leads to the frequency spectrum on the right side of Figure 2.3, where the left, symmetric half of the spectrum is omitted. Obviously, the information is being placed in sidebands beside the carrier frequency.

(Modulated) Manchester Coding

For the uplink, the described load modulation is utilised to transmit Manchester encoded data, modulated with a subcarrier of $\frac{f_c}{16} = 847.5$ kHz, which shall be synchronous to the field of the reader. Figure 2.4 illustrates the generation of the modulated code. One bit duration equals eight subcarrier-periods at the data rate of $\frac{f_c}{128} = 106$ kBit/s.

- *Logic 1*: Falling edge at the centre, i.e., modulation with the subcarrier for the first half of the bit period

[9] On Off Keying
[10] corresponding to 80dB
[11] all real world signals have a symmetric frequency spectrum

- *Logic 0*: Rising edge at the centre, i.e., modulation with the subcarrier for the second half of the bit period
- *SOC*: Equals *logic 1* (see above)
- *EOC*: No modulation for a full bit period

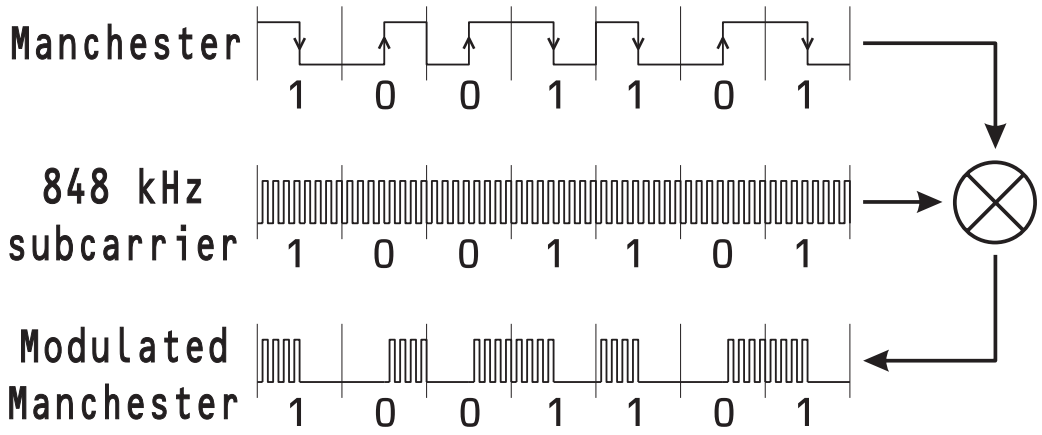


Figure 2.4: (Modulated) Manchester Coding

Manchester coding may be alternatively viewed as a phase encoding, where each bit is encoded by a positive 90 degree phase transition or a negative 90 degree phase transition, and therefore is sometimes referred to as biphase coding.

Note that, when Manchester coding is employed, a reader can easily detect two cards sending distinct bits simultaneously, as this leads to a modulation for a full bit period. This is of use during the anticollision phase of the ISO 14443 protocol.

2.2.3 Initialisation Phase

Collisions between two tags being in the same field, answering simultaneously to a request of a reader, and thus preventing it from acquiring valid data from any of the tags, usually don't play a role due to the short operating range. Hence, the anticollision part of the protocol is not explained here, and, in the following brief description of a typical communication sequence, it is assumed that only one card is present in the field of a reader. The following section shall give only an idea of the protocol – further details can be found in part 3 of the standard [22].

Initialisation Sequence

When getting in the proximity of a reader, into an energizing magnetic field greater than H_{min}[12]$= 1.5\ \frac{A}{m}$ (details in the standard [22], part 2), the card powers up and gets into

[12]a maximum unmodulated operating field, with a value of $H_{max} = 7.5\ \frac{A}{m}$, is also defined

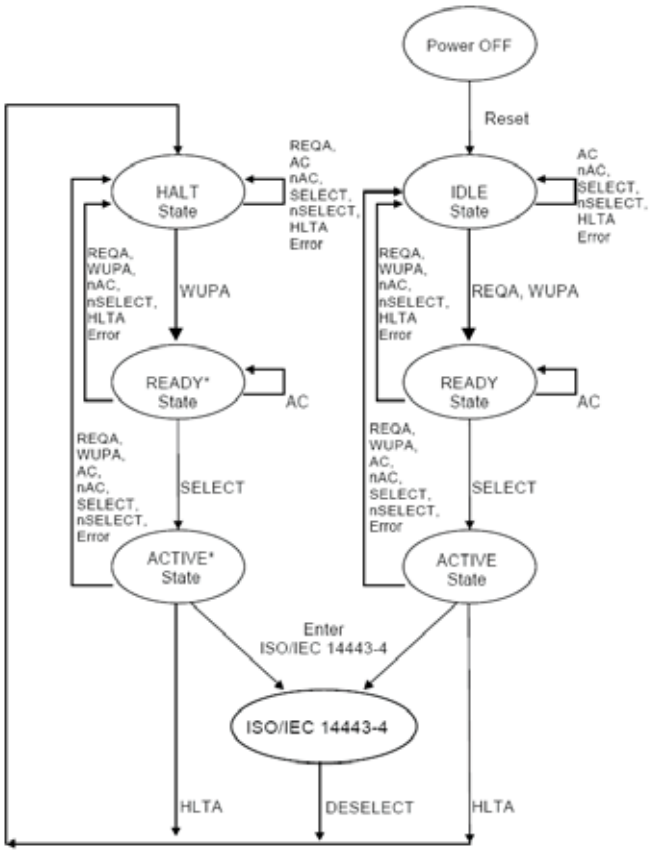


Figure 2.5: States of a tag during the initialisation phase

the *idle* state.

A *REQA*[13] or *WUPA*[14] induces emission of an *ATQA*[15] and a change into the *ready* state, where the card waits for a *SEL*[16] of Cascade Level n (CL_n) with the parameter NVB[17] being 0x20, prompting the card to answer with its UID[18] of CL_n. The reader acquires this UID and can now issue a *SELECT* command with the UID of the tag.

The card answers to the *SELECT* command with its *SAK*[19] response, which indicates, whether the UID is already complete (or a higher cascade level has to be handled) and if it is part 4 - compliant. As the ISO allows for 3 different lenghts of the UID (4,7 or 10 bytes), the above process (*SEL* etc.) might have to be repeated up to 3 times, each time with a higher CL, until the card has received its complete UID and finally goes

[13]Request command, Type A
[14]Wake-Up command, Type A
[15]Answer To Request, Type A
[16]SELect code command
[17]Number of Valid Bits
[18]Unique IDentifier
[19]Select AcKnowledge

into the *active* state. From there on, commands according to a higher layer protocol (ISO14443 [22] part 4, or a proprietary protocol) can be issued.

In case of compliance to part 4, the reader sends an *RATS*[20] now, containing the maximum possible framesize it can handle, answered by an *ATS*[21] of the tag. The *ATS* defines the maximum framesize accepted by the tag, as well as the bit rate capabilities of the tag in both directions.

After having entered at least the *active* state, a card can enter a *halt* state for example by receiving a *HLTA*[22], from which it only answers to a *WUPA*, but not to a *REQA*. The rest is similar to the normal case described above. A card in any state receiving a *REQA* will become either *idle* or enter the *halt* state.

The concrete implementation of the necessary commands is specified in the ISO 14443.

UID Concerns

Every ISO 14443A compliant RFID tag has an own UID, which is often a fixed number, written into the ROM of the chip by the manufacturer, but can also be a random number, dynamically created every time the device powers up - important, for example, to prevent tracking of individuals by scanning the UID of their e-passport. If the first byte of the UID equals 0x08, it is a randomly generated number, otherwise it will be a proprietary fixed number. During tests with an e-passport, the described behaviour was verified.

2.2.4 Timing Specifications

As the timing requirements of the ISO14443A ([22], part 3 and 4) are important for the emulation of a tag or performing a relay attack, which is naturally inducing a certain delay, they are discussed here in detail.

Request Guard Time

Between the start bits of several consecutive *REQA* commands, a pause of $\frac{7000}{f_c} \approx 516\ \mu$s, called request guard time, has to be inserted.

Frame Delay Time

The frame delay time FDT is the time between two frames transmitted in opposite directions and specified in part 3 of the standard [22].

Tag → Reader: The time between the end of the last pause created by the reader until to the first edge of the answer of the tag shall be

[20] Request Answer To Select
[21] Answer To Select
[22] Halt command, Type A

After a *logic 1*:

$$FDT = \frac{(128 \cdot n + 84)}{f_c} \tag{2.2}$$

If the reader sent a *logic 0*:

$$FDT = \frac{(128 \cdot n + 20)}{f_c} \tag{2.3}$$

For specific commands like *REQA* or *WUPA*, the integer value n equals $n = 9$, which leads to a pause duration of $\frac{1236}{f_c} \approx 91.15\ \mu$s if the last bit sent by the reader was a *logic 1*, or $\frac{1172}{f_c} \approx 86.43\ \mu$s if it was a *logic 0*. For all other commands, $n \geq 9$ applies. In any case, the first edge of the answer of the tag has to be aligned to the bit grid defined above.

Furthermore ISO 14443 [22] part 4 defines an activation frame waiting time, which is the maximum time for a card to answer after the *EOC* of the reader's request and equals $\frac{65536}{f_c} \approx 4.8$ ms.

Reader → Tag: The minimum time between the last modulation of the tag until to the first gap in the field, generated by the reader, is

$$FDT = \frac{1172}{f_c} \approx 86.43\ \mu s \tag{2.4}$$

Note that for the time between a command of the reader and the answer of a tag, except for the case $n = 9$, only a bit grid with an upper bound is specified, whereas, in the opposite direction, solely a minimum time has to be considered.

3 System Design and Development

The system developed in this thesis consists of a multi purpose reader device, which is equipped with a microcontroller, an RF interface and the ability to do some signal processing. A second device, named fake tag, is able to perform load modulation and to gather the information sent by a remote reader. If properly fed with data, this fake tag appears like an authentic tag to an RFID reader. Between the two units, a communication link can be established, which is just a cable or can be wireless.

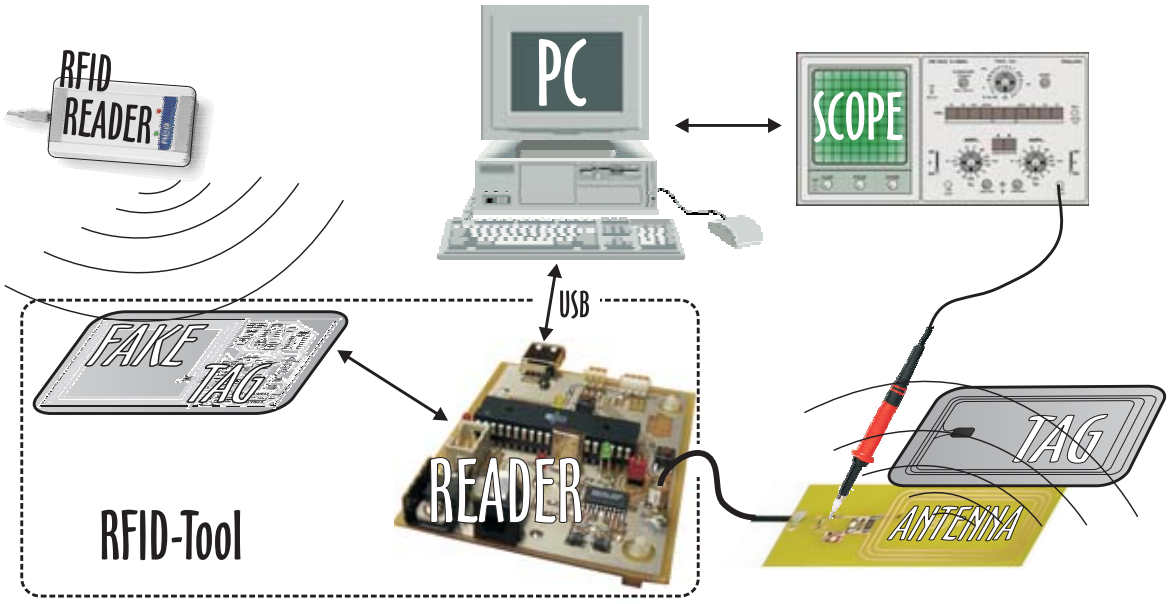


Figure 3.1: System Overview

The RFID tool can be integrated in a complete system, consisting of a PC, the developed reader and fake tag, a digital oscilloscope and more measurement equipment like near field probes to quantify electromagnetic emanation. Reader and scope are connected to the PC, which controls the process sequence and later combines and further handles the data acquired from scope and reader.

The developed hardware permits automatic recognition of the information interchanged and its transfer to a PC or specialised hardware [25] for cryptographic analysis, maybe real time decrypting of the data transmitted, or other processing.

In addition, stand alone operation of the RFID tool is possible, to execute man-in-the-middle attacks or store data acquired from RFID tags maybe without permission,

e.g., in the subway or other crowded places, where the required short reading distance can be accomplished. If the information is not encrypted, it could be modified and later replayed via the fake tag to make an RFID reader believe to have, for example, a valid ticket in its vicinity.

The RFID tool was built using electronic hobbyist equipment and materials, with commonly available components. Therefore, since the tool has been developed now, the reproduction is feasible without much competence, at a cost of well beyond 50 €.

3.1 The Fake Tag

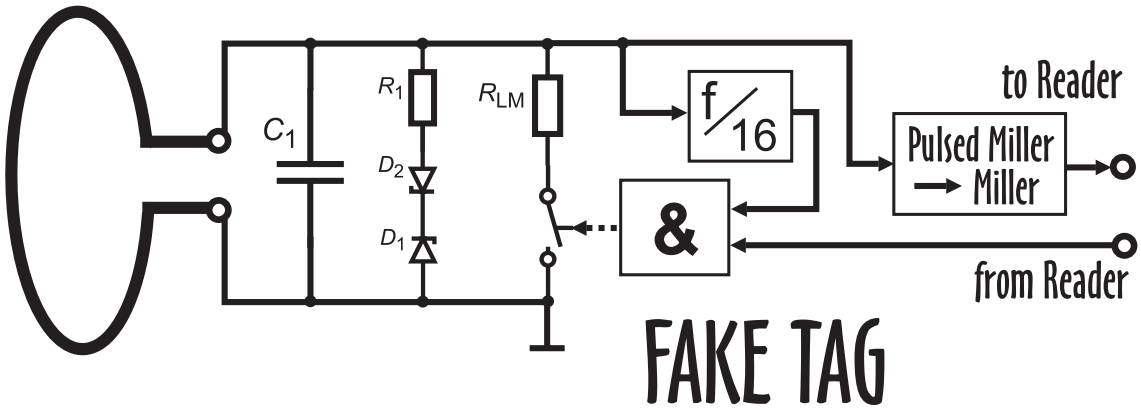


Figure 3.2: Operation Principle of the Fake-Tag

The Fake Tag, which is designed to appear like an authentic ISO 14443 compliant RFID transponder, is intended to cooperate with the developed RFID reader (see Section 3.2) and can be utilised for relay and replay attacks as well as for tag emulation. Unlike a normal (passive) tag, the fake tag described here has an own power supply[1], which can also be used for supplying an optional wireless module.

3.1.1 Parallel Resonant Circuit

To be able to communicate with a reader, a tag needs a coil as an antenna to establish the coupling to the counterpart of the reader. A capacitor is connected in parallel to this inductance to form a parallel resonant circuit with a resonant frequency corresponding to the carrier frequency of, in this case, 13.56 MHz.

For an ideal parallel resonant circuit, capacitance and inductance are selected according to equation 3.1, where f_0 denotes the carrier frequency of the reader, C the capacitance and L the inductance of the tuned circuit [50].

[1] can be a small lithium battery

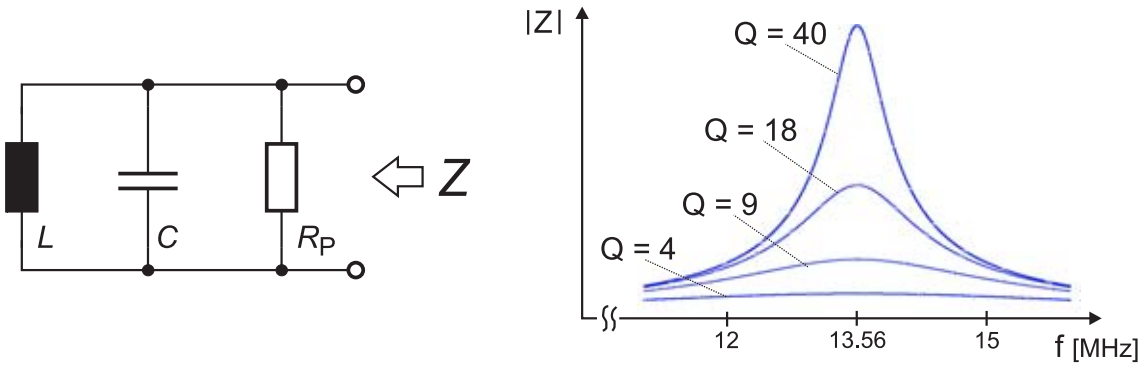


Figure 3.3: Parallel resonant circuit

$$f_0 = \frac{1}{2\pi \cdot \sqrt{LC}} \tag{3.1}$$

In practice, first the value for L is derived from the shape and dimensions of the coil[2], afterwards the optimal C is calculated according to equation 3.2, and then realised as a trimmable capacitor, so that the circuit can be tuned more precisely later on.

$$C = \frac{1}{(2\pi f_0)^2 \cdot L} \tag{3.2}$$

If the serial resistance of the coil, representing ohmic losses in the wire, is omitted, and only a parallel resistor R_P is taken into account, which incorporates the load and the parasitic parallel resistance of the capacitor, the circuit in Figure 3.3 is obtained. The input impedance, as a function of the angular frequency $\omega = 2\pi f$, can then be calculated following equation 3.3.

$$Z(j\omega) = \frac{j\omega L}{1 + j\frac{\omega L}{R_P} - \omega^2 LC} \tag{3.3}$$

The tuned resonant circuit behaves similar to a a band-pass filter, that only lets a certain frequency range pass through it.

Quality Factor and Bandwidth

The resistor R_P and the capacitor C determine the bandwidth B of the circuit [26], as defined in equation 3.4.

$$B = \frac{1}{2\pi \cdot R_P C} \tag{3.4}$$

[2]practical examples can be found in Section 3.1.8

Furthermore, a quality factor Q can be defined, which is usually the ratio of the energy stored to the energy dissipated in a system, but can also be related to the bandwidth, as shown in equation 3.5.

$$Q = \frac{f_0}{B} \tag{3.5}$$

Combining equations 3.1, 3.4 and 3.5, the quality factor Q of a parallel resonant circuit can be rewritten as in equation 3.6, i.e., proportional to the parallel resistance R_P.

$$Q = R_P \cdot \sqrt{\frac{C}{L}} \tag{3.6}$$

Clearly, once L and C are chosen, the Q factor is solely dependent on R_P. The impedance of a parallel tuned circuit reaches a maximum at the resonance frequency. It follows that the induced voltage also reaches a maximum. The amplitude of this maximum is a function of Q and hence the resistance of R_P, which is illustrated on the right side of Figure 3.3.

According to equation 3.7, the absolute value of the input impedance, i.e. at the resonant frequency, is equal to R_P.

$$|Z(j\omega_0)| = R_P \tag{3.7}$$

Furthermore, Figure 3.4 depicts the relationship between bandwidth and quality factor (see equation 3.5). The plots of the impedance of the tuned circuit, normalized to its maximum value, show: The larger the Q, the narrower the bandwidth B, which is of concern for the design of antennas for RFID systems.

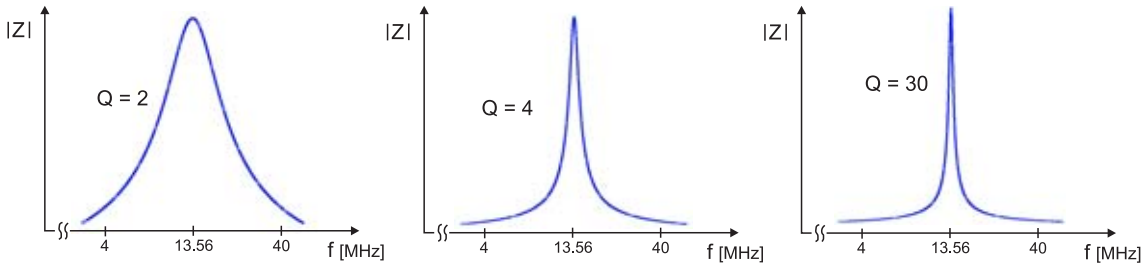


Figure 3.4: Impedance of a parallel resonant circuit, with Q varied

To sum up the coherences, in general, a large Q results in a greater maximum of the induced voltage and therefore a longer read range, at the cost of a decreased bandwidth. This is particularly important for the ISO 14443, because of the relatively high 847 kHz sub carrier frequency. Figure 3.5 illustrates the case at hand, where, for high Q factors, the information in the sidebands of the 13.56 MHz carrier frequency is strongly attenuated, compared to the carrier frequency, thus making it difficult for the reader to acquire the information sent by a tag.

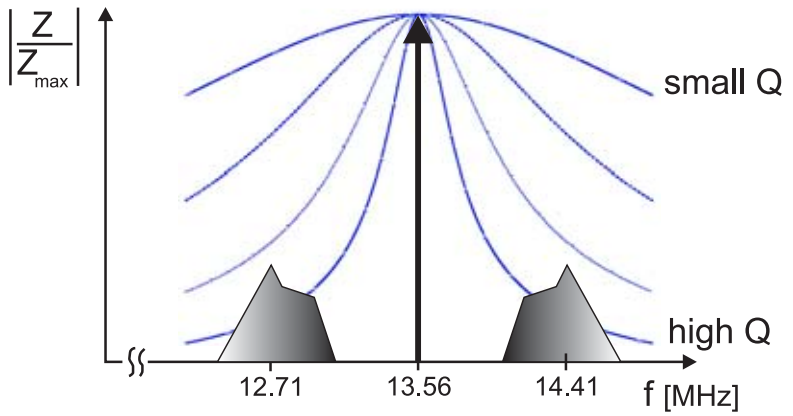


Figure 3.5: Influence of the Q factor on the received signal

For a real system, it is difficult to estimate the Q factor, as the load (resistance) varies significantly during operation of the tag, because it draws its energy from the field, and all its circuitry is connected in parallel to the LC-circuit. Therefore, in practice, the resistance for the optimal Q has to be found experimentally, i.e., by finding the best read range for the concrete system.

3.1.2 Protection Circuit

Due to resonance step up in the parallel resonant circuit [15], the amplitude of the voltage can become relatively large, which may damage the remaining circuitry, e.g., the inputs of the LM 311 comparator (see Section 3.1.7). To limit the maximum possible voltage and protect the sensitive devices, two Zener-diodes (D_1 and D_2) in opposite directions, i.e., anti-serial, and an optional resistor (R_1) in series, are connected in parallel to the LC-tank, as depicted in Figure 3.2.

In the forward direction, the characteristic curve of a Zener diode, presented in Figure 3.6, is similar to the curve of a standard pn-diode, i.e., the diode conducts, if the voltage U_D between anode(A) and cathode(K) becomes larger than $U_F \approx 0.7$ V. In the reverse direction, for a negative U_D, in contrast to a standard diode, which will very likely be destroyed once it starts conducting, a Zener diode is designed to operate with a low resistance in the corresponding operating point, r_z. Connecting two Zener diodes in an anti-serial manner results in no current through the path of the diodes, as if they were not present at all, unless the absolute voltage becomes greater than $U_Z + U_F$, when they suddenly start conducting. Most of the current from the antenna will then flow through the diodes and any too high voltage will be dissipated by them. Here, Zener diodes with a voltage $U_Z = 4.7$ V were chosen, so that no voltage greater than 4.7 V + 0.7 V = 5.4 V will be applied to the other devices on the fake tag, which is within the absolute maximum ratings of all devices present. For R_1, usually a piece of wire should be inserted, unless the maximum current through the diodes shall be limited.

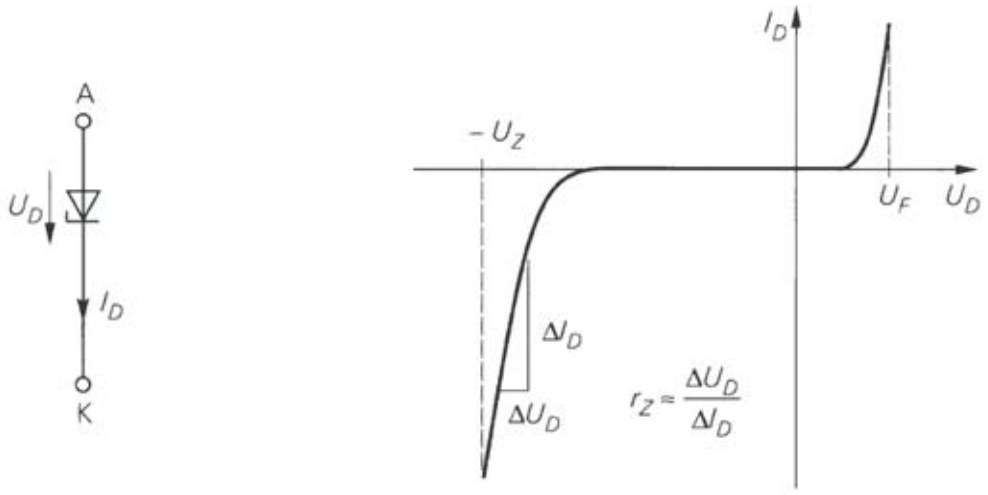


Figure 3.6: Typical characteristic curve of a Zener diode

3.1.3 Generation of a Subcarrier

The subcarrier with a frequency of $\frac{f_c}{16} = 847.5$ kHz is derived from the 13.56 MHz field of the reader. The voltage at the antenna is connected to the input of a 4-bit binary counter 74HC393 [37] through a resistor, which limits the maximum current into the input stage. The CMOS gates at the input of the 74HC393 are protected against damage, e.g. caused by high voltages, by means of internal protection diodes, as long as a maximum diode current of 20 mA is not exceeded [12].

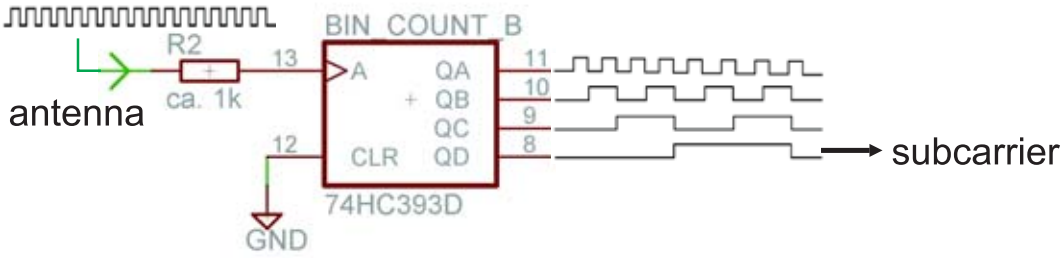


Figure 3.7: Frequency Division by 16 to obtain the Subcarrier

As depicted in Figure 3.7, the output Q_A halves the frequency of the input signal, Q_B halves the frequency of Q_A and so on. The fourth output of the binary counter, Q_D, toggles every $2^3 = 8$ clock cycles, which equals a frequency division by 16, i.e., the desired subcarrier.

3.1.4 Modulation with the Subcarrier

The modulation is achieved by ANDing the incoming Manchester coded signal with the subcarrier, which is output by the frequency divider. As depicted in Figure 3.2, a common 74HC08 [39], containing four two-input AND gates, provides the resulting modulated Manchester code at its output (compare with Figure 2.4). A pin-compatible 7409 chip, providing open collector outputs and thus incorporating switching capability, might be used instead of the 7408, if the induced voltage level is kept small enough.

3.1.5 Load Modulation

A resistor has to be connected in parallel to the antenna of the tag to achieve (resistive) load modulation of the field generated by the reader, as described theoretically in Section 2.2.2.

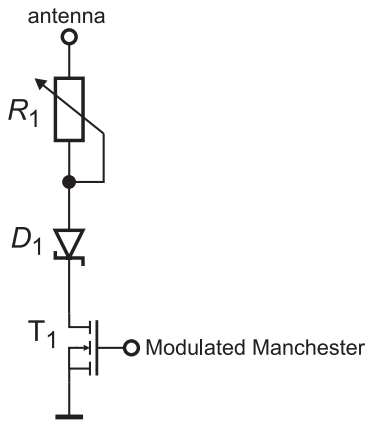


Figure 3.8: Realisation of the switch for the load modulation

Figure 3.8 illustrates, how the aforementioned switch is realised with an IRFD 110 [20] n-channel MOSFET[3], labeled with T_1, allowing for fast switching and a maximum drain-source voltage of 100 V whilst having a low on-resistance of 0.54 Ω. A likewise fast Schottky diode, D_1, in series with the adjustable load resistor R_1, prevents the internal avalanche diode of the MOSFET from conducting during the negative half cycle of the HF field, when a negative voltage is applied between drain and source, which would lead to irreversible damage of the transistor. The output of the AND gate (see Section 3.1.5) is connected to the gate of the transistor, which will toggle the load resistor on, when the gate-source voltage exceeds approximately 3 V. Accordingly, the 848 kHz modulated Manchester code is modulated onto the 13.56 MHz field of the reader and the information put into sidebands of the carrier frequency (compare with Figure 2.3).

Of course, as the n-channel transistor will only conduct when the voltage at the antenna is positive, load modulation only happens during one half cycle of the sine wave

[3]Metal-Oxide Semiconductor Field-Effect Transistor

of the field. Still, good results were obtained with the described circuit. During the pauses, the field is completely switched off for full periods, while in the load modulation case the amplitude at the antenna will rise again after one half cycle. So it is easier for the fake tag to distinct between gaps in the field and load modulation.

3.1.6 Acquire Miller Pulses from the HF field

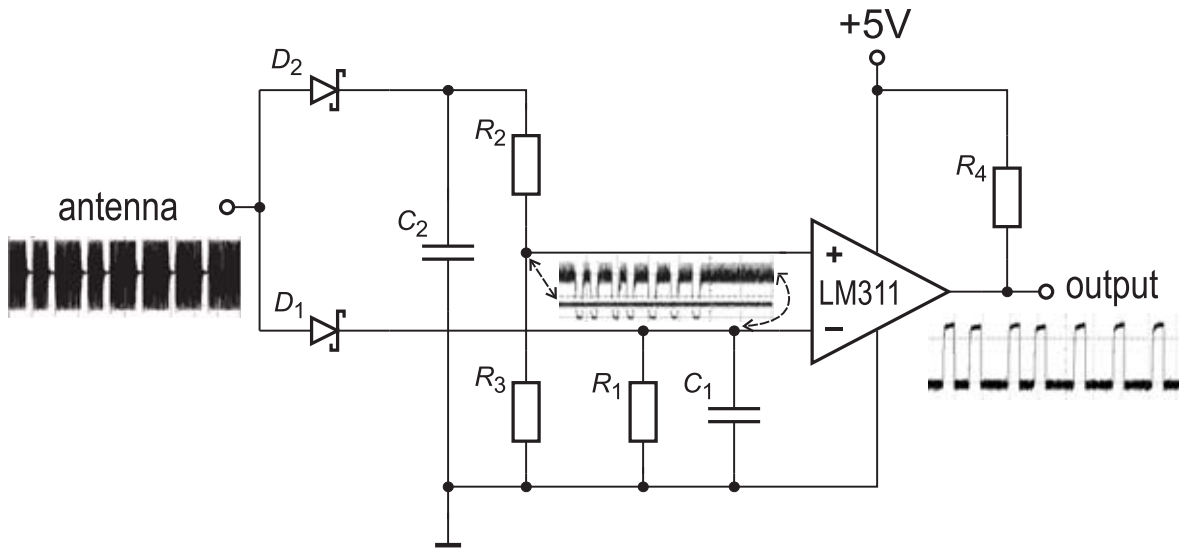


Figure 3.9: The adaptive envelope detector of the Fake Tag

The fake tag has to be able to distinguish between gaps in the HF field, caused by the reader sending data, and itself sending data, i.e. load-modulating the field. Furthermore, in addition to getting rid of the high frequent fraction of the field, a wide voltage range at the parallel LC-circuit must be handled, as the amplitude varies considerably with the distance between the two coils. To achieve this goal, an LM 311 comparator [33] is used, combined with two envelope detectors at its inputs, as depicted in Figure 3.9, which are both connected in parallel to the antenna. The LM 311 is operated from the single 5 V supply present on the PCB and, wired with an appropriate pull up resistor, R_4, at its output[4], capable of producing appropriate 0 and 5 V levels.

During the positive half cycle of the field, the capacitors of the detectors are rather quickly charged via the Schottky diodes. While the input at the diodes is negative, a reverse flowing current is blocked, so that the capacitors can only discharge by means of the connected resistors.

The detector at the negative input of the comparator, formed by D_1, $C_1 = 150$ pF and $R_1 = 1$ kΩ, is dimensioned for a fast response time and distinguishes between the field

[4]for fast reaction, during measurements a value of approximately 2.2 kΩ turned out to be optimal

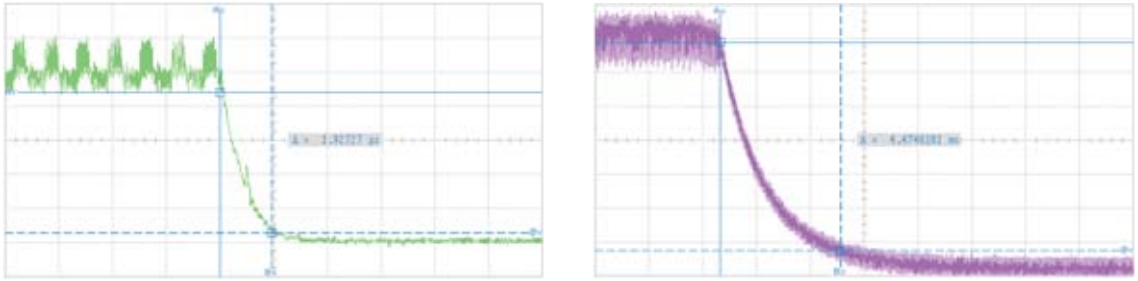


Figure 3.10: Fall times of the *RC*-circuits

being completely switched off and the load modulation case. With the time constant $\tau_1 = 150$ ns, derived in equation 3.8, a fall time of approximately 1.92 μs has been measured, as depicted on the left of Figure 3.10. Note, that the capacitor discharges so quickly, that the 13.56 MHz input signal from the antenna can be recognised in the waveform.

$$\tau_1 = R_1 \cdot C_1 = 1\ k\Omega \cdot 150\ pF = 150\ ns \tag{3.8}$$

$$\tau_2 = (R_2 + R_3) \cdot C_2 = (8.2\ k\Omega + 1.8\ k\Omega) \cdot 220\ nF = 2.2\ ms \tag{3.9}$$

The other envelope detector is formed by D_2, $C_2 = 220$ nF and the voltage divider consisting out of $R_2 = 8.2$ kΩ in series to $R_3 = 1.8$ kΩ. It has a rather large time constant of $\tau_2 = 2.2$ ms, calculated in equation 3.9, and averages the voltage at the antenna, which is then divided by a factor of 5.6, derived in equation 3.10, and then fed into the positive input of the LM 311.

$$(\frac{R_3}{R_2 + R_3})^{-1} = (\frac{1.8\ k\Omega}{8.2\ k\Omega + 1.8\ k\Omega})^{-1} = 5.55 \tag{3.10}$$

As shown in Figure 3.10 on the right side, for this *RC*-circuit, a fall time of approximately 4.5 ms has been measured. The resulting threshold voltage, appearing like a DC voltage during an established communication between reader and tag, is thereby adapted to the current field strength. This makes the circuit immune to noise caused by the HF field, extends the operating range and ensures fast reaction to the gaps in the field.

If the field is completely switched off, so that the voltage of the capacitor at the inverting input becomes smaller than the voltage at the non-inverting input, the output of the comparator will become high, indicating the beginning of a gap in the field, illustrated in the left of Figure 3.11. Zooming into the waveforms, on the right side of the figure, a delay of only 545 ns can be observed, induced by the complete envelope detection stage. The rise time of the output signal is slower compared to the fall time, originating in the open collector output of the LM 311.

3.1.7 Pulsed Miller → Miller

The conversion of the pulses received from the reader to normal Miller code is necessary to reduce the bandwidth needed for the transmission through the communication link (see Section 3.2.8). The output of the comparator is connected to the input of a positive edge triggered 7474 D-type flip flop [41], whose inverted output is fed back into the D input, as depicted in Figure 3.12, leading to a change of the logic state at the output on every rising edge occurring. The result of the obtained conversion from pulses into transitions is called a Miller coded signal and wired to the communication interface, to be forwarded to the reader, where the pulses are reestablished and fed into the DIN input of the RF transceiver and an input pin of the microcontroller (compare with Section 3.2.4).

The power-on state of the flip flop is undefined, but this does not mean a problem, because, as illustrated in Figure 2.2, Miller coded bits are represented by transmissions, not by levels. The measured function of the stage is presented in Figure 3.13, where the voltage of the antenna is on top, the acquired Miller pulses below, and the Miller coded signal with a low bandwidth, for transmission over the communication channel, at the bottom.

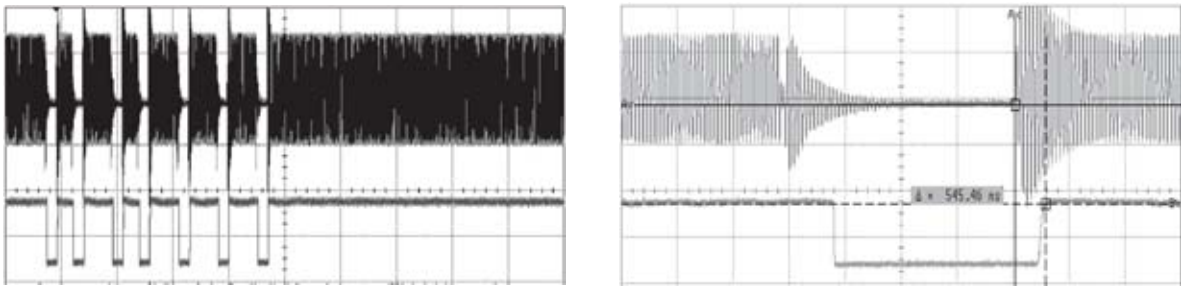


Figure 3.11: Delay induced by the envelope detector

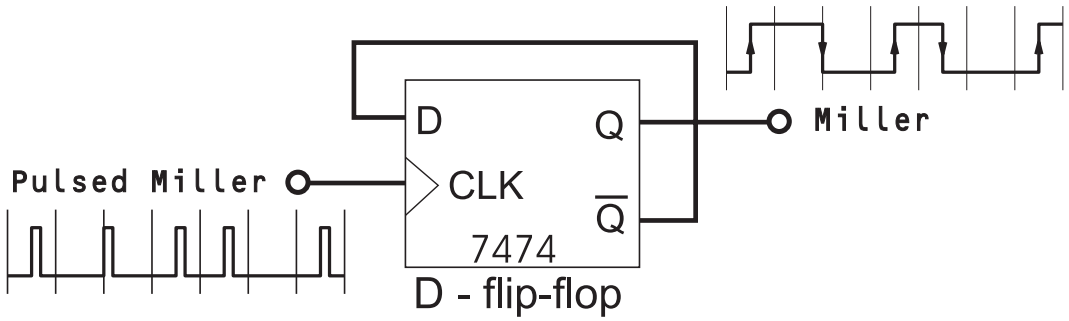


Figure 3.12: Conversion of Miller Pulses to normal Miller coded data

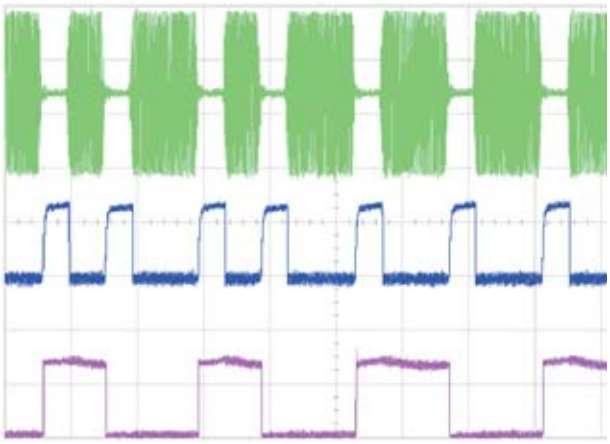


Figure 3.13: Transformation of the signal between antenna and communication interface

3.1.8 Fake Tag Design Flow

The Coffee Cup Tag

To perform first tests, regarding the performance and tunability of a self made parallel resonant circuit for 13.56 MHz, and to develop an expedient circuit to achieve proper load modulation, a simple but effective approach was chosen: A coffee cup, being the first obvious object at hand with the corresponding shape, was used to form a circular coil, and other components were wired directly to it, as shown in Figure 3.14.

If the diameter d of the wire used is much smaller than the diameter of the coil, the approximation in equation 3.11 can be used [15] for the calculation of the inductance L of a circular conductor loop.

$$L = N^2 \mu_0 R \cdot ln(\frac{2R}{d}) \tag{3.11}$$

The number of windings, N, was chosen equal to three, and the coated copper wire used has a diameter of $d = 0.5$ mm, while the radius of the coffee cup was found to be $R = 40$ mm. Inserting these values and the magnetic constant $\mu_0 = 4\pi \cdot 10^{-7} \frac{Vs}{Am}$, i.e. the permeability of vacuum, into equation 3.11, results in the inductance of the copper wire coil calculated in equation 3.12.

$$L = 9 \cdot 4\pi \cdot 10^{-7} \ \frac{Vs}{Am} \cdot 40 \cdot 10^{-3} \ m \cdot ln(\frac{2 \cdot 40 \cdot 10^{-3} \ m}{5 \cdot 10^{-4} \ m}) = 2.3 \ \mu H \tag{3.12}$$

The necessary parallel capacitor with a value of $C = 59.9$ pF, deduced from equation 3.2, is realised as a fixed 47 pF ceramic capacitor in parallel to an adjustable one with a range from 4 pF to 30 pF, hence tuning to resonance is possible.

The Coffee Cup Tag turned out to be suitably tunable and was initially capable of performing load modulation with a subcarrier, i.e., the subcarrier could be either

Figure 3.14: The Coffee Cup Tag

switched on or off, which then could be noticed at the amplitude of the measured field and at the signal at the DOUT pin of the reader (described below in Section 3.2). The form of the coil was later on fixed with superglue, to ensure mechanical long term stability.

The Fake Tag, Version 1

The Coffee Cup Tag was further extended with more components required for operation, resulting in a rather unconventional and unreliable appearance, depicted in Figure 3.15.

After testing several options for the circuit, the best variant was realised on a PCB, resulting in the first durable version of a device being able to emulate an ISO 14443 compliant RFID transponder, termed Fake Tag, which is presented in Figure 3.16.

The inductance of the coil was determined to $L = 1.25\ \mu$H, leading to a corresponding capacitance of $C = 110$ pF, again realised as a 100 pF fixed capacitor in parallel to a 6...30 pF variable capacitor. The one-sided layout, which was produced using the Layout

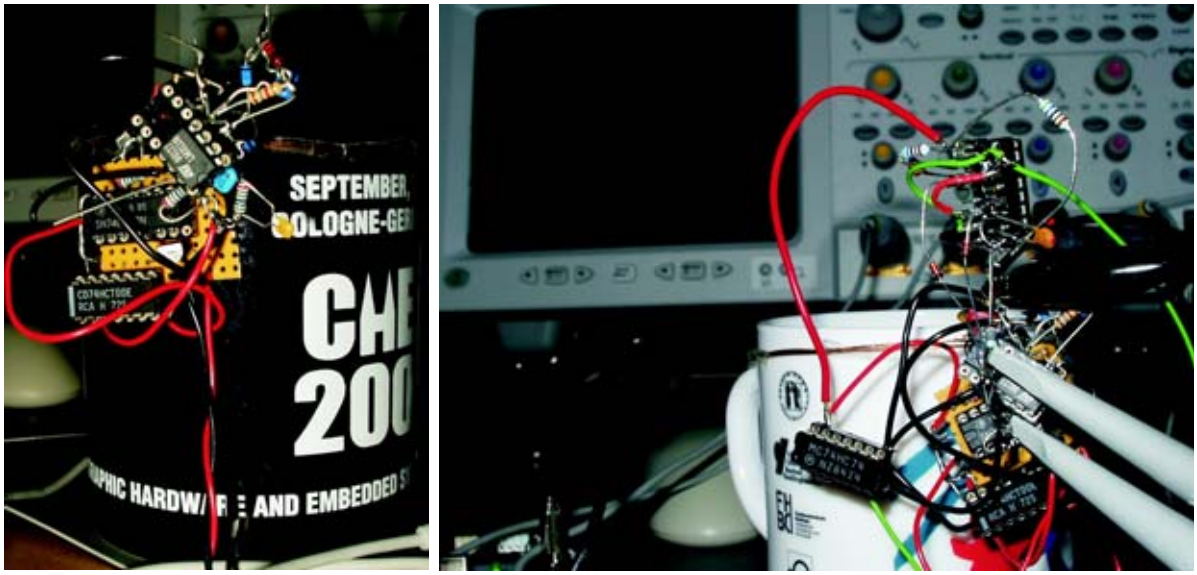


Figure 3.15: Experimental extensions of the Coffee Cup Tag

Editor EAGLE 4.13 from CadSoft[5], employs SMD[6] technology to keep the dimensions of the device small, and the wires short, which is particularly important for high frequent signals.

The Fake Tag, Version 2

For the second (and final) version of the Fake Tag, the complete circuitry is placed inside of the coil, thus achieving a larger coil area and longer operating range. Furthermore, as this time a two-sided layout has been designed, the number of windings of the antenna is doubled. Concerns about the strong magnetic field in the coil, potentially perturbing the functional performance of the designed circuit, turned out to be baseless during pertinent tests, if the integrated circuits are properly wired with bypass capacitors close to their pins, to reduce the noise in the supply voltage.

The resistor for the load modulation is realised as a variable SMD type, and the size of the PCB is adapted to fit into a standard cigarette packet (shown on the right of Figure 3.17), so that it can be easily hidden, e.g., during a real world relay attack.

For calculation of the inductance of the multilayer rectangular antenna, depicted in Figure 3.18, its spiral nature is neglected, i.e., the width and the height of the cross section is assumed to be much smaller than the width and the length of the coil, so that equation 3.13 can be used to find an estimation for the value of the inductance [26]. Inserting the dimensions in cm, the inductance is obtained in μH.

[5] `http://www.cadsoft.de`
[6] Surface Mounted Device

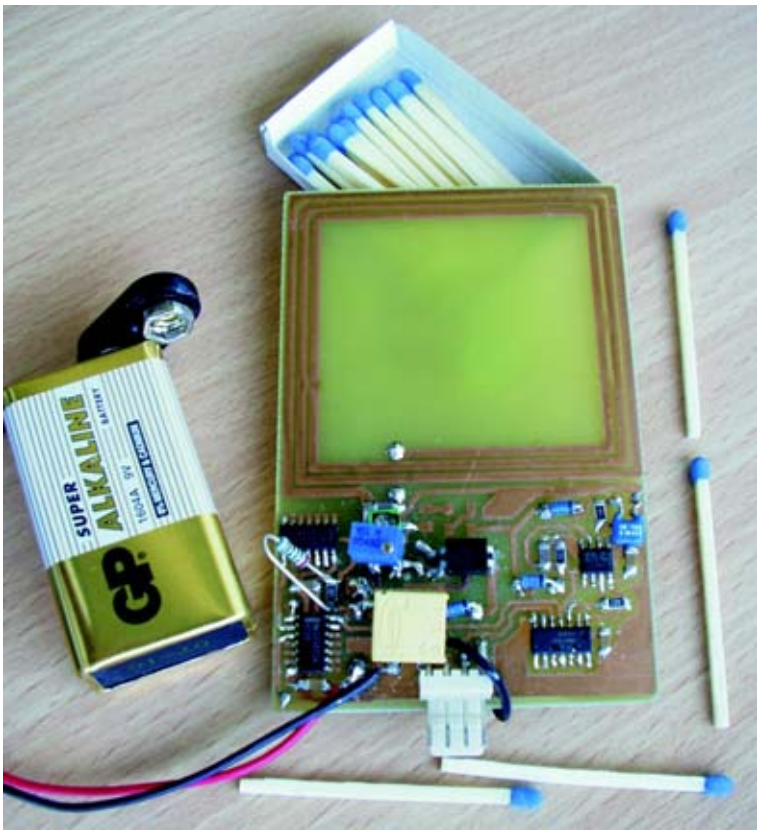


Figure 3.16: The Fake Tag, version 1

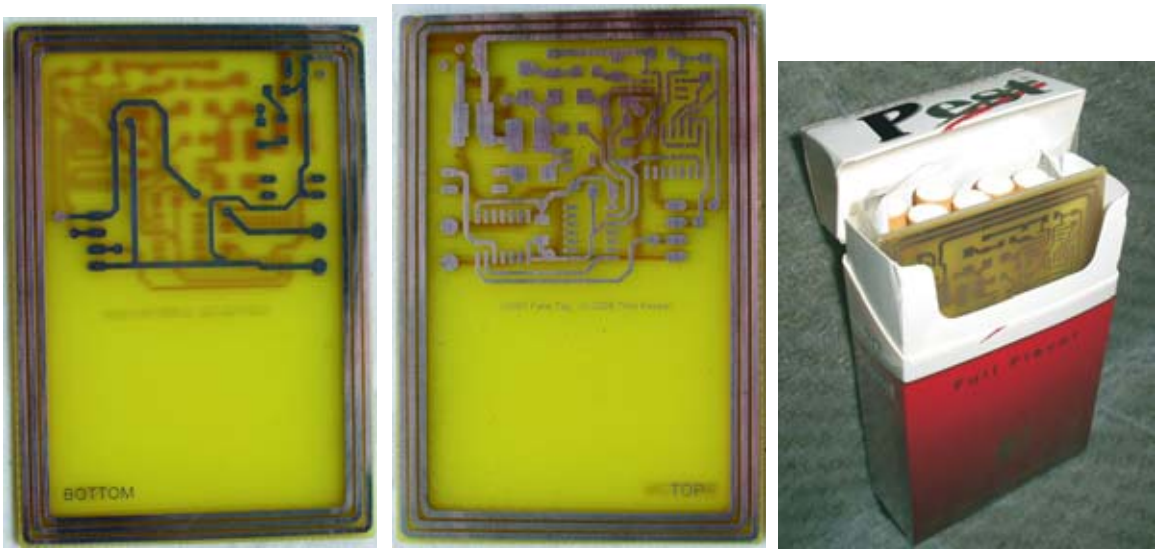


Figure 3.17: The PCB of version 2 of the Fake Tag

$$L = \frac{0.0276 \cdot (CN)^2}{1.908C + 9b + 10h} \tag{3.13}$$

If w denotes the width and l the length of the coil, while b and h refer to the width and the height of its cross-section, C in equation 3.13 is equal to $C = w + l + 2h$, i.e., $C = 5$ cm+7.5 cm+$2 \cdot 0.1$ cm= 12.7 cm. Accordingly, the second version of the Fake Tag, with the number of turns $N = 6$, the height of the cross-section $h = 0.1$ cm and the width of the cross-section $b = 0.4$ cm, has an inductance of $L = 5.56$ μH, as derived in equation 3.14.

$$L = \frac{0.0276 \cdot (12.7\ cm \cdot 6)^2}{1.908 \cdot 12.7\ cm + 9 \cdot +10 \cdot 0.1\ cm)} = 5.56 \mu H \tag{3.14}$$

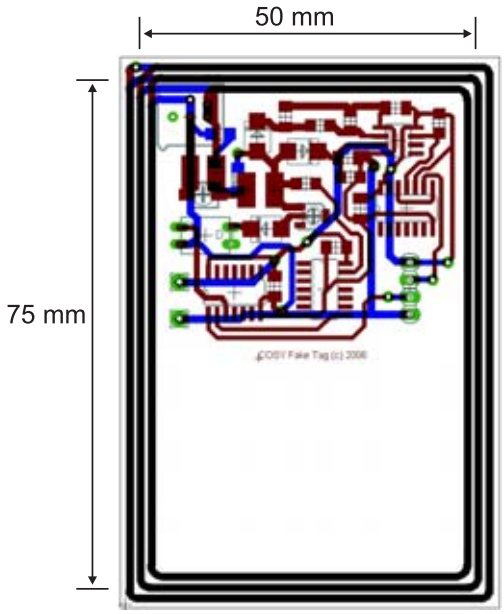


Figure 3.18: Layout and dimensions of the Fake Tag, version 2

As above, the value of the capacitor to be connected in parallel, for a resonance frequency of 13.56 MHz, is calculated from equation 3.2 and found to be approximately $C \approx 25$ pF, so that a single adjustable (SMD-) capacitor of 6...30 pF should be sufficient.

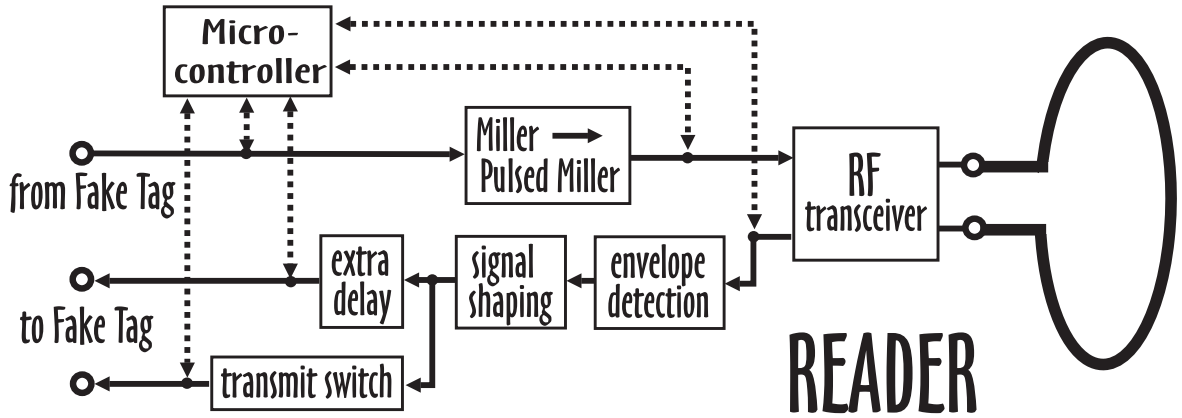


Figure 3.19: The Reader

3.2 The Reader

3.2.1 The RF Transceiver

The main part of the analogue front end is provided by the EM 4094 RF-transceiver[7] from EM Microelectronics, which possesses a 200 mW push pull transmitter operating at 13.56 MHz using an external quartz crystal, is capable of 100% ASK and ready for ISO 14443A operation at a price of less than 5 €. According to the fact sheet [11], the device is also dedicated for operation compliant to the ISO 14443B or ISO 15693 standards and provides interoperability with NFC devices. The received HF-Signal is demodulated and can be conditioned by an internal 400 kHz to 1 MHz lowpass filter, a 100-, 200-, or 300 kHz highpass filter and selectable receiver gain, thus being able to process the required subcarrier frequency of 848 kHz (see Section 2.2.2).

The chip is well suited for the application described here, as its operation is transparent, i.e., a high input level on the DIN pin will instantly switch off the field, while a low level switches it on, enabling flexible, direct control of the RF field. The 848 kHz signal received from the tag is output at the DOUT pin of the chip, from where it has to be further processed before being treated, e.g., by the microcontroller described in Section 3.2.9.

Several option bits need to be programmed into the chip to set it up for the desired operation mode, which is done after every power-on by the microcontroller, via a three pin serial interface.

Unfortunately, to gain access to the full data sheet of the EM 4094, an NDA[8] form, available from the website[9], has to be filled in. Note, that both Melexis' MLX90121 [31] and the S6700 Multi-Protocol Transceiver IC [47] from Texas Instruments offer possibil-

[7] transmitter and receiver
[8] Non Disclosure Agreement
[9] www.emmicroelectronics.com

ities very similar to the EM 4094 and are therefore suitable replacements.

3.2.2 Impedance Matching

For convenience, the output stage of the chip has been matched to feed the signal into a common 50 Ω coaxial cable, so different antennas can be tested by plugging them into the SMA connector placed on the PCB[10].

At the frequency of 13.56 MHz, the HF voltage has to be treated as an electromagnetic wave, and undesired effects like power reflections have to be taken into account. The reflection coefficient Γ, i.e., the ratio of the amplitude of the reflected wave to the incoming wave, is a measure for the reflected power. It can be derived from the output impedance of the source, Z_L, and the characteristic impedance of the transmission line connected to it, Z_0, according to equation 3.15.

$$\Gamma = \frac{Z_L - Z_0}{Z_L + Z_0} \tag{3.15}$$

For Z_L being equal to Z_0, the reflection coefficient will become zero, indicating that no power is reflected back into the source. Accordingly, to minimise losses and achieve the maximum possible power transmission from the output stage of the reader into the coaxial cable, the impedances have to be matched, which can be realised with a passive matching circuit using only a few components.

A method of visualising complex impedances and the corresponding reflection coefficient is the so called Smith Chart [29], depicted in Figure 3.21, in which the entire right half plane of the complex impedance plane is mapped into a circle. Before drawing the involved impedances into the chart, they have to be normalised to the impedance of the transmission line, Z_0 (which here equals 50 Ω), resulting in equation 3.16, where Z_L^* denotes the normalised impedance of the source, i.e. $Z_L^* = \frac{Z_L}{Z_0}$.

$$\Gamma = \frac{Z_L^* - 1}{Z_L^* + 1} \tag{3.16}$$

In a Smith Chart, impedances connected in series can directly be added, while those connected in parallel are obtained by adding the admittances, which are graphically created by rotating the impedance by 180°. The distance from the center of the chart to the outside of the circle is the reflection coefficient Γ, which is particularly convenient to perform impedance matching, as it is shown for the output stage in the following section.

3.2.3 The RF Output Stage

The output impedance of each of the antenna outputs ANT1 and ANT1 of the RF transceiver is assumed to be resistive with 10 Ω each. To eliminate the DC component,

[10]Printed Circuit Board

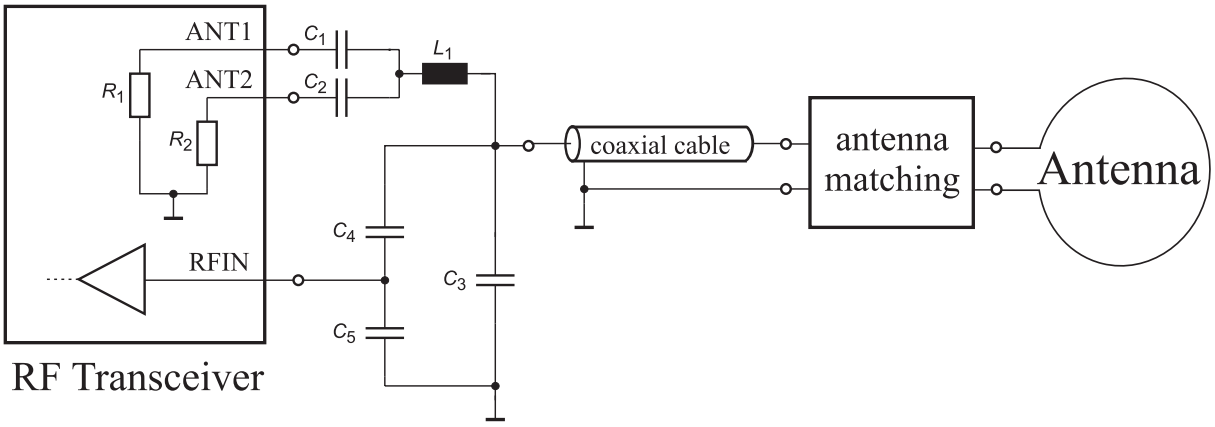


Figure 3.20: Schematic of the Output Stage

a 680 pF capacitor (C_1 and C_2 in Figure 3.20) is connected in series to each output, which at the frequency of 13.56 MHz results in an impedance equal to Z_1, as derived in equation 3.17.

$$\begin{aligned} Z_1 &= \frac{(R_1 + \frac{1}{j\omega C_1}) \cdot (R_2 + \frac{1}{j\omega C_2})}{(R_1 + \frac{1}{j\omega C_1}) + (R_2 + \frac{1}{j\omega C_2})} \\ &= \frac{(10\ \Omega + \frac{1}{j \cdot 2\pi \cdot 13.56\ MHz \cdot 680\ pF}) \cdot (10\ \Omega + \frac{1}{j \cdot 2\pi \cdot 13.56\ MHz \cdot 680\ pF})}{(10\ \Omega + \frac{1}{j \cdot 2\pi \cdot 13.56\ MHz \cdot 680\ pF}) + (10\ \Omega + \frac{1}{j \cdot 2\pi \cdot 13.56\ MHz\ \cdot 680\ pF})} \\ &= 5 - j \cdot 8.6302\ \Omega \end{aligned} \tag{3.17}$$

The normalised impedance, i.e. $\frac{Z_1}{50\ \Omega} = 0.1 - j \cdot 0.173\ \Omega$, is marked with an encircled 1 in Figure 3.21. An inductance of 285 nH is connected in series to obtain the impedance calculated in equation 3.18, where the (parasitic) resistance of the coil, $R_i = 0.45\ \Omega$, is taken into account.

$$R_i + j\omega L = 0.45\ \Omega + j \cdot 2\pi \cdot 13.56\ MHz \cdot 285\ nH = 0.45 + j \cdot 24.28\ \Omega \tag{3.18}$$

The normalised value, $0.009 + j \cdot 0.486\ \Omega$, is added to the impedance 1 in the Smith Chart, to obtain the point marked with a 2, corresponding to a normalised impedance of $0.109 + j \cdot 0.313\ \Omega$. To determine the total capacitance to be connected in parallel, now the admittance,labeled with 3, has to be taken by mirroring at the origin (dashed line in Figure 3.21). From here, the centre of the Smith Chart, where the reflection coefficient is $\Gamma = 0$, can obviously be reached by adding a normalised imaginary part of $j \cdot 2.85\ \Omega$, corresponding to an overall capacitance $C_{tot} = 669$ pF.

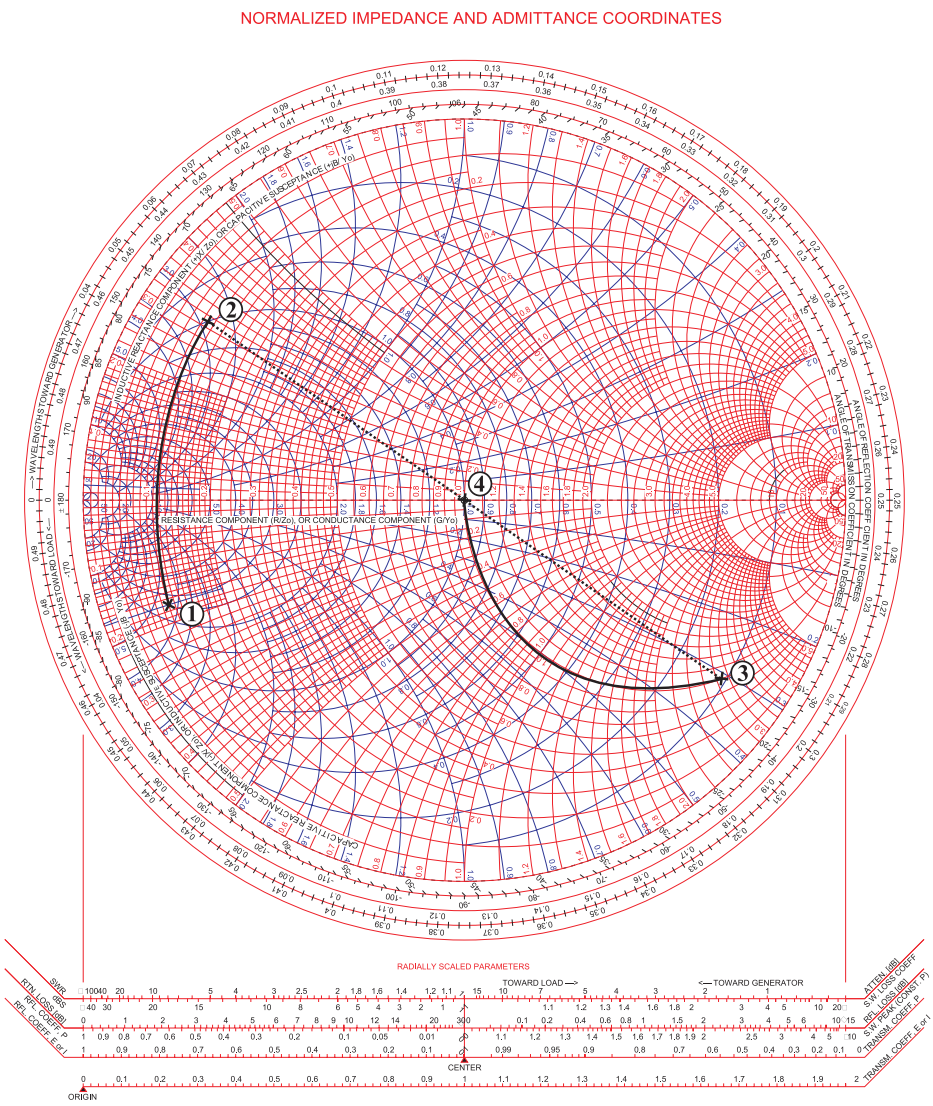


Figure 3.21: Impedance Matching with a Smith Chart

Reception Stage

During operation, due to resonance step up in the tuned circuit, peak to peak voltages U_{Ant} between 10 V and 15 V have been measured at the end of the antenna. As the amplitude at the RFIN input of the EM 4094, U_{RFIN}, must not exceed 5 V for proper reception of the incoming signal, C_4 and C_5 in Figure 3.20 form a capacitive voltage divider, through which the signal is fed into the RFIN pin.

$$U_{RFIN} = \frac{\frac{1}{j\omega C_5}}{\frac{1}{j\omega C_4} + \frac{1}{j\omega C_5}} \cdot U_{Ant} = \frac{C_4}{C_4 + C_5} \cdot U_{Ant}$$

$$= \frac{270\ pF}{270\ pF + 510\ pF} \cdot U_{Ant} = 0.346 \cdot U_{Ant} \tag{3.19}$$

As derived in equation 3.19, with $C_4 = 270$ pF and $C_5 = 510$ pF, the amplitude at the input of the transceiver is reduced to a reasonable level of approximately one third of the antenna voltage, thus meeting the specifications of the transceiver.

The equivalent capacitance of C_4 connected in series to C_5 is calculated according to equation 3.20.

$$\frac{C_4 \cdot C_5}{C_4 + C_5} = 177\ pF \tag{3.20}$$

Hence, a further capacitance C_3, with a value of 669 pF$-$177 pF$\approx$ 490 pF, is to be connected in parallel to obtain the total capacitance of $C_{tot} = 669$ pF, which is required for the desired impedance matching, as derived in Section 3.2.3.

With the above described method, the impedances of the amplifier of the transceiver and the coaxial cable are made equal, and power is transmitted with almost no losses through the waveguide to the antenna. There, a similar matching circuit is required, to adapt the antenna to 50 Ω. The required components can be found for each particular antenna, for example with the help of the Smith Chart, again.

3.2.4 Pulse Creation

In accordance to the ISO 14443A, pulses with a duration of approximately 2.5 μs have to be created. This is achieved using a monostable multivibrator (monoflop) of the 74123 type [42], wired with an external capacitor C_{EXT} and a resistor R_{EXT}, whose values are calculated after equation 3.21, out of the datasheet. In the equation, K is a voltage dependent constant, which, for a 5 V supply of the chip, is equal to 0.45, and t_W stands for the width of the output pulse.

$$t_W = K \cdot R_{EXT} \cdot C_{EXT} \tag{3.21}$$

Hence, with $C_{EXT} = 2.2$ nF and $R_{EXT} = 2.7$ kΩ, a pulse width of

$$t_W = 0.45 \cdot 2.7\ k\Omega \cdot 2.2\ nF = 2.67\ \mu s \tag{3.22}$$

is achieved.

As depicted in Figure 3.22 , one half of a 74123 (containing two monoflops) is connected to an output pin of the microcontroller. If it detects a rising edge at its input, a high pulse with the mentioned duration is emitted to the DIN input pin of the EM 4094, resulting in the field being switched off briefly. The workload for the microcontroller is lessened this way, so it has some time, for example to prepare the next data to be sent.

Still, as there is also a direct connection from an output pin of the μC to the DIN input, different pulse widths are achievable, at the cost of more processing time by the microcontroller.

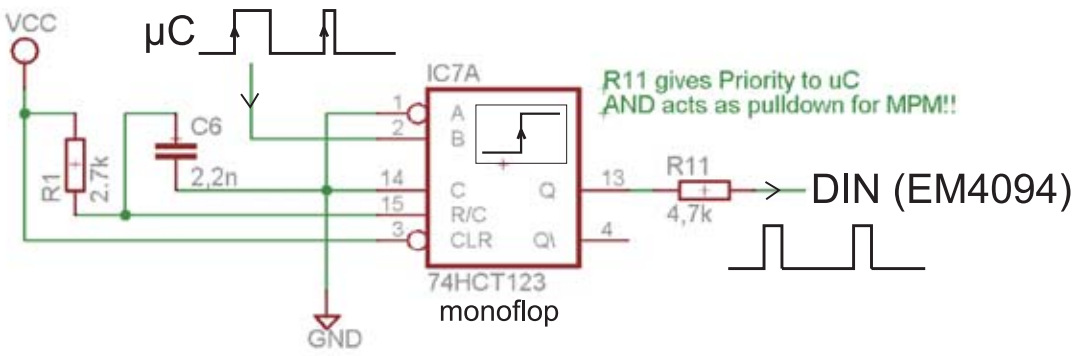


Figure 3.22: Wiring of the monoflop for generation of pulses

3.2.5 Miller → Pulsed Miller

Two more monoflops are utilised to convert the Miller coded data, received from the communication interface or generated by the microcontroller, into pulsed Miller coded data, as depicted in Figure 2.2, which is again fed into the DIN pin of the transceiver. The inputs of two chips are wired in such a manner, that a transition of any type leads to a pulse, as shown in Figure 3.23.

The pull-down resistor, required for an adequate low level at the output of the stage, can be found at the output of the monoflop in Figure 3.22, labeled with R_{11}. The output (pin 13) of the monoflop can be treated as a virtual ground (while it is not emitting pulses), because the 74123 data sheet [42] specifies a maximum output sink current of 25 mA, and the chip is therefore capable of pulling the left side of R_{11} close enough to 0 V, in the context of the here developed circuit.

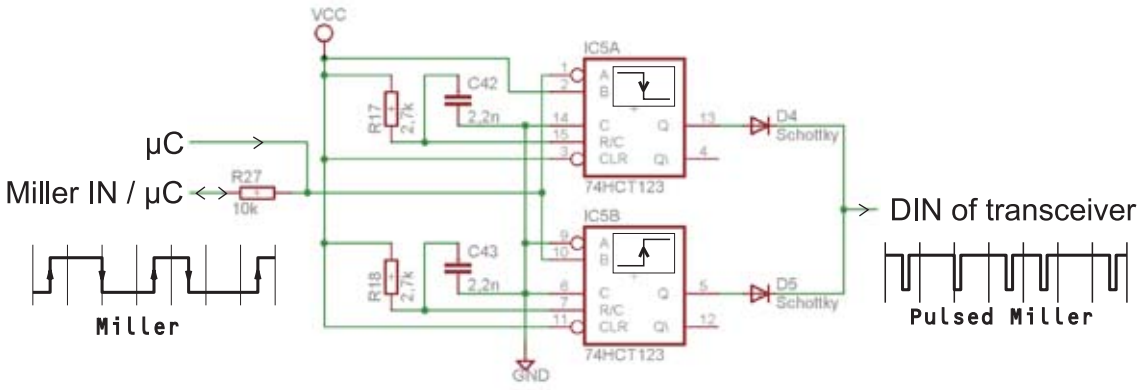


Figure 3.23: Recreation of pulses from the Miller coded input data

3.2.6 Modulated Manchester → Manchester

The output at the DOUT pin of the EM 4094 is modulated with a 848 kHz subcarrier, making it difficult to acquire the data on the side of the Atmel and requiring a high bandwidth for the communication channel to the fake tag (see Section 3.2.8). To lower the bandwidth significantly and make it easier for the μC to perceive the data sent by the tag, the modulated Manchester code is demodulated, as explained below in this section. For further details and explanations regarding the schematic and corresponding signal waveforms, please refer to Figure 3.25 and Figure 3.26, in which the whole demodulation process is illustrated.

Preparation of the DOUT Signal

Unfortunately, the output of the EM 4094 exhibits a non-ideal behaviour, as depicted in Figure 3.24. Deviant from the ideal waveform, the real signal may start with a high instead of a low level, and the last pulse of each half bit cycle is elongated. If directly fed into an envelope detector, the high level at the beginning of the non-ideal waveform would misleadingly result in the circuit indicating a modulation being present, while the last elongated pulse would lead to a longer delay of the output signal and hence a displaced transition (which should be at the centre of the bit period, compare with Section 2.2.2).

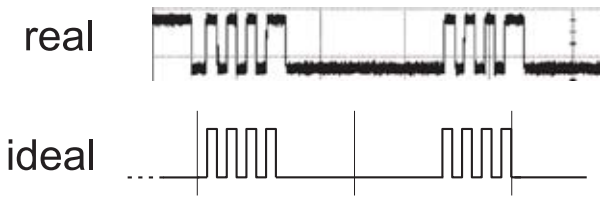


Figure 3.24: Ideal and real signal at the DOUT pin of the EM4094 transceiver

The mentioned behaviour is accounted for by using another 74123 monoflop, labeled with MONFLOP1B in Figure 3.25, which generates short pulses at every rising edge of the signal at the DOUT pin of the RF transceiver. With $C_{EXT} = 150$ pF and $R_{EXT} = 5.6$ kΩ, the pulse duration will be approximately 380 ns , as derived in equation 3.23.

$$t_W = 0.45 \cdot 5.6\ k\Omega \cdot 150\ pF = 378\ ns \tag{3.23}$$

Envelope Detection

The resulting waveform, labeled with 2 in Figure 3.26, is fed into a resistance-capacitance circuit via a diode, similar to the envelope detection circuit of the Fake Tag, described in Section 3.1.6. This time, as derived in equation 3.24, the voltage at the non-inverting

input of the comparator, U_+, is held on a constant level of $\approx$ 650 mV by a resistive voltage divider formed out of R_8 and R_9.

$$U_+ = 5\ V \cdot \frac{R_9}{R_8 + R_9} = 5\ V \cdot \frac{1.5\ k\Omega}{10\ k\Omega + 1.5\ k\Omega} = 652.2\ mV \tag{3.24}$$

During simulations with PSpice, a time constant τ_{man} of the RC-circuit (R_{10} and C_{13} in Figure 3.25), as derived in equation 3.25, turned out to be the optimal trade-off between reliability and fast reaction time of the circuit.

$$\tau_{man} = R_{10} \cdot C_{13} = 2.2\ k\Omega \cdot 470\ pF \approx 1\ \mu s \tag{3.25}$$

The corresponding signal for the voltage divider is labeled with a 4, while the voltage of the capacitor at the non-inverting input of the operational amplifier is marked with a 3 in Figures 3.25 and 3.26.

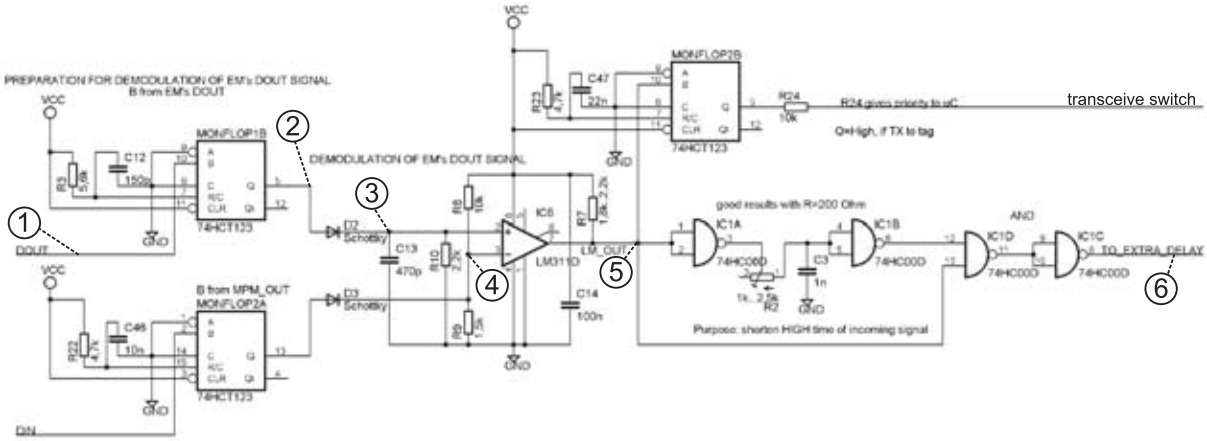


Figure 3.25: The envelope detector of the reader with surrounding circuitry

Depending on the voltage at the capacitor, an LM 311 voltage comparator [33] decides, whether the subcarrier is currently present or not, resulting in the output waveform labeled with [5]. Having a closer look, a longer high time compared to the low time of the signal, caused by the above described demodulation process, can be noticed.

Signal Shaping

The mentioned uneven high and low time is corrected by the circuit following the comparator, consisting out of a 7400 containing four two-input NAND gates [38], a variable resistor and a fixed capacitor. The signal coming in from the LM 311 is split up in two paths, one of which leads directly to one of the two inputs of an AND gate, formed out of two NAND gates. The inputs of the remaining two NAND gates of the 7400 are shorted, thus acting as inverters, which ensure steep edges of the signal passing through them,

while being delayed in between them by means of the *RC*-circuit, whose time constant τ can be adjusted with the variable resistor.

If a rising edge occurs at the input of the signal shaping stage, and therefore at one of the two inputs of the AND gate, the output will not change, i.e., be kept low, until the signal from the delayed path arrives from the output of the second inverter. Both inputs of the AND gate now being high, its output will eventually also become high, while the output level will at once change to low, if the incoming signal becomes low.

Hence, only the rising edge is delayed, whereas the point in time of the falling edge will remain unchanged, at last resulting in the high time of the signal being shortened by an adjustable amount, and, if properly set up, in normal Manchester encoded data at the output of the demodulation stage, labeled with a 6 in Figure 3.26.

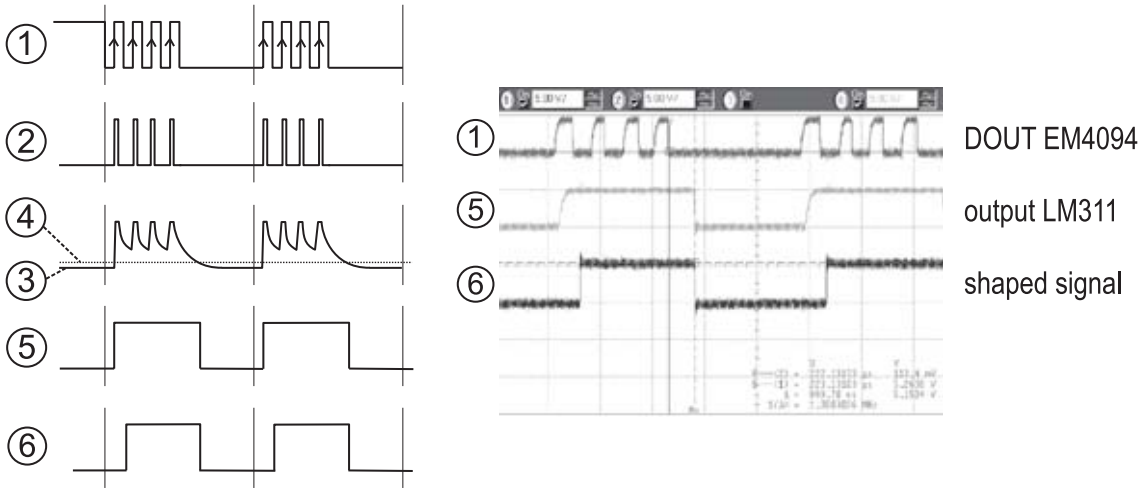


Figure 3.26: Step by step: Demodulation of the transceiver's DOUT signal

Blocking the DOUT During Transmission

An undesired effect is, that the EM 4094 senses its own RF output, leading to a meaningless signal at its DOUT pin during the transmission of data, which is not important for normal reader operation, but would lead to a faulty performance during a relay attack, if it was forwarded to the fake tag.

The situation is depicted in Figure 3.27: While pauses are created in the RF field (waveform at the top), the DOUT output (waveform in the middle) toggles randomly. Preventing this vacant signal from being relayed is the task of the monoflop at the left bottom of Figure 3.25, whose input is connected to the DIN pin of the EM 4094, thus emitting a high pulse with a duration of $t_{block} \approx 21\ \mu$s, according to equation 3.26, on occurrence of a rising edge at the DIN pin.

$$t_{block} = 0.45 \cdot R_{22} \cdot C_{46} = 0.45 \cdot 4.7\ k\Omega \cdot 10\ nF = 21.15\ \mu s \tag{3.26}$$

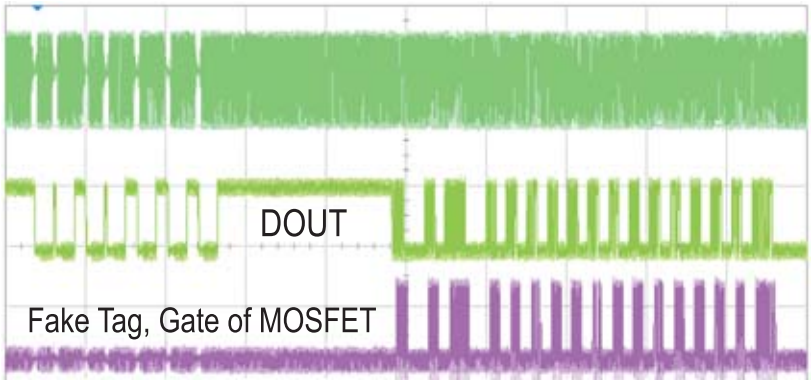


Figure 3.27: Antenna field, DOUT of EM4094 and relayed signal at the fake tag

The output of the monoflop is connected to the inverting input of the LM 311 via a Schottky diode, thus increasing the threshold voltage of the comparator during a pulse almost up to the level of the supply voltage, so that the output of the comparator is maintained low. The 74123 is retriggered on every rising edge at the DIN input, leading to a constant high output of the monoflop, preventing the data at the DOUT pin from being relayed until approximately 20 μs after the last pulse applied to the DIN input. The time t_{block} is chosen longer than two bit durations (9.44 μs at 106 kBit/s) and shorter than the minimum FDT of 86 μs (see Section 2.2.4), after which the tag will answer at the earliest.

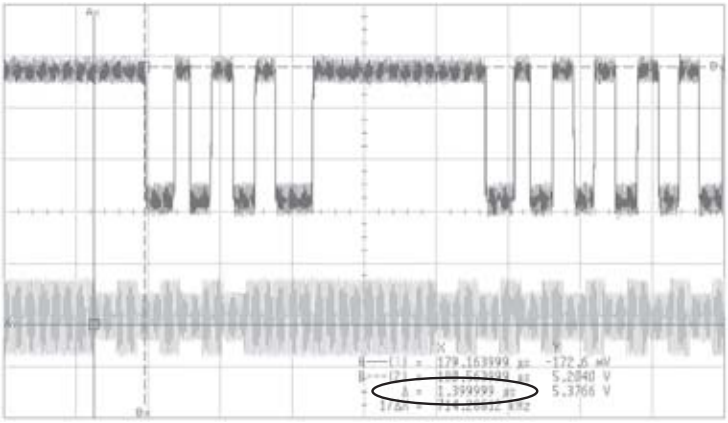


Figure 3.28: Delay induced by the Internal Signal Processing of the EM4094 Transceiver

Fortunately, the internal circuitry of the EM4094 transceiver induces a time delay, between a change of the field at the antenna and its effect at the DOUT pin. This is depicted in Figure 3.28, in which a reaction at the DOUT pin (upper waveform) commences some time after the load modulation at the antenna (lower waveform). During measurements, this delay time was found to be approximately 1.4 μs, while, according

to the data sheet [42], the propagation delay of a 74HC123, between a rising edge at the input and a pulse emerging at the output, is well below 100 ns, even under the worst conditions.

As the monoflop reacts much faster to the input data sent to the DIN pin, than the RF transceiver processes the information obtained from the field, relaying of the data from the DOUT pin to the fake tag is effectively blocked, long before the first impacts of the field being switched off are noticeable at the DOUT pin. The result can be surveyed in Figure 3.27 at the bottom, where, during a true working relay attack, no faulty signal is relayed to the gate of the transistor of the fake tag.

3.2.7 Extra Time Delay

As the demodulation of the signal received from the RF transceiver costs some time (in this case ≈ 1.5 μs), it can happen, that the answer of the tag is not accepted when being relayed to a remote reader (investigated in Section 4.3.1), because it is not well synchronised with the bit grid defined in Section 2.2.4.

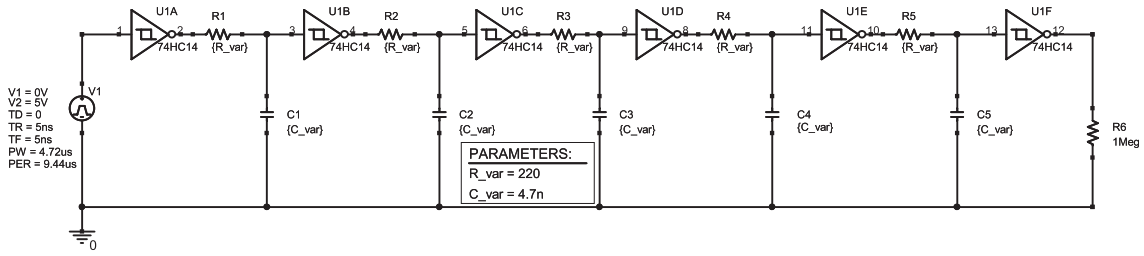


Figure 3.29: Schematic of the Extra Delay

For this case a delay circuit, depicted in Figure 3.29, has been developed, with which a short fixed time delay can be added to the outgoing signal, without altering the waveform. The delay can be varied from 0 to approximately 7 μs by setting a jumper on the PCB, so that a point in time during the bit period of $(106\frac{kBit}{s})^{-1} = 9.44$ μs can be adjusted, for which the relayed answer of a tag is accepted as valid.

The circuit consists of a 74HC74, containing six inverting Schmitt Triggers with a typical hysteresis voltage of 0.9 V [41], combined with six resistor-capacitor pairs, each having identical values. Due to the charging and discharging of the capacitor through the resistor, a time delay is created after every inverter. The optimal values, which are 220 Ω for the resistors and 4.7 nF for the capacitors, were found during simulations performed with PSpice, such, that the maximum possible time delay was achieved without a noticeable change of the waveform of the input signal. As the stages are connected in series, the achieved time delay for the whole circuit is equal to the sum of the six individual delays.

In Figure 3.30, some results of the simulations are presented. The upper left graph shows the input and the (dashed) output signal of a typical Manchester encoded signal

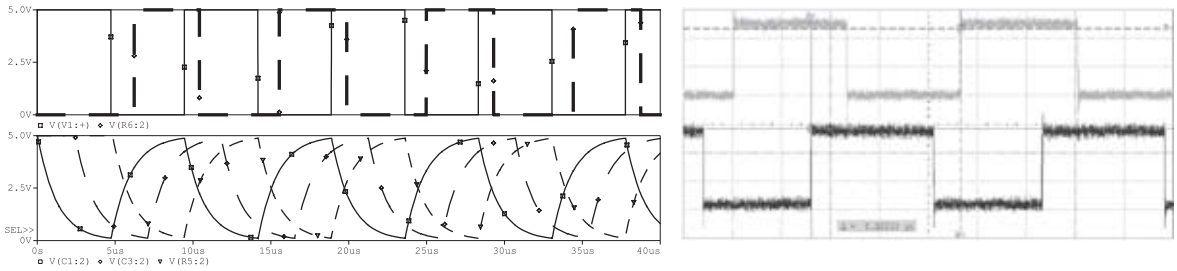


Figure 3.30: Simulation and Measured Performance of the Extra Delay

with a 106 kBit/s data rate. Below that, the voltages of the capacitors of every second inverter, i.e. C_1, C_3 and C_5 in Figure 3.29, are depicted. It is important, that the capacitor of every stage is charged and discharged to the same voltage levels, so that the waveforms of each stage look identical, except for a shift in time. If this criteria is not met, as will be the case for much larger values for the capacitors or resistors than the here chosen ones, the shape of the signal will be altered, e.g. the first pulse could be shortened.

By means of a jumper on the PCB (omitted in Figure 3.29), it is possible to choose between either no delay at all, i.e., bypass the extra time delay stage, or the signal present after the second (pin 4), the fourth (pin 8) or the last (pin 12) inverter of the 74HC74. On the right of Figure 3.30, the input signal (at the bottom), and the delayed output signal (at the top), are depicted. Together with the time delay caused by other parts of the developed circuit, e.g., the transceiver (see Figure 3.28) and the envelope detector (see Section 3.2.6), an overall delay greater than one bit duration is achieved, so that the relayed data can be aligned to the bit grid defined in Section 2.2.4.

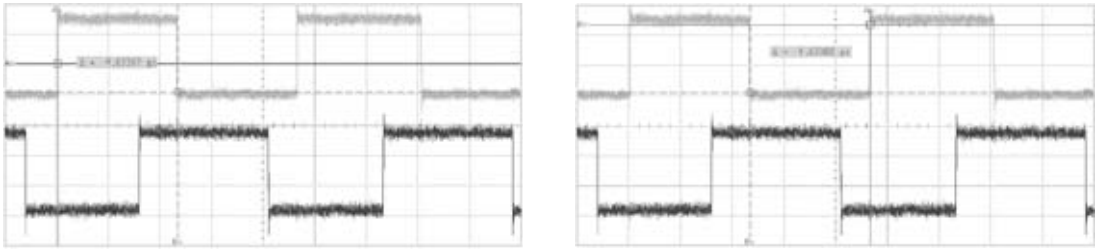


Figure 3.31: Manchester Coded Output of the Demodulation Stage

If the variable resistor of the signal shaping circuit, described in Section 3.2.6, is set up properly, an accurate Manchester encoded signal is obtained, as demonstrated in Figure 3.31, where both high and low time of the purposely delayed signal are found to be equal.

3.2.8 Communication Link Interface

An interface for a separate module to communicate with the RFID tool over an infrared or RF wireless link is installed on the PCB, providing data input and output pins, a serial programming interface and power supply. An additional output pin indicates, whether TX(transmit) or RX(receive) mode is required. The data pins can be driven directly by the peripheral circuitry of the RFID tool or steered by the I/O pins of the microcontroller, which allows for features like emulation of tags and microcontroller-based delaying of the interchanged signals.

Bandwidth Considerations

The bandwidth needed for the communication link is kept low, as due to the prior processing only Manchester or Miller encoded data is to be transferred. Miller or NRZ encoded data demands for a bandwidth of approximately the data rate, whereas a Manchester coded bit stream needs twice as much bandwidth, because, in the worst case, the amount of transitions is doubled (see Figures 2.2 and 2.4). The higher bandwidth required by the Manchester code could be circumvented by transforming Manchester to standard NRZ code, as NRZ coded data only needs half of the bandwidth demanded by the Manchester code. After equipping the wireless modules with the corresponding en- and decoding chips, e.g., from Intersil[11] or Data Delay Devices[12], cheap wireless RF modules available on the market with a maximum data rate of 115 kBit/s are sufficient, otherwise a bandwidth of at least $2 \cdot 106$ kBit/s= 212 kBit/s is theoretically required.

3.2.9 The Microcontroller

The RFID tool is based around an Atmel ATMega32 [6] microcontroller, clocked at 13.56 MHz, which is amongst others equipped with 32 kByte Flash RAM to store the code of a program, 2 kByte SRAM, 1 kByte EEPROM and an 8-channel, 10 Bit ADC[13]. It employs a RISC[14] structure, leading to often only one clock cycle ($\approx$ 73.7 ns) being needed for the execution of an instruction, therefore allowing relatively fast reaction to external signals, e.g., via interrupts. Every pin of the four general purpose byte I/O-ports provided by the Atmel is occupied in the developed application, emphasising the various potentials of the hardware. The wiring on the circuit board is carried out in such a way, that the microcontroller has preferential access to all relevant inputs and outputs, and so can forbid other devices on the board to control a certain signal. Hence, the respective pins of the μC have to be set to high impedance state, if another component shall have the priority.

[11] `http://www.intersil.com`
[12] `http://www.datadelay.com`
[13] Analog to Digital Converter
[14] Reduced Instruction Set Computer

3.2.10 The Programming Adapter

For flexible operation and testing, the software running on the microcontroller can be updated, without the need to remove it from the board, through a developed programming adapter, which is depicted in Figure 3.32 and can be plugged into the parallel port of a PC via the appropriate cable. The adapter is similar to the one described on the PonyProg2000 website[15] and compatible to the widespread Atmel STK200 AVR Starter Kit[16].

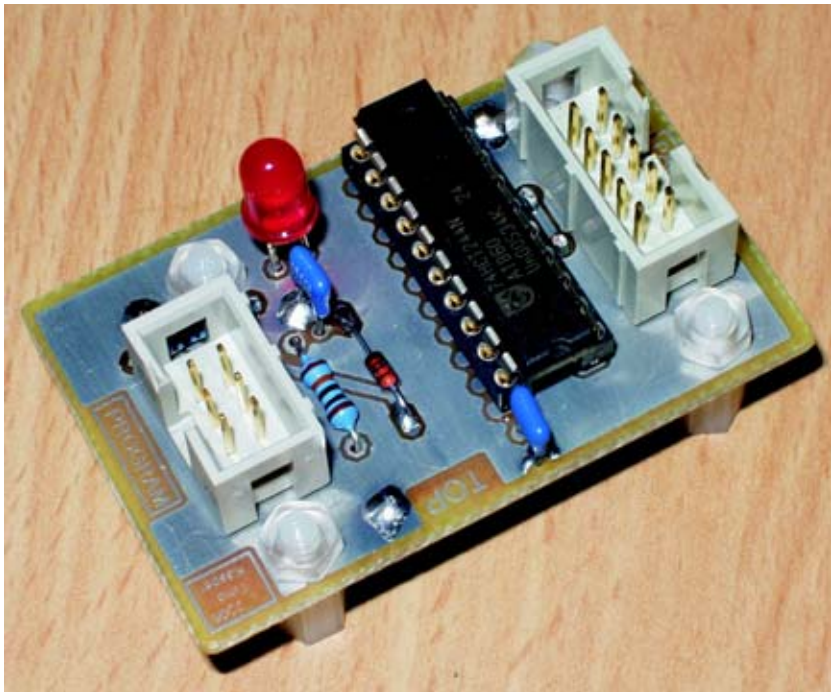


Figure 3.32: The readily assembled program adapter

Measuring the voltage levels of parallel ports of various PCs, it turned out, that sometimes a voltage of only approximately 3 V for a high logic level is delivered from the PC, which might not be accepted as a logic high by the (5 V-) CMOS compatible Atmel. Therefore, a 74HCT244 [40], containing eight 3-state buffers, is inserted between the parallel port and the appropriate pins of the ISP[17] interface of the microcontroller, i.e. MOSI[18], MISO[19], SCK[20] and Reset, to ensure adequate voltage levels for both directions. The schematic and the pin assignment for the cable to the parallel port of

[15] http://www.lancos.com/prog.html
[16] available from http://www.kanda.com
[17] In System Programming
[18] Master Out Slave In
[19] Master In Slave Out
[20] Slave Clock

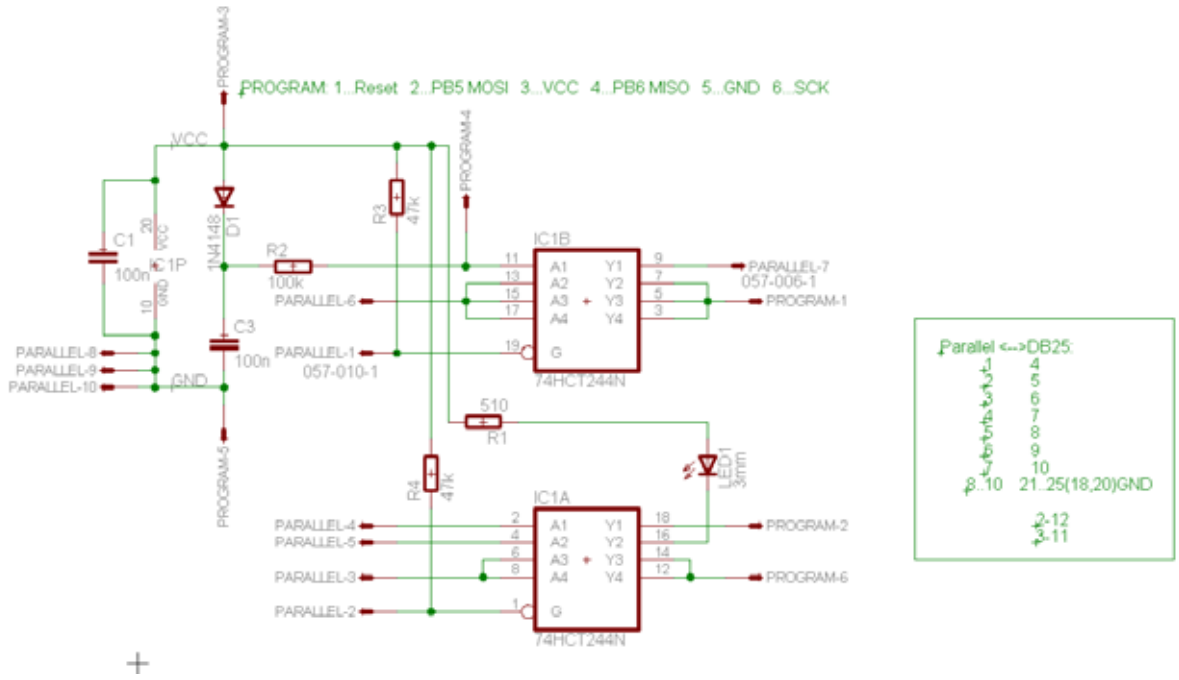


Figure 3.33: Schematic of the program adapter

the PC is presented in Figure 3.33. Integrated circuits of the HCT type accept the lower TTL[21] levels at their inputs, while CMOS compatible levels are output [1]. Note, that three outputs of the 74HCT244 are connected in parallel, wired to the Reset pin of the microcontroller, to achieve steep edges there. A red LED[22] on the programming adapter is lit, if a program is being downloaded into the Flash memory of the Atmel.

3.2.11 USB Port

Fast communication with a PC or other USB equipped hardware is made possible by the FT245R parallel to USB chip from FTDI[23]. The device allows to send or receive packets of eight data bits, adequate to the 8-bit architecture of the microcontroller, by pulling a read or write input pin high and low, once a corresponding strobe pin indicates whether the device is ready. Fetching or writing out one byte is possible in three clock cycles, as the minimum duration for a pulse to shift data from the input bus of the chip into the internal FIFO buffer is specified with 50 ns in the data sheet [16], which is automatically met with one clock cycle of the microcontroller taking 75 ns. So, using the supplied D2XX drivers, it is possible to exploit the maximum possible data transfer rate

[21] Transistor-Transistor-Logic
[22] Light Emitting Diode
[23] Future Technology Devices International – `http://www.ftdichip.com`

of 1 Megabyte/second between the RFID tool and a PC. In addition, so called VCP[24] drivers are available, providing access to the USB port as if it was a normal serial COM port, so that compability with programs implementing a normal serial port is given and any serial terminal program, e.g., Hyper Terminal[25], can be utilised. During tests, a serial connection with a baud rate of 921600 bit/second has been established.

3.2.12 Design of the Reader – Approach and Hints

1st Version

After analysing the various capabilities of the I/O ports of the microcontroller, and choosing the peripheral components described above, like the USB chip, the RF-transceiver, and more, a schematic for the first version of the reader was developed and entered into the EAGLE Layout Editor, to create the layout, again using SMD technology where possible.

During the layout process, the packages of the devices used had to be matched to the ones available on the market, and some new footprints had to be drawn and added to the EAGLE libraries. Care had to be taken, where vias from the top layer to the bottom layer of the board were needed, as with the used "home-brew" technology, the connection between the two layers is not achieved automatically after the etching of the PCB, but by afterwards soldering a little piece of wire, sticked through the board, for every via. This implicates, that vias are not possible in certain places, for example under the socket for the microcontroller, where the wire can not be reached and heated up properly with a soldering iron.

The dimensions of the reader are kept smaller than 80×100 mm, so that only half of a standard 160×100 mm card is occupied. After transferring the layout onto a double sided copper-clad board, coated with photo-resist, the PCB was etched and cut out. Afterwards, the holes for the vias and the other components were drilled and a thin plastic coating sprayed on the board, to protect it from corroding due to humidity. The utilised "Plastik" spray from Cramolin[26] turned out to be effortlessly solderable later on, when the components were assembled. The result of the effort, the low level reader version 1, is depicted in Figure 3.34.

For the power supply, a cheap unregulated mains adapter with a DC output voltage of approximately 7 to 9 V can be used and plugged into the DC socket on the board, as the voltage is regulated to constant 5 V by means of a 7805 voltage regulator, wired according to the data sheet [32]. A reset of the microcontroller and hence the peripheral devices can be triggered by means of a pushbutton on the board, as well as four LEDs and an additional pushbutton are provided for general user interaction.

[24] Virtual Com Port
[25] Terminal program delivered with Microsoft Windows
[26] `http://www.cramolin.de`

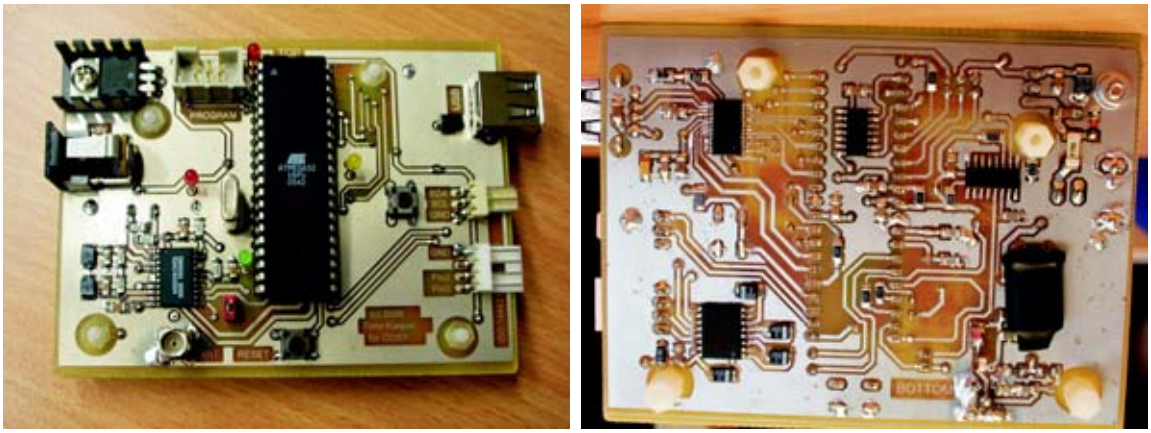


Figure 3.34: The completely assembled first version of the reader

An on board RS 232-compliant serial interface to a PC is realised using a MAX 232 RS 232 driver and receiver, wired amongst others with 1 μF electrolytic capacitors, as proposed in the data sheet [28]. With its internal voltage doublers and inverters, the 0 or 5 V output levels of the RX / TX pins of the internal UART[27] of the microcontroller are translated to the approximately $\pm$ 8 V being necessary for the the RS 232 interface.

To ensure operating stability and reduce the influence of the HF field on the analogue and digital circuitry [48], for each device on the designed PCB at least one bypass capacitor (often with a value of 100 nF) is placed as close as possible to the power pins of the particular chip. This, of course, also applies to the developed Fake Tag (see Section 3.1) and the program adapter (see Section 3.2.10).

The rather large DIP[28] was chosen for the microcontroller, as it simplifies the layout of the PCB[29] and facilitates measurements, as a scope probe can be directly clamped to the pins of the device, thus making additional test pins unnecessary. As other SMD components were placed underneath the chip, the increase in occupied board area is negligible.

2nd Version

After the development of the first fake tag (described in Section 3.1.8) and successfully testing the low level reader functionality, the labour was concentrated on the interaction between reader and fake tag. Working towards carrying out a relay attack, several improvements of the reader at hand became necessary, and were added to the circuit by means of discrete components. At the point, when the first relay attack had been

[27] Universal Asynchronous Receiver-Transmitter
[28] Dual In-line Package
[29] the pins of the chip are spaced far enough apart, so that connections can be made between them

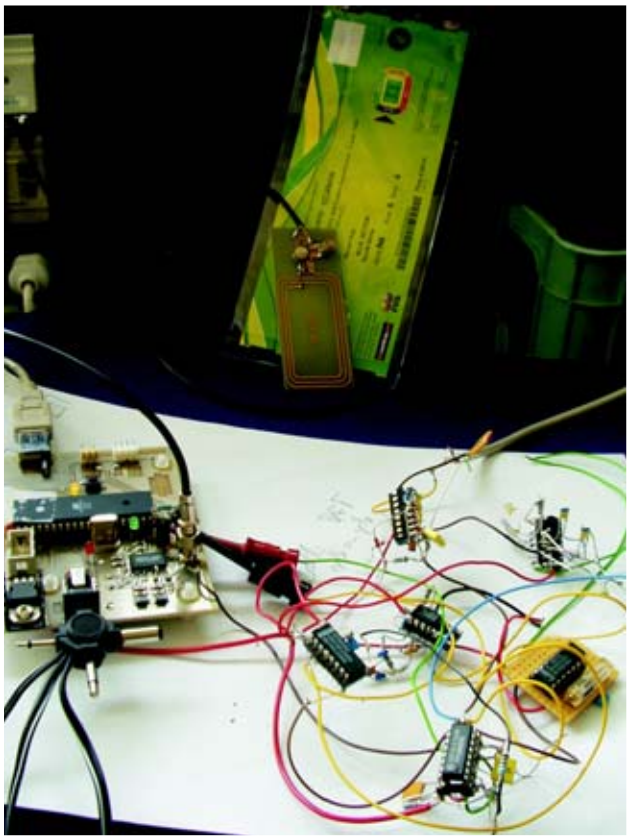


Figure 3.35: Experimental extensions of the first reader version

carried out successfully, these enhancements had grown to the freely wired and fairly unreliable "piece of art" depicted in the right bottom corner of Figure 3.35, this making a redesign inevitable.

All the components required for the extension of the reader were added to the existing Eagle schematic, while the standard serial interface was omitted, as the USB interface turned out to be an effective replacement. The outer dimensions of the PCB for the second version of the reader, which is depicted in Figure 3.36, have been retained unchanged, still being 80×100 mm.

3.3 Tuning the Antennas for Optimum Performance

For maximum operating range of both the reader and the Fake Tag, the respective antennas, some of which are depicted in Figure 4.7, have to be tuned to the present carrier frequency, matched to the input impedance, for example of a coaxial cable, and afterwards damped with a parallel resistor, to achieve the appropriate trade-off between bandwidth and amplitude.

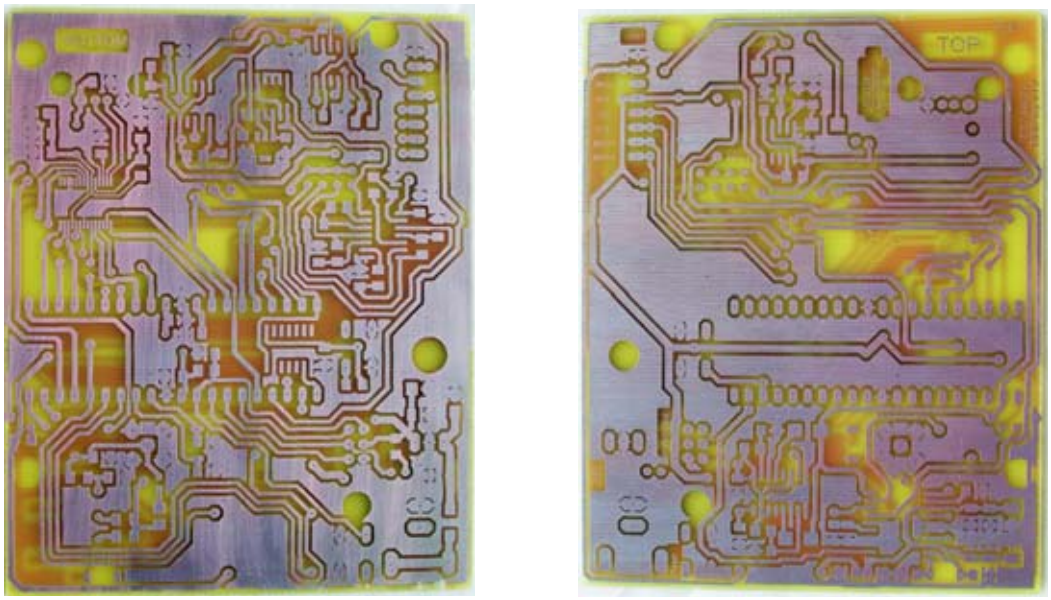


Figure 3.36: The PCB of the second version of the reader

Several tuning methods are proposed in the literature, for example in part 6 (test methods for proximity cards) of the ISO 10373 [21], which are fairly complicated and require special equipment. Instead, the system was tuned with common sense, using the instruments at hand, by the method described as follows.

The RF transceiver on the reader is used as a signal generator, set up to provide a pure sine wave with the desired frequency of 13.56 MHz at the antenna output, which will be very precise, as originated from a crystal oscillator. The antenna to be tuned is plugged into the socket on the PCB and placed in a neutral environment (no metal surfaces close to it, etc.). As depicted in Figure 3.37, a magnetic near field probe, which is connected to an oscilloscope, is fixed at a position above the centre of the antenna, where the waveform on the screen of the scope is not distorted, i.e. a pure sine wave is obtained. The used circular probe is termed RF-R50-1 and part of the near field probe set RF 2[30], manufactured by Langer EMV-Technik. Before the tuning process can start, the characteristics of the parallel resonant circuit and the operation principle of the RFID system have to be taken into account.

The quality factor, Q, can be set by adjusting a variable resistor in parallel to the L and C of the particular resonant circuit. As described in Section 3.1.1, for higher Q factors the bandwidth becomes narrower, while, at the same time, the maximum amplitude at the resonance frequency increases (compare with Figure 3.4). If the adjusted frequency of a tuned circuit with a high Q is slightly displaced from the desired carrier frequency, a significant change of the amplitude of the field will be noticeable, whereas for a low Q,

[30] `http://www.langer-emv.de/en/produkte/prod_rf2.htm`

the change in amplitude, due to detuning the frequency, will be much smaller. Hence, when the antenna is to be tuned to resonance with the trimmable parallel capacitor, the maximum possible Q factor is eligible, i.e., any parallel resistor, damping the circuit, should be removed before matching the frequencies.

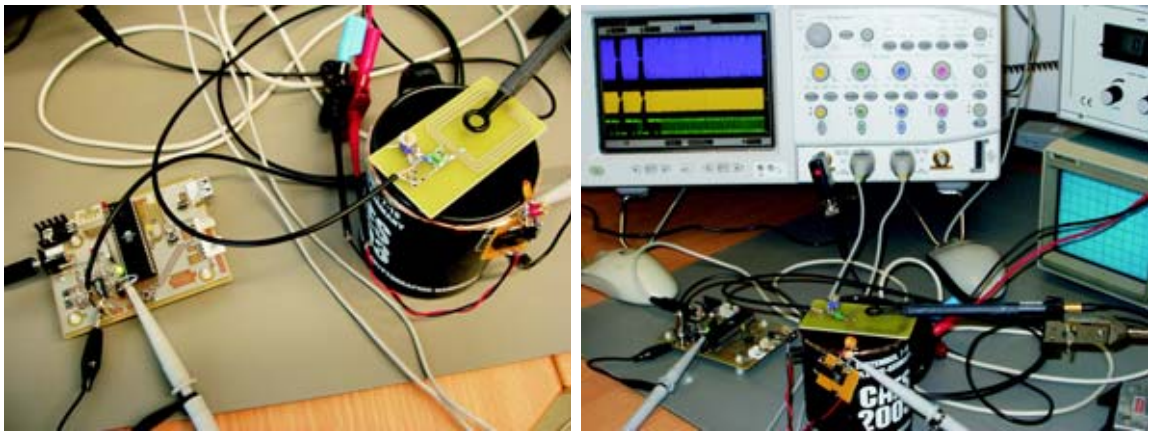


Figure 3.37: Setup for the tuning of the antennas

Usually, an antenna provides (at least) two trimmable capacitors - one connected in serial to the tuned circuit, for matching the input impedance to the coaxial cable, and the other one in parallel to the coil. While observing the signal amplitude on the oscilloscope, the values of the capacitors are altered in an iterative process, until the maximum amplitude is found, thus the antenna is tuned to the optimum value. This, of course, can be verified by testing the read range of a real (purchased) tag.

Once the antenna of the reader is adequately adjusted, as describe above, we proceed with the counterpart of the Fake Tag, where the same rules for quality factor and tuning sequence apply. In addition, the load modulation resistor (R_P in Figure 3.2) is to be adjusted, by observing the effect of the load modulation generated by the Fake Tag at the voltage of the antenna of the reader generating the field. Known from experience, a sufficient load modulation is achieved with a value of the resistor, where the first slight load modulation is noticeable on the screen of the scope.

After the LC-circuits are tuned to resonance, the Q factor has to be reduced by means of the parallel resistor, to lessen the damping of the information in the sidebands (compare with Figure 3.5) and make the system less sensitive to the environment, e.g. metal surfaces. The optimal setting can be obtained by finding the maximum read range, as a function of the Q factor and the load modulation, of a real RFID system, which is either the developed RFID reader in combination with a purchased RFID transponder, or the developed Fake Tag combined with a purchased (and hopefully properly tuned) reader.

The determined best setup has been preserved, using a permanent marker, by means of lines or dots on the components, indicating the required position of, for example, a

trimmable capacitor.

3.4 Software

In this section, an overview of the used programs and the interaction between the developed libraries and functions is given. The source code (see appendix) is commented in detail, so that it should be no problem to understand the concrete implementation, or conclude to the task of the not mentioned operations.

The developed software for the Atmel AVR microcontroller is mostly written in C[31] and compiled with the free avr-gcc compiler[32]. Some of the currently implemented functions are listed below.

- set up and control the EM 4094 RF transceiver,
- receive and send data via the USB port,
- precise wait/delay function,
- implementation of standard and short frames, including parity bit generation, as specified in ISO14443, part 3,
- generation and sending of Manchester coded data = tag emulation,
- generation and sending of Miller coded data = reader functionality,
- the user can switch between the operating modes like relay, tag emulation or reader mode.

3.4.1 Development Tools

The free WinAVR development environment[33] consisting of compiler, linker, libraries for various Atmel microcontrollers and an editor, called "Programmers Notepad", was found to be very helpful for comfortable programming and editing of the program code.

A so called *Makefile* has been created, with all the parameters necessary for the production of binary (.hex) files out of the C code, such as the microcontroller type, the processor frequency and the name of the file containing the required `main()` routine. The editor allows user defined tools to be implemented in the environment, so that compiling, linking and fast downloading of the binary file to the μC, with the help of the *Makefile*, is effectively done by just one keystroke.

[31] parts in (inline) assembler
[32] `http://www.avrfreaks.net/AVRGCC`
[33] `http://sourceforge.net/projects/winavr`

The AVRStudio Integrated Development Environment[34], available from the Atmel website[35], was found to be suitable for debugging purposes, for example by disassembling the generated code and analysing it with respect to the clock cycles needed for the execution. Furthermore, all the internal registers of the Atmel, as well as the I/O-ports, can be observed and arbitrarily modified.

Some options of the microcontroller, such as the clock source or whether a boot loader is used, have to be set by programming so-called flash fuses of the μC. This can be done with the PonyProg2000[36] by choosing "Configuration and Security Bits" from the "Command" menu. Previously, the "Interface Setup" has to be altered to "AVR ISP I/O", and the correct microcontroller type, i.e., AVR micro ATMega32, has to be selected. Note that for all fuses, "0" means programmed, while "1" means unprogrammed, as described in the data sheet [6]. The optimal setting for the Atmel on the reader board is, to leave all options unchecked in the PonyProg program, except for `CKOPT`, `BOOTSZ1` and `BOOTSZ0`.

3.4.2 Description of the Source Code

For convenience, the code is separated into several files, the contents of which are described below. More details about the internal structure of the Atmel ATMega32 can be found in the data sheet [6].

board.h

A header file `board.h` contains macros, declarations of variables and prototypes of functions, which are needed by the other libraries or the main program. Note, that variables or functions declared in the header file still have to be defined elsewhere - the declaration just ensures global access.

Worth mentioning is the definition of the global flags, which are stored in a bit-accessible I/O location of the μC, in this case the otherwise unused EEDR data register for the EEPROM. Setting or polling a flag placed in a bit-accessible location takes only one clock cycle, thus saving execution time, which is especially important for the real time processing of data in an ISR[37]. Currently, only two flags are employed: `ISRBusy` indicates, whether a particular ISR is finished with the processing of the current data. The other flag, `IsPause`, is used to hand over the current data to be sent to a ISR.

[34] IDE
[35] `www.atmel.com`
[36] http://www.lancos.com
[37] Interrupt Service Routine

em4094lib.c

The library `em4094.lib` was developed for controlling the EM 4094 chip and furthermore encoding and preparing the data to be sent, according to the ISO 14443. Calling `EM_Init()` results in the DDRs[38] of the pins of the Atmel, which are connected to the RF transceiver and the communication interface, being properly set. The EM 4094 is enabled, reset and then initialised (by means of other developed functions) with the appropriate option bits for ISO 14443 compliant operation. For comfortable programming, defines have been implemented for the most often used pins, e.g., a `DIN_LOW;` pulls the corresponding pin low and a `DIN_HIGH;` sets it to a high level.

Care has to be taken with regard to the data direction (input/output) of the DIN pin. During the initialisation process, the DIN pin is directly driven by the μC, and therefore used as an output. Later on, the pulses necessary for sending data are generated by an external device (see Section 3.2.4), so that the pin defined as `DIN_PULSE` is utilised. This implies, that the DIN pin has to be set to tri-state, i.e., input direction, as otherwise two outputs will try to drive the DIN pin simultaneously, which might damage the devices and is counterproductive to sending out valid data. This has to be done "manually" in the main routine, depending on the desired functionality.

After the initialisation, the three timers of the microcontroller are set up with the respective values. The (8-bit-)timer 0 is used, when Miller coded data is to be output. Timer 1, with 16 bit resolution, is utilised in the context of a wait-routine, when a longer, but still exact, pause is required (e.g., the FDT after sending out a REQA command). Timer 2, an 8-bit-timer again, is used for sending out Manchester encoded data, and, like timer 1, is set up for sending out data at a rate of 106 kBit/s. As the microcontroller is clocked with 13.56 MHz, and the data rate is derived from this frequency, the timing of the resulting data output is very accurate.

Efforts to encode the data from NRZ to Miller or Manchester code in real-time, by means of implementing a state machine in the ISRs, have turned out to be unavailing. As every register of the Atmel used in an ISR has to be pushed on the stack[39] before its execution, the extra time needed for the whole process does by far exceed the only 64 clock cycles until to the next call of the ISR, under any circumstances. Hence, the program had to be optimised for execution speed during the transmission of data, at the cost of memory[40] usage, and a sequence for pre-processing the data, described for sending out Miller coded frames[41], as follows, has to be obeyed.

First, the data to be sent is handed over to the function `EM_SendShortFrame` or `EM_SendStandardFrame`, depending on the kind of frame to be sent (see ISO 14443 [22], part 3). The seven bits needed for a short frame are passed over in the form of a byte,

[38] Data Direction Register
[39] and later on popped off the stack again
[40] there is plenty of memory available on the ATMega32 μC
[41] The principle and names of the functions are similar for Manchester coded data

where the MSB[42] is ignored and the LSB is sent first. For a standard frame, data is interchanged in the form of a pointer to an array, where the first byte in the array is sent first. The mentioned functions convert the data into the respective format of the frames and store the result in an array termed `Frame2Send`, in which the data is represented as listed below (compare to Section 2.2).

$$0 = SOC,\ 1 = One,\ 2 = Zero,\ 3 = EOC$$

Note that, as the EOC of Miller encoded data consists out of two bytes, with the first byte being equal to a *logic 0*, the EOC above corresponds to the second byte.

The (odd) parity bit for each data byte, needed for the creation of standard frames, is returned by the `EM_Parity` function:

```
uint8_t EM_Parity(uint8_t byte, uint8_t mode)
// returns Parity bit for odd(mode=1) or even(mode=0) parity
{
byte ^= (byte >> 4);
byte ^= (byte >> 2);
byte ^= (byte >> 1);
byte &= 1; //now byte contains the bit to add for EVEN number of ones
return (byte^mode);
}
```

The parity is effectively calculated as shown above, by XORing one bit with every other bit, which is the same as a modulo 2 addition of each bit of the data [35]. Please bear in mind, that a zero is coded as a "2" in the `Frame2Send` array, while the parity routine returns a "0".

In the next step, by means of the function `EM_DoTheMiller`, the data is encoded as required by the ISO 14443 standard, in such a way, that one byte in the array represents one half period of a bit to be sent, and thus the ISRs can effectively shift the data to the DIN pin of the chip. This "waste" of memory has been necessary due to the internal 8-bit structure of the microcontroller, which causes accessing a bit in the memory to take longer than accessing a byte. In the current implementation, insertion of an additional NOP[43] in the corresponding ISR, taking one clock cycle for execution, already leads, in the worst case, to invalid data being transmitted, because the pointer to the `Frame2Send` array has not been increased (as required for proper operation) before the subsequent call of the ISR.

A call of `EM_SendMiller` sets up the timer, such that the ISR is called every half bit period, i.e., 4.72 μs at a data rate of 106 kBit/s, and activates the correspondent interrupt, so that at last the data is forwarded to the RF transceiver, as follows.

[42]Most Significant Bit
[43]No Operation (computer processor instruction)

1. get the current value from the `Frame2Send` array
2. increase the pointer to the `Frame2Send` array
3. set the `IsPause` flag, according to the current value
4. set the `ISRBusy` flag to 1
5. wait for the `ISRBusy` flag to become 0, indicating that the ISR has finished transferring of the current data
6. proceed with 1 until all data is sent.

Simultaneously, the ISR of timer 0 is executed on every timer overflow occurring, simply performing the below listed operations.

1. if the `IsPause` flag is set, trigger a pulse at the DIN input of the RF transceiver
2. clear the `ISRBusy` flag

After all the data is sent out, the interrupt is deactivated again, to not disturb the other duties of the program.

ftlib.c

The library `ftlib.c` provides definitions, macros and functions for controlling of the FT 245R USB chip, and thus the interface for data transmission from and to the PC. One port of the microcontroller is connected to the eight data input/output pins of the FT 245R, while four more pins are needed to switch between reception or transmission of data, and for handshaking purposes.

The data to be sent or received is stored in an array termed `USBData`. An example for a macro definition, used to acquire data from the USB chip, is given below:

```
#define FT_READ(Pointer) RD_LOW; USBData[Pointer++]=FT_DATA_PIN; RD_HIGH
```

Issueing the command `FT_READ(CurrentByte);` in a function thus results in the data currently present at the `FT_DATA_PORT` being stored to the position in the `USBData` array, where `CurrentByte` points to, and in increasing `CurrentByte` afterwards.

The function `FT_Init` has to be called once to initialise the directions of the pins needed for communication and enable the internal pull up resistors of the μC, where necessary. Before the first call of the `FT_Send` function, to send an array of data bytes to the USB port, the `FT_WR_PIN` has to be pulled low. A similar pre-condition applies to the `FT_Send` function, for which the `FT_RD_PIN` is to be set to high before the first execution. The number of received bytes in the `USBData` array is returned, when all data is received from the PC, which is detected by occurrence of a time-out[44] of the timer 1.

[44] currently set to approximately 1 ms

etcetera.c

The serial port (only available in the reader version 1) is set up and controlled via several functions in the `etcetera.c` library[45]. Furthermore, a function `ETC_CheckButton()` is provided, which returns a value different from zero, if the push-button on the PCB was pressed down in the moment of its execution. The pins of the microcontroller connected to the LEDs on the reader are set up by calling `ETC_InitLEDs()`. Pulling the corresponding pin of the Atmel low results in the LED being switched on.

test.c

For testing of the so far implemented functions, a `test.c` program, containing the `main()` routine, was written. After calling the initialisation routines of the other libraries, described above, an endless loop is executed, in which a user interaction can take place.

If the reader is connected to a PC via USB, and the FTDI USB VCP driver is properly installed (see Section 3.2.11), a serial terminal program can be used to establish a connection with a baud rate up to 921600 bit/second. After triggering a reset via the reset push-button on the board, the reader will announce his presence by issueing a "Hello!" and the current state, which is the listening mode after a reset. By pressing a key, the user can change between several operation modes, which are listed below.

- 'l' - Listen mode - needed to for performing a relay attack or to acquire data
- 'r' - A *REQ* is instantly sent out
- 'w' - A *WUP* is instantly sent out
- 'a' - The reader waits for an incoming command from the Fake Tag. After a fixed time, an *ATQ* is sent out to the Fake Tag, confirmed by an "ATQ sent." via the USB port.
- 't' - likewise to 'a', but the delay time is increased each time 't' is pressed, and the value handed over to the wait routine is output via the USB port.

The chosen state is indicated by different LEDs being lit on the PCB and by the output of the reader in the terminal program.

[45] see comments in the C code for a description

4 Applications and Results

4.1 Low Level Reader

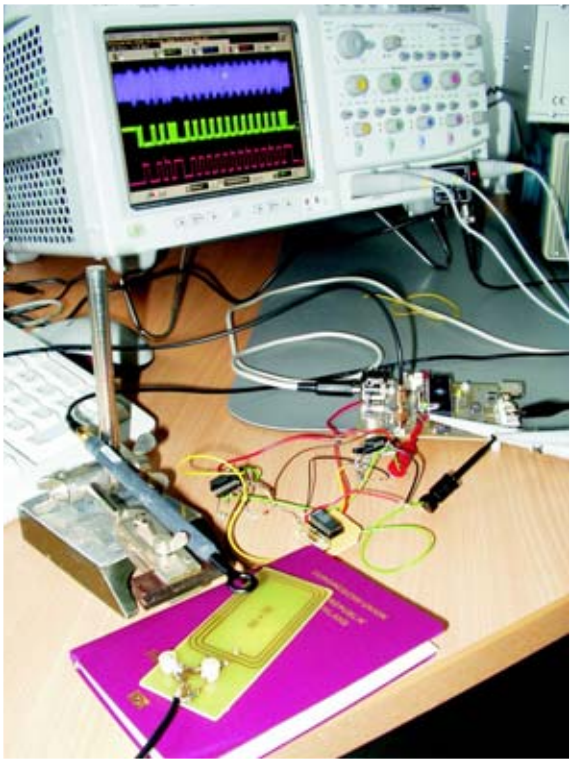


Figure 4.1: Testing the Low Level Reader with a German e-passport

The flexible low level reader function has been successfully tested with several ISO 14443 compliant tags, which all answered to the commands sent out by the reader. Figure 4.1 shows the measurement setup for tests with an electronic passport, issued by the Federal Republic of Germany, at the moment of the passport sending out an *ATQA*. A PCB type antenna serves as the coupling element to the antenna of the ISO 14443 chip, which was found to be embedded in the cover of the passport. On the top of the screen of the oscilloscope, the generated load modulation generated can be spotted in the field, which is measured by the magnetic near field probe in front of the picture. Below the waveform of the field is the modulated Manchester code at the DOUT output of the

RF transceiver, whereas the demodulated Manchester code, ready for acquisition by the microcontroller, can be seen at the bottom of the screen.

Parts of the ISO 14443 initialisation and transmission protocol have already been implemented (see Section 3.4). The exact behaviour can be flexibly steered by programming the microcontroller, so that more special functions, including changing the properties of the RF transceiver, can easily be implemented.

4.2 Relay Attack

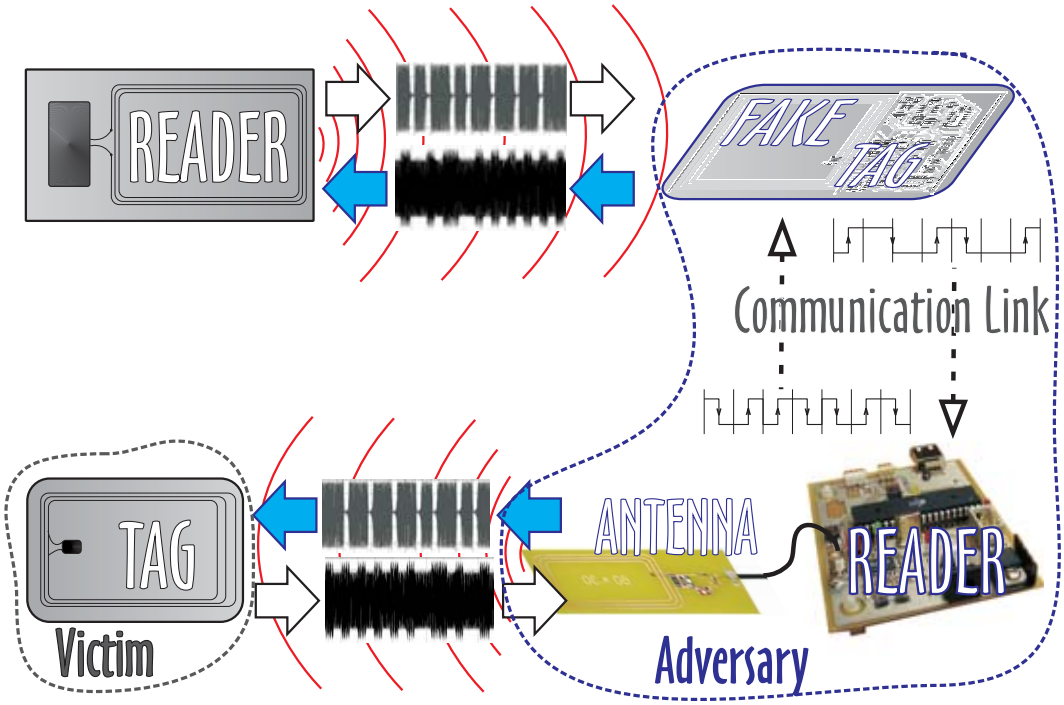


Figure 4.2: Principle of a Relay Attack

To perform a relay attack with the RFID tool, as illustrated in Figure 4.2, the offender, having the Fake Tag with him, needs an accomplice handling the designed reader. While the attacker maybe holds a colour copy of an authentic ticket, covering the Fake Tag (see Section 3.1)in his hands, into the field of the RFID reader at the entrance, the antenna of the accessory's reader is placed close enough to the contactless ticket of the victim, so that it powers up and gets into the idle state (see Section 2.2.3). The data being transferred by the reader at the entrance is acquired by the Fake Tag and forwarded on the bit layer through the communication link to the reader at the accomplice. Here, the data is retransmitted to the ticket of the victim, which then answers to the relayed requests of the remote reader. Sent over the communication link again, the answer is

relayed back via the fake tag to the reader at the entrance. As the exchanged data is further on relayed, both remote reader and the transponder in the ticket of the victim will be convinced, that they are in close vicinity to each other and share the same secret. After less than one second, the entrance could open up and the attacker gain access to a restricted area, without allowance.

A relay attack, as described above, has been successfully carried out using the developed tool with

- a digital passport (e-passport), issued in January 2006 by the Federal Republic of Germany, with an ISO 14443 chip in its cover,
- an ISO 14443 compliant student identity chip card employed for payments at the Ruhr-University in Bochum,
- Philips standard (classic) Mifare and DESFire cryptographically enabled smartcards,
- an Atmel AT88SC153 contactless smartcard,
- and tickets for the FIFA World Cup 2006 in Germany,

until to the point of at least reading out the UID (see Section 2.2.3) of the tags and furthermore, in the case of the Mifare classic, until to successful login to the card.

The assembly for executing a relay attack with the designed hardware is depicted in Figure 4.3. On the left of the picture, a ticket for the world championship is placed upright on the chair, with an antenna in front of it, which is establishing the RF connection between the developed reader and the tag implanted in the paper of the ticket. The so-called smart label, which is a combination of chip and antenna being manufactured thin enough to be embedded in paper, becomes visible, if the ticket is exposed to a source of light from behind, as demonstrated[1] in Figure 4.4. The reader is connected to the Fake Tag, which is positioned on top of a proprietary reader (in the right top corner of the picture), via a cable. With this reader, all 64 bytes stored on the Mifare Ultralight Chip (see Section 4.2.1) were successfully read out remotely, with the relay mode of the RFID tool, thus proving that the proprietary reader takes the Fake Tag as an authentic one.

The minimum overall delay induced by the depicted assembly, in the case of the reader being directly connected to the Fake Tag by means of a wire, is approximately 2 μs.

4.2.1 World Cup Ticket Remarks

It turned out, that a Philips Mifare Ultralight chip is embedded in the ticket [45], providing no encryption at all, so that in reality a relay attack is not necessary for an

[1] for protection of privacy, the personal data printed on the ticket has been altered

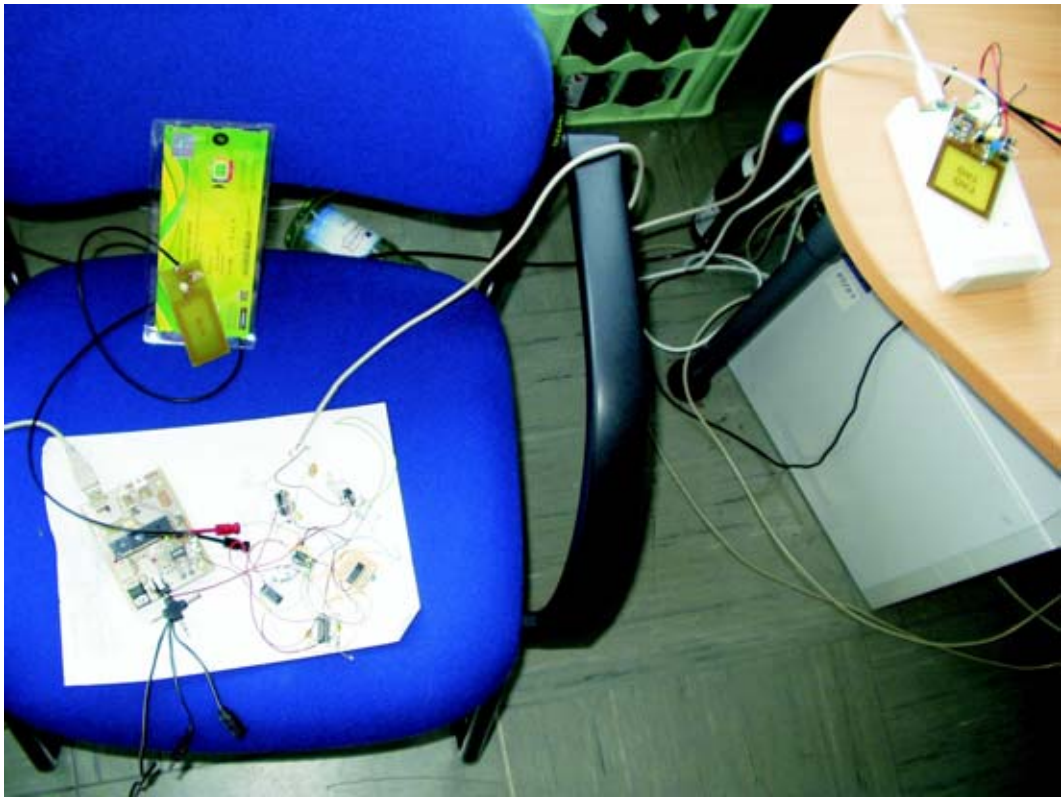


Figure 4.3: Relaying a ticket for the world championship

offender to spoof the RFID access control. Instead, a replay attack can be performed, the principle of which is much easier. The contents of a ticket, which in case of the Mifare Ultralight chip is the UID and a maximum of 64 bytes, which are stored in 16 blocks, have to be read out and stored in the memory of the microcontroller, by means of the developed reader[2]. The communication protocol, which would have to be implemented for the attack, is fully published in the data sheet [44]. At a crowded place or in a queue, where the short operating distance of the system can be achieved, the contents of several tickets could thus be recorded. At the entrance, where an RFID reader examines the content of the smart labels, the data of a ticket can be replayed via the Fake Tag, making the reader believe to have a valid ticket in its vicinity.

The security of this particular RFID ticketing system probably relies on the fact, that the fixed UID of the employed chip is usually determined during manufacturing of the smart label and cannot be changed later on. A combination of the UID with the data stored on the ticket will very likely refer to the entry of the purchaser of the ticket in a data base. Fortunately, the emulation of both the data stored on the smart label and the UID is effortlessly possible with the developed tool, thus making such a simple attack

[2]stand-alone operation without connection to a PC is possible

feasible. Hence, from my point of view, no security at all is provided by the described system.

Of course, tracking movements of individuals in the stadium, by means of their tickets, is possible, due to the fixed UID.

Figure 4.4: Sunlight from behind reveals the secrets of the world championship ticket

4.2.2 Timing

When performing a relay attack, an extra delay induced by the processing of the signal is inevitable. In practice, the minimum arising time delay will be one bit slice, as defined in Section 2.2.4, which is confirmed in Figure 4.5, where the FDT between a *REQA* command of the reader and the *ATQA* answer of the tag during a working relay attack is depicted. The first edge of the answer of the transponder, in the middle of the figure, occurs approximately 95 μs after the last gap in the field of a proprietary reader, on top of the figure, which corresponds to the mentioned bit slice duration.

Even if the reader scrutinised the timing requirements demanded by the ISO standard during the initialisation phase (see Section 2.2.4), the attack could be carried out successfully, as the (fixed) bit sequence of an *ATQA* command could be stored on the μC and then sent out fast enough after an incoming *REQA* request, to meet the timing specification.

4.2.3 Implications on Privacy and Security

Employing ISO 14443 RFID tags in security sensitive applications should be regarded very critically, as the assumption of the contactless interface being secure, i.e., resistant to relay attacks, is proven to be wrong. The card identified by a reader does not have to

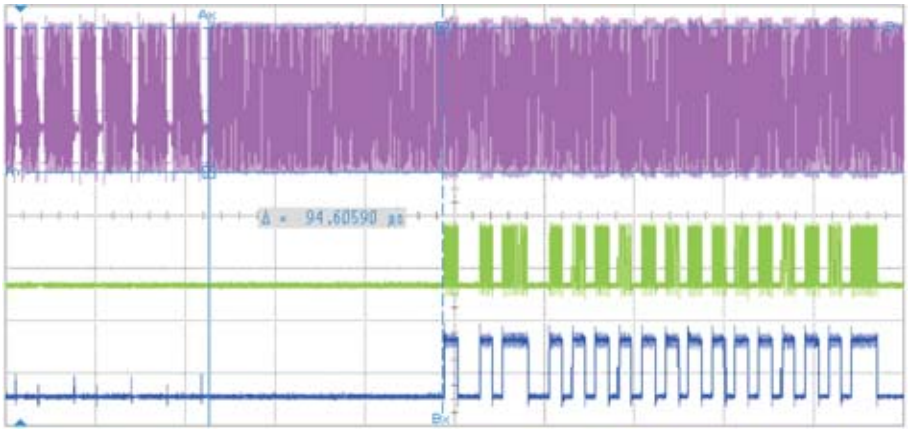


Figure 4.5: Induced delay during a relay attack

be in the direct vicinity of the reader, as declared by many manufacturers, but can be elsewhere and its data relayed from large distances without the permission of the owner. More endeavours have to be made to enhance the security of the interface, apart from limiting the reading range[3] to 4 cm [3].

When security or privacy is necessary, an ISO 14443 compliant RFID transponder should not be able to become active until the owner has performed a certain action, i.e. press a button or open the cover of his e-passport, as this would almost eliminate the possibility of performing a relay attack.

4.3 Timing Analysis of a Commercial RFID reader

4.3.1 Tag Emulation Measurements

Using the tag emulation mode of the RFID tool, timing measurements have been performed with an ACG[4] Dual 2.1 passport reader module. The *ATQA* answer of the developed fake tag to a *REQA* issued by the ACG reader was intentionally delayed and the reaction of the reader, i.e. if the answer was accepted as valid or not, was analysed. The ISO 14443 requires the tag to answer after exactly 86.9 μs.

4.3.2 Results

As presented in Figure 4.6, the Fake Tag was not only recognised as an authentic tag, but its *ATQA* answer accepted as valid, even after more than 200 μs. Every 9.44 μs, which is equal to one bit period of the 106 kBit/s data rate, during a time-slice of approximately 2.5 μs, the answer of the tag was repeatedly accepted. Compliance with the strict timing

[3] anyway being a contradiction to just waving a credit card to carry out payments, as advertised
[4] http://www.acg.de

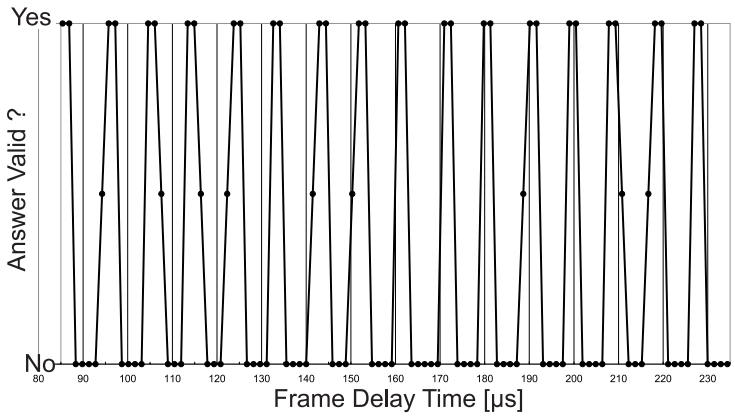


Figure 4.6: Measured behaviour of the ACG reader

requirements of the ISO 14443 (see Section 2.2.4 above) during the initialisation phase could not be observed, thus facilitating relay attacks (see Section 4.2).

4.4 Antenna Tests

Various kinds of antennas were built, as depicted in Figure 4.7, some layouted on a PCB as proposed in an application report [49], as well as self developed ones made out of thin copper wire. They were tuned to resonance with the carrier frequency of the reader and to the 50 Ω coaxial cable, and tested with regard to operating range and the influence of different environments.

It turned out, that the tuning can significantly alter the read range of the low level reader. Sometimes longer distances were achieved, when the antenna of the reader was slightly detuned. During tests with a 60×30 mm rectangular PCB antenna, for one particular setting of the trimmable capacitors, a read range from 0 to 5 cm was achieved, i.e. the contactless smartcard could be placed directly on top of the antenna. When the antenna was tuned for greater read ranges of approximately 10 cm, the card had to be placed at least 3-5 cm away from the antenna for successful communication – placing the card directly above the antenna, in this case, resulted in no valid data being acquired by the RF transceiver.

To be able to compare the achieved maximum and minimum ranges, also as a function of the tuning, the antenna of the reader was put on the table, and the smartcard with the transponder was positioned above it by means of a tripod, as depicted in Figure 4.8. The achieved read ranges were read off the scale of a ruler, also fixed to the table.

The achieved operating range is best, if the antenna size of the tag matches the dimensions of the antenna of the reader, and the antennas are properly aligned to each

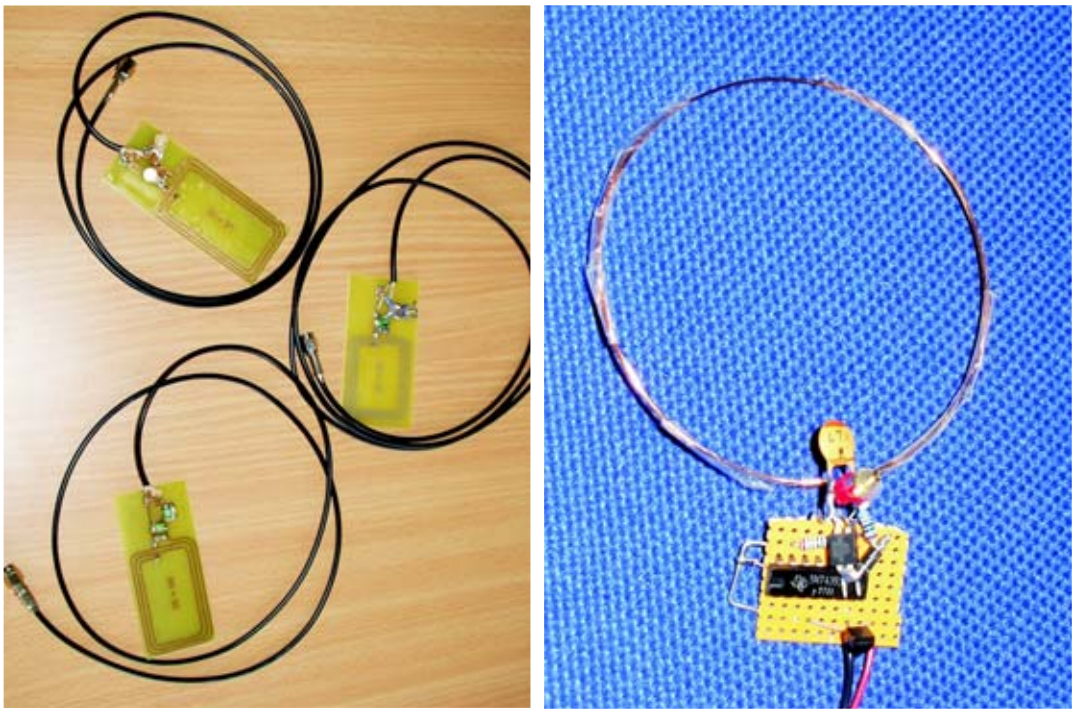


Figure 4.7: Wire and PCB antennas with different dimensions

other, seemingly as then the best inductive coupling is achieved.

4.4.1 Enhance Privacy Protection

Placing the antenna above a metal surface resulted in noticeable decreased field strength, i.e., reading range. Further investigations showed, that aluminum foil shields the device and so protects it from being reached by an unauthorised reader. A single layer of foil wrapped around a tag completely prevents reading out its data, this therefore being an appropriate countermeasure against relay attacks.

One might have the idea of putting only one slice of metal foil in his purse, close to where the contactless card is, i.e., shield the tag from only one side. This results in the reading range being only slightly (approximately 10 %) decreased, thus not providing secure protection.

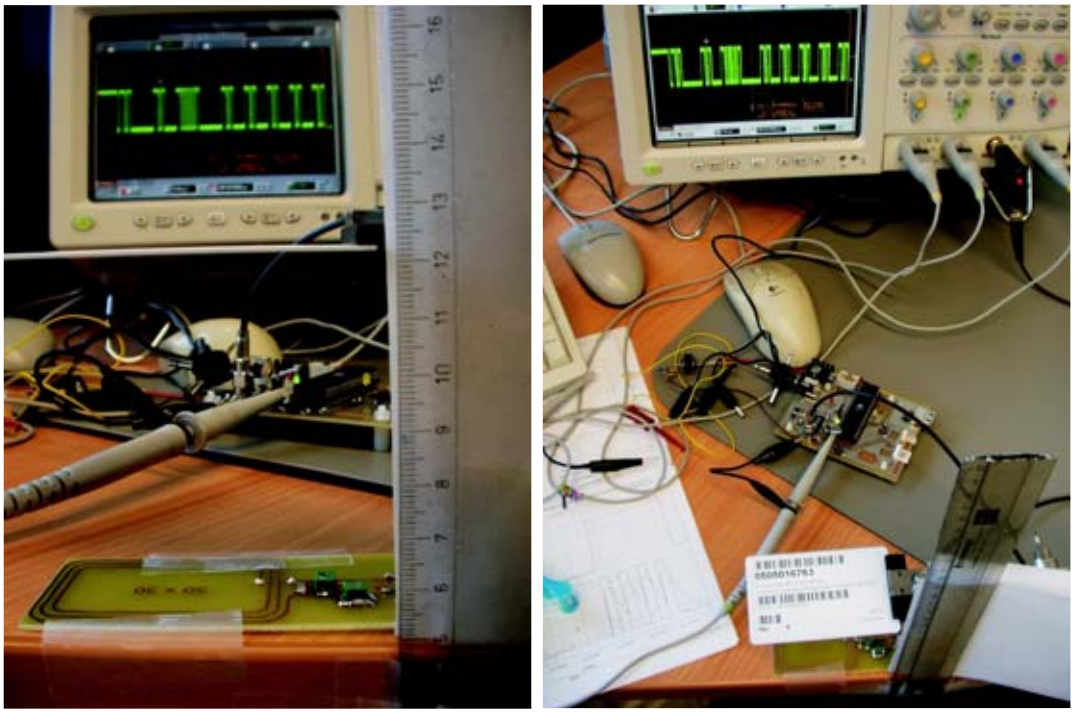


Figure 4.8: Setup for range measurements

5 Future Prospects

Before realising any new improvements or enhancements, the produced circuit boards of the second version of the reader and the Fake Tag should be assembled with the components[1] and tested, as more reliable operation is expected on the long run. If wireless RF communication between Fake Tag and reader is desired, suitable wireless modules can be purchased from Semtech[2] (e.g., the DP 1205 drop-in RF transceiver module), Radiotronix[3] and other companies. For a simplified development of embedded systems in the future, versions of the layouts and schematics are existing, in which the fragments of the designed circuit, e.g., voltage regulation, μC or RF transceiver, are separated from each other.

The current software has to be adapted to the new hardware, as several connections to the microcontroller have changed, i.e., other pins and ports are used for certain signals. Furthermore, the program on the microcontroller should be extended for acquiring Manchester and Miller coded data, and shifting it over to the PC in real-time. The usage of so-called anticollision frames, as defined in the ISO 14443, may be required for some commands during the initialisation phase of the transmission protocol. An example algorithm, written in C, for the generation of a CRC-16 checksum, which is necessary for these frames, can be found in part 3, annex B of the standard [22].

If a synchronous operation of the microcontroller with the field generated by the RF transceiver is desired, a 13.56 MHz signal can be utilised, which is provided at the DOUT1 output of the EM 4094 during reception of data from an RFID tag. Before being fed into a "'Counter Clock Source"' pin of the μC, the frequency of the signal is divided by 16 (compare with Section 3.1.3), thus being slow enough to serve as the external clock for the counter T0 of the Atmel.

5.1 Improved Man in the Middle Attack

5.1.1 Data Logging

As, in the first place, the μC is not needed for a relay attack (see Section 4.2), it may be used for logging the data interchanged between tag and reader. All that has to be done, is to reprogram the microcontroller with the appropriate software, because all the

[1] which have already been purchased
[2] www.semtech.com
[3] www.radiotronix.com

required signals are already interconnected. The recorded data can be sent over the USB port to a PC or other cryptographic hardware. This can be helpful to come to know about proprietary, not publicised protocols, as well as to store the acquired information for further processing and analyses, such as key-search with cost-effective hardware [25].

5.1.2 Active MITM

Furthermore, the μC could perform an active task by adding an extra delay to the signals or modify the data relayed → an active man-in-the-middle attack. During the initialisation, the induced time delay could be eliminated, as described in Section 4.2.2, by replaying previously stored commands, instead of the "real" relayed answer of the tag, at a desired point in time after the request of the reader.

With little effort, higher data rates than 106 kBit/s could be circumvented, by sending counterfeited data packets at the appropriate point of the protocol, where reader and tag agree about the transmission speed, which will usually happen at an early stage of the particular protocol, where the data is still unencrypted.

Altering data of an encrypted information exchange is lot harder and only a theoretical construct, even with the help of specialised cryptographic hardware. The key for encryption of the current session would have to be found out by the attacker in real-time, so that the acquired data could be decrypted, altered, and then encrypted again, before being relayed. If state-of-the-art cryptography is employed, this goal is impossible to achieve.

5.2 Increasing the Range

At the moment, the achieved read range with the developed reader and the antennas used is approximately 5-10 cm. As shown by Kirschenbaum and Wool [24], it is possible to extend this range to approximately 25 cm. The utilised power amplifier, proposed in a Melexis application note [30], can be modified slightly to fit into the design described here, as well as the copper tube antenna suggested by TI [49] could be connected. It would be interesting to verify, if the read range of contactless creditcards is really limited to the advertised 4 cm [3], which does not seem to be very likely.

Instructions for building a device for passive sniffing, i.e., eavesdropping of the communication between an ISO 14443 compliant reader and a corresponding tag, are given by Milosch Meriac[4]. A similar circuit could be added to the low level reader, while the functions to acquire Manchester- or Miller encoded data could remain unchanged.

[4] `http://rfiddump.org/rfid-22C3.pdf`

5.3 Improvement of DEMA

An improved DEMA, as described above in Section 1.3.1, can be executed, with the RFID tool providing the contactless interface to the smartcard, as well as a reliable trigger signal for the oscilloscope, issued by the microcontroller. Due to the flexibility of the low level reader, the performed operations can be aborted at any point in time, bringing advantages with regard to timing.

The major improvement of the developed system with respect to a DEMA is, that the time consuming extraction of the challenges from the waveforms recorded during the communication, is no more necessary. Instead, these numbers can now be directly obtained via USB or freely chosen by the attacker. Thus, a significant increase in the speed for performing a DEMA is expected.

5.4 Power Analysis

It is promising, to use the RFID tool for execution of a (remote) power analysis similar to the one described above in Section 1.3.3. The DOUT1 pin of the EM 4094 can be programmed to act as direct analogue output of the received signal, which is proportional to the energy consumed by a tag. The signal can then either be acquired by the Atmel's internal ADC, i.e. the decision about a peak occurring or not, might be made by the microcontroller itself, or, if a better resolution is required, by an attached oscilloscope.

5.5 Fault Attacks

As the designed low level reader can be arbitrarily programmed, fault attacks are feasible, in which the device is forced to show erroneous performance. An error is introduced deliberately, by either sending invalid data to the tag or perturb other parameters like the power supply. The faulty behaviour eventually simplifies cryptanalysis, e.g., to deduce a secret key in combination with a DEMA.

5.6 Implementation of any Protocol

Any protocol based on the physical interface of the ISO 14443 can be implemented, by performing a corresponding software update of the microcontroller on the low level reader. This of course includes newly developed protocols, but also existent ones, for example those for MRTDs[5] like the e-passport. A comprehensive collection of libraries written in C, including the reader side protocol stack of the ISO 14443A (librfid) and the necessary functions for communication with the e-passport(libmrtd), has been developed

[5]Machine Readable Travel Documents

by Harald Welte and is freely available from the internet[6]. The adaptation to the developed reader should be fairly straightforward.

The RFID tool may also be extended for a prototype implementation of a distance bounding protocol [18] or other countermeasures against relay attacks, but this will probably bring a modification or extension of the hardware with it.

[6] `http://openmrtd.org`

6 Conclusion

A cost effective RFID low level reader and a fake tag have been designed and developed, which can be used for various promising purposes.

With the produced hardware, relay attacks between diverse RFID tags and a commercial ISO 14443 RFID reader have been carried out successfully.

Furthermore, possessing the RFID tool, one is in the position to emulate an RFID tag and thus perform replay attacks.

For fast communication between the low level reader and a PC, or other hardware, a USB port is provided on the circuit board.

It was discovered, that a commercial RFID reader does not obey certain timing requirements specified in the ISO 14443 standard and so eases relay attacks. Even if an RFID reader scrutinised the timing constraints, a method is proposed to still carry out the attack successfully with the designed hardware.

Employing ISO 14443 RFID tags in security sensitive applications should be regarded very critically, as the physical interface is proven to be insecure against relay attacks. The card identified by a reader does not have to be in its direct vicinity, as the data can be forwarded from large distances without permission and even without notification of the owner.

Various kinds of antennas, both for the reader and the Fake Tag, were built, tuned to resonance and tested with regard to operating range and the influence of different environments.

If an RFID tag is indispensable, a metal shielding is suggested, to prevent unauthorised usage of a tag. For security sensitive applications, it is proposed that a tag should not be able to become active, unless the owner has performed an action, e.g., press a button or open the cover of his electronic passport.

Any new protocol based on the ISO 14443A standard can be implemented, while existing protocols can be reverse engineered by means of logging the interchanged data.

A DEMA can be sped up significantly with the capabilities of the reader, just as fault attacks and remote power analysis can be investigated with RFID tags. A bunch of other analyses and investigations, that can be performed with the RFID tool, is proposed.

The developed tool will hopefully be useful in the research for new RFID security measures, help to find flaws in todays RFID systems and improve the security and privacy of future applications.

A Bibliography

[1] Logic threshold voltage levels. http://www.interfacebus.com/voltage_threshold.html.

[2] MasterCard and Visa agree to a common contactless communications protocol. http://www.corporate.visa.com/md/nr/press252.jsp.

[3] Texas Instruments to deliver RFID solution for MasterCard PayPass. http://www.ti.com/rfid/docs/news/news_releases/2005/rel01-17-05a.shtml.

[4] Philips' MIFARE identification chips just the ticket for London's Oyster Smart Card. http://www.semiconductors.philips.com/news/content/file_910.html, 2002.

[5] Use of contactless ICs in machine readable travel documents - annex 1. Technical report, ICAO, 2004. http://www.icao.int/mrtd/download/documents/Annex%20I%20-%20Contactless%20ICs.pdf.

[6] Atmel. ATMega32 data sheet. http://www.atmel.com/dyn/resources/prod_documents/doc2503.pdf.

[7] S. Bono, M. Green, A. Stubblefield, A. Juels, and Avi. Security analysis of a cryptographically - enabled RFID device. http://rfidanalysis.org/DSTbreak.pdf, Jan 2005.

[8] BSI - German Ministry of Security. ePass - Der Reisepass mit biometrischen Merkmalen. http://www.bsi.de/fachthem/epass/.

[9] D. Carluccio. Electromagnetic Side Channel Analysis for Embedded Crypto Devices. Master's thesis, Chair for Communication Security at the Ruhr University Bochum, 2005.

[10] D. Carluccio, K. Lemke, and C. Paar. Electromagnetic Side Channel Analysis of a Contactless Smart Card: First Results. In *ECRYPT Workshop on RFID and Lightweight Crypto*, pages 44–51, Graz, Austria, July 2005. ECRYPT. http://www.iaik.tu-graz.ac.at/research/krypto/events/RFID-SlidesandProceedings/Proceedings-WSonRFIDandLWCrypto.zip.

[11] EM Microelectronics. EM4094 fact sheet. http://www.emmicroelectronics.com/webfiles/product/rfid/ds/EM4094_fs.pdf.

[12] Fairchild Semiconductors. Application note 313: DC electrical characteristics of MM74HC high speed logic. http://www.fairchildsemi.com/an/AN/AN-313.pdf.

[13] A. Fettweis. *Elemente nachrichtentechnischer Systeme*. Teubner, second edition, 1996.

[14] T. Finke and H. Kelter. Radio Frequency Identification Abhörmöglichkeiten der Kommunikation zwischen Lesegerät und Transponder am Beispiel eines ISO14443-systems. http://www.bsi.de/fachthem/rfid/Abh_RFID.pdf. BSI - German Ministry of Security.

[15] K. Finkenzeller. *RFID Handbook: Fundamentals and Applications in Contactless Smart Cards and Identification*. John Wiley and Sons, 2nd edition, 2003.

[16] FTDI. FT245 USB chip data sheet. http://www.ftdichip.com/Documents/DataSheets/DS_FT245R_v105.pdf.

[17] G. Hancke. A practical relay attack on ISO 14443 proximity cards. http://www.cl.cam.ac.uk/~gh275/relay.pdf, 2005.

[18] G. P. Hancke and M. G. Kuhn. An RFID distance bounding protocol. In *Proceedings of IEEE/Create-Net SecureComm*, pages 67–73. IEEE Computer Society Press, 2005.

[19] Hitachi. World's smallest and thinnest 0.15 x 0.15 mm, 7.5m thick RFID IC chip. http://www.hitachi.com/New/cnews/060206.html.

[20] International Rectifier. Data sheet for IRFD110 N-channel MOSFET. http://www.irf.com/product-info/datasheets/data/irfd110.pdf.

[21] ISO/IEC 10373 - 6. Identification cards - test methods - part 6: Proximity cards, 2001.

[22] ISO/IEC 14443. Identification cards - Contactless integrated circuit(s) cards - Proximity cards - Part 1-4. www.iso.ch, 2001.

[23] Z. Kfir and A. Wool. Picking virtual pockets using relay attacks on contactless smartcard systems. Cryptology ePrint Archive, Report 2005/052, 2005. http://eprint.iacr.org.

[24] I. Kirschenbaum and A. Wool. How to build a low-cost, extended-range RFID skimmer. Cryptology ePrint Archive, Report 2006/054, 2006. http://eprint.iacr.org/.

[25] S. Kumar, C. Paar, J. Pelzl, G. Pfeiffer, A. Rupp, and M. Schimmler. How to break DES for 8,980. In *International Workshop on Special-Purpose Hardware for Attacking Cryptographic Systems — SHARCS'06, Cologne, Germany*, April 2006.

[26] Y. Lee. Antenna circuit design for rfid applications. `http://ww1.microchip.com/downloads/en/AppNotes/00710c.pdf`. Microchip Application Note 710.

[27] T. Lohmann, M. Schneider, and C. Ruland. Analysis of power constraints for cryptographic algorithms in mid-cost RFID tags. In J. Domingo-Ferrer, J. Posegga, and D. Schreckling, editors, *Smart Card Research and Advanced Applications*, volume 3928 of *Lecture Notes in Computer Science*, pages 278–288. Springer, 2006.

[28] MAXIM. Data sheet for the max232: 5v-powered, multichannel rs-232 drivers/receivers. `http://pdfserv.maxim-ic.com/en/ds/MAX220-MAX249.pdf`.

[29] MAXIM. Application note 742, impedance matching and the smith chart: The fundamentals. `http://www.maxim-ic.com/appnotes.cfm/appnote_number/742`, 2001.

[30] Melexis. Application note: A power booster for the MLX90121. `http://www.melexis.com/prodfiles/0003881_AN90121_4_1.pdf`.

[31] Melexis. MLX90121 datasheet. `http://www.melexis.com/prodfiles/0004755_MLX90121_REV6.pdf`.

[32] National Semiconductor. Data sheet for the 7805 voltage regulator and others. `http://cache.national.com/ds/LM/LM341.pdf`.

[33] National Semiconductor. Datasheet for LM311 voltage comparator. `http://www.national.com/pf/LM/LM311.html#Datasheet`.

[34] Y. Oren and A. Shamir. Power analysis of RFID tags. `http://www.wisdom.weizmann.ac.il/~yossio/rfid/`.

[35] C. Paar. Lecture Notes Applied Cryptography and Data Security, Dec 2004.

[36] C. Paar. Lecture Notes Implementation of Cryptographic Algorithms, 2004.

[37] Philips. Data sheet for 4 bit binary ripple counter 74393. `http://www.semiconductors.philips.com/pip/74HC393D#datasheet`.

[38] Philips. Data sheet for 7400 quad 2-input NAND gate. `http://www.semiconductors.philips.com/acrobat_download/datasheets/74HC_HCT00_3.pdf`.

[39] Philips. Data sheet for 7408 AND gate. `http://www.semiconductors.philips.com/pip/74HC08N`.

[40] Philips. Data sheet for 74HC(T)244 3-state octal buffer. http://www.semiconductors.philips.com/acrobat_download/datasheets/74HC_HCT244_3.pdf.

[41] Philips. Data sheet for D type flip-flop 7474. http://www.semiconductors.philips.com/pip/74F74.html#datasheet.

[42] Philips. Data sheet for monostable multivibrator 74HC/HCT123. http://www.semiconductors.philips.com/pip/74HCT123D#datasheet.

[43] Philips. Near Field Communication. http://www.semiconductors.philips.com/products/identification/nfc/.

[44] Philips. Data sheet for MIFARE Ultralight Contactless Single-trip Ticket IC. http://www.semiconductors.philips.com/acrobat_download/other/identification/M028630.pdf, 2003.

[45] Philips. Philips scores in German stadiums. *On the move*, page 3, Mar 2006.

[46] M. R. Rieback, B. Crispo, and A. S. Tanenbaum. The evolution of RFID security. *Pervasive Computing*, 5(1), Jan-Mar 2006.

[47] Texas Instruments. S6700 RFID transceiver datasheet. http://www.ti.com/rfid/docs/manuals/pdfSpecs/RI-R6C-001A.pdf.

[48] Texas Instruments. The bypass capacitor in high speed environments, Nov 1996.

[49] Texas Instruments. HF antenna cookbook technical application report. http://www.ti.com/rfid/docs/manuals/appNotes/HFAntennaCookbook.pdf, 2004.

[50] U. Tietze and C. Schenk. *Halbleiter - Schaltungstechnik*. Springer, eleventh edition, 2001.

B Layout and Schematics

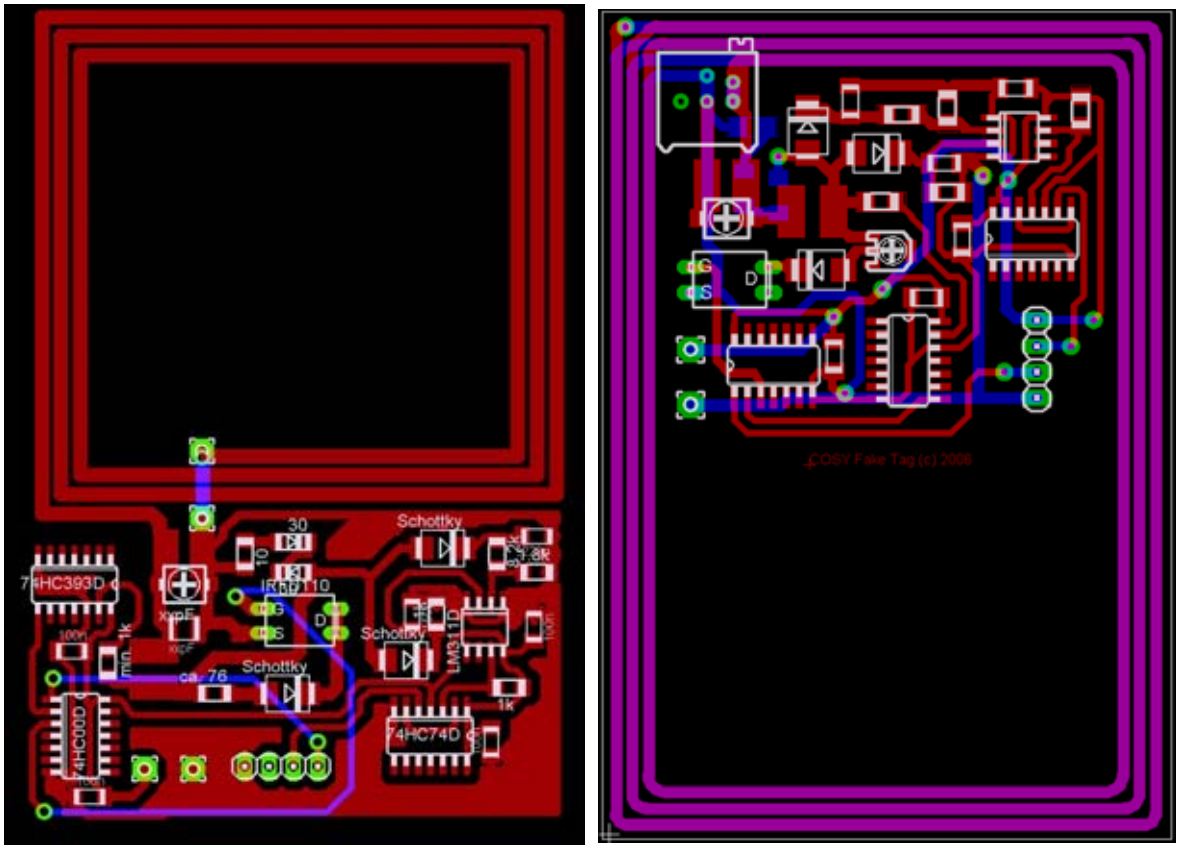


Figure B.1: Layout of the Fake Tag, Version 1 and Version 2

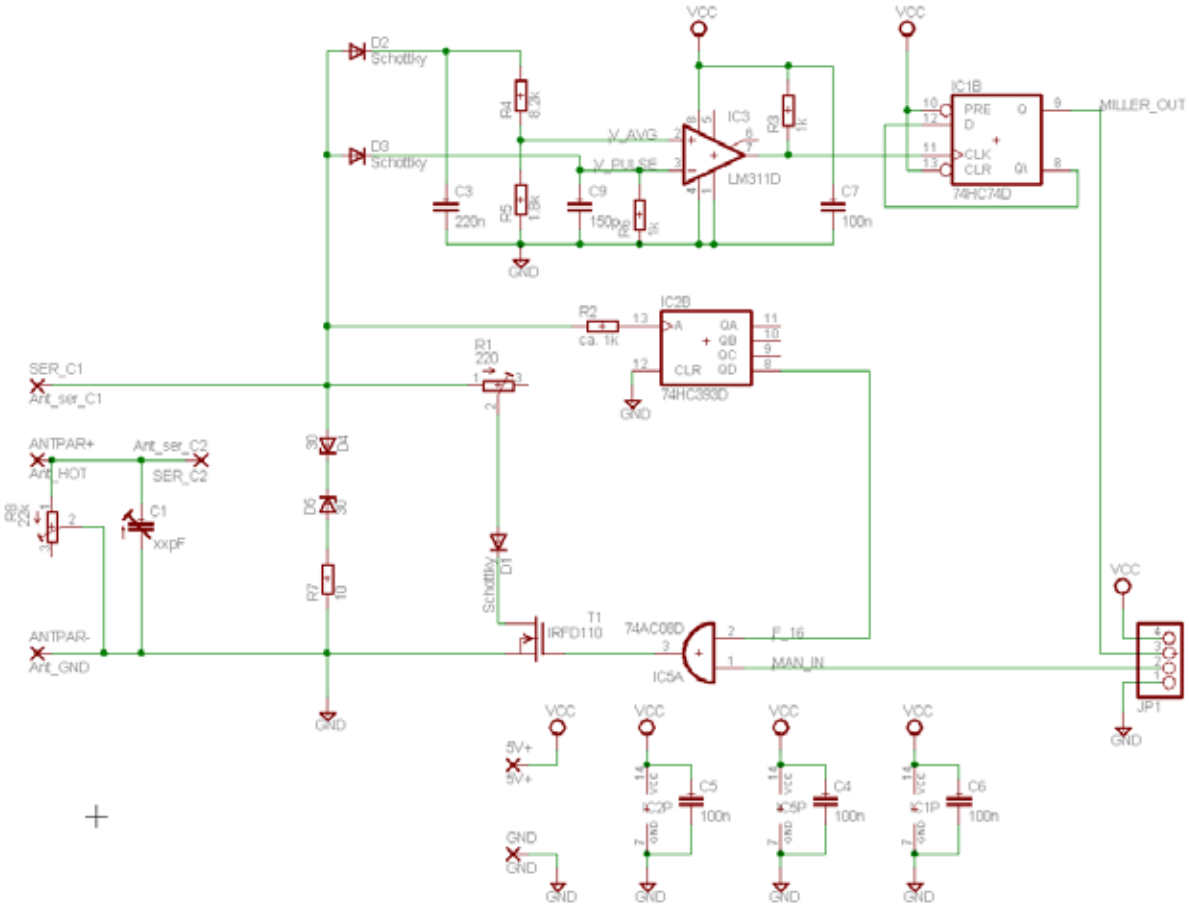


Figure B.2: Schematic of the Fake Tag, Version 2

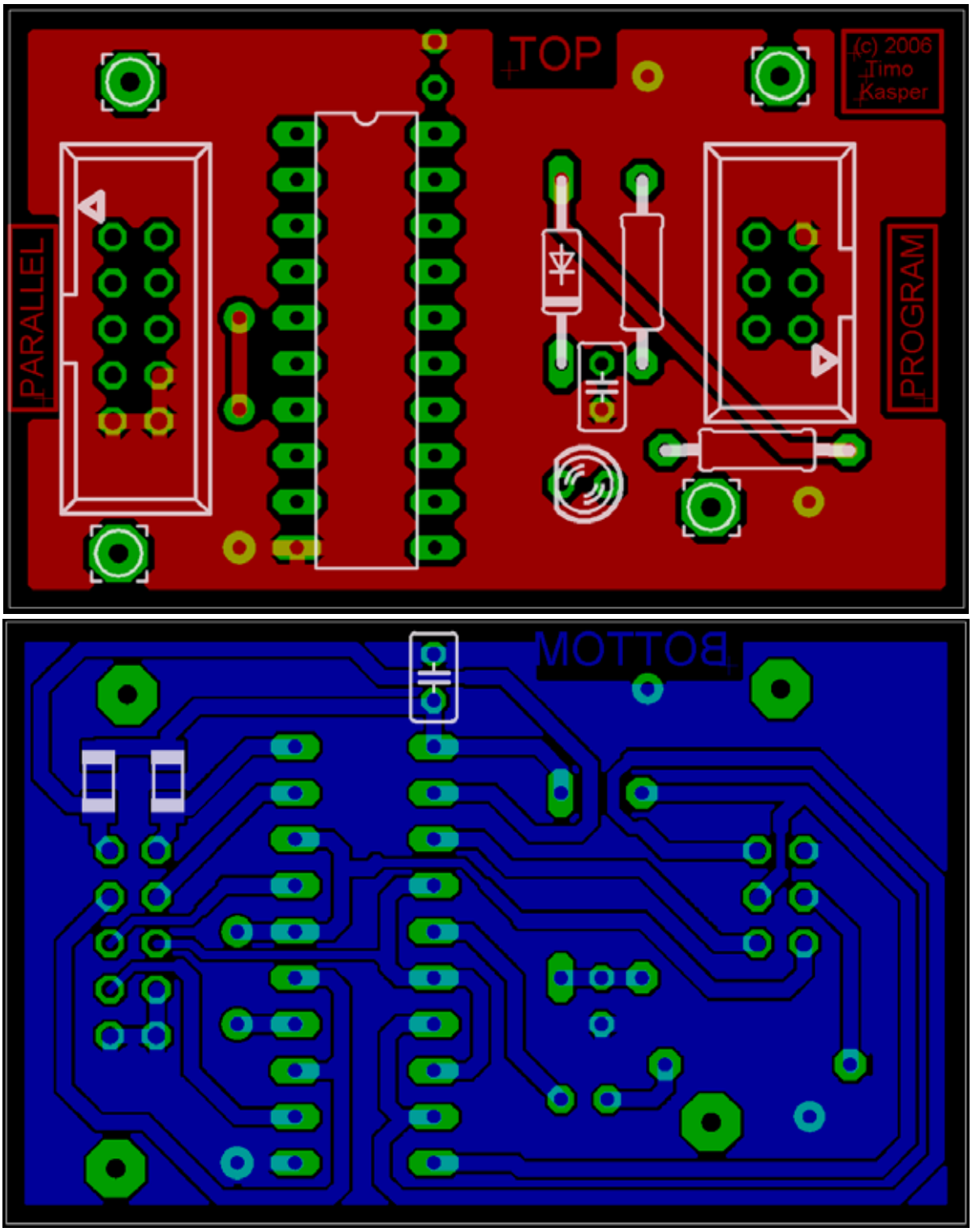


Figure B.3: Top and Bottom Layer of the Program Adapter

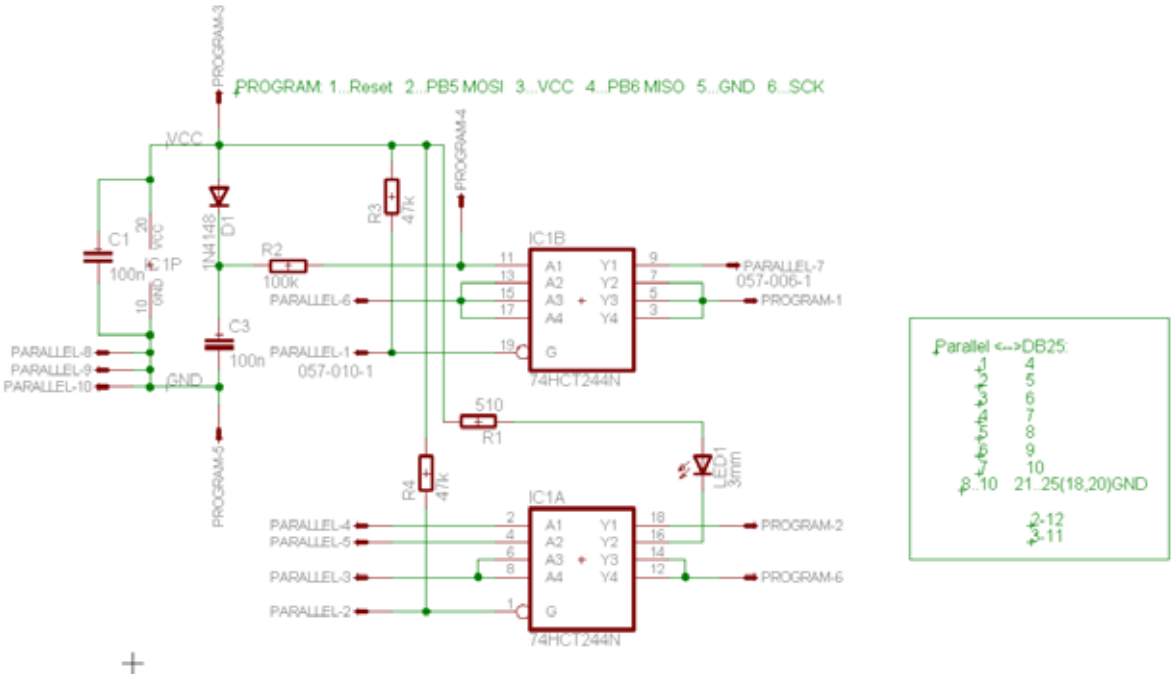


Figure B.4: Schematic of the Program Adapter

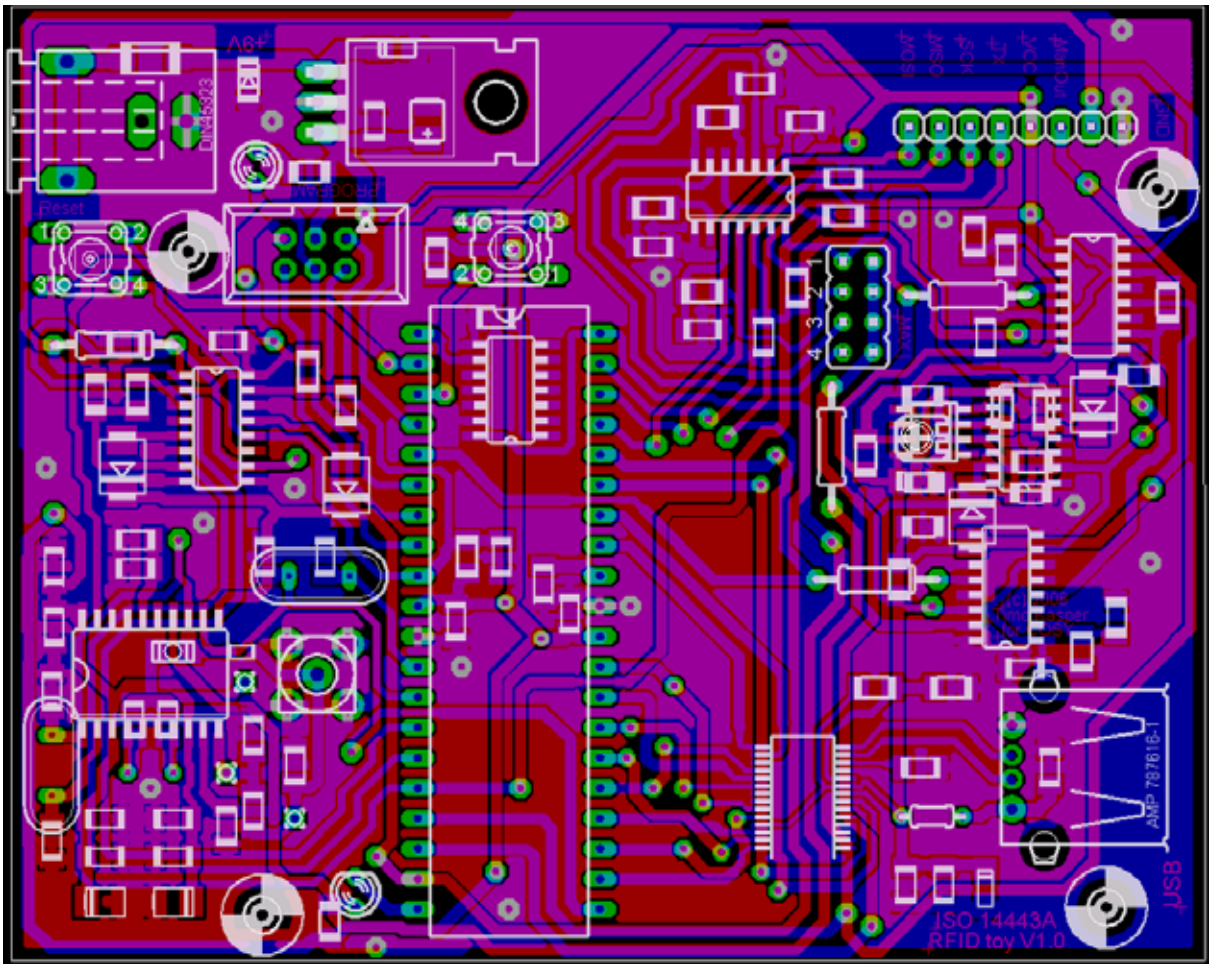


Figure B.5: Layout of the Reader, Version 2

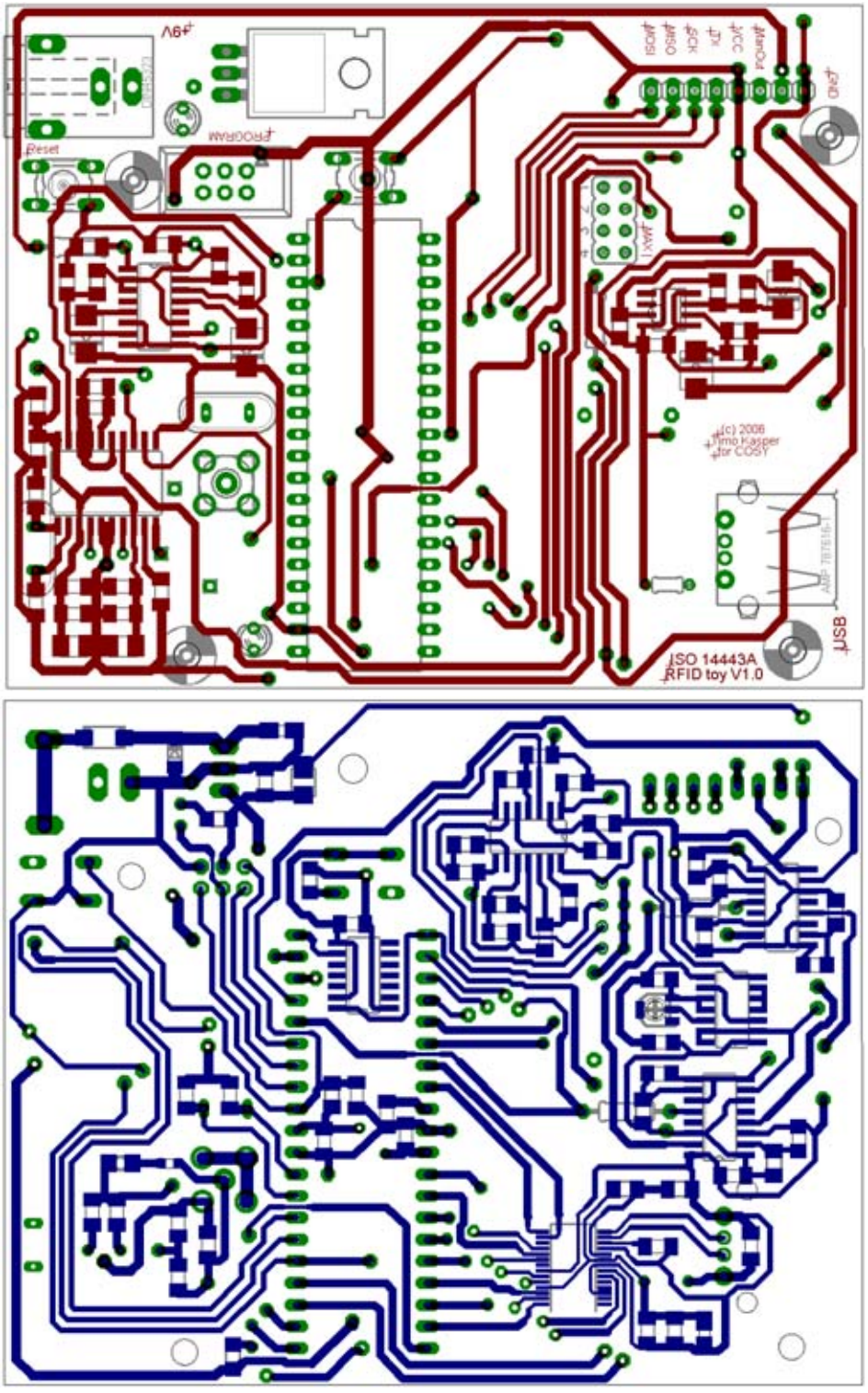


Figure B.6: Top and Bottom Layer of the Reader, Version 2

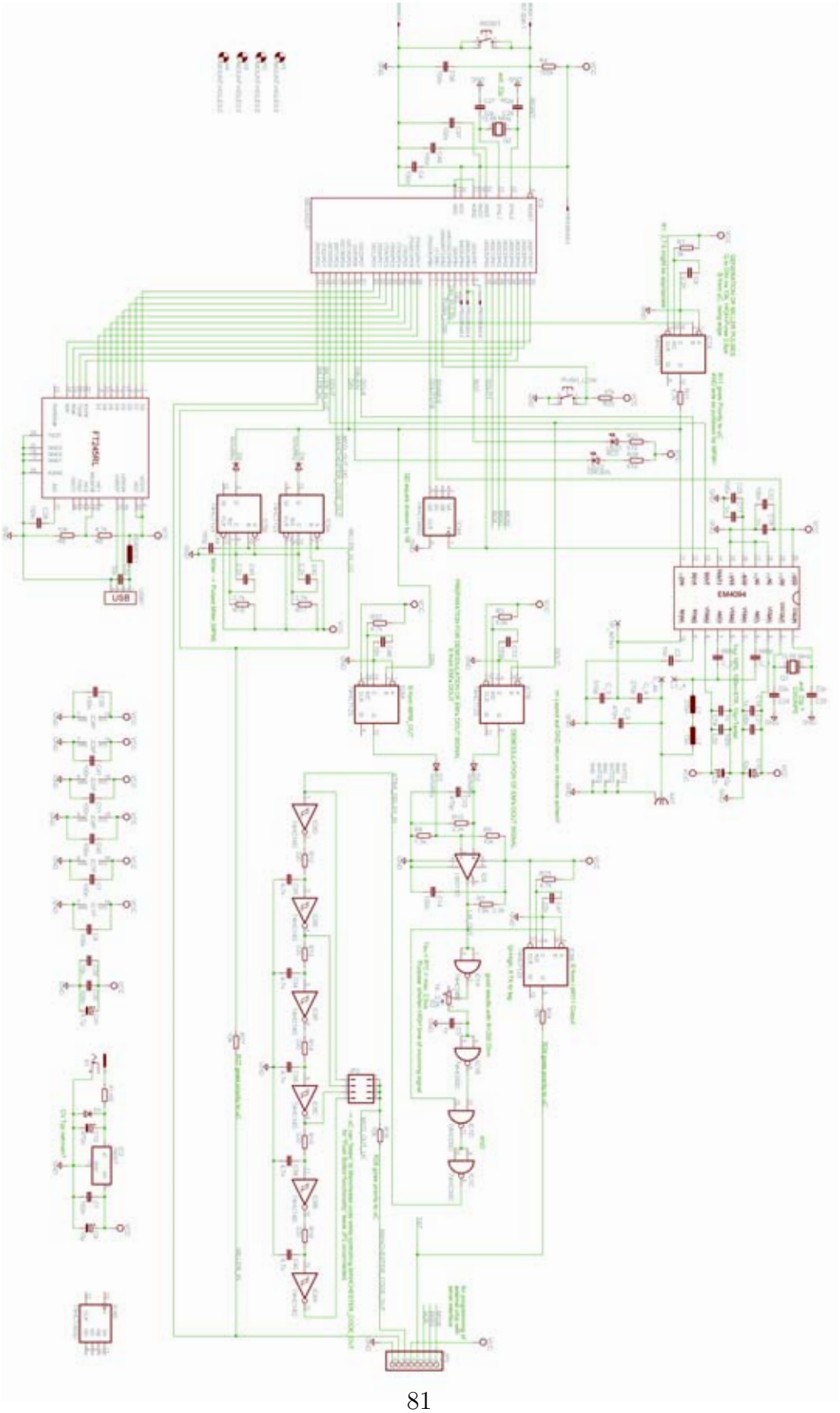


Figure B.7: Schematic of the Reader, Version 2

C Source Code Version 0.95

The source code Version 0.95 of the program on the microcontroller is compatible with the reader, version 1, and all functions implemented have been tested for proper function. Note that, as the software is still in the test phase, unnoticed bugs are possible.

C.1 board.h

```
// V0.95

#include <avr\io.h>  // loads the C type defined in the makefile
#include <stdint.h>  // #include <inttypes.h>

 //********* MACROS **********
 #define setBit(Byte, BitNo)   (Byte |=  (1<<BitNo))    // sets the BitNo in Byte
 #define clrBit(Byte, BitNo)   (Byte &= ~(1<<BitNo))   // clears the BitNo in Byte
 #define chkBit(Byte, BitNo)   (Byte & (1<<BitNo))    // true (!=0) , if Bit is set.

 //********* GLOBAL FLAGs in IO Locations **********
 #define Flag EEDR

 // defines used in EM4094lib
 #define ISRBusy 1
 #define IsPause 2


 //********** PROTOTYPES from EM4094lib **********
 void EM_InitOptionbits(void);
 void EM_Reset(void);
 void EM_Init(void);
 void EM_InitChip(void);
 void EM_Shiftdata(void);

 void EM_SetTimer0(uint8_t start, uint8_t max);
 void EM_InitTimer0(void);
 void EM_SetTimer1(uint16_t start, uint16_t max);
 void EM_InitTimer1(uint16_t cycles);
 void EM_SetTimer2(uint8_t start, uint8_t max);
 void EM_InitTimer2(void);

 void EM_SendStandardFrame (uint8_t *Data2Send, uint8_t DataLength);
 void EM_SendShortFrame (uint8_t);

 uint8_t EM_DoTheMiller(void);
 void EM_SendMiller(uint8_t);

 void EM_SendMan(uint8_t ManLength);
 uint8_t EM_DoTheMan(void); // ...chester

 void EM_Wait(uint16_t duration); // 16 Bit Timer1

 uint8_t EM_Parity(uint8_t byte, uint8_t mode);
```

```
//********** PROTOTYPES from etcetera **********
 void ETC_InitLEDs(void);
 void ETC_Init(void);
 void ETC_InitUSART(uint16_t baud);
 uint8_t ETC_CheckButton(void);

uint8_t ETC_ReceiveByte(void);
void ETC_TransmitByte (uint8_t data);


//********** PROTOTYPES from ftlib **********
void FT_Init(void);
void FT_Send(uint8_t *USBData, uint8_t length);
uint8_t FT_Receive(uint8_t *USBData);
void FT_InitChip(void);


 //********** VARIABLES from etcetera **********
 // obsolete flags to be optimised / put into "Flag"
 extern volatile uint8_t GogoLED; // flag for RunLEDs - used in former version of em4094lib & test.
 extern volatile uint8_t RXflag; // flag for USART Receive
 extern volatile uint8_t TXflag; // flag for USART Transmit
 extern volatile uint8_t TXdata; // Transmitted data
 extern volatile uint8_t RXdata; // Received data

 //********** VARIABLES from EM4094lib *********
   extern uint8_t MillerArray[256];
   extern uint8_t Frame2Send[256];

 // note: extern DECLARES variables for use in other libraries, still they must be DEFINED elsewhere!!
```

C.2 em4094lib.c

```
/*****************************************************************************
 *
 * Title: Library for EM Microelectronics EM4094 chip
 *
 * Author: Timo Kasper
 *
 * Date: 051224 (yymmdd)
 *
 * Version: 0.95
 *
 * Purpose: Control EM4094 Chip with Atmel Mega Microcontroller
 *
 * Software: avrgcc compiler
 *
 * Hardware: ATMega32 (can be other ATMegas) + EM4094 IC
 *
 *
 *
 * Demands: + library shall be adaptable to "any" pin assignment C<==>EM4094.
 *          + the chip's possibilities shall be accessible via the library functions.
 *          + All functions of this library have a default value.
 *   (see comments of individual functions)
 *
 *
 * Note: contents of this library are strictly confidental.
 *
 * (c) 2005 Timo Kasper
```

```
*
*****************************************************************************
*
* The Serial interface is used to control the EM4094 option bits setting.
*
* After reset, the DIN signal is shifted to the internal register on every rising edge in
* DCLK. During first 31 DCLK transitions the DIN data is read to the chip while during the
* 32nd transition the chip enters the normal mode.
*
* During normal mode:
* DIN is the modulation input (high DIN: low reader filed for ASK or no field for OOK).
* DCLK must be low in normal mode.
* DOUT and DOUT1 are data and clock outputs in normal mode.
*
* The EM4094 system selection bits (in the sequence send to chip) are:
* Bit 1 Power up flag
* Bit 2 Modulation index selection 0
* Bit 3 Modulation index selection 1
* Bit 4 Modulation index selection 2
* Bit 5 Short circuit protection enable
* Bit 6 Single or dual RF driver selection
* Bit 7 Dual driver in phase or phase Opposite
* Bit 8 Filter zero selection 1
* Bit 9 Filter zero selection 2
* Bit 10 Filter low pass selection 400kHz
* Bit 11 Receive gain selection 0 (LSB)
* Bit 12 Receiver gain selection 1
* Bit 13 Receiver gain selection 2 (MSB)
* Bit 14 AM PM input channel selection
* Bit 15 AGC on off selection
* Bit 16 AGC attack mode selection
* Bit 17 AGC decay mode selection
* Bit 18 AGC attack rate (lsb)
* Bit 19 AGC attack rate (msb)
* Bit 20 AGC decay wait (lsb)
* Bit 21 AGC decay wait (msb)
* Bit 22 Output selection direct sub-carrier or BPSK 848kHz
* Bit 23 BPSK automatic frequency adjust
* Bit 24 Output selection analog
* Bit 25 Hold delay after modulation selection
* Bit 26 Oscillator gain selection
* Bit 27 External oscillator
* Bit 28 -> 31 Test mode selection (all LOW for normal operation)
*
*****************************************************************************
* board's pinout:
*
* C           |    USB
* ------------+--------------------
* PA0...PA7  <-->   D0...D7
* PC5               RXF#
* PC4               TXE#
* PC6               RD#
* PC7               WR
*
* C           |    LEDs etc.
* ------------+--------------------
* PB4         -->   Rot1, LowActive
* PD4         -->   Rot2, LowActive
* PC3         -->   Gelb, LowActive
* PD3         -->   Grn, LowActive
* PC2         <--   Pushbutton, LowActive
*
```

```
* C           |    EM4094
* -------------+--------------------
* PD7         -->   DCLK
* PD6,PD2     <--   DOUT
* PB2         <--   DOUT1
* PB0         <--   DOUT1 / 16 (!)
* PD5         -->   DIN
* PB3         -->   DIN Pulse
* (note: JP2 ON --> DIN Pulse ACTIVE --> PD5 has to be set "TriState"!!)
* PB1         -->   ENABLE
*
* C           |    Serial
* -------------+--------------------
* PD1 (TXD)   -->   MAX232 T1IN
* PD0 (RXD)   <--   MAX232 R1OUT
*
* PC1 (SDA)  (<)--> 2WireInterface Pin1 SDA
* PC0 (SCK)  <--(>) 2WireInterface Pin2 SCK
*
*******************************************************************************/


// in board.h, extern variables etc. (everything that is used by all 3 libraries) are declared.



//********** INCLUDES **********
#include <stdint.h>  // #include <inttypes.h>
#include <avr\io.h>  // loads C type defined in makefile
#include <avr\delay.h>
#include <avr\signal.h> // necessary for ISRs
#include <avr\interrupt.h> // necessary for sei() / cli()
#include "board.h"




// ********** DEFINES **********
#define EOFrame 0xFF

#define MillerValue 63  // for setting of timer 0
#define WaitValue 2300

#define EM_SHDEL 1        // Shift delay (used during shifting serial data)

#define EM_EN_PORT PORTB // Enable Port
#define EM_EN_PIN PINB      // Enable Pin
#define EM_EN_DDR DDRB      // Enable DDR
#define EM_EN 1          // Enable Bit

#define EM_DIN_PORT PORTD // DIN Port
#define EM_DIN_PIN PIND      // DIN Pin
#define EM_DIN_DDR DDRD      // DIN DDR
#define EM_DIN 5          // DIN Bit

#define EM_DIN_PULSE_PORT PORTB // DIN_PULSE Port
#define EM_DIN_PULSE_PIN PINB      // DIN_PULSE Pin
#define EM_DIN_PULSE_DDR DDRB      // DIN_PULSE DDR
#define EM_DIN_PULSE 3          // DIN_PULSE Bit

#define EM_DCLK_PORT PORTD // DCLK Port
#define EM_DCLK_PIN PIND      // DCLK Pin
#define EM_DCLK_DDR DDRD      // DCLK DDR
```

```
 #define EM_DCLK 7          // DCLK Bit

 // DOUT is connected to 2 pins: PD2, PD6 --> DOUT_1, DOUT_2 (!= DOUT1)
 #define EM_DOUT_1_PORT PORTD // DOUT Port
 #define EM_DOUT_1_PIN PIND      // DOUT Pin
 #define EM_DOUT_1_DDR DDRD      // DOUT DDR
 #define EM_DOUT_1 2         // DOUT_1 Bit

 #define EM_DOUT_2_PORT PORTD // DOUT Port
 #define EM_DOUT_2_PIN PIND      // DOUT Pin
 #define EM_DOUT_2_DDR DDRD      // DOUT DDR
 #define EM_DOUT_2 6          // DOUT_2 Bit

 #define EM_DOUT1_PORT PORTB  // DOUT1 Port (don't mix up with DOUT_1 !)
 #define EM_DOUT1_PIN PINB       // DOUT1 Pin
 #define EM_DOUT1_DDR DDRB       // DOUT1 DDR
 #define EM_DOUT1 2        // DOUT1 Bit

 #define EM_DOUT1_16_PORT PORTB // DOUT1/16 Port
 #define EM_DOUT1_16_PIN PINB      // DOUT1/16 Pin
 #define EM_DOUT1_16_DDR DDRB      // DOUT1/16 DDR
 #define EM_DOUT1_16 0          // DOUT1/16 Bit

//For Manchester Code Output
 #define EM_MAN_OUT_PORT PORTC
 #define EM_MAN_OUT_PIN PINC
 #define EM_MAN_OUT_DDR DDRC
 #define EM_MAN_OUT 0



 //********* MACROS **********
 //(setBit/clrBit/chkBit(Byte, BitNo) defined in board.h)
 //(only needed for Output ports/pins, NOT for INPUT (e.g. DOUT1,DOUT)

 #define EN_HIGH   setBit (EM_EN_PORT, EM_EN)
 #define EN_LOW   clrBit (EM_EN_PORT, EM_EN)
 #define DIN_HIGH   setBit (EM_DIN_PORT, EM_DIN)
 #define DIN_LOW   clrBit (EM_DIN_PORT, EM_DIN)
 #define DCLK_HIGH   setBit (EM_DCLK_PORT, EM_DCLK)
 #define DCLK_LOW   clrBit (EM_DCLK_PORT, EM_DCLK)

 #define MAN_OUT_LOW clrBit (EM_MAN_OUT_PORT, EM_MAN_OUT)
 #define MAN_OUT_HIGH setBit (EM_MAN_OUT_PORT, EM_MAN_OUT)

//pulsed DIN:
 #define DIN_P_HIGH   setBit (EM_DIN_PULSE_PORT, EM_DIN_PULSE)
 #define DIN_P_LOW   clrBit (EM_DIN_PULSE_PORT, EM_DIN_PULSE)


 //Timers: 0. Init Timer Values ,1. INT on, 2. Timer start ; Do S.TH. ; 3. INT off 4. Timer stop.

 //T0
 #define INT_T0_ON setBit (TIMSK,OCIE0)  // ENable Timer0 Interrupt; (TIMSK, Bit OCIE0=1)
 #define INT_T0_OFF clrBit (TIMSK,OCIE0) // DISable Timer interrupt; (TIMSK, Bit OCIE0=0)

 #define T0_START setBit (TCCR0,CS00)   // No Prescaler, TIMER0 RUN
 #define T0_STOP clrBit(TCCR0,CS00);clrBit(TCCR0,CS01);clrBit(TCCR0,CS02)  // Timer0 STOP

 //T1
 #define INT_T1_ON setBit (TIMSK,OCIE1A)  // ENable Timer1 Output Compare A Interrupt; (TIMSK, Bit OCIE0=1)
 #define INT_T1_OFF clrBit (TIMSK,OCIE1A) // DISable Timer1 Output Compare A interrupt; (TIMSK, Bit OCIE0=0)
```

```
 #define T1_START setBit (TCCR1B,CS10)   // No Prescaler, TIMER1 RUN
 #define T1_STOP clrBit(TCCR1B,CS10);clrBit(TCCR1B,CS11);clrBit(TCCR1B,CS12)  // Timer0 STOP

 //T2
 #define INT_T2_ON setBit (TIMSK,OCIE2)  // ENable Timer2 Interrupt; (TIMSK, Bit OCIE2=1)
 #define INT_T2_OFF clrBit (TIMSK,OCIE2) // DISable Timer2 interrupt; (TIMSK, Bit OCIE2=0)

 #define T2_START setBit (TCCR2,CS20)   // No Prescaler, TIMER2 RUN
 #define T2_STOP clrBit(TCCR2,CS20);clrBit(TCCR2,CS21);clrBit(TCCR2,CS22)  // Timer2 STOP

 /* if short delay is needed (without using interrupts):
    void _delay_ms   (    double     __ms    )
    The maximal possible delay is 262.14 ms / F_CPU in MHz.

    void _delay_us  (   double   __us   )
   The maximal possible delay is 768 us / F_CPU in MHz.
note: (F_CPU has to be set correctly ion the makefile)
 */

// Note (if needed) NAKED_ISR --> NO registers saved at beginning,
// NO RETI at the END!! (Do manually!)
#define NAKED_ISR(vector) \
 void vector (void) __attribute__ ((naked)); \
 void vector (void)


//********** VARIABLE DEFINITIONS **********

// DO THE ARRAYS HAVE TO BE INITIALIZED ?

 uint8_t Frame2Send[256]; // ends with EOFrame (DEFINED above)
 // Frame2Send data format: 0=SOC, 1=One, 2=Zero, 3=EOC2 (EOC1 equals "0")

 // 1 Byte IS EQUAL TO ONE HALF BIT :-(
  uint8_t MillerArray[256];
  uint8_t ManArray[256];


// Bit field
volatile struct
{
  // UNSIGNED char ensures, that no bits are used as a +/- flag.
  volatile uint8_t Bit1 : 1; // Power up flag
  volatile uint8_t Bit2 : 1; // Modulation index selection 0
  volatile uint8_t Bit3 : 1; // Modulation index selection 1
  volatile uint8_t Bit4 : 1; // Modulation index selection 2
  volatile uint8_t Bit5 : 1; // Short circuit protection enable
  volatile uint8_t Bit6 : 1; // Single or dual RF driver selection
  volatile uint8_t Bit7 : 1; // Dual driver in phase or phase Opposite
  volatile uint8_t Bit8 : 1; // Filter zero selection 1
  volatile uint8_t Bit9 : 1; // Filter zero selection 2
  volatile uint8_t Bit10 : 1; // Filter low pass selection 400kHz
  volatile uint8_t Bit11 : 1; // Receive gain selection 0 (LSB)
  volatile uint8_t Bit12 : 1; // Receiver gain selection 1
  volatile uint8_t Bit13 : 1; // Receiver gain selection 2 (MSB)
  volatile uint8_t Bit14 : 1; // AM PM input channel selection
  volatile uint8_t Bit15 : 1; // AGC on off selection
  volatile uint8_t Bit16 : 1; // AGC attack mode selection
  volatile uint8_t Bit17 : 1; // AGC decay mode selection
  volatile uint8_t Bit18 : 1; // AGC attack rate (lsb)
  volatile uint8_t Bit19 : 1; // AGC attack rate (msb)
  volatile uint8_t Bit20 : 1; // AGC decay wait (lsb)
  volatile uint8_t Bit21 : 1; // AGC decay wait (msb)
```

```
  volatile uint8_t Bit22 : 1; // Output selection direct sub-carrier or BPSK 848kHz
  volatile uint8_t Bit23 : 1; // BPSK automatic frequency adjust
  volatile uint8_t Bit24 : 1; // Output selection analog
  volatile uint8_t Bit25 : 1; // Hold delay after modulation selection
  volatile uint8_t Bit26 : 1; // Oscillator gain selection
  volatile uint8_t Bit27 : 1; // External oscillator
  volatile uint8_t Bit28 : 1; // Test mode selection (all LOW for normal operation)
  volatile uint8_t Bit29 : 1; // Test mode selection (all LOW for normal operation)
  volatile uint8_t Bit30 : 1; // Test mode selection (all LOW for normal operation)
  volatile uint8_t Bit31 : 1; // Test mode selection (all LOW for normal operation)
// only 4 Byte of memory used
//access with Option.Bit2=0 (e.g.)

} Option;


// ********************************** ISRs **********************************
// keep ISRs short, eventually use global flags

SIGNAL (SIG_OUTPUT_COMPARE0) //from iom32.h, BIG LETTERS!!
{
    // 8 Bit Timer
    // Purpose: Output of "modified miller" coded data

// 1 Byte of MillerArray contains 1 Nibble Data (4 Bit Periods)
    // Count DOWN the Current Bits and UP the Bytes to keep order...

if (chkBit(Flag,IsPause)) // if Current "HalfBit" == 1
{ // Send Pause
DIN_P_HIGH;
DIN_P_LOW;
}
clrBit(Flag,ISRBusy);
}


SIGNAL (SIG_OUTPUT_COMPARE1A)
{
    // 16 Bit Timer
    // Purpose: WAIT (i.e. just clear flag)  for a period between 75 ns and 4,8 ms with a resolution of 75 ns
// e.g. 170 us is approx. 2305 clock cycles.

// Clear Flag
clrBit(Flag,ISRBusy);
}


SIGNAL (SIG_OUTPUT_COMPARE2)
{
    // 8 Bit Timer
// Purpose: Output of Manchester coded data

if (chkBit(Flag,IsPause))
 MAN_OUT_HIGH; // if IsPause=1 : Man.-Pin = HIGH
    else
// asm volatile("nop\n\t"::);

 MAN_OUT_LOW;  // else (IsPause=0) Man.-Pin = LOW

clrBit(Flag,ISRBusy);
}
```

```
// ******************************** FUNCTIONS ********************************

void EM_InitOptionbits(void)
//sets "save" (working with existing board) default values for optionbits.
{
  Option.Bit1=1; //Power Up

  Option.Bit2=1; //OOK modulation
  Option.Bit3=0;
  Option.Bit4=0;

  Option.Bit5=1; //short circuit protection ENabled

  Option.Bit6=1; //use ant1 AND ant2
  Option.Bit7=0; // * IN PHASE driving!!

  Option.Bit8=0; // * 300KHz receiving filter
  Option.Bit9=0;

  Option.Bit10=0; //high cut off frequency

  Option.Bit11=0; //nominal gain
  Option.Bit12=0;
  Option.Bit13=0;

  Option.Bit14=0; //RF input 1 selected

  Option.Bit15=1; // * AGC ACTIVATED
  Option.Bit16=0; //AGC: attack always
  Option.Bit17=0; //AGC: fast decay
  Option.Bit18=0; //AGC: "fast attack"
  Option.Bit19=0;
  Option.Bit20=0; //AGC: decay wait 44us
  Option.Bit21=0;

  Option.Bit22=0; //BPSK decoder: off (direct subcarrier output)
  Option.Bit23=0; //BPSK: no auto freq. adjust
  Option.Bit24=1; // ** ENABLED    BPSK: analogue output disabled
  Option.Bit25=0; //BPSK: no hold delay (?)

  Option.Bit26=1; // * oscillator: high gm
  Option.Bit27=0; //oscillator: use internal (quartz) oscillator

  Option.Bit28=0; //bits for test mode --> all LOW (for normal operation)
  Option.Bit29=0;
  Option.Bit30=0;
  Option.Bit31=0;
}

void EM_InitChip(void)
// to be configured as INPUT: DOUT_1,2 (both ports) , DOUT1
// to be configured as OUTPUT: EN, DIN (*), DIN_PULSE, DCLK
// (*) if JP2 is plugged on --> DIN must TRISTATE while sending miller data!!
// device is enabled after InitChip

// note: ORDER of commands is important (for outputs):
// 1st: set PullUp  (PORT)
// THEN set direction (DDR)
// otherwise "LOW-Peak" might occur ?!.

{
```

```
// 1. port/pin directions
  clrBit(EM_EN_PORT, EM_EN); // Low
  setBit(EM_EN_DDR, EM_EN); // Output

  clrBit(EM_DCLK_PORT, EM_DCLK); // Low
  setBit(EM_DCLK_DDR, EM_DCLK); // Output


  // During Initializing (Shift Data etc.) port DIN has to be used as OUTPUT!!!
  clrBit(EM_DIN_PORT, EM_DIN); // Low
  setBit(EM_DIN_DDR, EM_DIN); // Output

  clrBit(EM_DIN_PULSE_PORT, EM_DIN_PULSE); // Low
  setBit(EM_DIN_PULSE_DDR, EM_DIN_PULSE); // Output
  // --> No output from 74123 if no rising edge is generated willingly
  // note 10k "pull Down" resistor (former Jumper2) when output 74123=LOW.

  setBit(EM_DOUT_1_PORT, EM_DOUT_1); // Pull Up active
  clrBit(EM_DOUT_1_DDR, EM_DOUT_1); // Input

  setBit(EM_DOUT_2_PORT, EM_DOUT_2); // Pull Up active
  clrBit(EM_DOUT_2_DDR, EM_DOUT_2); // Input


  setBit(EM_DOUT1_PORT, EM_DOUT1); // Pull Up active
  clrBit(EM_DOUT1_DDR, EM_DOUT1); // Input

  setBit(EM_DOUT1_16_PORT, EM_DOUT1_16); // Pull Up active
  clrBit(EM_DOUT1_16_DDR, EM_DOUT1_16); // Input

// Manchester:
  clrBit(EM_MAN_OUT_PORT, EM_MAN_OUT); // Low
  setBit(EM_MAN_OUT_DDR, EM_MAN_OUT); // Output

  // 2. Enable Chip
  EN_HIGH;
  //_delay_ms(EM_SHDEL);
}


// Timer0
void EM_InitTimer0(void)
// after T = PRESCALER/f_cpu [s], the timer value is incremented (due 2 prescaler)
// hence time until next interrupt is T_int=OCR0*T

// INTERRUPTS ARE NOT GLOBALLY ENABLED!!!
// Timer is still STOPPED
{
  EM_SetTimer0 (0, MillerValue); // set Timer to ZERO and OCR for 106 KBit/s

  TCCR0=0; //set to zero first - TIMER STOP while CS0/1/2 == 0
  TCCR0 |= (1<<WGM01); // CTC mode set (Clear Timer on Compare match)
  //--> TOP = OCR0, Timer is reset to zero when compare match, counting UP.
  TIMSK=0;
  TIMSK |= (1<<OCIE0); // interrupt if compare match
  //not bothered about overflow...
}

void EM_SetTimer0(uint8_t start, uint8_t max)
{
  TCNT0=start; // set Timer value to 'start'
  OCR0=max;  // set Output Compare Register to 'max'
}
```

```
//Timer1
void EM_InitTimer1(uint16_t cycles)
// Output Compare match Interrupt after XX cycles
{
  EM_SetTimer1 (0, cycles); // set Timer to ZERO and OCR according to cycles

  TCCR1A=0; //everything zero

  TCCR1B=0; //set to zero first
  TCCR1B |= (1<<WGM12); // CTC mode set (Clear Timer on Compare match)

  //note: TIMSK is set to ZERO in InitTimer0
  TIMSK |= (1<<OCIE1A); // interrupt if compare match OCR1A (16Bit!)
}


void EM_SetTimer1(uint16_t start, uint16_t max)
// 16 Bit WRITE --> high byte first, then low byte
// 16 Bit READ --> low Byte first, then high byte.
// high byte is stored in TEMP
// In c fortunately the compiler handles the 16 Bit access...
{
  TCNT1=start; // set Timer value to 'start'
  OCR1A=max;  // set Output Compare Register to 'max'
}


//Timer2
void EM_InitTimer2(void)

{
  EM_SetTimer2 (0, MillerValue); // set Timer to ZERO and Output Compare Register - calculation see above

  TCCR2=0;
  TCCR2 |= (1<<WGM21); // CTC mode set (Clear Timer on Compare match)
  //TIMSK set to ZERO in InitTimer0
  TIMSK |= (1<<OCIE2); // interrupt if compare match
}


void EM_SetTimer2(uint8_t start, uint8_t max)
{
  TCNT2=start; // set Timer value to 'start'
  OCR2=max;  // set Output Compare Register to 'max'
}

void EM_Reset(void)
 // High level on DCLK pin and rising edge on DIN pin causes serial interface reset.
 // After Reset, DCLK is put LOW again, so data is shifted into the chip
 // with the next DCLK_HIGH. Note: Port direction has to be set appropriately!
 {
    DIN_LOW; //otherwise no rising edge guaranteed...
DCLK_HIGH;
DIN_HIGH; //now device is reset
DIN_LOW;
DCLK_LOW; //now everything is prepared for shifting data.
 }


/* void Switch2normal()
During normal mode: DIN is the modulation input (high DIN: low reader filed for ASK or no field for OOK).
```

```
DCLK must be low in normal mode.
DOUT and DOUT1 are data and clock outputs in normal mode.*/


void EM_Shiftdata(void)
// shifts the previously set option bits to the device
// premises: serial Reset is performed AND DCLK is LOW.
// AFTER this function, EM4094 chip is in NORMAL MODE! (DCLK=LOW,DIN=LOW)

// sequence for 1 bit:
// 1. put (next) option bit to DIN-Pin
// 2. wait EM_SHDEL ms
// 3. DCLK (LOW)-->HIGH (--> data is in shift register)
// 4. wait EM_SHDEL ms
// 5. DCLK (HIGH)-->LOW
// goto 1.
{
 char count;

 for (count=1;count <= 31;count++)
 {
    switch(count)
{
  // set corresponding option bit
  case 1: if (Option.Bit1) DIN_HIGH; else DIN_LOW;  break;
  case 2: if (Option.Bit2) DIN_HIGH; else DIN_LOW;  break;
  case 3: if (Option.Bit3) DIN_HIGH; else DIN_LOW;  break;
  case 4: if (Option.Bit4) DIN_HIGH; else DIN_LOW;  break;
  case 5: if (Option.Bit5) DIN_HIGH; else DIN_LOW;  break;
  case 6: if (Option.Bit6) DIN_HIGH; else DIN_LOW;  break;
  case 7: if (Option.Bit7) DIN_HIGH; else DIN_LOW;  break;
  case 8: if (Option.Bit8) DIN_HIGH; else DIN_LOW;  break;
  case 9: if (Option.Bit9) DIN_HIGH; else DIN_LOW;  break;
  case 10: if (Option.Bit10) DIN_HIGH; else DIN_LOW;  break;
  case 11: if (Option.Bit11) DIN_HIGH; else DIN_LOW;  break;
  case 12: if (Option.Bit12) DIN_HIGH; else DIN_LOW;  break;
  case 13: if (Option.Bit13) DIN_HIGH; else DIN_LOW;  break;
  case 14: if (Option.Bit14) DIN_HIGH; else DIN_LOW;  break;
  case 15: if (Option.Bit15) DIN_HIGH; else DIN_LOW;  break;
  case 16: if (Option.Bit16) DIN_HIGH; else DIN_LOW;  break;
  case 17: if (Option.Bit17) DIN_HIGH; else DIN_LOW;  break;
  case 18: if (Option.Bit18) DIN_HIGH; else DIN_LOW;  break;
  case 19: if (Option.Bit19) DIN_HIGH; else DIN_LOW;  break;
  case 20: if (Option.Bit20) DIN_HIGH; else DIN_LOW;  break;
  case 21: if (Option.Bit21) DIN_HIGH; else DIN_LOW;  break;
  case 22: if (Option.Bit22) DIN_HIGH; else DIN_LOW;  break;
  case 23: if (Option.Bit23) DIN_HIGH; else DIN_LOW;  break;
  case 24: if (Option.Bit24) DIN_HIGH; else DIN_LOW;  break;
  case 25: if (Option.Bit25) DIN_HIGH; else DIN_LOW;  break;
  case 26: if (Option.Bit26) DIN_HIGH; else DIN_LOW;  break;
  case 27: if (Option.Bit27) DIN_HIGH; else DIN_LOW;  break;
  case 28: if (Option.Bit28) DIN_HIGH; else DIN_LOW;  break;
  case 29: if (Option.Bit29) DIN_HIGH; else DIN_LOW;  break;
  case 30: if (Option.Bit30) DIN_HIGH; else DIN_LOW;  break;
  case 31: if (Option.Bit31) DIN_HIGH; else DIN_LOW;  break;
    }

    // shift once
    DCLK_HIGH;
    DCLK_LOW;
 } //end for (--> next bit)

    DIN_LOW;  // appears reasonable ;-)
```

```
// shift one more time to enter normal mode !
    DCLK_HIGH;
    DCLK_LOW;
}


void EM_Init(void)
 {

  // set variables (struct)
  EM_InitOptionbits();

  // set Port directions and Enable device
  EM_InitChip();
  // enable Device is included in InitChip

  // reset serial interface of device
  EM_Reset();

  // shift (previously set) data to the device.
  EM_Shiftdata();

  // after shiftdata: wait for DOUT=HIGH? - see below
  // for test purposes: delay instead
_delay_ms (1);

  EM_InitTimer0(); // Initialise the timer used to control MILLER communication
  EM_InitTimer1(WaitValue); // Initialise the "wait" timer
  EM_InitTimer2(); // Initialise the timer used to control MANCHESTER communication


  // IF power Down mode (bit 1 L-->H or EN L-->H) --> startup procedure of chip
  // --> DOUT pin = high for 100us, then chip goes 2 normal mode.

  // IF short circuit detected between startup --> DOUT remains LOW, DOUT1 goes HIGH.

  // note: normal mode --> dout1=clock output 13.56MHz
  // and high DIN = No field (in OOK) --> set DIN to low for tests.

  // --> eventually take this into account by
  // 1. make sure, that power down mode. (e.g. EN=LOW)
  // 2. Init and WAIT for DOUT rising edge for say 1-2 ms.
  // 3. if not, detect error (maybe even look for DOUT1=HIGH?) --> red LED on.
  }


void EM_SendShortFrame (uint8_t Byte2Send) // only 7 Bit, MSB will be ignored.
// Short frame:  [SOC b1...b7 EOC]  (LSB first)
// routine puts the above pattern into (Frame2Send) array
// 0=SOC, 1=One, 2=Zero, 3=EOC2 (EOC1 equals "0")
{
   uint8_t FramePointer=0;

   Frame2Send[FramePointer++]=0; // SOC

    for (uint8_t CurrentBit=0;CurrentBit<=6;CurrentBit++){
if (Byte2Send&(1<<CurrentBit)) {  // Bit "CurrentBit" is SET
// ********** SEND ONE **********
Frame2Send[FramePointer++]=1;
}
else { // Bit "CurrentBit" is NOT SET
// ********** SEND ZERO **********
```

```
Frame2Send[FramePointer++]=2;
}
} // END FOR

Frame2Send[FramePointer++]=2; // EOC part 1 = "Zero"
Frame2Send[FramePointer++]=3; // EOC part 2

    Frame2Send[FramePointer]=EOFrame;
}



//*****************************************************************************************
void EM_SendStandardFrame (uint8_t *Data2Send, uint8_t DataLength)
// DataLength=No. of Bytes in Data2Send Array
// limited to max. 28 data Bytes (array size!)

// Standard frame:  [SOC b1...b8 P b1...b8 P  ...(n times)...  EOC]  (LSB first, P=odd parity bit)
// routine puts the above pattern into (Frame2Send) array FIRST BYTE of Data2Send IS SENT FIRST!!
// 0=SOC, 1=One, 2=Zero, 3=EOC
{
   uint8_t DataPointer=0;
   uint8_t FramePointer=0;
   uint8_t tmp;

   Frame2Send[FramePointer++]=0; // SOC

   do
   {
    for (uint8_t CurrentBit=0;CurrentBit<=7;CurrentBit++){
    // Bit2Send = Byte2Send & (1<<CurrentBit); --> LSB sent first if CurrentBit counted UP
if (Data2Send[DataPointer]&(1<<CurrentBit)) {  // Bit "CurrentBit" is SET
// ********** SEND ONE **********
Frame2Send[FramePointer++]=1;
}
else { // Bit "CurrentBit" is NOT SET
// ********** SEND ZERO **********
Frame2Send[FramePointer++]=2;
}
} // END FOR

   // Add (Odd) Parity Bit
   // Careful: if Parity returns Zero, a "2" wants to be put in the frame array !
   tmp=EM_Parity(Data2Send[DataPointer],1);
   if (tmp==0) Frame2Send[FramePointer++]=(tmp+2); // Zero is "2" in frame array
   else Frame2Send[FramePointer++]=tmp;

   } while (++DataPointer < DataLength); // DataPointer must never become == DataLength in do..while loop

Frame2Send[FramePointer++]=2; // EOC part 1 = "Zero" (for miller compability)
Frame2Send[FramePointer++]=3; // EOC part 2 = "Real EOC indicator"
    Frame2Send[FramePointer]=EOFrame;
}


uint8_t EM_Parity(uint8_t byte, uint8_t mode)
// returns Parity bit for odd(mode=1) or even(mode=0) parity
// !! CAUTION: for Zero, a 0 is returned, but in Frame2Send Zero equals 2 !!
{
byte ^= (byte >> 4);
byte ^= (byte >> 2);
byte ^= (byte >> 1);
```

```
byte &= 1; //now byte contains the bit to add for EVEN number of ones
return (byte^mode);
}


uint8_t EM_DoTheMiller(void) // return: length of Miller Array
// 0=SOC, 1=One, 2=Zero, 3=EOC2 (EOC1 equals "0")
// input: next Data to send (coded like above) = Frame2Send
// output: Miller Array with information for ISR: "pause or nothing"
// 2DO: 1. get 1st(next) Byte from array
// 2. Decide What to send
//      3. put next 2 Bytes into MillerArray
{
  uint8_t LastBit=0; // 0=Zero (!), 1=One
  uint8_t FramePointer=0;
  uint8_t MillerByte=0;

  while (Frame2Send[FramePointer] != EOFrame)
  {
   switch (Frame2Send[FramePointer++])
   {
    // send SOC
    case 0:   MillerArray[MillerByte++] = 1; // SetBit / Pulse
MillerArray[MillerByte++] = 0; // ClrBit / NO Pulse
LastBit=0; break;

  // send ONE (Pause in middle of Bit period)
    case 1:   MillerArray[MillerByte++] = 0; // ClrBit / NO Pulse
MillerArray[MillerByte++] = 1; // SetBit / Pulse
LastBit=1; break;

// send ZERO
case 2:  if (LastBit){ // LB==1
MillerArray[MillerByte++] = 0; // ClrBit / NO Pulse
MillerArray[MillerByte++] = 0; // ClrBit / NO Pulse

}
else { // LB==0
MillerArray[MillerByte++] = 1; // SetBit / Pulse
MillerArray[MillerByte++] = 0; // ClrBit / NO Pulse

} // end if
LastBit=0; break;

    // send EOC part 2
case 3:   MillerArray[MillerByte++] = 0; // ClrBit / NO Pulse
MillerArray[MillerByte++] = 0; // ClrBit / NO Pulse
LastBit=0; break;
   } // end switch

  } // end while

  return MillerByte; // So ISR / SendMiller knows, when to STOP.
}


void EM_SendMiller(uint8_t MillerLength)
// calling sequence:
// 1. sendshortframe  2. dothemiller  3. sendmiller
// during SendMiller Timer Interrupt is activated and then deactivated again.

{
   uint8_t MillerPointer=0;
```

```
   EM_SetTimer0 (0, MillerValue); // (Re)set Timer0 and Output Compare Register
   setBit(Flag,ISRBusy);

// hier "schlimme" nderung (zeitmssig)?
// IsPause=MillerArray[MillerPointer++]; // Prefetch  1st bit value for ISR
//   if (MillerArray[MillerPointer++]){setBit(Flag,IsPause);}
//   else {clrBit(Flag,IsPause);}

   INT_T0_ON;
   T0_START;

  do {
    if (MillerArray[MillerPointer++]){setBit(Flag,IsPause);}
    else {clrBit(Flag,IsPause);}
setBit (Flag,ISRBusy);
do {} while (chkBit(Flag,ISRBusy)) ; // react to current Half Bit Period

// IsPause=MillerArray[MillerPointer++]; // Prefetch NEXT bit value for ISR

  } while (--MillerLength);

    INT_T0_OFF;
T0_STOP;
}


uint8_t EM_DoTheMan(void) // ...chester.  return: length of ManArray
// 0=SOC, 1=One, 2=Zero, 3=EOC
// input: next Data to send (coded like above) = Frame2Send
// output: Manchester Array with information for ISR: HIGH or LOW Half Bit Period
// note: modulation with f/16 is done externally with a binary counter.
{
  uint8_t FramePointer=0;
  uint8_t ManByte=0;

  while (Frame2Send[FramePointer] != EOFrame)
  {
   switch (Frame2Send[FramePointer++])
   {
    // send SOC (= Logic ONE)
    case 0:   ManArray[ManByte++] = 1;
ManArray[ManByte++] = 0;
break;

  // send ONE
    case 1:   ManArray[ManByte++] = 1;
ManArray[ManByte++] = 0;
break;

// send ZERO / EOC ?
// note: Miller compability: case 3(EOC) is already included in case 2 !
case 2:    if (Frame2Send[FramePointer]==3) { // if next data to be sent = EOC
ManArray[ManByte++] = 0; // ignore this ZERO and put EOC instead
ManArray[ManByte++] = 0;
}
else {  // this IS a Zero...
ManArray[ManByte++] = 0; // ...so let's send it.
ManArray[ManByte++] = 1;
}
break;

   } // end switch
```

```
  } // end while

  return ManByte; // So ISR / SendMan knows, when to STOP.
}


void EM_SendMan(uint8_t ManLength)
// calling sequence:
// 1. send short/standard frame  2. length=dothemiller  3. sendmiller(length)
// during SendMiller Timer Interrupt is activated and then deactivated again.

{
   uint8_t ManPointer=0;
   EM_SetTimer2 (0, MillerValue); // (Re)set Timer0 and Output Compare Register
   setBit(Flag,ISRBusy);

   INT_T2_ON;
   T2_START;

  do {
    if (ManArray[ManPointer++]){setBit(Flag,IsPause);}
    else {clrBit(Flag,IsPause);}
setBit (Flag,ISRBusy);
do {} while (chkBit(Flag,ISRBusy)) ; // react to current Half Bit Period

  } while (--ManLength);

    INT_T2_OFF;
T2_STOP;
}

void EM_Wait(uint16_t duration)
{
   setBit(Flag,ISRBusy);

   EM_SetTimer1(0, duration);

   INT_T1_ON;
   T1_START;

   while (chkBit(Flag,ISRBusy)); //Wait fot ISR

   INT_T1_OFF;
   T1_STOP;
}
```

C.3 etcetera.c

```
/* ********************************************************************************
 *
 * Title: Library with certain useful self-developed functions
 *
 * Author: Timo Kasper
 *
 * Date: 051224 (yymmdd)
 *
 * Version: 0.95
 *
 * Purpose: Put all useful things and those for testing in here.
 *
 * Software: avrgcc compiler
 *
```

```
 * Hardware: ATMega32 (can be other ATMegas)
 *
 *
 * (c) 2005 Timo Kasper
 *
 * ********************************************************************************/


 // function prototypes put into board.h for convenience


 //********** INCLUDES **********
 #include <stdint.h>  // #include <inttypes.h>
 #include <avr\io.h>  // loads C type defined in makefile
 #include <avr\signal.h> // necessary for ISRs
 #include <avr\interrupt.h> //necessary for sei() / cli()
 #include "board.h"


 // ********** DEFINES **********

 #define RED1_PORT PORTB //  Port
 #define RED1_PIN PINB      //  Pin
 #define RED1_DDR DDRB      //  DDR
 #define RED1 4          //  Bit

 #define RED2_PORT PORTD //  Port
 #define RED2_PIN PIND      //  Pin
 #define RED2_DDR DDRD      //  DDR
 #define RED2 4          //  Bit

 #define GREEN_PORT PORTD //  Port
 #define GREEN_PIN PIND      //  Pin
 #define GREEN_DDR DDRD      //  DDR
 #define GREEN 3          //  Bit

 #define YELLOW_PORT PORTC //  Port
 #define YELLOW_PIN PINC      //  Pin
 #define YELLOW_DDR DDRC      //  DDR
 #define YELLOW 3          //  Bit

 #define BUTTON_PORT PORTC //  Port
 #define BUTTON_PIN PINC      //  Pin
 #define BUTTON_DDR DDRC      //  DDR
 #define BUTTON 2          //  Bit



 // ********** VARIABLE DEFINITIONS **********
 // !!all variables read by ISR and main routine have to be volatile!!
 // (otherwise data might be put into registers and overwritten before ISR reads them)
 // (makes the access to these variables slower-not allowed to put in registers!!)
 volatile uint8_t LEDstate, dir;

/* not needed here but might be useful elsewhere:
typedef union
{
  uint16_t i16;
  struct
  {
    uint8_t i8l;
uint8_t i8h;
  };
```

```
} convert16to8;

convert16to8 baud;
*/



// ********** FUNCTIONS **********
void ETC_InitLEDs(void)
 // inits Port directions etc.
 {
  // to be optimised: write ONE BYTE instead of multiple bits.

  clrBit(RED1_PORT, RED1); // High
  setBit(RED1_DDR, RED1); // Output

  clrBit(RED2_PORT, RED2); // High
  setBit(RED2_DDR, RED2); // Output

  clrBit(YELLOW_PORT, YELLOW); // High
  setBit(YELLOW_DDR, YELLOW); // Output

  clrBit(GREEN_PORT, GREEN); // Low
  setBit(GREEN_DDR, GREEN); // Output


  setBit(BUTTON_PORT, BUTTON); // Pull Up active
  clrBit(BUTTON_DDR, BUTTON); // Input / Tristate

   LEDstate=0; //initialize running light
   dir=1; // 1 is count UP  0 is count DOWN.
 }


uint8_t ETC_CheckButton(void)
{   // TRUE if pressed
uint8_t Button;
Button = chkBit(BUTTON_PIN,BUTTON);
return (!Button);
}


void ETC_Init(void)
// Inits LEDs, USART

{
  ETC_InitLEDs();
  ETC_InitUSART(ubrr_setting); // different values have to be tested...
}


// OBSOLETE:

 // Baud rates: Refer to table p.164 ATMega32 datasheet for setting of UBRR
 // and aberration from exact value
 // For normal (not 2x) mode: UBRR = fosc/(16*baudrate) - 1
 // possible: #define baudrate 115200
 //           #define ubrr_setting f_cpu / (16 * baudrate ) - 1
 // but note: Truncation--> not properly rounded int.

// #define ubrr_setting 6 // 6 is approx.115200 bps @ 13.56MHz (soll 6,35)

#define ubrr_setting 14 // 14 is approx. 57600 bps @ 13.56MHz (soll 13,71)
```

```
// #define ubrr_setting 87 // 87 is approx. 9600 bps @ 13.56MHz

// #define ubrr_setting 705 // 705 is approx. 1200 bps @ 13.56MHz

 //UART
 #define INT_TX_ON setBit (UCSRB,UDRIE)  // ENable TX Interrupt
 #define INT_TX_OFF clrBit (UCSRB,UDRIE) // DISable TX Interrupt

 #define INT_RX_ON setBit (UCSRB,RXCIE)  // ENable RX Interrupt
 #define INT_RX_OFF clrBit (UCSRB,RXCIE) // DISable RX Interrupt

 #define TX_ON setBit (UCSRB,TXEN)  // ENable TX
 #define TX_OFF clrBit (UCSRB,TXEN) // DISable TX

 #define RX_ON setBit (UCSRB,RXEN)  // ENable RX
 #define RX_OFF clrBit (UCSRB,RXEN) // DISable RX


 volatile uint8_t GogoLED=0;
 volatile uint8_t TXdata; // for TX ISR
 volatile uint8_t RXdata; // for RX ISR

 // ********************************** ISRs **********************************
// keep ISRs short, eventually use global flags

SIGNAL (SIG_USART_RECV) // received char ready to be picked up
// RXC flag is ("automatically") cleared when reading UDR
// (ONE if new data is available)
// for test purposes: just read out UDR into Rxdata and set flag

{
  RXdata=UDR;
  // clrBit(Flag,ISRBusy); not needed: look for RXC to become ZERO  clrBit(Flag,ISRBusy);
}


SIGNAL (SIG_USART_DATA) // data buffer ready for new char to be transmitted
// UDRE flag is ("automatically") cleared when writing UDR
// --> chkBit(UCSRA,UDRE) is ZERO, when new data has been transferred to UDR
// AFTER RESET, UDRE FLAG IS AUTOMATICALLY SET
{
  UDR=TXdata;
  //not needed: look for UDRE to become ZERO  clrBit(Flag,ISRBusy);
}



uint8_t ETC_ReceiveByte(void)
{
  while (!chkBit(UCSRA,RXC));
  return UDR;
}

void ETC_TransmitByte (uint8_t data)
{
  while (!chkBit(UCSRA,UDRE));
  UDR=data;
}


// ********** USART functions **********
// save time --> no parity.
```

```
void ETC_InitUSART(uint16_t baud)
  // !Note: baud is NOT the baudrate, but the value to be written into UBRR!
  // Here comes a lot of code with not really much effect...

  // what about setting the port directions where the RX/TX pins are?
{
  uint8_t tmp;

  tmp=0; // just to be on the safe side
  // No doubling of transmission speed --> better durability against unprecise baud rate generation
  clrBit(tmp,U2X);
  // No Multiprocessor mode
  clrBit(tmp,MPCM);
  UCSRA=tmp;


  tmp=0; // just to be on the safe side
  // DISable RX complete interrupt
// is already cleared  clrBit(tmp,RXCIE);
  // DISable USART data register empty interrupt
// is already cleared  clrBit(tmp,UDRIE);
  // DISable Receiver
// is already cleared  clrBit(tmp,RXEN);
  // Enable Transmitter
// is already cleared  clrBit(tmp,TXEN);
  // No 9th Bit needed
//  clrBit(tmp,UCSZ2);
  UCSRB=tmp;

  tmp=0x80; // URSEL=MSB is One --> write to UCSRC
  // Set Asynchronous mode
  clrBit(tmp,UMSEL);
  // No Parity
  clrBit(tmp,UPM1);
  clrBit(tmp,UPM0);
  // 1 Stop Bit
  clrBit(tmp,USBS);
  // 8 Data Bits
  setBit(tmp,UCSZ1);
  setBit(tmp,UCSZ0);
  UCSRC=tmp;

  tmp=0; // URSEL=MSB is Zero --> write to UBRRH
  tmp|=(0x0F & (uint8_t)(baud>>8)); // cast --> only lower bits are taken into account
  // only write lower 4 Bits !!
  UBRRH=tmp; // write higher 4 Bits of UBRR
  UBRRL=(uint8_t)baud; // write lower Byte of UBRR

}


/*
void ETC_RunLEDs(void)
{
  if (GogoLED)
  {
    uint8_t temp;
temp=SREG;
cli(); // deactivate interrupts - "disturbance free code" begins
  GogoLED=0; // clear flag

  // make LEDs running:
```

```
  if (dir) LEDstate++; else LEDstate--;
  PORTA = ((PORTA&0xF0) | (0x0F& ~(1<<LEDstate))); //LEDs ON if pin LOW.
  // High nibble of port a remains unchanged (="Pull-upped"), low nibble=LEDs.

  if ((LEDstate == 3)&&  dir) dir=0;//count up AND "last LED" --> change direction.
  if ((LEDstate == 0)&& ~dir) dir=1;//count down AND "first LED" --> change direction.
      // if fourth LED=on then dir=down. if first LED=on then dir=up.

//"disturbance free code" ends
SREG=temp; // interrupt handling as it was before.
   }
}
*/
```

C.4 ftlib.c

```
/*****************************************************************************
 *
 * Title: Library for FT245RL chip
 *
 * Author: Timo Kasper
 *
 * Date: 051225 (yymmdd)
 *
 * Version: 0.95
 *
 * Purpose: Control FTDI FT245RL Chip with Atmel Mega Microcontroller
 *
 * Software: avrgcc compiler
 *
 * Hardware: ATMega32 (can be other ATMegas) + FT245RL IC
 *
 *
 *
 * Demands: + library shall be adaptable to "any" pin assignment C<==>FT245RL.
 *          + the chip's possibilities shall be accessible via the library functions.
 *
 *
 *  Receive data at uC from PC:
 *    1. wait while RXF# is H (L=data available)
 *       might not be necessary: >= 2 clockcycles between #RD H->L and RXF# L->H
 *    2. #RD=H
 *    3. #RD H --> L : Fetch current data from PC# into USB chip
 *    4. #RD L : data at D[7...0] is valid until #RD=H
 *    5. Fetch data into uC
 *       DURATION until next RD (step3) must be at least 50ms+80ms=130ms (>=2 clockcycles)
 *    proceed with 1 until all data is received, i.e. RFX# L for a long time
 *
 *  Send data from uC to PC:
 *    1. {wait while TXE# is H (L=chip ready for new data)}
 *       might not be necessary: >= 2 clockcycles between WR H->L and TXE# L->H
 *    2. WR = H
 *    3. Write valid data to D[7..0]
 *    4. WR H --> L : Transfer data into USB chip
 *    proceed with 1 until all data sent.
 *
 *
 *****************************************************************************
 * pinout: (version 1)
 *
 * C           |    USB
```

```
 * -------------+--------------------
 * PA0...PA7  <-->   D0...D7
 * PC5               RXF#
 * PC4               TXE#
 * PC6               RD#
 * PC7               WR
 *
 ********************************************************************************/



 //********** INCLUDES **********
 #include <stdint.h>  // #include <inttypes.h>
 #include <avr\io.h>  // loads C type defined in makefile
 #include <avr\delay.h>
 #include <avr\signal.h> // necessary for ISRs
 #include <avr\interrupt.h> // necessary for sei() / cli()
 #include "board.h"

 // in board.h, extern variables etc. (everything that is used by all libraries) are declared.

 /* if short delay needed (without using interrupts):
    void _delay_ms   (    double     __ms    )
    The maximal possible delay is 262.14 ms / F_CPU in MHz.

    void _delay_us  (   double   __us   )
   The maximal possible delay is 768 us / F_CPU in MHz.
note: (F_CPU has to be set correctly)
 */


 // ********** DEFINES **********

 //    value for RXtimer:
//     x=14000 (--> wait approx. 1ms) should be OK.
 #define RXTimer 14000

// working with Bytes --> No need for data pins to be bit adressable
 #define FT_DATA_DDR DDRA      // DDR

 // !! note "non-standard" names
 #define FT_DATA_PORT PORTA // Port for output
 #define FT_DATA_PIN PINA // Pin for input

 //The Rest
 #define FT_TXE_PORT PORTC // Port
 #define FT_TXE_PIN PINC      // Pin
 #define FT_TXE_DDR DDRC      // DDR
 #define FT_TXE 4          // Bit

 #define FT_RXF_PORT PORTC // Port
 #define FT_RXF_PIN PINC      // Pin
 #define FT_RXF_DDR DDRC      // DDR
 #define FT_RXF 5          // Bit

 #define FT_RD_PORT PORTC // Port
 #define FT_RD_PIN PINC      // Pin
 #define FT_RD_DDR DDRC      // DDR
 #define FT_RD 6          // Bit

 #define FT_WR_PORT PORTC // Port
 #define FT_WR_PIN PINC      // Pin
```

```
 #define FT_WR_DDR DDRC      // DDR
 #define FT_WR 7          // Bit

 #define RD_HIGH   setBit (FT_RD_PORT, FT_RD)
 #define RD_LOW   clrBit (FT_RD_PORT, FT_RD)
 #define WR_HIGH   setBit (FT_WR_PORT, FT_WR)
 #define WR_LOW   clrBit (FT_WR_PORT, FT_WR)

 #define DATA_INPUT FT_DATA_PORT = 0xFF; FT_DATA_DDR = 0x00  // Active Pull Ups, Inputs
 #define DATA_OUTPUT FT_DATA_PORT = 0x00; FT_DATA_DDR = 0xFF  // Low, Outputs

//********* MACROS **********
 //(setBit/clrBit/chkBit(Byte, BitNo) defined in board.h)

// faster - note that between 2 subsequent calls of the 2 macros below must be >=2 cycles
 #define FT_WRITE(CurrentByte) WR_HIGH; FT_DATA_PORT=USBData[CurrentByte++]; WR_LOW
 #define FT_READ(CurrentByte) RD_LOW; USBData[CurrentByte++]=FT_DATA_PIN; RD_HIGH

// might be needed somewhere: asm volatile("nop\n\t"::);

//********** VARIABLE DEFINITIONS **********
//(none)

// ******************************** FUNCTIONS *********************************

void FT_InitChip(void)
// to be configured as INPUT: TXE, RXF
// to be configured as OUTPUT: RD,WR
// Data Bits are initialized as INPUTS

{// port/pin directions
//INPUTS
  setBit(FT_TXE_PORT, FT_TXE); // Pull Up active
  clrBit(FT_TXE_DDR, FT_TXE); // Input

  setBit(FT_RXF_PORT, FT_RXF); // Pull Up active
  clrBit(FT_RXF_DDR, FT_RXF); // Input

  DATA_INPUT; // all data bits pullup active,inputs

//OUTPUTS
  clrBit(FT_RD_PORT, FT_RD); // Low
  setBit(FT_RD_DDR, FT_RD); // Output

  clrBit(FT_WR_PORT, FT_WR); // Low
  setBit(FT_WR_DDR, FT_WR); // Output
}


uint8_t FT_Receive(uint8_t *USBData) // data received is put in USBData, lenght(bytes, max. 256!) is returned.
{
 /*   !NOTE RD has to be H before first call (and RXF=L)!
 *    ! TIMER1 has to be initialised (EM_InitTimer1(WaitValue)) before calling !
 *
 *  Receive data at uC from PC:
 *    1. #RD=H
 *    2. #RD H --> L : Fetch current data from PC# into USB chip
 *    3. #RD L : data at D[7...0] is valid until #RD=H
 *       DURATION until next RD (step2) must be at least 50ms+80ms=130ms (>=2 clockcycles)
 *    proceed with 1 until all data is received, i.e. RFX# L for a long time
 */
   uint8_t CurrentByte=0;
```

```
   DATA_INPUT; //set direction of corresponding uC pins to input

   EM_SetTimer1(0, RXTimer); // set Timer1 Start and Timeout Value for "all data received"

   // start RX timeout timer and interrupt here, so that ISRBusy=0 when time out.
   // (Timer 1 ISR will clear flag)

   setBit(Flag,ISRBusy);
   // ISRBusy Flag will only be cleared (ny ISR) of timeout occurs

   INT_T1_ON;
   T1_START;

do
{
  if (!chkBit(FT_RXF_PIN,FT_RXF)) // if data available
   {
// Reset RX Timeout-Timer
TCNT1=0;
FT_READ(CurrentByte); // CurrentByte is automatically incremented
   }

    } while (chkBit(Flag,ISRBusy)); // while no RX timeout

     INT_T1_OFF; //RX Timer not needed any more.
     T1_STOP;

return CurrentByte; // equals number of received bytes in USBData array
}


void FT_Send(uint8_t *USBData, uint8_t length) // data to be sent (length < 256!) is in USBData
{
 /*   !NOTE WR has to be L before first call (and TXE = L)!
 *    1. WR = H
 *    2. Write valid data to D[7..0]
 *    3. WR H --> L : Transfer data into USB chip
 *    proceed with 1 until all data sent.
 */

DATA_OUTPUT; //set direction of corresponding uC pins to output

    while (chkBit(FT_TXE_PIN,FT_TXE)); // wait for TXF to become LOW


    for (uint8_t CurrentByte=0;CurrentByte<length;) //note: CurrentByte is automatically incremented below!!
{
FT_WRITE(CurrentByte); // CurrentByte is automatically incremented
} // END FOR
}

void FT_Init(void)
 {
  // set Port directions
  FT_InitChip();

  // RD has to be set HIGH before first call of FT_Receive!
  RD_HIGH;

  // WR has to be set LOW before first call of FT_Send!
  WR_LOW;
 }
```

C.5 test.c

```
// V 0.95

#include <avr\io.h>
#include <stdint.h>
#include "em4094lib.c"
#include <avr\delay.h>
#include "etcetera.c"
#include "board.h"
#include "ftlib.c"

#define onlyones 0xFF
#define onlyzeroes 0x00
#define onlymix 0xAA
#define onlymix2 0x55
#define pattern1 0x64

#define REQA 0x26
#define WUPA 0x52

// "0010011" to be sent --> (reverse,LSB sent first!) (0)1100100 = 0x64

//********************************** main **********************************

int main (void)
{
  uint8_t ML_tmp=0;
  uint16_t tmp=0;
  uint16_t delay=2020;

  uint8_t rcv_length=0;

  uint8_t ATQA[2]={0x04, 0x00}; // "Real Tag"'s answer to REQA

//  uint8_t USBTest[5]={0x41, 0x42, 0x43, 13,10}; // ASCII "A" and "B",CR,LF

   const char* Text ="Current "  "state: " "\n\r";

   uint8_t USBHello[]={10,13,'H','E','L','L','O','!',10,13};
   uint8_t USBRCV[100];

   USBRCV[0]='l';
   rcv_length=1;

  uint8_t state='l'; //default state is "listen"


// ----------- INITIALISATION -----------
  _delay_ms (100); // wait for EM4094 startup
  EM_Init(); // init communication with chip
  ETC_Init(); // init the rest

  // now: make DIN ready for "listening" mode
  // as DIN_Pulse / 74123 makes the pulses...
  clrBit(EM_DIN_DDR, EM_DIN); // Input
  clrBit(EM_DIN_PORT, EM_DIN); // Tristate (Pull Up = off)

  FT_Init(); // Init Directions etc. of FT chip
```

```
  // Timer0 Interrupt = OFF
  INT_T0_OFF; // not really needed...

  sei(); // Enable Global Interrupts

  FT_Send(USBHello,10); // message after reset
  FT_Send((uint8_t*)Text,17); // String Test ("Current Mode:")


// ----------- main loop -----------

while(1) //forever
  {

   switch(state)
{
  // set corresponding option bit
  case 'l': // listen
        FT_Send(USBRCV,rcv_length); // send received data = new state
    FT_Send("-->LISTEN\r\n",11); // display current mode;

    // Do nothing but switch yellow LED on
setBit(RED1_PORT, RED1); // High
setBit(RED2_PORT, RED2); // High
setBit(GREEN_PORT, GREEN); // High
clrBit(YELLOW_PORT, YELLOW); // Low = ON

  break;

  case 'r':  // request
        FT_Send(USBRCV,rcv_length); // send received data = new state
    FT_Send("-->REQA\r\n",9); // display current mode;

setBit(RED1_PORT, RED1); // High
setBit(RED2_PORT, RED2); // High
clrBit(GREEN_PORT, GREEN); // Low = ON
setBit(YELLOW_PORT, YELLOW); // High
EM_SendShortFrame(REQA);  //Send REQA
ML_tmp=EM_DoTheMiller();
EM_SendMiller(ML_tmp);

  break;

  case 'w':  // wake up
        FT_Send(USBRCV,rcv_length); // send received data = new state
    FT_Send("-->WUPA\r\n",9); // display current mode;

setBit(RED1_PORT, RED1); // High
setBit(RED2_PORT, RED2); // High
clrBit(GREEN_PORT, GREEN); // Low = ON
setBit(YELLOW_PORT, YELLOW); // High
EM_SendShortFrame(REQA);  //Send REQA
ML_tmp=EM_DoTheMiller();
EM_SendMiller(ML_tmp);

  break;

  case 'a': //answer to request
        FT_Send(USBRCV,rcv_length); // send received data = new state
    FT_Send("-->ATQA, waiting for rising edge at DIN pin: ",45);

clrBit(RED1_PORT, RED1); // Low = ON
setBit(RED2_PORT, RED2); // High
```

```
setBit(GREEN_PORT, GREEN); // High
setBit(YELLOW_PORT, YELLOW); // High
        // Prepare Data 2 send
EM_SendStandardFrame(ATQA,2); //Send ATQA
ML_tmp=EM_DoTheMan();
            //wait 4 DIN (very simple but working)
while(!chkBit(EM_DIN_PIN,EM_DIN)){}
EM_Wait(delay); // ging: EM_Wait(2280) ??;
EM_SendMan(ML_tmp);
    FT_Send("ATQA sent.\n\r",12);
  break;


  case 't': // test
        FT_Send(USBRCV,rcv_length); // send received data = new state
    FT_Send("-->ATQA, waiting for rising edge at DIN pin: ",45);

setBit(RED1_PORT, RED1); // High
clrBit(RED2_PORT, RED2); // Low = ON
setBit(GREEN_PORT, GREEN); // High
setBit(YELLOW_PORT, YELLOW); // High
        // Prepare Data 2 send
EM_SendStandardFrame(ATQA,2); //Send ATQA
ML_tmp=EM_DoTheMan();
            //wait 4 DIN (very simple but working)
while(!chkBit(EM_DIN_PIN,EM_DIN)){}
EM_Wait(delay);
EM_SendMan(ML_tmp);

FT_Send("ATQA sent with new delay ",25);

//Current delay
tmp=delay/1000;
USBHello[0]='0'+tmp;
tmp=(delay%1000)/100;
USBHello[1]='0'+tmp;
tmp=(delay%100)/10;
USBHello[2]='0'+tmp;
tmp=(delay%10);
USBHello[3]='0'+tmp;
USBHello[4]=13; //13=CR
USBHello[5]=10; //10=LF

FT_Send(USBHello,6);

delay=delay+20;
  break;

  default:
        FT_Send(USBRCV,rcv_length); // send received data = new state
    FT_Send(" is an invalid command.\n\r",25); // display current mode;
  break;
    }

// switch yellow LED on and green off --> green LED flash indicates received byte
setBit(GREEN_PORT, GREEN);
clrBit(YELLOW_PORT, YELLOW); // Low = ON

  _delay_ms (100);


//wait for input from USB chip
  rcv_length=0;
```

```
  while (rcv_length==0){  rcv_length=FT_Receive(USBRCV); }
    // switch green LED on --> byte received
clrBit(GREEN_PORT, GREEN); // Low=ON

    state=USBRCV[0];

    _delay_ms (100);


// TESTING
    // DIN_HIGH;
// artificial pause so scope can trigger on pause
/* for (int i=0;i<=50;i++) {DIN_P_HIGH;
DIN_P_LOW; // asm volatile("nop\n\t"::);
 } */

  } //end of while(1)


} //end of main routine
```

C.6 Makefile

```
#-----------------------------------------------------------------------------
# On command line / in Programmers Notepad:
#
# make all = Make software.
#
# make clean = Clean out built project files.
#
# make coff = Convert ELF to AVR COFF.
#
# make extcoff = Convert ELF to AVR Extended COFF.
#
# make program = Download the hex file to the device, using avrdude.
#                Please customize the avrdude settings below first!
#
# make debug = Start either simulavr or avarice as specified for debugging,
#              with avr-gdb or avr-insight as the front end for debugging.
#
# make filename.s = Just compile filename.c into the assembler code only.
#
# make filename.i = Create a preprocessed source file for use in submitting
#                   bug reports to the GCC project.
#
# To rebuild project do "make clean" then "make all".
#-----------------------------------------------------------------------------


# MCU name
MCU = atmega32

# Processor frequency.
#     This will define a symbol, F_CPU, in all source code files equal to the
#     processor frequency. You can then use this symbol in your source code to
#     calculate timings. Do NOT tack on a 'UL' at the end, this will be done
#     automatically to create a 32-bit value in your source code.
F_CPU = 13560000

# Output format. (can be srec, ihex, binary)
```

```
FORMAT = ihex


# Target file name (without extension).
TARGET = test


# List C source files here. (C dependencies are automatically generated.)
SRC = $(TARGET).c
# Timos test project consists solely of one file ;-)


# List Assembler source files here.
#     Make them always end in a capital .S.  Files ending in a lowercase .s
#     will not be considered source files but generated files (assembler
#     output from the compiler), and will be deleted upon "make clean"!
#     Even though the DOS/Win* filesystem matches both .s and .S the same,
#     it will preserve the spelling of the filenames, and gcc itself does
#     care about how the name is spelled on its command-line.
ASRC =


# Optimization level, can be [0, 1, 2, 3, s].
#     0 = turn off optimization. s = optimize for size.
#     (Note: 3 is not always the best optimization level. See avr-libc FAQ.)
OPT = s


# Debugging format.
#     Native formats for AVR-GCC's -g are dwarf-2 [default] or stabs.
#     AVR Studio 4.10 requires dwarf-2.
#     AVR [Extended] COFF format requires stabs, plus an avr-objcopy run.
DEBUG = dwarf-2


# List any extra directories to look for include files here.
#     Each directory must be seperated by a space.
#     Use forward slashes for directory separators.
#     For a directory that has spaces, enclose it in quotes.
EXTRAINCDIRS =


# Compiler flag to set the C Standard level.
#     c89   = "ANSI" C
#     gnu89 = c89 plus GCC extensions
#     c99   = ISO C99 standard (not yet fully implemented)
#     gnu99 = c99 plus GCC extensions
CSTANDARD = -std=gnu99



# Place -D or -U options here
CDEFS = -DF_CPU=$(F_CPU)UL


# Place -I options here
CINCS =



#---------------- Compiler Options ----------------
#  -g*:          generate debugging information
#  -O*:          optimization level
```

```
#  -f...:        tuning, see GCC manual and avr-libc documentation
#  -Wall...:     warning level
#  -Wa,...:      tell GCC to pass this to the assembler.
#    -adhlns...: create assembler listing
CFLAGS = -g$(DEBUG)
CFLAGS += $(CDEFS) $(CINCS)
CFLAGS += -O$(OPT)
CFLAGS += -funsigned-char -funsigned-bitfields -fpack-struct -fshort-enums
CFLAGS += -Wall -Wstrict-prototypes
CFLAGS += -Wa,-adhlns=$(<:.c=.lst)
CFLAGS += $(patsubst %,-I%,$(EXTRAINCDIRS))
CFLAGS += $(CSTANDARD)


#---------------- Assembler Options ----------------
#  -Wa,...:   tell GCC to pass this to the assembler.
#  -ahlms:    create listing
#  -gstabs:   have the assembler create line number information; note that
#             for use in COFF files, additional information about filenames
#             and function names needs to be present in the assembler source
#             files -- see avr-libc docs [FIXME: not yet described there]
ASFLAGS = -Wa,-adhlns=$(<:.S=.lst),-gstabs


#---------------- Library Options ----------------
# Minimalistic printf version
PRINTF_LIB_MIN = -Wl,-u,vfprintf -lprintf_min

# Floating point printf version (requires MATH_LIB = -lm below)
PRINTF_LIB_FLOAT = -Wl,-u,vfprintf -lprintf_flt

# If this is left blank, then it will use the Standard printf version.
PRINTF_LIB =
#PRINTF_LIB = $(PRINTF_LIB_MIN)
#PRINTF_LIB = $(PRINTF_LIB_FLOAT)


# Minimalistic scanf version
SCANF_LIB_MIN = -Wl,-u,vfscanf -lscanf_min

# Floating point + %[ scanf version (requires MATH_LIB = -lm below)
SCANF_LIB_FLOAT = -Wl,-u,vfscanf -lscanf_flt

# If this is left blank, then it will use the Standard scanf version.
SCANF_LIB =
#SCANF_LIB = $(SCANF_LIB_MIN)
#SCANF_LIB = $(SCANF_LIB_FLOAT)


MATH_LIB = -lm



#---------------- External Memory Options ----------------

# 64 KB of external RAM, starting after internal RAM (ATmega128!),
# used for variables (.data/.bss) and heap (malloc()).
#EXTMEMOPTS = -Wl,-Tdata=0x801100,--defsym=__heap_end=0x80ffff

# 64 KB of external RAM, starting after internal RAM (ATmega128!),
# only used for heap (malloc()).
#EXTMEMOPTS = -Wl,--defsym=__heap_start=0x801100,--defsym=__heap_end=0x80ffff
```

```
EXTMEMOPTS =



#---------------- Linker Options ----------------
#  -Wl,...:     tell GCC to pass this to linker.
#    -Map:      create map file
#    --cref:    add cross reference to  map file
LDFLAGS = -Wl,-Map=$(TARGET).map,--cref
LDFLAGS += $(EXTMEMOPTS)
LDFLAGS += $(PRINTF_LIB) $(SCANF_LIB) $(MATH_LIB)



##---------------- Programming Options (avrdude) ----------------
#
## Programming hardware: alf avr910 avrisp bascom bsd
## dt006 pavr picoweb pony-stk200 sp12 stk200 stk500
##
## Type: avrdude -c ?
## to get a full listing.
##
#AVRDUDE_PROGRAMMER = stk500
##
## com1 = serial port. Use lpt1 to connect to parallel port.
#AVRDUDE_PORT = com1    # programmer connected to serial device
#
#AVRDUDE_WRITE_FLASH = -U flash:w:$(TARGET).hex
##AVRDUDE_WRITE_EEPROM = -U eeprom:w:$(TARGET).eep
#
#
## Uncomment the following if you want avrdude's erase cycle counter.
## Note that this counter needs to be initialized first using -Yn,
## see avrdude manual.
##AVRDUDE_ERASE_COUNTER = -y
#
## Uncomment the following if you do /not/ wish a verification to be
## performed after programming the device.
##AVRDUDE_NO_VERIFY = -V
#
## Increase verbosity level.  Please use this when submitting bug
## reports about avrdude. See <http://savannah.nongnu.org/projects/avrdude>
## to submit bug reports.
##AVRDUDE_VERBOSE = -v -v
#
#AVRDUDE_FLAGS = -p $(MCU) -P $(AVRDUDE_PORT) -c $(AVRDUDE_PROGRAMMER)
#AVRDUDE_FLAGS += $(AVRDUDE_NO_VERIFY)
#AVRDUDE_FLAGS += $(AVRDUDE_VERBOSE)
#AVRDUDE_FLAGS += $(AVRDUDE_ERASE_COUNTER)


#---------------- Programming Options for usage of uisp ------------------
# more info: UISP --help

UISP_PROGRAMMER = stk200

UISP_PORT = 0x378
# programmer connected to lpt1

UISP_FILENAME = $(TARGET).hex
# filename of binary file wo be written to device
# at the moment nothing written to eeprom - i.e. simplified
```

```
UISP_TARGET_DEVICE = $(MCU)
# might be subject to change ( e.g. atmega32 or $(MCU) )

UISP_FLAGS = -dprog=$(UISP_PROGRAMMER) -dpart=$(UISP_TARGET_DEVICE) -dlpt=$(UISP_PORT)
UISP_FLAGS += --erase --upload --verify
UISP_FLAGS += if=$(UISP_FILENAME)
# not bothered about fuses etc. at the moment.


#---------------- Debugging Options ----------------

# For simulavr only - target MCU frequency.
DEBUG_MFREQ = $(F_CPU)

# Set the DEBUG_UI to either gdb or insight.
# DEBUG_UI = gdb
DEBUG_UI = insight

# Set the debugging back-end to either avarice, simulavr.
DEBUG_BACKEND = avarice
#DEBUG_BACKEND = simulavr

# GDB Init Filename.
GDBINIT_FILE = __avr_gdbinit

# When using avarice settings for the JTAG
JTAG_DEV = /dev/com1

# Debugging port used to communicate between GDB / avarice / simulavr.
DEBUG_PORT = 4242

# Debugging host used to communicate between GDB / avarice / simulavr, normally
#     just set to localhost unless doing some sort of crazy debugging when
#     avarice is running on a different computer.
DEBUG_HOST = localhost



#============================================================================


# Define programs and commands.
SHELL = sh
CC = avr-gcc
OBJCOPY = avr-objcopy
OBJDUMP = avr-objdump
SIZE = avr-size
NM = avr-nm
UISP = uisp
REMOVE = rm -f
COPY = cp
WINSHELL = cmd


# Define Messages
# English
MSG_ERRORS_NONE = Errors: none
MSG_BEGIN = -------- begin --------
MSG_END = --------  end  --------
MSG_SIZE_BEFORE = Size before:
MSG_SIZE_AFTER = Size after:
MSG_COFF = Converting to AVR COFF:
MSG_EXTENDED_COFF = Converting to AVR Extended COFF:
```

```
MSG_FLASH = Creating load file for Flash:
MSG_EEPROM = Creating load file for EEPROM:
MSG_EXTENDED_LISTING = Creating Extended Listing:
MSG_SYMBOL_TABLE = Creating Symbol Table:
MSG_LINKING = Linking:
MSG_COMPILING = Compiling:
MSG_ASSEMBLING = Assembling:
MSG_CLEANING = Cleaning project:



# Define all object files.
OBJ = $(SRC:.c=.o) $(ASRC:.S=.o)

# Define all listing files.
LST = $(SRC:.c=.lst) $(ASRC:.S=.lst)


# Compiler flags to generate dependency files.

#!!!!!!!!ERROR here ????!!!!!!!!!!!!!
# GENDEPFLAGS = -MD -MP -MF .dep/$(@F).d

GENDEPFLAGS = -MD -MP -MF $(@F).d

# Combine all necessary flags and optional flags.
# Add target processor to flags.
ALL_CFLAGS = -mmcu=$(MCU) -I. $(CFLAGS) $(GENDEPFLAGS)
ALL_ASFLAGS = -mmcu=$(MCU) -I. -x assembler-with-cpp $(ASFLAGS)




# Default target.
all: begin gccversion sizebefore build sizeafter end

build: elf hex eep lss sym

elf: $(TARGET).elf
hex: $(TARGET).hex
eep: $(TARGET).eep
lss: $(TARGET).lss
sym: $(TARGET).sym



# Eye candy.
# AVR Studio 3.x does not check make's exit code but relies on
# the following magic strings to be generated by the compile job.
begin:
@echo
@echo $(MSG_BEGIN)

end:
@echo $(MSG_END)
@echo


# Display size of file.
HEXSIZE = $(SIZE) --target=$(FORMAT) $(TARGET).hex
ELFSIZE = $(SIZE) -A $(TARGET).elf
```

```
AVRMEM = avr-mem.sh $(TARGET).elf $(MCU)

sizebefore:
@if test -f $(TARGET).elf; then echo; echo $(MSG_SIZE_BEFORE); $(ELFSIZE); \
$(AVRMEM) 2>/dev/null; echo; fi

sizeafter:
@if test -f $(TARGET).elf; then echo; echo $(MSG_SIZE_AFTER); $(ELFSIZE); \
$(AVRMEM) 2>/dev/null; echo; fi



# Display compiler version information.
gccversion :
@$(CC) --version




# program the device
program: $(TARGET).hex
$(UISP) $(UISP_FLAGS)




# Generate avr-gdb config/init file which does the following:
#     define the reset signal, load the target file, connect to target, and set
#     a breakpoint at main().
gdb-config:
@$(REMOVE) $(GDBINIT_FILE)
@echo define reset >> $(GDBINIT_FILE)
@echo SIGNAL SIGHUP >> $(GDBINIT_FILE)
@echo end >> $(GDBINIT_FILE)
@echo file $(TARGET).elf >> $(GDBINIT_FILE)
@echo target remote $(DEBUG_HOST):$(DEBUG_PORT)  >> $(GDBINIT_FILE)
ifeq ($(DEBUG_BACKEND),simulavr)
@echo load  >> $(GDBINIT_FILE)
endif
@echo break main >> $(GDBINIT_FILE)

debug: gdb-config $(TARGET).elf
ifeq ($(DEBUG_BACKEND), avarice)
@echo Starting AVaRICE - Press enter when "waiting to connect" message displays.
@$(WINSHELL) /c start avarice --jtag $(JTAG_DEV) --erase --program --file \
$(TARGET).elf $(DEBUG_HOST):$(DEBUG_PORT)
@$(WINSHELL) /c pause

else
@$(WINSHELL) /c start simulavr --gdbserver --device $(MCU) --clock-freq \
$(DEBUG_MFREQ) --port $(DEBUG_PORT)
endif
@$(WINSHELL) /c start avr-$(DEBUG_UI) --command=$(GDBINIT_FILE)




# Convert ELF to COFF for use in debugging / simulating in AVR Studio or VMLAB.
COFFCONVERT=$(OBJCOPY) --debugging \
--change-section-address .data-0x800000 \
--change-section-address .bss-0x800000 \
--change-section-address .noinit-0x800000 \
--change-section-address .eeprom-0x810000
```

```
coff: $(TARGET).elf
@echo
@echo $(MSG_COFF) $(TARGET).cof
$(COFFCONVERT) -O coff-avr $< $(TARGET).cof


extcoff: $(TARGET).elf
@echo
@echo $(MSG_EXTENDED_COFF) $(TARGET).cof
$(COFFCONVERT) -O coff-ext-avr $< $(TARGET).cof



# Create final output files (.hex, .eep) from ELF output file.
%.hex: %.elf
@echo
@echo $(MSG_FLASH) $@
$(OBJCOPY) -O $(FORMAT) -R .eeprom $< $@

%.eep: %.elf
@echo
@echo $(MSG_EEPROM) $@
-$(OBJCOPY) -j .eeprom --set-section-flags=.eeprom="alloc,load" \
--change-section-lma .eeprom=0 -O $(FORMAT) $< $@

# Create extended listing file from ELF output file.
%.lss: %.elf
@echo
@echo $(MSG_EXTENDED_LISTING) $@
$(OBJDUMP) -h -S $< > $@

# Create a symbol table from ELF output file.
%.sym: %.elf
@echo
@echo $(MSG_SYMBOL_TABLE) $@
$(NM) -n $< > $@



# Link: create ELF output file from object files.
.SECONDARY : $(TARGET).elf
.PRECIOUS : $(OBJ)
%.elf: $(OBJ)
@echo
@echo $(MSG_LINKING) $@
$(CC) $(ALL_CFLAGS) $^ --output $@ $(LDFLAGS)


# Compile: create object files from C source files.
%.o : %.c
@echo
@echo $(MSG_COMPILING) $<
$(CC) -c $(ALL_CFLAGS) $< -o $@


# Compile: create assembler files from C source files.
%.s : %.c
$(CC) -S $(ALL_CFLAGS) $< -o $@


# Assemble: create object files from assembler source files.
```

```
%.o : %.S
@echo
@echo $(MSG_ASSEMBLING) $<
$(CC) -c $(ALL_ASFLAGS) $< -o $@

# Create preprocessed source for use in sending a bug report.
%.i : %.c
$(CC) -E -mmcu=$(MCU) -I. $(CFLAGS) $< -o $@


# Target: clean project.
clean: begin clean_list end

clean_list :
@echo
@echo $(MSG_CLEANING)
$(REMOVE) $(TARGET).hex
$(REMOVE) $(TARGET).eep
$(REMOVE) $(TARGET).cof
$(REMOVE) $(TARGET).elf
$(REMOVE) $(TARGET).map
$(REMOVE) $(TARGET).sym
$(REMOVE) $(TARGET).lss
$(REMOVE) $(OBJ)
$(REMOVE) $(LST)
$(REMOVE) $(SRC:.c=.s)
$(REMOVE) $(SRC:.c=.d)
$(REMOVE) .dep/*



# Include the dependency files.
-include $(shell mkdir .dep 2>/dev/null) $(wildcard .dep/*)


# Listing of phony targets.
.PHONY : all begin finish end sizebefore sizeafter gccversion \
build elf hex eep lss sym coff extcoff \
clean clean_list program debug gdb-config
```