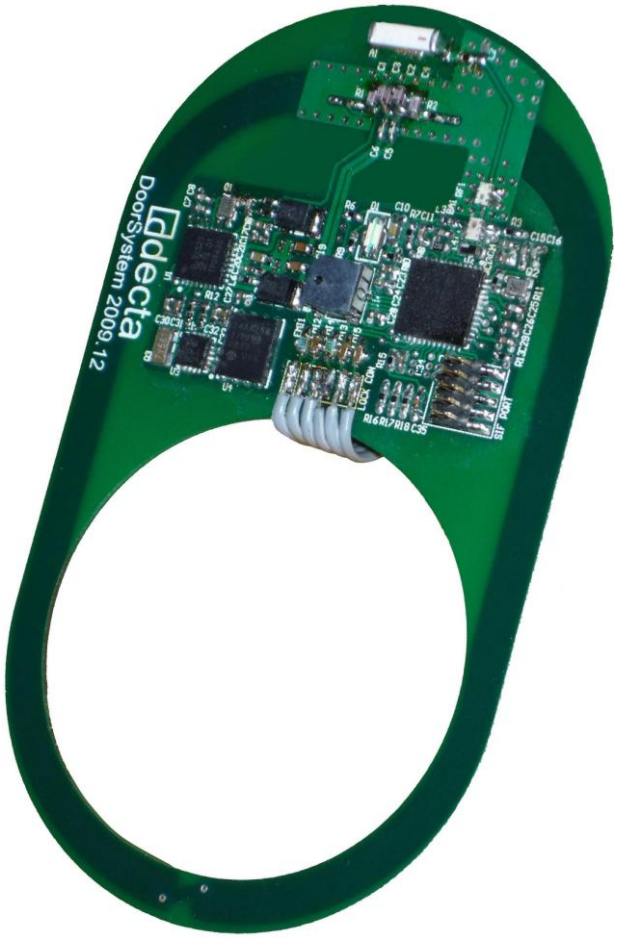
CHALMERS



Embedded Wirelessly Maintained RFID Card Entry System

Master of Science Thesis in Integrated Electronic System Design

MIKAEL HÖGRUD
JOHAN RIISBERG-JENSEN

Embedded Wirelessly Maintained RFID Card Entry System

MIKAEL HÖGRUD
JOHAN RIISBERG-JENSEN

Examiner: LARS SVENSSON

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Cover:
The fully functional hand-soldered production prototype, which was the result of the project detailed in this thesis. See page 72 for further information.

Department of Computer Science and Engineering
Göteborg, Sweden September 2010

Abstract

This thesis details the development of the new hardware and firmware of a next-generation door-entry system for a Swedish company named Decta AB. The developed system employs MiFare Radio-Frequency-IDentification (RFID) tags as keys, and uses ZigBee wireless communication for maintenance operations, such as granting or revoking user access etc. The solution is battery operated, and is required to have an expected operating time in excess of two years, using two Alkaline AA-cells (LR6).

The system features on-board memory used to store user-access permissions, time schedules, and log data, as well as an elaborate self-managing sorting algorithm, designed to provide fast user lookup, be power-efficient, and let the list remain searchable even during sorting and defragmentation operations.

In order to meet the demanding battery operating-time requirement, an infrared optical proximity detector was implemented in order to detect users, thereby allowing the system to sleep for extended periods of time.

Thanks to the system's small size, low cost, and the combination of attractive main features, it seems reasonable to assume that it offers an edge on the otherwise highly competitive market for door entry systems.

Keywords: ***RFID***, ***MiFare***, ***ISO14443***, ***ZigBee***, ***802.15.4***, ***entry system***, ***battery operated***, ***optical proximity detection***

Sammanfattning

Den här avhandlingen beskriver utvecklingen av ny hård- och mjukvara för ett nästa-generations dörrlåssystem, framtaget för ett svenskt bolag vid namn Decta AB. Det framtagna systemet använder MiFare Radio Frequency IDentification (RFID) taggar som nycklar, och använder sig av ZigBee trådlös kommunikation för att möjliggöra underhållsarbete som t ex att bevilja eller neka åtkomst för användare av systemet. Lösningen är batteridriven och specificerad till att ha en förväntad drifttid på som minst två år vid strömförsörjning via två vanliga alkaliska LR6-celler (AA).

Systemet är utrustat med inbyggt minne som används för att spara åtkomst-rättigheter, tidsscheman och logg-data. Tillika är systemet även utrustat med en innovativ själv-skötande listsorterings-algoritm, ämnad att hitta användare snabbt, vara energieffektiv, samt låta listan förbli sökbar även under sorterings- och defragmenterings-operationer.

För att möta de tuffa kraven gällande drifttid vid batteriförsörjning så infogades en infraröd optisk detektor till designen, varvid systemet tilläts att sova under längre tidsperioder än annars.

Tack vare systemets nätta storlek, låga kostnad och kombinationen av åtråvärda egenskaper, så verkar det rimligt att anta att det erbjuder vissa fördelar gentemot andra produkter på den annars väldigt konkurrens-utsatta marknaden för dörrlåssystem.

Nyckelord: ***RFID***, ***MiFare***, ***ISO14443***, ***ZigBee***, ***802.15.4***, ***dörrlåssystem***, ***batteridrivet***, ***optisk detektion***

Preface

This report is the result of a Master's thesis project carried out at Decta AB, in Gothenburg, Sweden. It constitutes the final part of the Master of Science Programme in Integrated Electronic System Design at the department of Computer Science and Engineering, at Chalmers University of Technology.

We would like to thank Mattias Karlsson and Peter Hagéus for providing the opportunity and means necessary for carrying out this thesis, as well as Lars Svensson who served to act as our examiner. We would also like to thank Jian Yang and Ahmed Hussain who helped us measure the ZigBee antenna-structure at Chalmers' facilities.

Table of Contents

List of Figures

List of Tables

Glossary

0x	Prefix indicating hexadecimal notation
AC	Alternating Current
ACK	ACKnowledged
AES	Advanced Encryption Standard
API	Application Programming Interface
APS	APplication Support
ASCII	American Standard Code for Information Interchange
ASK	Amplitude Shift Keying
ATQA	Answer To reQuest, type A
BALUN	BALanced to UNbalanced
BCC	Block-Check Character
BCD	Binary Coded Decimal
CPWG	Co-Planar WaveGuide
CRC	Cyclic Redundancy Check
CS	Chip Select
CSMA-CA	Carrier-Sense Multiple-Access Collision-Avoidance
DC	Direct Current
DES	Data Encryption Standard
DMA	Direct Memory Access
DSSS	Direct Sequence Spread Spectrum
EAP-E	Telegesis Ethernet Access Point
EEPROM	Electrically Erasable Programmable Read-Only Memory
EMC	Electro-Magnetic Compliance
EMI	Electro-Magnetic Interference
EPID	Extended Personal-area-network IDentification number
EUI	Extended Unique Identifier
FIFO	First-In-First-Out
GPIO	General Purpose Input/Output
HID	Human-Interface Device
I^2C	Inter-Integrated Circuit
IC	Integrated Circuit
ICT	In-Circuit Testing
ID	IDentification (number)
IEEE	Institute of Electrical and Electronics Engineers
IF	Intermediate Frequency
IR	InfraRed
ISM	Industrial Scientific Medical
KiB	1 KibiByte = 2^{10} bytes
Kibit	1 Kibibit = 2^{10} bits
LC	Inductor-Capacitor
LED	Light-Emitting Diode
LNA	Low-Noise Amplifier
MAC	Medium Access Control
MCU	Micro-Controller Unit
Mibit	1 Mebibit = 2^{20} bits
MIC	Message Integrity Check
mil	1 mil = 1/1000 inch
MISO	SPI Master-In Slave-Out

MOSI	SPI Master-Out Slave-In
NACK	Not ACKnowledged
NIST	National Institute of Standards and Technology
NVB	Number of Valid Bits
NWK	NetWorK
OEM	Original Equipment Manufacturer
OOK	On-Off Keying
O-QPSK	Offset-Quadrature Phase-Shift Keying
OUI	Organizational Unique Identifier
PA	Power Amplifier
PANID	Personal Area Network IDentification number
PC	Personal Computer
PCB	Printed Circuit Board
PCD	Proximity Coupling Device
PHY	PHYsical
PICC	Proximity Integrated Circuit Card
POE	Power-Over-Ethernet
RAM	Random Access Memory
RC	Resistor-Capacitor
REQA	REQuest to answer, type A
RF	Radio Frequency
RGB	Red Green Blue
RTC	Real-Time Clock
RX	Receive
SAP	Service Access Point
SCL	I^2C Serial CLock
SCLK	SPI Serial CLocK
SDA	I^2C Serial DAta
SEL	SELect
SIF	Serial InterFace
SOC	System-On-Chip
SPI	Serial Peripheral Interface
SS	Slave Select
TC	Trust Center
TWI	Two-Wire Interface
TX	Transmit
UART	Universal Asynchronous Register Transfer
UID	User-ID number
VCO	Voltage Controlled Oscillator
VNA	Vector Network Analyzer
WPAN	Wireless Personal Area Network
WUPA	Wake UP, type A
ZC	ZigBee Coordinator
ZCL	ZigBee Cluster Library
ZDO	ZigBee Device Object
ZED	ZigBee End-Device

1. Introduction

This chapter gives the motivations and reasons behind the development work carried out in this project.

1.1 Background

Decta AB, formerly the product development section of Confidence Sweden AB, situated in Västra Frölunda, Gothenburg, plans to develop a next-generation door-entry system. The new system, which is to be based on Radio-Frequency-IDentification (RFID) technology, is meant to implement a set of features that in combination are believed to offer a competitive advantage over the older line of products, as well as similar product offerings from competing companies.

The previous system relied on magnetic-strip cards, which required a bulky card-reader and lacked means of convenient communication for administration purposes. This lack of direct communication between doors and the card-issuer meant that, if a card, for any reason, had to be blocked before its preset expiration time, the system block-list had to be manually updated by an operator, walking from door to door with an administration card carrying the card number of the card to be blocked out.

Furthermore, the information needed to determine whether a card was valid or not, resided on the magnetic strip of the card itself, making the system reliant on the inherent security functions of the employed card standard. Thus, in order to both ease administration and increase security, it is desirable that system-critical information, such as when the card is valid and where, is moved from being stored on the card itself, to being stored inside the door-node. This relocation of data would allow for convenient changes in a user's access-rights, while increasing the difficulty of issuing fake cards.

If the system was to be employed in, for instance, a hotel, guests would only be allowed passage through a door if they could present an RFID-card with a valid card-number during a valid period of time. At check-in, the hotel clerk would give a guest a disposable RFID-card, whose card number could be activated or deactivated from the reception-desk wirelessly, via a certain Personal-Computer (PC) administration central. If the guest would like to stay longer than was initially agreed upon, or perhaps change room during the stay, the administrator would be able to accommodate those changes without the hassle of having to issue a new card, or even being in physical contact with the guest's card.

The complete door-lock system consists of mechanical assemblies, including handles and a lock-cylinder, which offer the possibility of manually unlocking the door, as well as two Printed Circuit Boards (PCB) realizing the desired electrical functions. One PCB is to be placed on the inside of the door, and is used to control the motor of the lock mechanism and provide the battery-based power-supply for the entire system. The other PCB, which is the result of this project, is to be placed around the lock-cylinder on the outside of the door. For aesthetic reasons, Decta would like to have the RFID-antenna placed around the lock-cylinder. The impact on communications from having the lock-cylinder, which is made out of chromed brass, in such close proximity to the antenna is

unclear, but if the possible degradation is only marginal, it can be considered acceptable.

A big challenge, apart from making the system reliable and secure, is to make it power efficient so that battery-life is extended for as long as possible. Power efficiency is a priority, because in a network with tens or hundreds of doors, frequent exchange of batteries might result in high operating costs, as well as a recurring annoyance for system administrators.

1.2 Purpose, Goal and Delimitation

The purpose of this project is to develop the complete, next-generation embedded hardware and firmware for a new door-entry system. Starting from abstract performance specifications, to manufacturing of a final production prototype, excluding mechanical assemblies and the development of the power supply and lock-mechanism controller-card. The PC administration software is also outside the scope of this project. Security concerns have been considered throughout the project, but some specific functions have been left unimplemented for reasons of the public nature of a thesis report and corporate confidentiality issues.

2. System Analysis and Method

In this chapter, the system and its environment are analyzed, and the development methodology used during the project work is laid out.

2.1 System Analysis

In a building employing Decta's new entry-system, a user may be authorized to enter certain areas during certain times, and otherwise prohibited to do so. For instance, an employee may be granted access to an office all day, but only allowed access to certain store rooms during working hours. If the employee were to quit his/her job, he/she could either be forced to hand in his/her key-card, or more desirable in case the card was forgotten or lost, have his/her key-card disabled.

In order to efficiently and cost-effectively deactivate a key-card and administer access rights for new ones, wireless networking capabilities are desired. In general, a system is considered to be an online system if the device is able to go online and query a central database whether a person is granted access or not. Conversely, a system is considered an offline system if it already has all the data necessary for making the decision without going online. An offline system has the advantage of not depending upon a functioning network, but possibly poses higher requirements on data storage in each single door-node. Different combinations of online and offline operation are of course also possible. A system querying a centralized database only when a card is not found in its internal storage, might be an appealing solution based on such a combination.

In order to give a more thorough picture of how a system such as this works, a few main concepts need to be defined and explained:

Gateway

A gateway is originally defined as the interconnecting node between two non-compatible networks. In the context of this project, it loosely refers to a maintenance central run on a PC, consisting of all the software and hardware required to manage the system and be able to communicate that with the nodes on the wireless network. The gateway would for instance update the door-nodes with the latest user-data configurations and time schedules, as well as receive logged data about who entered which doors at what time and so on. In an online system, the gateway would also have to maintain a database about users' access permissions.

User (or User-Number)

A user refers to a single key-card number, also called a Unique IDentifier (UID), or loosely, the individual carrying the key-card in question.

Door (or Door-Node)

A door refers to a single lock unit (an end-node in a networking context).

Time Schedule

A time schedule is a collection of one or several intervals in time, during which door access is permitted, for example:

Time Schedule	Time Interval(s)
TS1	Monday–Friday , between 8.00 a.m.–5.00 p.m.
TS2	Saturday–Sunday December 24th June–August
TS3	January–April, Odd weeks, Wednesdays, between 2.00 p.m.–3.00 p.m. December
TS4	Year 2009 Year 2010 Year 2012
TS5	(always)

Table 1: Time Schedules

Group

A group is a collection of doors, for example:

Group	Door(s)
G1	D2, D3
G2	D5

Table 2: Groups

Each door may belong to a single group and is then aware of which group it belongs to. Group belongings may be used as a way of addressing a cluster of door-nodes when intending to transmit the same data to all of them. The concept of groups is not explored extensively in this project, even though the functionality is supported and may be refined further.

Access Zone

An access zone refers to the association of one or many door-nodes with one time schedule, as exemplified in Table 3 below. Access zones are used inside the gateway only.

Access Zone	Door(s)	Time Schedule
AZ1	D2, D5	TS3
AZ2	D1, D2, D3, D4, D5	TS1
AZ3	D2, D3	TS2
AZ4	D2	TS4

Table 3: Access Zones

Access Configuration

An access configuration refers to the association of one user with one or many access zones, as exemplified in Table 4 below. Access configurations are used inside the gateway only.

Access Configuration	User	Access Zone
AC1	U1	AZ2
AC2	U2	AZ1, AZ2, AZ3

Table 4: Access Configurations

User-Data Configuration (or simply User-Data)

User-data configurations refer to the association of a user with a derived time schedule. In this example door D3 needs to store the following user-data configurations:

User-Data Configuration	User	Time Schedule
UDC1	U1	TS1
UDC2	U2	(TS1|TS2)=TS12

Table 5: User-Data Configurations

User-data configurations are primarily used inside door-nodes, but are issued from the gateway, which derives them from the stored access configurations and access zones.

An administrator setting up a system, might start by choosing a few doors and deciding that they should belong to the same access zone and be accessible only during a certain period of time. He/she may then continue, and go through a list of users, assigning to each of them a number of such access zones they are to be authorized to enter. This user-specific data, labelled access configurations, together with the list of access zones, provide the gateway with the information necessary to derive which users may be authorized to enter which doors at what times. These door-node–specific derivations called user-data configurations must, in an offline system, be sent out along with their associated time schedules, to each single door-node, where they must be saved. In an online system on the other hand, that information may simply be queried each time a user presents his or her card at the door.

Thus, in an offline system, each door-node must be able to accommodate memory large enough to store the user-data configurations and time schedules. Each door-node must of course also be able to keep track of what time it is, so that it can be determined whether a user is to be allowed entry at that time or not. For this reason, it is necessary to incorporate a real-time clock into an offline system. It is also usually desirable to keep track of which users enter the door, necessitating also a log-entry section. All of this functionality may simply be left out of an online system, as it can simply ask for, or forward that information.

Furthermore, security concerns differ for an online system, where communication between doors and the central gateway becomes sensitive, compared to in an offline system where protection of the local storage of information becomes critical instead. An offline system may also become more reliant on the security-mechanisms embedded in the key-card itself, since it might then not only be used to identify the user, but may also contain user-access rights.

Management of access zones and access configurations are up to the PC gateway software and lies beyond the scope of this project, but are defined and mentioned for the purpose of understanding the structure of a complete entry system.

2.2 Method

The applied working method relied on a top-down–bottom-up approach, where system-level requirements were analyzed, and necessary block-level functionality identified and modeled when considered useful. Low-level parts were then selected to accommodate for higher-level functionality while meeting the overall required performance goals, while at the same time also striving towards minimum cost. Furthermore, the constituent parts were integrated on a prototype PCB, which was used for firmware development and initial functionality demonstrations, before the actual design of the compact embedded-system PCB was undertaken. To facilitate initial development, a complete pre-manufactured ZigBee-module was used for the prototype, but was later replaced by a complete custom design. Firmware functionality was encapsulated and divided into different layers of abstraction, allowing for easy functional extension, design reuse, as well as changes to hardware while keeping higher-level functional behaviour the same and shell integrity intact. The list-sorting algorithm was first developed in MATLAB, to verify functionality, and then recreated in C. This approach offered the possibility of creating dual test-cases, where a test-case in MATLAB could be compared to that written in the C language.

3. System Requirements

In this chapter, the requirements on system performance and capability (as stated by Decta), and their implications are analyzed.

3.1 Given Requirements

The system requirements specified by Decta were:

- Minimum two-year operating time using two AA-sized battery cells
- MiFare RFID touchless–identification-card operation
- ZigBee wireless networking communication ability
- Offline operation with the capability of upgrading to make online queries
- Able to fit into standard door sockets, thus compatible with common door parts
- Manual mechanical unlocking ability
- Moderate to high total security
- Attractive and neat design
- Relatively low manufacturing cost

Furthermore, the size and shape of the final PCB was fixed at 87x50 mm, and two Integrated Circuits (IC) were preselected, namely the MFRC523 MiFare interrogator from NXP [1], and the ZigBee System-On-Chip (SOC) EM250 transceiver from Ember [2]. The pre-selection of the two ICs was due to the fact that development tools and samples had already previously been acquired by Decta.

3.2 Requirement Analysis

Clearly, the most difficult problem to solve will be controlling the power consumption, since it must be factored into design-decisions at every level, affecting not only the choices of hardware to be used in the system, but also how the software running on the hardware is allowed to operate. For example, unnecessary operations must be kept to a minimum and the system must spend as much time as possible sleeping. Fortunately, it is a probable scenario, that having a user at the door requesting access can be viewed as a comparatively rare event, occurring only at most every fifteen minutes or so, on average per day. However, on the other hand, since RFID-proximity cards of the type expected to be used by the system do not have an internal energy source, but rather relies on the card-reader to supply them with sufficient energy during transactions, it is likely that a large part of the active current consumption will come from energizing the RFID cards. This situation is especially troublesome, since the presence of any card within communication range is normally detected by polling, whereby the card-reader first supplies enough energy in order for a card in close proximity to wake up. If no card is present, that energy is wasted. Furthermore, if the polling is too infrequent, the user is bound to perceive the system as sluggish, thus offering an unsatisfactory service level. For this reason, it ought to be investigated if user-proximity can be detected by some other, more energy efficient, means.

In order to be compatible with standard door sockets, the pre-assigned board-space limits must not be exceeded. The total available board-space is only on the order of 40 cm^2, including the space needed for two antenna-structures and the protruding lock-cylinder needed for manual unlocking operation. Thus, the system requirements will force a rather high degree of integration, increasing the complexity of the layout work especially. In order to keep manufacturing costs low, it is also desirable to mount all components on one side of the board, and eliminate through-hole parts altogether. Since the system to be developed is meant to have offline capabilities but be wirelessly maintained, it must have all the hardware necessary to support full offline operation. This requirement necessitates the implementation of a real-time clock, as well as memory large enough to store the necessary data in the form of permitted cards, time schedules and log entries. Because of the wireless functionality, online operation may then be easily implemented at a later stage by simple firmware updates.

4. Technologies

In this chapter, a review of the characteristics of the major technologies employed in this project is conducted.

4.1 Description of ZigBee

ZigBee is a networking protocol aimed primarily at the market for relaying wireless sensor and control data.

Overview

The intention of ZigBee is to try and fill out a niche that other popular networking protocols fail to cover. It tries to do this by aiming for a few characteristics that, in combination, separates it somewhat from many of the more well-known networking protocols commonly in existence as of this day. These characteristics include:

- Low power consumption
- Low data rate
- High reliability
- Highly secure
- Medium to high range
- Self-healing mesh networking capability
- Cost effective
- Open and global standard

The ZigBee protocol itself is governed by the ZigBee Alliance , which is an association consisting of several companies cooperating in order to maintain and develop the ZigBee standard and proliferate its use. The ZigBee Alliance state the following as their primary focus areas [3]:

- Defining network, security and application software layers
- Providing interoperability and conformance testing specifications
- Promoting the ZigBee brand globally to build market awareness
- Managing the evolution of the technology

Any company may join the ZigBee Alliance and receive a Promoter, Participant or Adopter-class membership, granted that they pay the associated yearly fee. The different membership classes grant different amounts of influence on development of current and future versions of the ZigBee stack. Becoming a member is also necessary in order for Original Equipment Manufacturers (OEM) to ship products with the ZigBee logotype. Furthermore, the ZigBee Alliance also provides certification for both radio and stack chip manufacturers, as well as OEMs using ZigBee inside their products [4].

The radio part of ZigBee is based on the Institute of Electrical and Electronics Engineers (IEEE) 802.15.4 radio standard. Task group 4 of the IEEE 802.15 Working group for Wireless Personal Area Networks (WPAN), states that they were initially "chartered to investigate a low data rate solution with multi-month to multi-year battery

life and very low complexity, operating in an unlicensed, international frequency band. Potential applications are sensors, interactive toys, smart badges, remote controls, and home automation" [5]. Some technical features of the underlying 802.15.4 standard are:

- Data rates of 250, 40 and 20 kbit/s
- 16 channels in the 2.4 GHz Industrial, Scientific and Medical (ISM) band, 10 channels in the 915 MHz band, and one channel in the 868 MHz band
- 16 and 64-bit, 'short' and 'long' addressing modes
- Support for critical latency devices, such as joysticks
- Uses Offset-Quadrature–Phase-Shift Keying (O-QPSK) and Direct-Sequence-Spread-Spectrum (DSSS) technology
- Carrier-Sense-Multiple-Access–Collision-Avoidance (CSMA-CA) channel access
- Automatic network establishment by the coordinator node
- Fully handshaked protocol for transfer reliability
- Power management to ensure low power consumption

ZigBee Network Topology

A ZigBee node can be one of three types: ZigBee Coordinator (ZC), ZigBee Router (ZR), or ZigBee End-Device (ZED). There can only be one coordinator within a ZigBee network. The coordinator is the only node type that may form a network. It is considered to be the main node, and may never sleep. The other node types can join a network already formed by a coordinator, but cannot form a network on their own. The router node type is like the coordinator able to route traffic on to other nodes, but is also disallowed from sleeping. The end-device type however, may sleep, but as a consequence can't route any traffic.

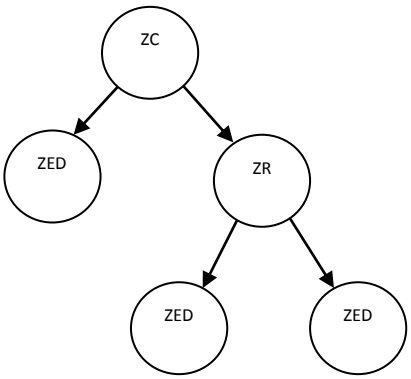


Figure 1: Typical ZigBee Network Setup

Messages may be sent as unicasts, groupcasts or as broadcasts. To unicast a message, simply means to send a message from one node to another, complete with end-to-end acknowledgments. To perform a groupcast (or multicast), means to send a message to all the nodes belonging to a certain group, and to perform a broadcast, means to send a message to all the nodes within the network. Groupcasts and broadcasts are not acknowledged, because that would likely flood the network with excessive amounts of traffic. Furthermore, there is also the possibility of sending broadcasts with certain

radiuses, allowing the broadcasts to reach only a certain number of hops away from the sending node.

Each ZigBee network has a 2-byte Personal-Area-Network–IDentification number (PANID), and should also have a unique 8-byte Extended–Personal-area-network–IDentification number (EPID), used to distinguish exactly the desired network amongst all other possible networks. EPIDs are not managed by any standards body, and may be chosen arbitrarily.

Each node in a ZigBee network has a unique Medium-Access-Control (MAC) address, sometimes also called Extended Unique Identifier (EUI), IEEE, or simply long address. MAC addresses are unique and never change, and are used to address a node before it has joined a network and received a short address. The MAC address consists of a total of eight bytes, of which the first three bytes constitute an Organizational Unique Identifier (OUI), and the remaining five bytes may be chosen arbitrarily by an OEM. OUIs are governed and sold by IEEE.

Upon joining a network, each node receives a 2-byte so called network (NWK) address (or short address). The network address is used when addressing nodes within a WPAN, and is preferred over the longer MAC address in order to conserve precious over-the-air packet space, which may only be up to a maximum of 127 bytes.

As indicated previously, each node may also belong to a group, for which a 2-byte group-ID number is defined. The group ID may be used to filter out all messages intended for the group.

ZigBee Stack Architecture

The internal ZigBee architecture is specified as a layered structure, where functionalities are built on top of each other as illustrated in Figure 2 below. The two bottom layers, called the PHYsical (PHY) layer and the MAC layer, are defined by the 802.15.4 radio specification [5], and the rest are defined by the ZigBee specification [6]. Together, these layers make up the ZigBee stack. Each layer has its own level of functionality, and data and commands can travel between the layers via certain Service Access Points (SAP).

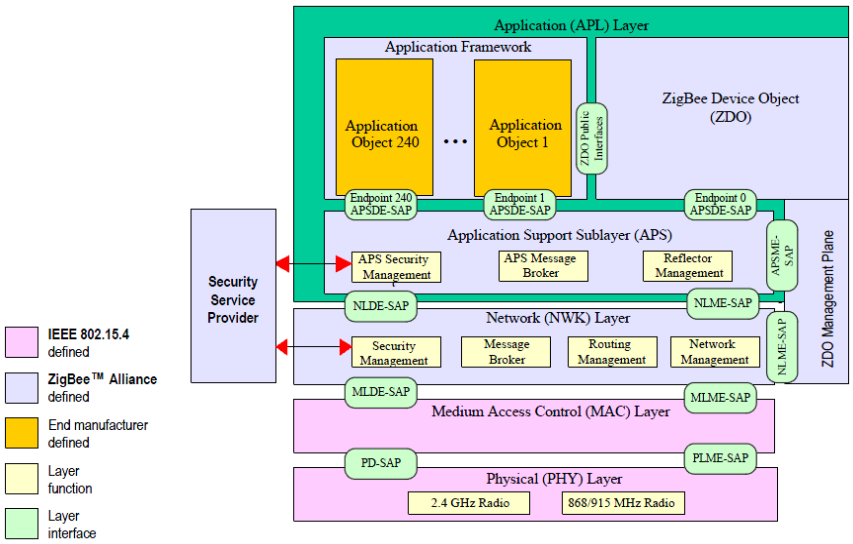


Figure 2: ZigBee Architecture [6]

The PHY layer is responsible for translating data to and from a transceivable format. The MAC layer is, among other things, responsible for discovering networks, and contains some of the functionality required for forming or joining a network. Continuing on to the ZigBee-defined layers, the Network (NWK) layer is responsible for the routing and mesh-routing capabilities, and adds further network, reliability and security functionality on top of the MAC layer. The APplication-Support (APS) layer acts as a filter for data being transmitted to different user-applications, and also keeps a binding table for which nodes, a node wishes to interact with. The ZigBee Device Object (ZDO) can interact with different layers, and is responsible for higher-level network management. The Application Framework contains the different application end-points on which applications may reside, and introduces the so called ZigBee Cluster Library (ZCL) in an attempt to simplify and standardize development and compatibility between different ZigBee product vendors. A Security Service Provider interacting with the ZDO, the NWK and the APS layers is also depicted in Figure 2, and is specified by the ZigBee Alliance.

End-points are identified by a 1-byte number and allow for different applications to reside on the same node. End-points 1 through 240 may be used for applications, while end-point 0 is reserved for ZDO. End-points may use different profiles, indicated by a 2-byte–profile-ID number. A profile describes a set of related functions, for example the 'Home Automation' profile, which defines functions that may be encountered in a ZigBee-automated house. Public profiles, specified by the ZigBee Alliance, which use the ZCL, have a profile ID in the range of 0x0000 to 0x7FFF, while OEMs may define their own private profiles in the range of 0xBF00 to 0xFFFF. A profile defines a set of clusters, which in turn are identified by a 2-byte number. Clusters contain both commands and data (attributes) and have direction. A cluster may for instance describe on/off functionality and be used by devices such as a switch and a light. The switch has the cluster listed as an output, while the light has the same cluster listed as an input. The switch will then be able to find a light through ZigBee's binding mechanism, but not another switch because an output must be matched to an input. The attribute contains the state of the device, in this case, whether the light is on or off. Furthermore, commands may be issued to manipulate or retrieve cluster attributes. The 'On' command may for example change the on/off attribute to a binary one. End-points also

contain a device-ID number used to distinguish devices that work in a similar way using the same clusters, but are meant to appear different to the end-user.

There currently exists three slightly different versions of the ZigBee stack software; ZigBee 2006, ZigBee 2007 and ZigBee Pro. Each stack builds on the features of the previous versions, but on the other hand tends to consume more bandwidth and make a larger memory footprint than its predecessor(s).

Security

ZigBee uses the National Institute of Standards and Technology (NIST) Advanced Encryption Standard (AES) 128-bit encryption scheme for encryption of all traffic. The inherent encryption makes communicating with ZigBee very secure, by default.

AES is a symmetric block-cipher encryption-algorithm adopted by NIST on November 26, year 2001. AES, which is sometimes interchangeably called Rijndael after its conceivers Vincent Rijmen and Joan Daemen, was selected the winner in an open competition for an algorithm to replace NIST's previous standard encryption algorithm, simply called Data Encryption Standard (DES). AES is a substitution-permutation cipher that can process data blocks of 128 bits, using cipher keys with lengths of 128, 192, or 256 bits [7]. AES is considered very secure after thorough evaluations, and is fast in both software and hardware implementations.

128-bit AES Encryption and Message Integrity Checks (MIC) are always enabled in ZigBee. Extra higher-level–security features, including Trust Centers (TC), may be enabled in the ZigBee-Pro stack version, as required. In standard-security mode, all nodes that are allowed to join a network are considered trustworthy. In certain situations however, it may be desirable to have nodes join the network without automatically trusting each other internally. The motivation for using a TC in these cases, is that two nodes may wish to have a private conversation without any of the other nodes eavesdropping. The TC can then be used as a means of establishing a private Link key that none of the other nodes know about.

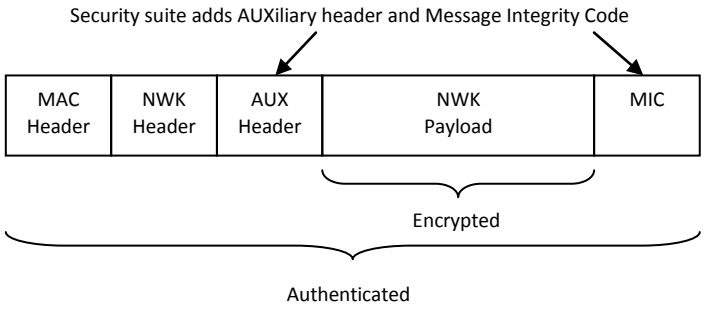


Figure 3: ZigBee Message-Frame Structure [4]

Figure 3 describes how a ZigBee over-the-air packet is structured. The entire packet is authenticated, and the NWK payload is encrypted. Each message is assigned a rolling 8-bit sequence-number, which can be used to keep track of sent messages.

4.2 Description of MiFare

MiFare is a registered trademark of NXP Semiconductors, and represents one of the most common forms of contactless-card technologies used today [8]. It is based on selected parts of ISO14443. ISO14443 was in turn developed by Working Group 8 of Subcommittee 17 in ISO/IEC's Joint Technical Committee 1, and was first published in April, year 2000. It has since undergone a few minor revisions but remained fundamentally intact. The standard is divided into four separate parts, each dealing with separate issues concerning the communication and operation of Proximity–Integrated-Circuit Cards (PICC). The four parts are:

1. Physical Characteristics [9]
2. Radio Frequency Power and Signal Interface [10]
3. Initialization and Anti-Collision [11]
4. Transmission Protocol [12]

Part one and four is of little relevance to this project, since the physical dimensions of the PICCs themselves are beyond influence, and the transmission protocols outlined in part four are supplanted by the proprietary transmission protocols and cryptographic algorithms used by MiFare-compliant cards and interrogators. Part two details the minimum and maximum signal-strength, the bandwidth, and the modulation schemes that must be adhered to, in order to maintain proper operation of cards within the Proximity Coupling Device's (PCD), i.e. card-reader's field. Part three details the low-level–logic functionality of the cards, as well as the algorithm used to select one card by itself, even though several cards are being presented to the reader at the same time.

PICCs require power to operate, even though the levels are small. These passive devices operate without an internal battery, deriving the power to operate from the electromagnetic field generated by the PCD. Consequently, dimensioning of the antenna structure is critical, since it must not only carry the communication with the card, but in addition also needs to supply it with energy. For this reason, passive devices offer, in principle, unlimited operational lifetime, but have short read-ranges and require higher field-strengths from the PCD.

The energy transfer is made possible through electromagnetic induction between the reader antenna and the card coil, both of which constitute loop-antennas, effectively forming an air-cored transformer when the card is in close enough proximity to the reader device. For the energy transfer to occur efficiently, both coils must be suitably shaped and tuned to approximately the same operating frequency. The transformer coupling is used to induce a pulsating current in the card loop-antenna, where it is rectified and used to drive the internal logic, the state machines and the cryptographic engine of the card. Fundamentally, for communication to be possible, two criteria must be fulfilled. First the reader must be able to provide the card with sufficient power to operate, and secondly, it must be able to pick up the card's response. Together, these two criteria combine to determine the achievable operating range.

ISO14443-type cards operate within the globally available, and unlicensed Radio-Frequency (RF) ISM-band, at 13.56 MHz. ISO14443 specifies two fundamental modes of operation; A and B, that use different communication and card-selection procedures. This project focuses on mode A, which is the only form used by MiFare-compliant cards.

When communicating, reader to card, ISO14443A specifies the use of two-level Amplitude-Shift-Keying (ASK) modulation, with a 100% modulation ratio and modified Miller-code for data transfer. The Miller code is modified in such a way as to provide a constant average value of the signal amplitude, regardless of the transmitted data [10]. This alteration is due to the fact that the PICC derives its energy-supply from the field, and thus benefits from as constant a power-transfer level as possible.

Data transfer, card to reader, is carried out differently. In general, the card response is on the order of 60 dB weaker than the signal level generated by the reader itself [13]. In order to detect this comparatively weak signal, a well-designed receiver circuit must be employed. Instead of using direct load-modulation, MiFare uses an 847.5 kHz sub-carrier. The sub-carrier is then modulated by using On-Off Keying (OOK) with Manchester-encoded data. Manchester coding uses the transition edges to encode data, and is self-clocking, which means that the clock-signal can be derived from the data as it is being transmitted. The data-rates for both directions are 105.9 kHz, and the link is half duplex with the PCD always initiating communication.

The anti-collision protocol is based on the principle of a simple state machine (see appendix A). A card placed in the field of the reader, powers up and enters its IDLE state. Typically, the reader sends a REQuest-to-answer–type-A (REQA) command, or a Wake-UP–type-A (WUPA) command, which places any idle card in range into the READY state. The card then returns an Answer-To-reQuest–type-A (ATQA) command so that the reader knows that it has at least one card within range. If multiple cards are present, a binary search algorithm is used to select a unique card [11]. This procedure is described below.

Once one or several ISO14443A-compliant card(s) acknowledge presence, the reader goes on to transmit a SELect (SEL) command, and a Number-of-Valid-Bits (NVB) parameter. The PICCs in proximity then respond with their UID. If a collision occurs, i.e. several PICCs respond differently, this is detected by the PCD, which retransmits the SEL command, followed by the updated NVB plus one, the received parts of the UID, and a Block-Check Character (BCC). The PICCs then compare the UID against their own, and the PICC with the matching UID responds with the remainder of its UID. This process is repeated until no more collisions occur and one full UID has been received. The matching PICC may then be selected, whereby, in addition to previous steps, two Cyclic-Redundancy–Check (CRC) bytes are appended, following the BCC. After these events have occurred, the PICC responds with a Select AcKnowledge (SAK) command and moves on to its ACTIVE state [11]. The reader and card will then negotiate the protocol parameters used for data communication. It is at this point that card-type and MiFare-specific procedures supplant those detailed in ISO14443, part four. The state machines for both the PCD and the PICC are given in appendix A.

Security

A wide variety of different MiFare cards exists. MiFare Ultralight-type cards do not support encryption at all, but are very cheap and derive security from one-time programming areas on their memory. MiFare Classic is ubiquitous and is used in Sweden by for instance Västtrafik public transportation services; however, the proprietary cryptographic Crypto-1 algorithm, used in the Classic-type cards, has been shown to be vulnerable to several different types of attacks [14] [15] [16]. Security concerns have even been admitted by NXP themselves [17]. NXP does, however, offer

several higher-security card models at the present time, with the next-generation MiFare Plus cards meant to be the first-hand replacement for the aging line of Classic cards [18]. MiFare Plus is backwards compatible with Classic cards, but in addition also offers the possibility of using AES 128-bit encryption at the application layer.

MiFare Antenna Theory

The use of an inductive loop-antenna offers a high degree of miniaturization, since the loop inductance can, with the aid of an external capacitor, be forced into oscillating at the operating frequency, as long as its inherent self-resonance frequency is higher than the desired frequency. Thus, the antenna can be made small, even though the wavelength at which it operates is orders of magnitude larger than the circumference of the antenna itself. This ability comes at the cost of poor radiation efficiency however [19].

In order to derive the magnetic field generated by a loop-antenna, Biot-Savarts law can be used. It states the magnetic flux density resulting from a steady current flow, through for instance a conductor:

$$\boldsymbol{B} = \int \frac{\mu_0 \mu_r I \, d\boldsymbol{l} \times \hat{\boldsymbol{r}} \cdot r}{4\pi r^3} \qquad (1)$$

Here $\boldsymbol{B}$ is the magnetic flux density, $\mu_0\mu_r$ is the permeability, I is the current, $d\boldsymbol{l}$ is the infinitesimal length element, and $\hat{\boldsymbol{r}} \cdot r$ is the distance vector. If the magnetic flux density is instead expressed as magnetic field intensity $\boldsymbol{H}$, the analysis can be continued without regard to the physical properties of the space surrounding the antenna [20].

$$\boldsymbol{B} = \mu_0 \mu_r \boldsymbol{H} \qquad (2)$$

Thus, combining Equation (1) and Equation (2) results in Equation (3):

$$\boldsymbol{H} = \int \frac{I d\boldsymbol{l} \times \hat{\boldsymbol{r}}}{4\pi r^2} \qquad (3)$$

If the disparity in size between wavelength and antenna radius is sufficiently large ($\frac{\lambda}{10} \gg r$), and the number of loop turns is low, the current distribution may be assumed to be constant across the antenna [21]. This assumption simplifies the analysis, since it allows phase differences along the antenna conductors to be disregarded.

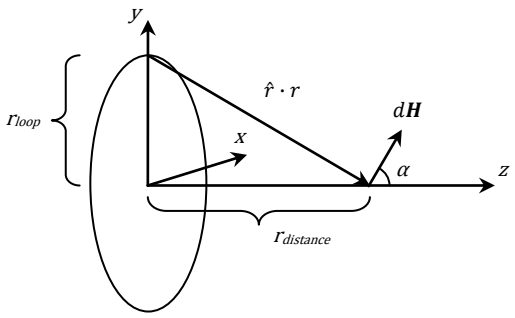


Figure 4: Antenna Geometry

Thus, solving Equation (3) for a small circular loop-antenna with N turns, along the center z-axis, as illustrated in Figure 4, results in Equation (4):

$$|\boldsymbol{H}_z| = \left| \int_0^{2\pi r_{loop}} \frac{NI_{loop} \cos\alpha}{4\pi {r_{distance}}^2} dl \right| = \left| \frac{NI_{loop} {r_{loop}}^2}{2\sqrt{\left({r_{loop}}^2 + {r_{distance}}^2\right)^3}} \right| \quad (4)$$

This expression could be further simplified by the elimination of the r_{loop} term in the denominator in such cases where $r_{distance}$ is much greater than the radius of the antenna. This is, however, not expected to be the case here. For more complex antenna shapes, the field intensity can be derived from superposition of suitable wire fragments representing the antenna structure [20].

Using Equation (4), the following plot of how the magnetic field intensity diminishes with increasing distance can be generated. In order to comply with ISO14443-2, the field intensity at the PCD must be within the range of 1.5–7.5 A/m [10]. Note how the field at the center of a smaller loop is higher than that of a larger loop, and that the field remains roughly constant up to $r_{loop} = r_{distance}$.

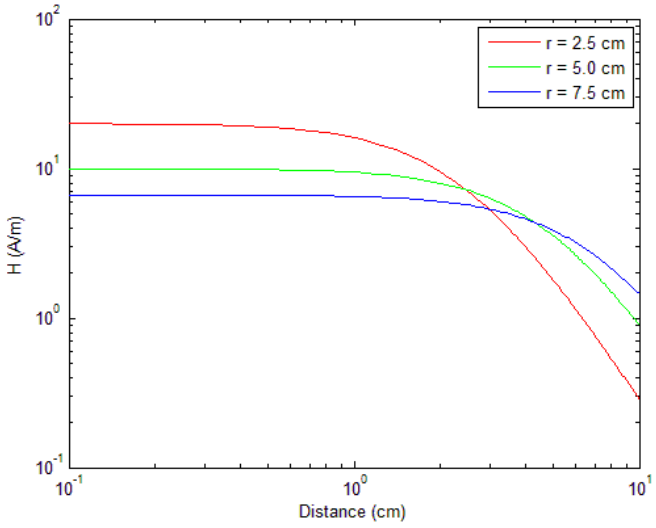


Figure 5: Magnetic Field Intensity versus Distance

Again plotting Equation (4), though this time with constant distance and varying radius, shows that it is possible to select the radius of the antenna in order to optimize the field intensity at a given distance. It is interesting to note that a too large antenna may lack the necessary minimum field level to energize a card, unless the current cannot be increased correspondingly. In these plots the current has been set to 0.5 A, and the number of loop turns to two.

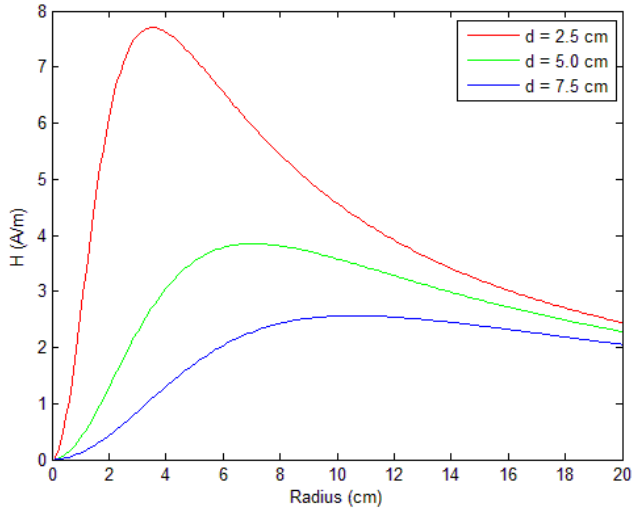


Figure 6: Magnetic Field Intensity at a Fixed Distance versus Antenna Radius

The disparity between the antenna loop-length and the wavelength at the operating frequency, also allows the antenna structure to be treated as a lumped-circuit equivalent with R, L and C components. Thus, determining the inductance of a loop-antenna can be done by applying the definition of inductance:

$$L = \frac{N\phi}{i} \tag{5}$$

Here L is the inductance, N is the number of wire turns, ϕ is the magnetic flux through the area enclosed by the loop, and i is the current through the wire.

$$\phi = \iint_s \boldsymbol{B} \cdot d\boldsymbol{s} \tag{6}$$

The magnetic flux, ϕ, of Equation (5), can in turn be calculated by integrating $\boldsymbol{B}$ over the area enclosed by the antenna, as described in Equation (6). Thus, it can be easily seen that Equation (5) and Equation (6) can be combined and used to provide an actual figure for the calculated inductance.

Several empirical formulas for determining the loop inductance from the antenna shape and conductor diameter are also available [22] [23] [24]. These are, however, not very precise, and because the inductance of the coil will be affected by its surrounding environment, it is recommended to measure the actual antenna under controlled circumstances. The measurement setup should preferably reflect the characteristics of the future operating environment of the system.

The radiation resistance for a magnetic loop can be determined from Equation (7), where A is the area enclosed by the loop, and λ is the wavelength [19]:

$$R_{rad} = \frac{320\pi^4 A^2}{\lambda^4} \tag{7}$$

For a small loop-antenna, the radiation efficiency is low, and subsequently, the radiation resistance is very low, typically on the order of only a few mΩ. Thus, antenna resistance is primarily made up of parasitic resistance from the wire making up the antenna coil, and can be calculated from the relation given in Equation (8), where ρ is the resistivity of the conductor, l_{ant} is its length, and A its cross-sectional area.

$$R_{loss} = \frac{\rho l_{ant}}{A} \tag{8}$$

Other factors contributing to antenna conductor-resistance are: the proximity effect, which causes an increase in resistance from close-coupling between parallel antenna traces, and the skin-effect, stemming from the cancellation of the field inside the conductor itself, forcing current to flow only at its surface. Both these effects have significant influence on the resistance seen at the operating frequency, causing it to be higher than the calculated or measured DC resistance [21].

Furthermore, parasitic capacitances resulting from conductor-conductor overlap can be estimated by using empirical formulas such as those given by Aerts et al. [21]. Parasitic capacitance is not necessarily undesirable, as long as the desired resonance frequency is lower than the natural resonance frequency of the loop. In such cases, it can be regarded as making a contribution to the capacitance otherwise needed to force the antenna to resonate at a lower frequency.

The Quality-factor (Q) describes how (under-)damped a resonating system is, and for the antenna-system analyzed here, it can be stated as follows:

$$Q = \frac{\text{Energy stored}}{\text{Energy dissipated per cycle}} = \frac{2\pi f_c L}{R_{ant}} = \frac{f_{resonance}}{f_{bandwidth}} \tag{9}$$

The Q-factor of a MiFare interrogator is critical, since in order to keep the driving power low, a high Q increases the oscillating reactive current in the antenna, thereby increasing the range over which a transponder can be efficiently energized [25]. However, since a high Q also limits the effective bandwidth of the antenna, there is a limit that must not be exceeded in order to assure proper operation. More specifically, the requirement stems from the anti-collision mechanism, which is based on the timing accuracy of the transponder responses [10].

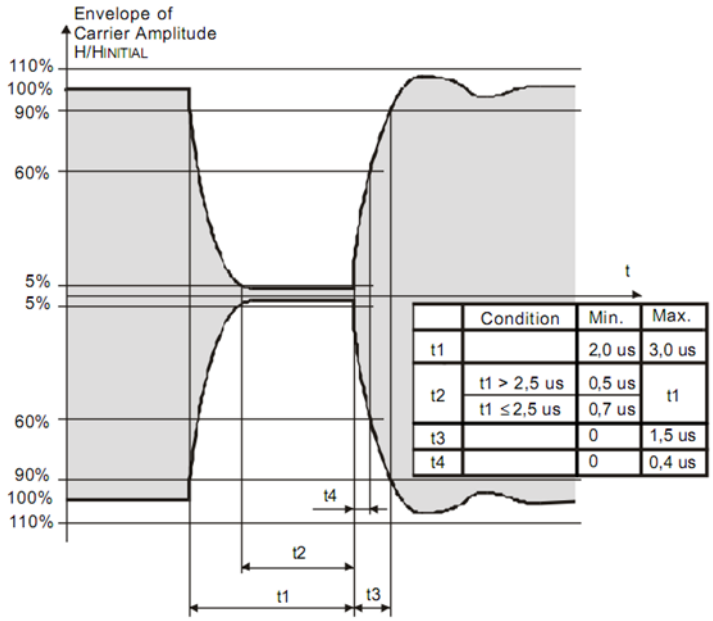


Figure 7: Carrier Envelope at 105.9 kbit/s Data Rate [10]

Figure 7 shows the amplitude pulse-parameter specifications at a data rate of 105.9 Kb/s, as laid out in part two of ISO14443 [10]. According to Gebhart et al. [25], for an idealized case, it is possible to determine the acceptable upper limit on the Q-factor from the modulation parameters. The limit can be found by describing the falling and rising edges of the envelope with an exponential function with a time-constant determined from the system bandwidth:

$$u_F(t) = e^{-t/\tau} = e^{-t\frac{2\pi f_c}{Q}} \tag{10}$$

$$u_R(t) = 1 - e^{-t\frac{2\pi f_c}{Q}} \tag{11}$$

Taking the logarithm of both sides of Equation (10) and rewriting the result, produces Equation (12):

$$\tau = \frac{Q}{2\pi f_c} \qquad Q = 2\pi f_c \tau \tag{12}$$

Using the pulse-parameter specifications of Figure 7, and Equation (10) and Equation (11), Equation (13), Equation (14) and Equation (15) can be derived:

$$(t_1 - t_2) = \frac{Q}{2\pi f_c} \cdot [\ln(0.90) - \ln(0.05)] \tag{13}$$

$$(t_3) = \frac{Q}{2\pi f_c} \cdot [\ln(0.95) - \ln(0.10)] \tag{14}$$

$$(t_4) = \frac{Q}{2\pi f_c} \cdot [\ln(0.95) - \ln(0.40)] \qquad (15)$$

These expressions set the acceptable upper limit on the Q-value. The value t_4 poses the most stringent requirement on the steepness of the edge, and thus:

$$Q \leq \frac{2\pi f_c}{\ln(0.95) - \ln(0.4)} \cong 39.4 \qquad (16)$$

The Q-value must not be allowed to be greater than this, and some safety margin for component tolerances and component aging, as well as influence of nearby objects, ought to be factored in. NXP recommends the Q-value of an antenna not to exceed 30–35 [13], which is valid for a communication speed of 105.9 kbit/s. If this speed were to be increased, the antenna Q-value would have to be decreased.

Although the loop will have an internal resistance stemming from the conductor itself, an external resistor often also has to be added to adjust Q down to the acceptable range determined previously. The resistor may be placed either in series or in parallel with the antenna [24].

In practice, an external capacitor must be added to the loop in order to obtain a tuned Inductor-Capacitor (LC) circuit and make it resonate at the desired frequency. Several different arrangements can be made, but the four depicted in Figure 8 are the most fundamental ones [21]:

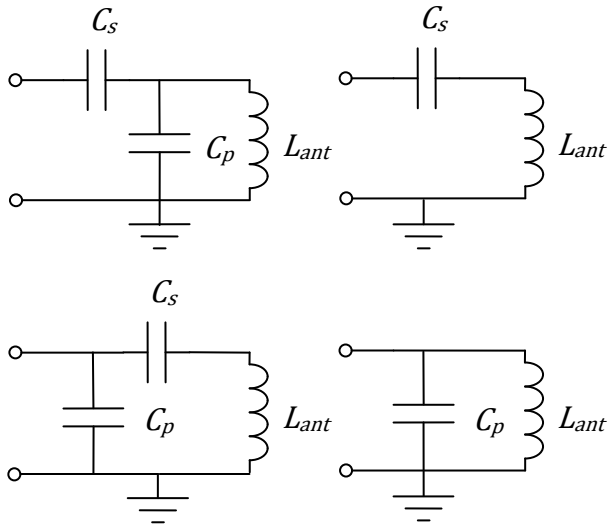


Figure 8: Antenna LC-Resonance Arrangements

The suitability of the different combinations is dependent on the characteristics of the source available to drive the antenna. In case the source can deliver an unlimited amount of current, the bottom right setup is ideal. Conversely, if the source provides the antenna with a high driving voltage, the upper right setup is the most suitable. The two remaining alternatives reduce the need for high voltage or current, and thus, they constitute the preferred choices. In cases where the feed-line is long enough that transmission-line effects must be dealt with, the upper left alternative provides more degrees of freedom in transforming the impedance that the antenna presents [21].

As implied by the low radiation resistance, energy is not transmitted to the PICC by radiation primarily, but rather transferred through magnetic coupling. In essence, the PICC and the PCD make up a loosely-coupled air-cored transformer, with both sides tuned to resonate at approximately the same frequency [26].

The voltage induced in two coupled coils can be determined from Equation (17), Equation (18), and Equation (19) below, where the mutual inductance, M, describes how the change in current in one inductor induces a voltage in the other. The coupling factor, k, always range between zero and one, where zero corresponds to no coupling at all, and one corresponds to total coupling, whereby all the flux lines generated by the first coil are enclosed by the second. The coupling factor is determined by the physical properties of the antennas, as well as their distance and positioning with regards to each other [26].

$$\mathrm{v}_1(t) = L_1 \frac{di_1}{dt} + M \frac{di_2}{dt} \tag{17}$$

$$\mathrm{v}_2(t) = L_2 \frac{di_2}{dt} + M \frac{di_1}{dt} \tag{18}$$

$$M = k\sqrt{L_1 L_2} \tag{19}$$

To facilitate analysis, Equation (17) and Equation (18) can be rewritten into Equation (20) and Equation (21):

$$\mathrm{v}_1(t) = (L_1 - M) \frac{di_1}{dt} + M \frac{d(i_1 + i_2)}{dt} \tag{20}$$

$$\mathrm{v}_2(t) = (L_2 - M) \frac{di_2}{dt} + M \frac{d(i_1 + i_2)}{dt} \tag{21}$$

The above equations describe a transformation that is illustrated in Figure 9 below. Please note that this transformation can give rise to negative inductance values for certain parameter combinations.

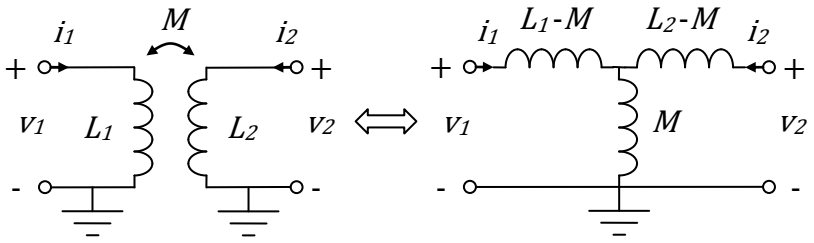


Figure 9: Coupled-Inductors Transformation

Using this transformation to describe the inductive coupling between the PCD and the PICC, the circuit model depicted in Figure 10 can be drawn:

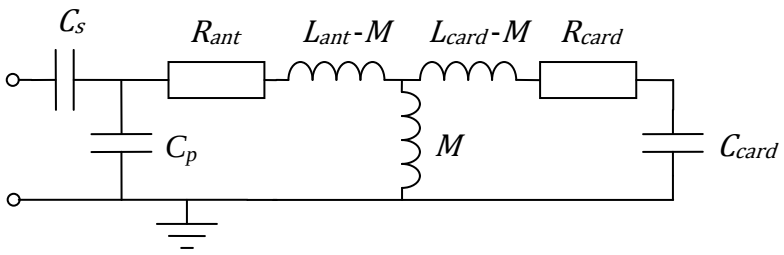


Figure 10: Coupled PCD-PICC Antenna Model

Based on this model, the total transfer-function can be derived, and is shown in Equation (22) below. R_{ant} and R_{card} are added to give the system realistic Q-values.

$$\frac{v_o}{v_i} = \frac{C_p \| \left(R_{ant} + L_{ant} + M \| (L_{card} + R_{card} + C_{card})\right)}{C_s + C_p \| \left(R_{ant} + L_{ant} + M \| (L_{card} + R_{card} + C_{card})\right)} \quad (22)$$

The amount of mutual inductance between the two antenna coils, determine the shape of the transfer function. For a tuned system with no coupling at all, the transfer function shows only a single amplitude peak at the resonance frequency. As the mutual inductance is increased beyond a certain level however, the system appears to have two separate resonance frequencies; one corresponding to the PCD, and one to the PICC. This effect is illustrated by plotting the transfer-function given in Equation (22) for a few different values of the coupling coefficient. The result can be seen in Figure 11 below:

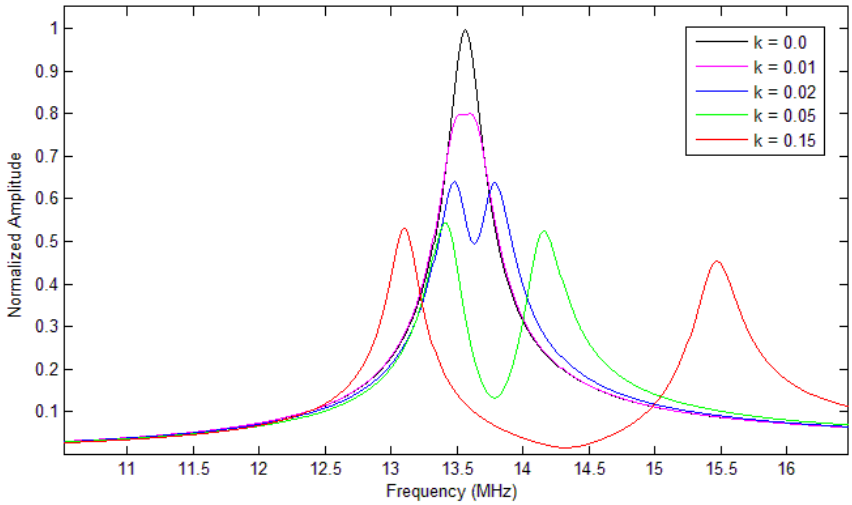


Figure 11: Coupling-Factor Influence on PCD-PICC Tuning

Thankfully, PICCs have built-in overvoltage protection in the form of Zener clamping diodes across their power supplies. Thus, when placed in close proximity of the PCD, the diodes conduct and reduce the PICC's Q-value. In most cases this is sufficient to alleviate the impact of the coupling factor's harmful effect, and maintain the system in a functional state [26].

4.3 Description of I^2C and SPI Buses

The Inter-Integrated Circuit (I^2C) bus is a very common half-duplex 2-wire serial bus interface, originally introduced by Philips Semiconductors for in-house use for internal IC control and communication [27]. The I^2C bus in its original form is capable of bi-directional data transfers with a speed of up to 100 kbit/s in standard mode, and 400 kbit/s in fast mode. A 3.4 Mbit/s high-speed mode has since been added, and further variations with different speed and timing constraints exist. It is also not uncommon for somewhat compatible two-wire interfaces to, rather anonymously, go by the moniker of TWI, which is short for Two-Wire Interface.

Communication on an I^2C bus is based on a master/slave relationship between the different devices, with the master always initiating communication. There is, however, also the possibility of having multiple masters on one bus through arbitration. Data traffic over I^2C is by necessity half duplex, since data can travel in only one direction at a time. Originally, I^2C offered only a 7-bit address field, putting a theoretical limit on the number of devices on a bus to no more than 128, though a few reserved addresses further reduced this number. Since then, the possibility of using 10-bit address fields has been added, making it possible for over a thousand devices to share the same I^2C bus. The word-length for I^2C is fixed at eight bits, with the addition of an acknowledge-bit from the receiver at the end of each byte.

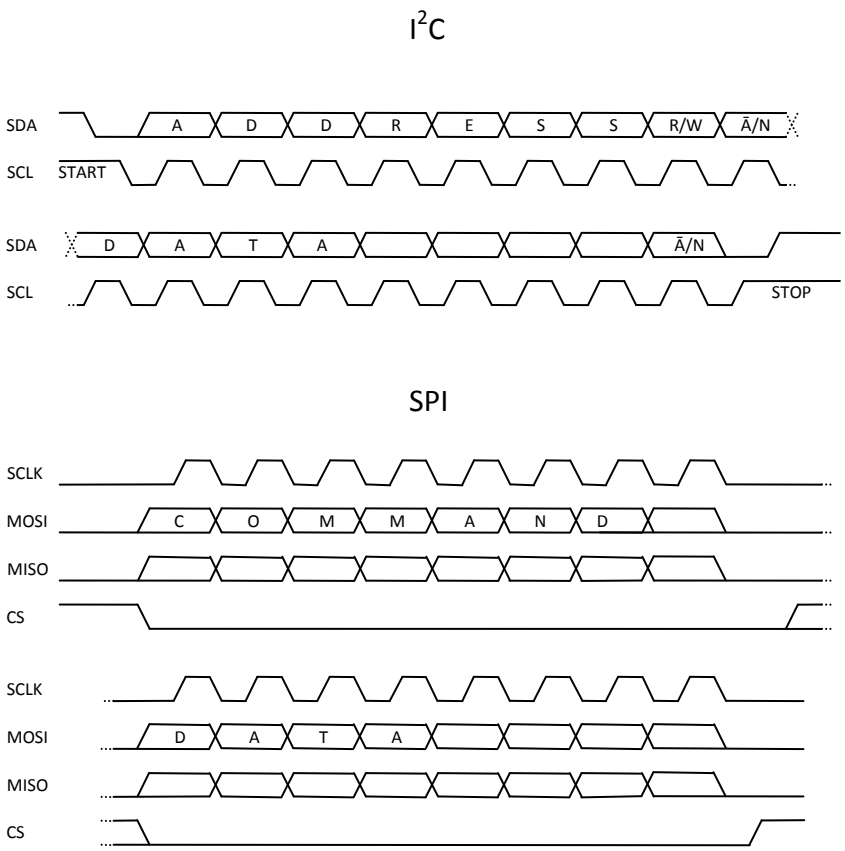


Figure 12: Simplified I²C and SPI Signal Diagrams

At the physical layer, an I²C bus consists of a clock-line (SCL) and a data-line (SDA). Both lines are open-drain, requiring external pull-up resistors. Having the data-line pulled to ground while the clock-line goes high, is interpreted as a logical zero, while leaving the line drawn high by the pull-ups while the clock line goes high, is interpreted as a logical one. The open-drain signaling technique is also used as a clear-channel–access method, to ascertain whether the medium is available or busy, by simply checking that none of the pins have been pulled low before attempting to use the bus. Transitions on the data line must be performed while the clock is low, since transitions while it is high indicate start and stop conditions. See the upper part of Figure 12 for an illustration of a simplified I²C transmission sequence. Systems requiring high speed may make the addition of current source pull-ups to enable faster rise times, or support a higher bus capacitance for a given speed.

An interesting feature of the I²C bus protocol is clock-stretching. An addressed slave-device may hold the clock-line low after receiving or sending a byte, indicating that it is not yet able to process more data. The master that is communicating with the slave, wants to raise the clock to transfer the next bit, but must verify that the clock-line was actually raised. If the slave is clock-stretching, the clock line will still be low because the connections are open-drain. The same is true if a second, slower master, wants to raise the clock at the same time.

The Serial Peripheral Interface (SPI) bus is most commonly recognized as a 3-wire synchronous full-duplex serial bus introduced by Motorola. SPI allows for faster interfacing compared to I^2C, and many variations exist.

The three signal lines consist of two data-lines named Master-Output-to-Slave-Input (MOSI), Master-Input-to-Slave-Output (MISO), and a 'Serial CLocK' (SCLK) clock-line. In a typical hardware setup, the data-lines form an inter-device serial shift register or a circular buffer. Thus, for every bit of data sent, one bit is received in return. See the lower part of Figure 12 for a simplified illustration of what an SPI transmission sequence might look like. In most cases, transmissions consist of 8-bit words, and a master could at leisure initiate a string of such transmissions until the desired length of data has been transmitted. However, other word-sizes are also possible, such as 16 or even 12-bit.

If more than one slave exists on the same bus, separate Slave-Select (SS) signals are required to enable or deselect each device present on the bus, rendering the SPI bus somewhat impractical if many devices are to be connected together. Furthermore, for this to be possible, every slave on the bus that has not been activated using its SS line must disregard the input clock and any signal seen on its MOSI input, and must be certain not to drive its MISO output. The master must thus make certain to select only one slave at a time, in order to avoid slaves interfering with each other.

5. Design Considerations

In this chapter, system-level requirements and considerations are analyzed further, in light of the capabilities of the employed technologies.

5.1 System-Level Requirements

Before venturing further and constructing any actual demonstration prototype, it is advisable to try and estimate, on a system-level, what the limitations and requirements of a working implementation of the product will be.

Battery Life

A rule of thumb, as specified by Decta, is that an entry system should be expected to be able to process at most one-hundred door accesses per day. In addition to this, normal maintenance operations such as adding and removing cards, updating time schedules and retrieving log entries should be accounted for. Since the power consumed by the door-locking mechanism and controller-card is beyond control, only the power consumed by the front-end module will be considered.

Looking into the specified performance of a standard Alkaline 1.5 V Duracell MX1500 AA cell [28], a complex relationship between load, self discharge current, operating temperature and voltage level at the battery terminal emerges. In addition, the shape of the discharge current waveform plays a role in determining the achievable battery life. For instance Castillo et al. [29] reports beneficial effects on alkaline battery life from using pulsating loads with prolonged intermittent rest periods, in comparison to using a constant load with the same amount of total energy draw.

Since the system is expected to primarily reside indoors, battery analysis is greatly simplified as the system can be assumed to always operate at room temperature. The shape of the load current waveform is not as easily predicted, but with repeated polling for proximity cards, the waveform is likely to consist of spikes measured in milliseconds, with rest periods measured in seconds.

In order to be able to draw roughly 2.5 Ah from one Duracell MX1500 cell, the battery must be operated down to about 1.1 V. What this implies is that, on average, the system, using two cells connected in series, must not draw more than:

$$I_{avg} = \frac{1}{T}\int_{t=0}^{T} \mathrm{I}(t)\,dt = \frac{1}{2\ \mathrm{year}\cdot 365\ \frac{\mathrm{day}}{\mathrm{year}}\cdot 24\ \frac{\mathrm{h}}{\mathrm{day}}}\cdot 2.5\ \mathrm{Ah} = 142\ \mu\mathrm{A} \qquad (23)$$

Thus, the system must adhere to this average current limit while remaining in operation down to a supply-voltage of 2.2 V. If this cutoff voltage can be lowered, additional current draw and operating time can be realized, since the cells can deliver approximately 20% more energy if they can be operated down to 0.9 V per cell.

Texas Instruments has a new line of DC-DC boost converters which looks very promising for this kind of battery-powered application [30]. For as long as the battery voltage remains higher than a preset threshold, the converter is turned off, drawing only a fraction of a micro-Ampere in leakage current with the load directly connected, via a bypass switch, to the batteries. Once the threshold is reached, the switch is opened and the converter is turned on, allowing for the equipment to keep on working at a more optimal voltage-level, as well as letting the battery be depleted to a deeper level of discharge. It should be noted however, that efficiency degrades as the disparity in voltage increases.

Thus, nevertheless, of overall primary concern is minimizing the average current draw by having the system spend as much time as possible sleeping, and by keeping any current-consuming activities as short as possible.

Storage Capabilities

A standard MiFare Ultralight card has a UID consisting of a 7-byte string [31]. This is the longest UID used by any MiFare card, although the underlying standard allows for ten bytes [11]. Thus, in order to identify one unique card among others, this string would have to be stored, retrieved and identified. Usable log information would at least need to store the card number together with the corresponding time and date of entry. Disregarding Binary-Coded–Decimal (BCD) form, simply storing all numbers in separate bytes, time and date would require:

$$\text{year} + \text{month} + \text{day} + \text{hour} + \text{minute} = 5 \text{ bytes} \qquad (24)$$

Thus, a complete log entry would at minimum need to consist of twelve bytes. Furthermore, time schedules have to be added to define when entry is permitted. At minimum, a time schedule must consist of a start and a stop time, defining a period where entry is granted. Thus, at minimum, that would require ten bytes of space, but since it is likely that multiple time intervals will be required, such as for instance when allowing access during intermittent break hours, but denying access at other times, around 60 bytes would seem sufficient to construct reasonably arbitrary intervals.

Thus, to be able to handle 2000 UIDs, 500 log entries and 50 different time schedules, the memory would at minimum need to store on the order of:

$$2000 \cdot 7 + 500 \cdot 12 + 50 \cdot 60 = 23{,}000 \text{ bytes} \qquad (25)$$

It also seems likely that some overhead for data formatting and unanticipated needs ought to be added to this number.

Another consideration affecting memory-size requirements is the possible implementation of the feature of being able to remotely update the firmware on an issued ZED, as the EM250 comes with a pre-implemented stand-alone boot loader. For boot-loading capability with the EM250, if the ZED is more than one hop away from the ZC, sufficient memory must be available to store the entire boot-file locally, before the ZED can be re-programmed.

5.2 Hardware Considerations

The necessary top-level functional blocks for an EM250-based entry-system can be illustrated as in Figure 13 below:

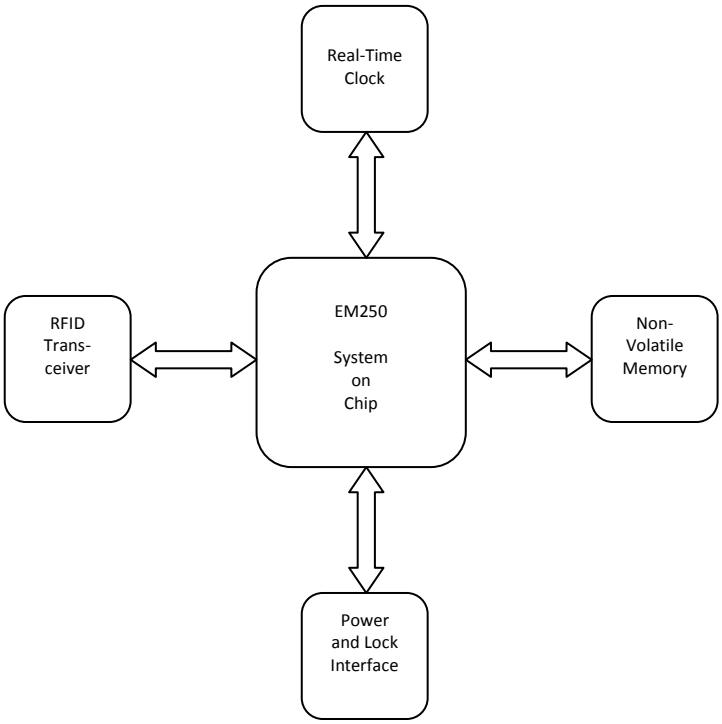


Figure 13: Conceptual Block Diagram

In order to conserve General Purpose Input and Output (GPIO) pins, it is convenient to interface any peripherals through a serial bus such as I^2C or SPI, allowing several devices to share the same pins.

5.3 Firmware Considerations

The functionality of a real-time clock can to the most extent be simulated in software on an embedded system. Running the clock in software would ease the physical hardware design of the system somewhat, and would also lower the total system cost by eliminating some of the hardware. However, since the long term accuracy of such an arrangement can be hard to guarantee, it was decided that a real-time clock should be integrated onboard instead. Since the space needed for the clock would not affect the manufacturing cost of the board (The minimum board-space is already determined by the MiFare antenna), the possibility of leaving the clock un-mounted and implementing its functionality entirely in software still remains in the future.

The EM250 offers the opportunity to utilize some of its 128 KiB Flash as a simulated EEPROM. This memory, however, although employing the use of wear-levelling algorithms (avoiding writing too many times to any one place on memory), is still severely restricted in both capacity (8 KiB) and guaranteed number of actual write-cycles (on the order of 1,000 cycles). In use-cases where only a few cards are expected to be needed, and obsolete card-numbers can be readily discarded from memory, it

might present a valid option, but for most conceivable cases, an external EEPROM or Flash memory remains the better option. Also, if over-the-air–updating capabilities are required, an external memory of no less than 1 Mibit must be available, i.e. it must be of at least the same size as the internal EM250 Flash memory.

6. Demonstration-Board

In this chapter, the proof-of-concept demonstration-board hardware is developed, its parts selected, and its capabilities summarized.

6.1 Overview

During initial prototyping, a pre-manufactured ZigBee module from Telegesis was used. This module, named ETRX2, consists of an EM250 SOC with RF and decoupling components integrated [32]. Employment of this module eased the requirements on the prototype board considerably, and a simple one-layer board could be used. For convenient interconnection of the different surface-mounted devices, off-the-shelf adapter boards were used. Full documentation for the demonstration-board is available in appendix B.

6.2 Parts Selection

The ZigBee transceiver and the MiFare interrogator were selected beforehand by Decta. The rest of the parts were determined in the scope of this thesis project, and comparisons between different parts that were made to aid selection, are attached in the demonstration-board–documentation section in Appendix B.

ZigBee Transceiver

The EM250 ZigBee transceiver from Ember corporation is a complete SOC [2]. Its main features are summarized below:

- 12 MHz 16-bit XAP2b Harvard-Architecture Micro-Controller Unit (MCU)
- 5 kB of SRAM and 128 kB of Flash memory
- Integrated 802.15.4-compliant transceiver module
- Two serial-controllers with Direct Memory Access (DMA), capable of relaying Universal Asynchronous Register Transfer (UART), I^2C, or SPI communication
- Two 16-bit general-purpose timers
- Watchdog timer, and power-on-reset circuitry
- Receiving sensitivity of -98 dBm in boost mode
- Transmit power of +5 dBm in boost mode
- 12-bit Integrated Analog-to-Digital–Converter (ADC) module
- Hardware AES 128-bit encryption accelerator
- 17 GPIOs, of which four may be used as interrupts capable of waking the system from sleep
- Internal Resistor-Capacitor (RC) oscillator for low-power operation
- Two sleep modes: ‘Processor idle’, and ‘deep sleep’
- Non-intrusive ‘Serial InterFace’ (SIF) debug-interface

The current consumption for the EM250, operating on a supply-voltage of 3.0 V, typically lies below 36.0 mA with the radio busy with either transmission or reception. When only running the micro-controller, the EM250 current consumption is typically lower than 8 mA. In deep-sleep mode using the internal RC-oscillator for timed wake-

up however, it only consumes around 1.0 µA, making it suitable for extended operation in battery-operated applications.

The radio transmitter uses differential outputs specified to work with a 200 Ω impedance. The setup will require the use of a 'BALanced to UN-balanced' signal-line converter (BALUN), acting as an impedance-matching transformer, in order to interface with a 50 Ω Microstrip-line and a ceramic chip-antenna of the same impedance. Disregarding antenna gain and external losses from the BALUN, feed-line, and EMC filters, the EM250 in itself, provides a link-budget reaching 100 dB without resorting to boost-mode. The receiver Low-Noise Amplifier (LNA) shares one of the Transmit (TX) pins, and does not require any additional external components. Additionally, the EM250 contains a band-pass filter and an Intermediate-Frequency (IF) amplifier stage, converting the received signal to baseband, where an ADC digitizes the signal.

The EM250 is able to operate in a supply-voltage range of 3.6 V down to 2.1 V, and comes packaged in the QFN48 casing.

MiFare Interrogator

The MFRC523 is one of the latest offerings from NXP semiconductors' line of MiFare-compliant PCD transceivers [1]. Some of its features are summarized below:

- Support for ISO/IEC 14443A and B
- Operating distance up to 50 mm, depending on antenna size and tuning
- Selectable Interface (SPI, I^2C, or serial UART)
- 64-byte send-and-receive FIFO
- Configurable interrupt modes
- Programmable Timer
- Hardware CRC Co-Processor
- Hardware MiFare Crypto-1 encryption engine

In order to determine whether a card is present, a PCD must conduct a poll consisting of a minimum 5 ms energization of the field, in order to transfer enough energy to any present PICC to enable it to wake up. After this has happened, the issuing of a REQA command, asking the card to make its presence known, and then finally a 1 ms waiting-period for any PICC to respond, must follow [11]. In total, a minimum of roughly 6 ms is needed to ascertain whether a card is present or not. Furthermore, the interval between polls need to be sufficiently short for any human user not to experience the system as sluggish. Too short a polling interval, however, increases the power consumption unnecessarily, since the generation of the field to energize cards is one of the main power drains of the entire system. Looking into the datasheet of the MFRC523, it states that the worst-case continuous operating current is 160 mA. If the system were to poll for a card for 6 ms every second, it would result in a much too large average current:

$$160\ \text{mA} \cdot 6\ \text{ms} \cdot \frac{1}{1\ \text{s}} = 960\ \mu\text{A} \qquad (26)$$

A possible solution for this is to increase the battery storage capability by replacing the AA/LR6 cells with C/LR14 or D/LR20-type batteries, since these provide significantly more energy, and would allow for the required two-year operating time to be realized. Another alternative would be to add other, more battery-efficient, means of activating the system once a user is in the vicinity of the reader.

The antenna drive circuit, with adjustable drive-strength, is differential to maximize the use of the limited supply-voltage. The output waveform consists of out-of-phase square waves, switched between V_{DD} and Ground, with a fundamental frequency of 13.56 MHz. This arrangement generates several harmonics, and NXP recommends the addition of an LC EMC-filter directly at the driver outputs, to cope with spurious radiation and harmonics requirement limits [13].

Finally, the MFRC523 is guaranteed to operate on a supply-voltage in the range of 3.3 V down to 2.5 V. The chip comes in a HVQFN32 casing.

Peripheral Device Selection

Initial part selection was made by looking through product offerings from some of the more common chip manufacturers, such as Texas Instruments, Microchip, STMicroelectronics and NXP semiconductors, in conjunction with checking availability from some of the larger vendors conveniently available in Sweden, such as DigiKey, Mouser, Farnell, Elfa etc. Prices as well as relevant performance data were compiled into lists to ease comparison. The parts were evaluated on the following criteria:

1. Worst-case current consumption during sleep
2. Physical space requirements
3. Cost
4. Availability

EEPROM

Electrically Erasable, Programmable, Read-Only-Memory–type (EEPROM) memories are commonly used in embedded systems for long-term data storage of limited amounts of data. They can have data retention times on the order of hundreds of years, tolerate millions of erase/write cycles, and are, in most cases, able to write and rewrite bytes individually.

The 24AA256 32 kiB EEPROM from Microchip was selected as the main non-volatile memory of the device, since it offers the best fit to the applied cost function [33]. On a further note, Microchip offers functionally identical memories, differing only in memory capacity, ranging from 1 kiB to 128 kiB in the same form factor. This circumstance adds the flexibility of making changes to the memory capacity even after the final board has been manufactured. The main features of the 24AA256 chip are:

- Maximum standby current of 1.0 µA at 5.5 V
- I^2C-compatible serial bus interface, running at a maximum of 400 kHz
- 1,000,000 guaranteed erase/write cycles
- Schmitt-trigger inputs for noise immunity
- Data retention time greater than 200 years

The memory also shows acceptable current consumption during operation: 3 mA maximum write current, and 0.4 mA maximum read current at 5.5 V. Even though current specifications taken from the datasheet were measured at 5.5 V, the chip is capable of operating down to 2.5 V if the I^2C bus speed is 400 kHz. This voltage limit is sufficient for the application at hand, because other parts are expected to set a higher limit. If the speed is instead lowered down to 100 kHz, the chip may operate all the way down to 1.8 V.

The chip contains an internal buffer capable of temporarily holding up to 64 bytes of data waiting to be written using a self-timed erase/write-cycle mechanism, thereby relaxing timing concerns associated with sending data over an I^2C bus. Bytes may be written individually, or an entire page of 64 bytes may be written all at once. Regarding read operations, the 24AA256 supports both random and sequential read operations. Random read allows for reading any memory location on-chip, while sequential read may be used to collect bulk data. It is even possible to retrieve the entire contents on memory during one sequential-read operation. The µSOP variant of the chip was used for the demonstration-board.

Real-Time Clock

In essence, a real-time clock is a device of moderate complexity meant to keep track of time in terms of hours, minutes and seconds, or in other words, deliver the current time in human terms, on request, often while also providing calendar and alarm functionality. Communication with any particular device is often manufacturer-specific, and may occur over different kinds of buses. With the relatively low need for absolute accuracy of the real-time data in a system such as this (the user will not notice if the clock is off by several seconds), the comparatively slow speed and possible latency, even on a crowded I^2C bus, should not pose a problem. However, long term accuracy is of interest, since systems can be expected to run stand-alone for prolonged periods of time. To remedy this, most clocks allow for internal calibration, whereby the internal logic can be made to either skip a few beats every now and then if the crystal runs too fast, or double-count some beats if, conversely, the crystal runs too slow.

For the demonstration-board, the Seiko S-35390A real-time clock was selected [34]. The chip has the following main features:

- Maximum timekeeping current consumption of 0.93 µA at 3.0 V
- I^2C-compatible serial bus interface
- Built-in clock correction functionality
- Calendar with leap-year–calculation function up to the year of 2099
- Basic dual-alarm functionality
- Two configurable interrupt pins, capable of relaying either alarm signals or outputting a user-set frequency

The S-35390A keeps track of year, month, date, day of the week, hour, minute and second data stored in internal registers in BCD form. Data may be read out all at once, or limited to just reading out the time portion, containing hour, minute, and second information. If a more exact reference oscillator is available in the complete system employing the S-35390A, the clock may be corrected on the order of a few ppm, by means of adjustments made to the internal divider-circuit working on the 32.768 kHz

external oscillator signal. Because it is not recommended to load the crystal oscillator with the extra capacitance associated with a test probe directly, when doing frequency measurements, a user-set frequency may instead be output to the interrupt pins and connected to a frequency counter. In other case, the 32.768 kHz signal frequency may be skewed, and the measured frequency prove inaccurate. The S-35390A chip uses a 3-bit subset of the I^2C address-byte for the purpose of specifying commands. This property slightly limits the number of addressable units that may be connected to the same I^2C bus. The S-35390A can operate on supply-voltages in the range of 5.5 V down to 1.3 V, making it eminently suitable for battery operation. Finally, the S-35390A is available in three different casings, of which the SOP-8 version was chosen.

Flash Memory

A still vacant serial interface allowed for using an optional Flash memory, if desired. Flash memories allow for a more cost-effective solution in cases where excessive amounts of data must be stored locally.

Flash, although still a type of EEPROM, offers larger storage capacity in a comparatively smaller form factor, at a lower purchase price. The drawback is shorter operating life, as compared to regular EEPROMs, but not enough to prohibit use in most cases, considering the short (on the order of a few years) life cycle of most embedded systems. A Flash memory commonly has a data retention time of 10–20 years, and can usually withstand on the order of hundreds of thousand write/erase cycles. However, larger chunks of memory space often need to be erased simultaneously, and bytes thus cannot be rewritten individually.

The M25PE10 1-Mibit Flash memory from Numonyx was determined suitable for the initial prototype, and was incorporated into the vacant serial interface [35]. Its main features are:

- Power-down current consumption of maximum 10 µA over the entire operational voltage range
- SPI-compatible serial bus interface, running at speeds of up to 75 MHz
- Page erasable
- More than 100,000 guaranteed erase/write cycles
- Data retention time greater than 20 years

The M25PE10 can operate in a supply-voltage range from 3.6 V down to 2.7 V, of which the lower limit is a bit high, but still acceptable for the sake of this application. Note that if this memory is used, then this voltage-limit will determine the entire system's lower operational supply-voltage boundary. The memory has a page size of 256 bytes, and an internal data-buffer that is capable of holding one page prior to it being read or written. Data can be written to the M25PE10 byte or page-wise, but can only be erased bulk-wise, in chunks of minimum one page or greater. Writing data requires that the data-bits have previously been erased, since with this type of memory, it is only possible to write zeros in place of ones, and not ones in place of zeroes. This actuality slightly limits the usability of this type of memory, and must be taken into account when developing applications that use it. Although the M25PE10 is available in several different casings, the comparatively large SO8N was used to simplify hand soldering. Further drawbacks exist; an erase/write cycle takes around 11 ms, which is

around twice that of the selected EEPROM. The current consumption during both sleep and active state is also much higher.

Human-Interface–Device (HID) Peripherals

To signal activity to a human user, the demonstration-board contains one red and one green LED, a buzzer and initially two pushbuttons (one extra was added at a later stage). The push-buttons were added to enable external input and easy reset functionality. These buttons, as well as the rest of the components of a general nature, were not selected using any particular selection criteria, but were rather chosen out of suitability, availability and convenience.

6.3 MiFare Antenna

For simulation and modeling purposes, it is convenient to study only one half of a differential circuit. The relations below can be used to convert the loop-antenna in Figure 14 into the single-ended form of Figure 15. The values for C_p and C_s remains the same, only there are two of them each on the real board.

$$L_{ANT} = L_{ant} + L_{ant} \tag{27}$$

$$R_{ANT} = R_{ant} + R_{ant} \tag{28}$$

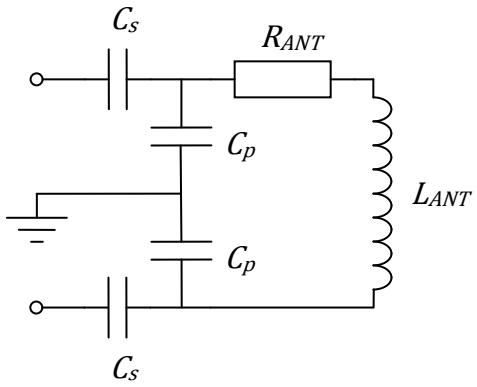


Figure 14: Differential Antenna Model

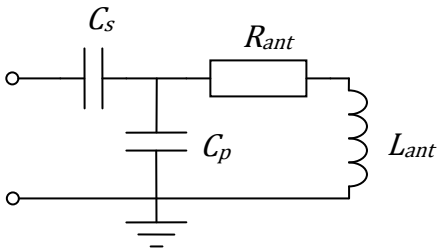


Figure 15: Single-ended Antenna Model

In order to enhance the current flow through the antenna coil, passive current-enhancing techniques can be employed by the addition of series and parallel capacitances. These serve to increase the oscillating reactive currents. It can be shown that for the series-parallel combination showed in Figure 15 above, the current through the antenna circuit is maximized when the following condition is met [21]:

$$\omega^2 C_s = \frac{1 - 2\omega^2 L_{ant} C_p + \omega^2 {C_p}^2 \left(\omega^2 {L_{ant}}^2 + {R_{ant}}^2\right)}{L_{ant} - C_p\left(\omega^2 {L_{ant}}^2 + {R_{ant}}^2\right)} \qquad (29)$$

Although other combinations are possible, this is the best choice, since in the case a transmission line is needed to connect the antenna with the driver, it allows for matching to an arbitrary feed-line impedance [21]. In this case, however, transmission-line effects on the antenna feed-line have not been taken into detailed consideration, since the traces on the board are much shorter than the operating wavelength, even with regard to harmonics. Figure 16 shows the MFRC523 and its associated parts:

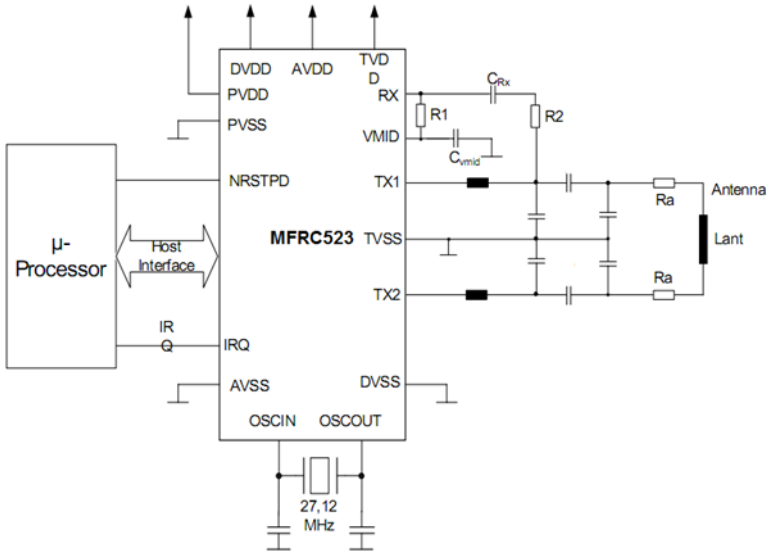


Figure 16: Simplified MiFare Transceiver Schematic [1]

The receiver circuit in the MFRC523 needs to be fed with the signal centered in between the analog supply-voltage and ground. The required midpoint voltage is generated from an internal reference, but requires an external filtering capacitor. The signal tapped from the antenna must be attenuated so as not to exceed the dynamic range of the input, and be superimposed onto the center voltage using AC-coupling.

The EMC filter consists of an LC-filter connected directly to the TX-output pins of the MFRC523. The cutoff frequency for this filter is designed to lie above 14.4 MHz, encompassing the 847.5 kHz sub-carrier used for card-to-reader communication. Thus, with the filter also present, it too needs to be taken into consideration when deciding upon appropriate matching components. The resulting single-ended circuit is depicted in Figure 17 below:

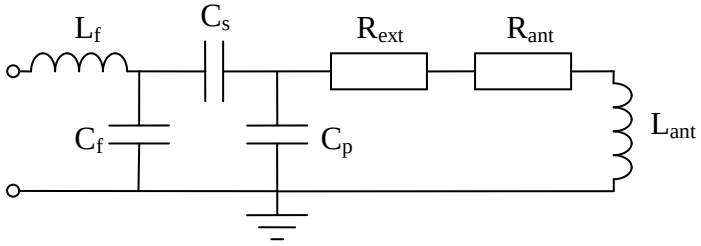


Figure 17: Complete Single-Ended MiFare Antenna Schematic

The transfer function for this antenna-system, assuming a coupling constant equal to zero, can thus be written as:

$$\frac{v_o}{v_s} = \frac{R_{ant} + \mathrm{j}\omega L_{ant}}{R_{ext} + R_{ant} + \mathrm{j}\omega L_{ant}} \cdot \frac{(R_{ext} + R_{ant} + \mathrm{j}\omega L_{ant}) \| \left(\frac{1}{\mathrm{j}\omega C_p}\right)}{\frac{1}{\mathrm{j}\omega C_s} + (R_{ext} + R_{ant} + \mathrm{j}\omega L_{ant}) \| \left(\frac{1}{\mathrm{j}\omega C_p}\right)} \cdot \frac{\left(\frac{1}{\mathrm{j}\omega C_s} + (R_{ext} + R_{ant} + \mathrm{j}\omega L_{ant}) \| \left(\frac{1}{\mathrm{j}\omega C_p}\right)\right) \| \left(\frac{1}{\mathrm{j}\omega C_f}\right)}{R_s + \mathrm{j}\omega L_f + \left(\frac{1}{\mathrm{j}\omega C_s} + (R_{ext} + R_{ant} + \mathrm{j}\omega L_{ant}) \| \left(\frac{1}{\mathrm{j}\omega C_p}\right)\right) \| \left(\frac{1}{\mathrm{j}\omega C_f}\right)} \tag{30}$$

Plotting the above equation over C_s and C_p, using suitable component values, yields the following surface:

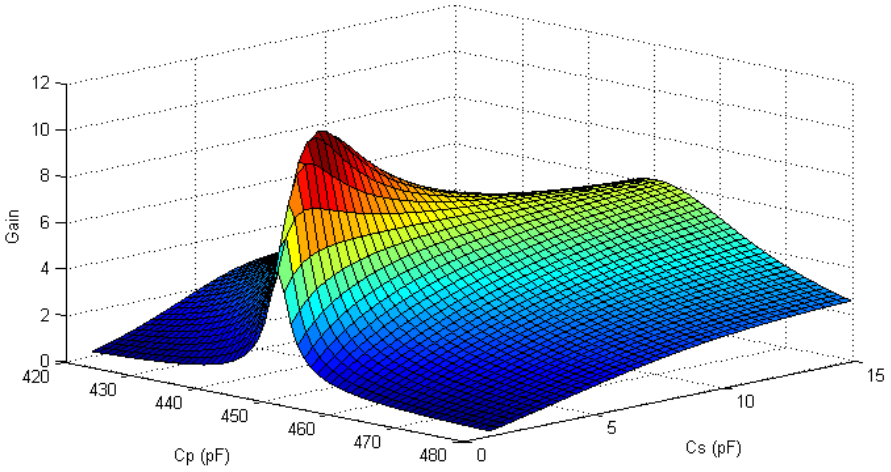


Figure 18: Antenna Gain versus C_p and C_s

Based on the recommendations found in the Philips antenna design guide [13], a loop-antenna consisting of two turns of copper on an FR4-board, was devised for the demonstration-board. Using the formulas found in Microchip's design note on RFID coil design [22], the inductance L_{ANT} was calculated to lie around 0.6 µH. The resistance R_{ANT} was instead measured to be around 0.2 Ω at DC.

Plotting Equation (30) versus frequency, using suitable component values, results in the graph shown in Figure 19 below:

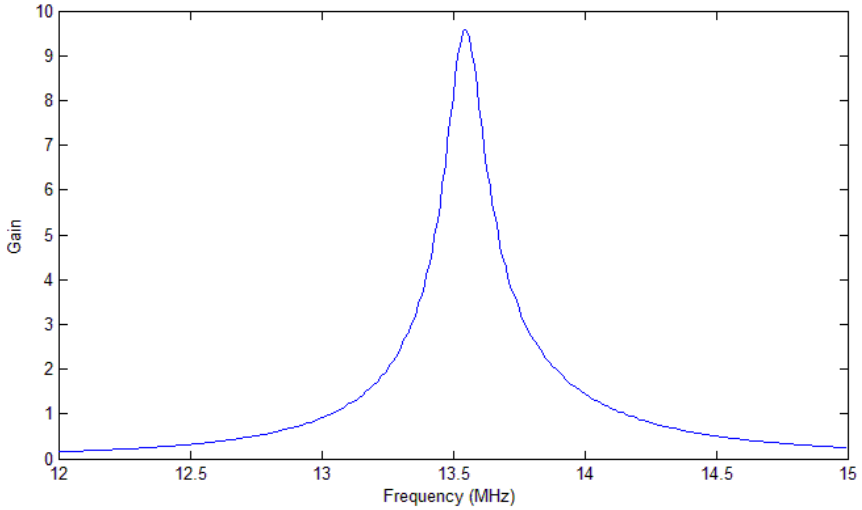


Figure 19: Antenna-Gain versus Frequency

6.4 Test-Support Peripherals

To facilitate testing of issues relating to supply-voltages, a simple adjustable regulator was implemented on the demonstration-board. This arrangement made it possible to test both system-level operations as a whole, with diminishing battery-voltage levels, as well as more specifically determining how this affects the operating-range of the MiFare interrogator.

6.5 Demonstration-Board Layout

The demonstration-board layout was developed in CadSoft Eagle [36], and made to fit on a standard Euroboard-sized PCB (100x160 mm). For the sake of simplicity, all surface-mounted ICs were mounted on prefabricated adapters, while other surface-mounted components, for instance decoupling capacitors and antenna matching components, were mounted directly on the board itself, as to not increase current-loop areas more than necessary. Two complete demonstration-boards were assembled, of which one is depicted in Figure 20 below. Demonstration-board layout and documentation can be found in appendix B.

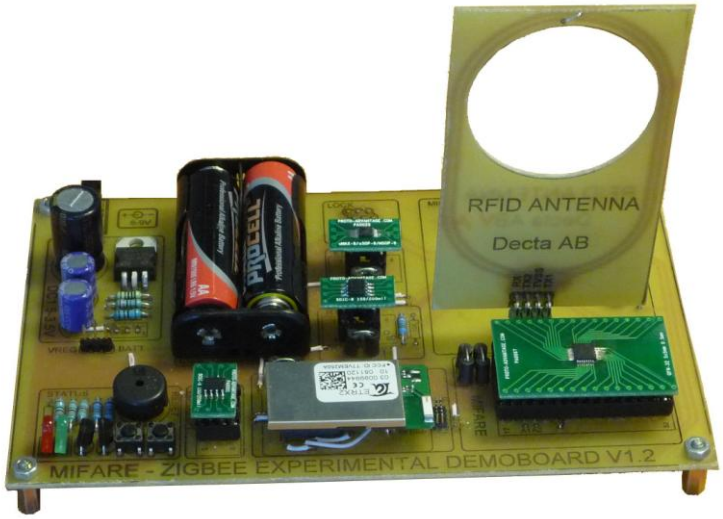


Figure 20: Demonstration-Board

6.6 Demonstration-Board Findings

Different measurements were taken on the demonstration-board in order to confirm operation and test the feasibility of complying to previously estimated current-budgets, before deciding on taking the design to the next phase.

In order to perform sleep-current measurements on the demonstration-board setup, a KM-48 Ampere-meter from Kyoritsu was connected in series with the power supply [37]. The measurements were taken when the device was in battery-operated mode, in order to eliminate any impact from residual currents running through the Ampere-meter via an external voltage-supply unit with comparatively lower impedance to ground. The ICs on the demonstration-board were added successively, in order to monitor their contribution to the total current draw. The results are depicted in Table 6 below:

EM250:	S-35390A:	MFRC523:	24AA256:	M25PE10 Standby:	M25PE10 Sleeping:	Measured Supply Current (µA):
•						2
•	•		•			3
•	•	•	•			4
•	•	•	•	•		16–18
•	•	•	•		•	12

Table 6: Demonstration-Board Sleep-Current–Measurement Results

Results show that the measured sleep-currents come relatively close to what can be expected when comparing to figures from the datasheets for the different components.

The mean allowable current-budget is 142 µA when two serially connected AA-cells are operated down to 2.2 V according to Equation (23). However, the figures above do not include the higher RFID poll current that is drawn at repeated intervals. It should also be noted that the correct operation of the MiFare MFRC523 interrogator is not guaranteed below 2.5 V. If the optional Flash memory is used instead of the EEPROM, then this limit is raised even further, up to 2.7 V.

For reading-range measurements, two cards were used as references; one MiFare Ultralight credit-card–sized PICC, and one smaller MiFare Classic key-chain–sized tag.

The MiFare inductive loop-antenna was tuned to achieve maximum peak-to-peak voltages over its loop end-points, as this corresponds rather well to maximum read-out range. These values were subsequently recorded for varying parallel capacitance values (C_p), as depicted in Figure 21 below:

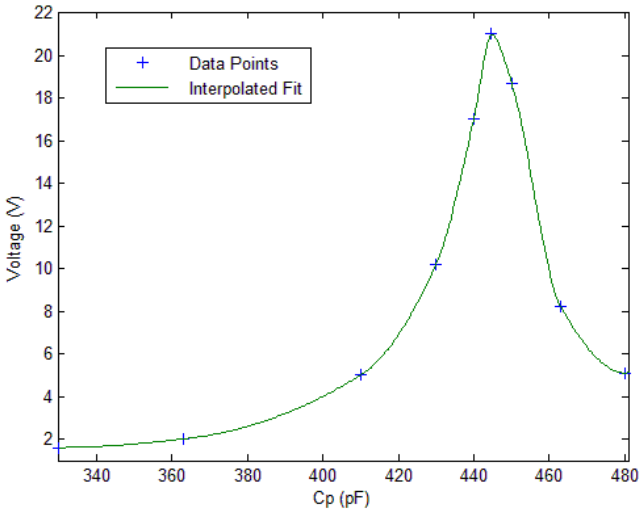


Figure 21: Demonstration-Board Single-Ended Antenna Voltage versus C_p

With L_f =2.2 µH, C_f =56 pF, C_s =4.7 pF and C_p =444.7 pF, using the maximum-power setting for the MFRC523, a maximum reading-range of about 9.0 cm was achieved for the large Ultralight type card, and 6.0 cm for the smaller Classic-type tag. At this power setting, the MFRC523 consumed about 53 mA with the transmitter active, but lower power consumption could be obtained by using lower power settings at the cost of decreased read-out range.

Furthermore, using the expression for finding the resonance frequency of an LC pair, and rewriting it to determine the inductance for known values of C and ω results in an L_{ant} equal to 0.62 µH, showing a close correspondence to the calculated value.

$$\omega = \frac{1}{2\pi\sqrt{LC}} \tag{31}$$

$$L = \frac{1}{(2\pi\omega)^2 C} \tag{32}$$

The default-power setting turned out to be a good compromise, as the current consumption for the MiFare transceiver with this setting was 38 mA, while the reading-range was 8.2 cm for the large card and 6.2 cm for the small card.

A poll interval of 750 ms was determined to be the acceptable upper limit in order for the system not to be perceived as sluggish by end-users, though this value is of course subjective by nature.

The minimum time required for determining whether a card is present or not is 6 ms. This time, together with a repetition rate of 0.75 seconds, results in an expected average current draw of:

$$38\ \text{mA} \cdot 6\ \text{ms} \cdot \frac{1}{0.75\ \text{s}} = 304\ \mu\text{A} \tag{33}$$

While the measured current consumption for the MFRC523 is significantly less than the stated worst-case value from the datasheet, it results in an average value clearly above the 142 µA current-budget, indicating the need for either larger batteries or an alternative card-detection method. For this reason, it was decided that the Si1102 InfraRed (IR) proximity detector from Silicon Labs should be used [38].

Adding the protruding lock-cylinder to the antenna changed the inductance of the loop-antenna significantly, forcing a retuning of the antenna with a correspondingly larger parallel capacitance, to once again make it resonate at 13.56 MHz. However, once the antenna was retuned, the performance was degraded, though not prohibitively so. The maximum read-out range was then measured to be 8.1 cm for the large card, and 5.7 cm for the small Classic-type tag.

Overall, no unsurmountable obstacles were encountered, making it possible to continue with the full production prototype. Metal in the form of the lock-cylinder in the center of the RFID antenna, surprisingly enough, did not constitute as large a difficulty as was initially expected.

7. Firmware

In this chapter, firmware development is detailed.

7.1 Application Programming Interface

In order to ease the development process of a system based on a certain technology, thus making the technology more attractive to customers, hardware vendors often provide customers with a higher-level–language library to interface to that technology and provide premade functions for it. By doing so, they help customers avoid falling into common programming pitfalls. Such a library of interfacing functions is called an Application Programming Interface (API), and in the case of Embers EM250 chip, the provided API is called EmberZNet. EmberZNet contains a rich set of well documented high-level functions for interfacing between the C programming language and the EM250 ZigBee SOC hardware.

7.2 Development Environment

The firmware was developed using Ember's InSight Development-Environment kit depicted in Figure 22 below. Development started by using the included pre-manufactured EM250 Breakout boards, but as soon as the demonstration-board hardware was available, development shifted onto this platform. The development kit consists of both hardware and software components for the PC.

Figure 22: Ember EM250 Development-Kit [60]

The Ember InSight development-environment–kit hardware consists of:

- 3 pieces of EM250 InSight Adapters
- 3 pieces of EM250 Breakout Boards
- 1 piece of MC Card to SMA Cable
- 3 pieces of InSight Port Cable
- 3 pieces of Power Supplies and Battery Packs
- 3 pieces of Extended Debug Cables
- 1 piece of 8-Port Switch with four Power-over-Ethernet (POE) ports
- 10 pieces of EM250 Chips

The Ember InSight development-environment–kit software consists of:

- InSight Desktop
- EmberZNet ZigBee-Pro Stack
- xIDE Compiler
- Ember AppBuilder

The most important piece of hardware mentioned above is the InSight Adapter, which can be connected to a PC over Ethernet and further on to an EM250 (or EM260) chip via the SIF interface InSight port. Unless too much current is tapped from this port, it may also be utilized as a power supply. The InSight Adapter can be used for uploading firmware, debugging applications, and reporting time-stamped radio traffic, sent or received over the air.

The Breakout boards present a quick way of getting started with programming the EM250 chip, as they add peripheral connection abilities and input/output devices like buttons, LEDs, a buzzer etc. All the initial tests, such as sending simple messages etc., were done using the breakout boards.

The compiler environment is called xIDE, and consists of a C-compiler, an Assembler, and an integrated debugger. InSight Desktop, on the other hand, can be used for uploading firmware and monitor network activity. These two programs, together with the EmberZNet ZigBee-Pro–Stack C-library, make up for a complete development solution facilitating developing ZigBee-based firmware for the EM250 platform. The Ember AppBuilder application is a quick program-building application used for assembling ready-to-use standard ZCL applications that may be compiled in the xIDE environment. This program, however, presented a much too crude way of building applications in order for it to be usable in this project, and thus was never used other than for learning purposes.

Other hardware that facilitated firmware development was:

- Telegesis Ethernet Access Point (EAP-E) [39]

Other software that facilitated firmware development was:

- MATLAB [40]
- PuTTY Terminal software [41]
- Bray Terminal software [42]
- Lantronix Com-Port–Redirector Manager [43]
- Lantronix Device Installer [43]

7.3 Program Structure

System firmware was built around an infinite main program loop, in which function calls are made and state machines are ticked, in order to divide processing time between all the different activities (refer to appendix A for a graphical description of the different state machines). The main program loop was lifted out and put inside a function, so that it could be called from different locations. In this way, important scheduled system activities could be called and executed from various places, without having to leave the

current position inside a particular function. It also allowed for easier test runs, where for instance, a user-data configuration could be added, the main loop called to run the necessary state machines and system functions, and then another user-data configuration could be added etc., all without ever leaving the test function. However, because recursive main-loop operation is not generally desirable in an embedded system, that functionality was tightly controlled, and it was made sure that the system was disallowed from receiving further commands while already busy interpreting previous ones. Stack-memory allocations were also adjusted correspondingly.

7.4 Data Management

Data is generally stored in slots on non-volatile memory. Each slot is associated with a bit in a bit-field at the beginning of each page of memory, indicating whether that slot is being used or not. This method of storing data allows for storing any kind of data permutation inside the assigned slot, without disqualifying certain special numbers. The concept is illustrated in Figure 23 below. If the number of data-slots is not an integer multiple of eight, then the spare bits in the bit-field are disregarded. Please note that the number of slots that can fit into one page depends on the size of each slot, as well as the page-size of the employed non-volatile memory.

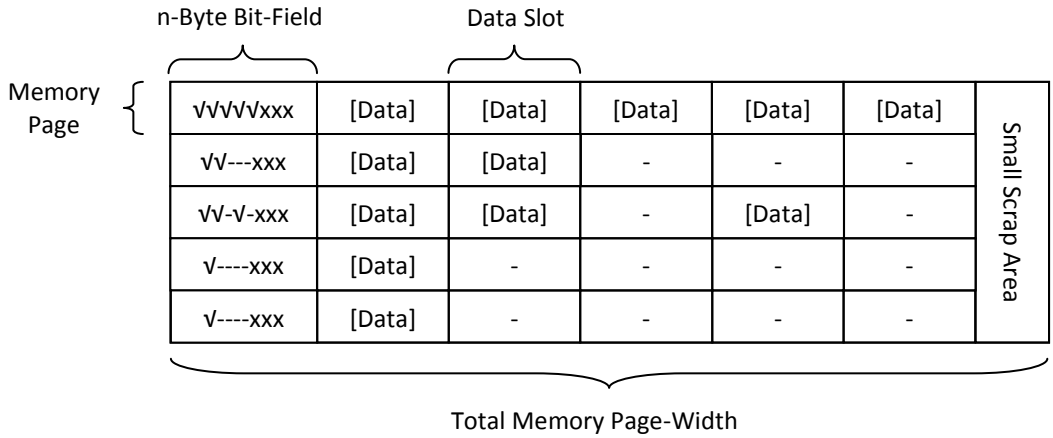


Figure 23: Data-Slot Storing Mechanism

The non-volatile memory, where data is kept while the system is sleeping or powered off, has been structured into four distinct sections used for storing different kinds of data. The sections are:

The User-Data Section

This section is used for storing user-data configurations, that is, UIDs together with corresponding time-schedule numbers.

The Log Section

This section is used for storing information about which UIDs have been associated with opening the door at what time, together with a status byte for each entry, indicating further what has actually happened.

The Time-Schedule Section

This section is used for storing time schedules, i.e. representations of intervals in time during which certain UIDs are permitted access.

The Information Section

This section is used for storing the values of important system variables that need to be kept safe during a power-down or reset event.

The number of pages assigned to each section, as well as the length of user-data etc., is all fully customizable at either runtime or compilation time. For instance, the number of pages assigned to storing user-data may be set via a ZigBee command, following a format-memory command, while the length of user-data is not considered to change during runtime, and thus, is only settable at compile time.

User-Data Handling

Because user-data is stored on external non-volatile memory such as EEPROM or Flash, and because interfacing these possibly very large memories via serial protocols (such as I^2C or SPI) takes time, a custom algorithm for placing and finding user-data configurations swiftly was devised. The algorithm was designed to balance different important aspects that need to be taken into consideration in order for the system to work seamlessly. The list-system maintains a partially sorted list of user-data configurations for fast searchable access, and is self-sorting and self-defragmenting as determined necessary. It is also fully searchable at all times, even during sorting and defragmentation operations. The non-volatile memory, as described previously, has been divided into memory pages and slots which form user-data rows (simply called pages) and columns. Because the system is rather scarce on Random-Access Memory (RAM), most sorting and defragmentation operations must take place directly on EEPROM or Flash, and because these types of memories can only withstand a limited number of write cycles, care must be taken to assure that unnecessary write operations are avoided during sorting or defragmentation of the list. For the purpose of simplifying the sorting procedures, each user-data–configuration slot is assigned a certain index number. Slots are indexed in the manner shown in Table 7 below:

#0	#1	#2	#3	#4
#5	#12	#19	#26	#33
#6	#13	#20	#27	#34
#7	#14	#21	#28	#35
#8	#15	#22	#29	#36
#9	#16	#23	#30	#37
#10	#17	#24	#31	#38
#11	#18	#25	#32	#39

Table 7: Slot Index-Numbering

Adding and Placing User-Data Configurations

The procedure for adding user-data configurations was conceived in such a way as to try and avoid unnecessary operations for as long as possible before any rearrangement procedure becomes necessary. A user-data configuration is placed in the list depending upon the value of its UID. Because the sorting algorithm works on the UIDs of the user-data configurations, and for reasons of simplicity, the explanation below simply uses the term 'number' to denote an element that is really a representation of a complete user-data configuration. Thus, returning to the add-and-place procedure, consecutive numbers are simply added one after the other in the currently active column. If a column gets filled up, a rearrangement procedure is initiated, and the next column becomes the active column as soon as rearrangement has finished. User-data-configurations with user-numbers that are lower than the highest consecutive number, generally gets placed un-sorted two steps to the right of the active column, in the same page as their closest lower sorted number. They remain there until a rearrange procedure gets initiated, which may happen in a number of ways. The procedure is illustrated in the following few scenarios below. Please note that, before any number can be added, the entire list must be searched to make sure that the number is not already present in the list. This procedure is performed automatically every time a number is added.

20				
30				
40		47	45	
50				
60				

Table 8: List-Management Example

In Table 8 above, numbers 20, 30, 40, 50 and 60 have been added consecutively. Then numbers 47 and 45 were added, and got placed to the right of their closest lower number, which is 40 in this case. The sorted section is marked as dark grey and buffer space is marked as light grey.

20				
30				
40		47	45	
50				
60				
70				
80				

Table 9: List-Management Example

In Table 9, numbers 70 and 80 have also been added, triggering a rearrange operation, described in Table 10 and Table 11, as well as in the text below.

20	70			
30	80			
40		47	45	
50				
60				

Table 10: List-Management Example

The rearrange procedure starts by moving numbers 50 through 80 two steps forward (index-wise), to make room for numbers 47 and 45, as in Table 10 above. This relocation divides the list into two sorted sections, which are both binary searchable. After that, numbers 47 and 45 are Quicksorted internally, so that they can be placed into the two newly made-available slots. Two new numbers, 49 and 48, are then added to create Table 11, again triggering a rearrange procedure.

20	70			
30	80			
40				
45				
47			49	**48**
50				
60				

Table 11: List-Management Example

18				
20	50		25	
30	60		34	
40	70		73	**42**
45	80			
47				
48				
49				

Table 12: List-Management Example

In Table 12, five new numbers have been added. Number 18, which is smaller than the very first number, 20, has been placed in the empty first row designated to this purpose. Numbers 25, 34, 73 and 42 have been placed in empty slots to the right of their corresponding closest lower sorted number. When number 42 was added, it triggered a new rearrange procedure resulting in Table 13 below:

18				
20	47	80		
25	48			
30	49			
34	50			
40	60			
42	70			
45	73			

Table 13: List-Management Example

18	7	4	16	15
20	47	80		
25	48			
30	49			
34	50			
40	60			
42	70			
45	73			

Table 14: List-Management Example

In Table 14, numbers 7, 4, 16, 15 have been added to the first row. Then number 3 is added, causing also the first row to be sorted and put into the sorted section of the list. Number 3 is actually not added to the first row until the rearrange procedure resulting in the grey portions of the list visible in Table 15 below, has finished, in which numbers 90, 100, 110, 120 and 130 have also been added.

3				
4	30	49	100	
7	34	50	110	
15	40	60	120	
16	42	70	130	
18	45	73		
20	47	80		
25	48	90		

Table 15: List-Management Example

Now with exception to the first row, there is no more room for placing numbers that are less than the highest sorted number, because some buffer space is needed in order to move numbers around. If, for example, number 32 is added at this point, then all numbers higher than 32 must be moved one step upwards in order to provide space for this number. This procedure for 'moving in' numbers is not very efficient, but is only employed once the list is already starting to fill up, and again divides the list into two binary-searchable sections, still making it fully searchable during processing. See Table 16 and Table 17 below:

3				
4	30	48	90	
7		49	100	
15	34	50	110	
16	40	60	120	
18	42	70	130	
20	45	73		
25	47	80		

Table 16: List-Management Example

3	1	2		
4	30	48	90	160
7	32	49	100	170
15	34	50	110	180
16	40	60	120	190
18	42	70	130	200
20	45	73	140	210
25	47	80	150	220

Table 17: List-Management Example

In Table 17, number 32 has been added successfully, as well as numbers 1, 2 and 140 through 220. The list is now considered full, with the possible exception of number 0, which may still be added to the first row.

Removing User-Data Configurations

In Table 18 below, numbers 3, 1, 7, 15, 16, 45, 220 and 210 have all been removed from the previous table. Numbers outside of the sorted section simply gets deleted or moved one step to the left as required, while numbers in the sorted section are kept, but disabled via the use-mask bits in the beginning of each memory page. The reason the numbers are kept, is to ensure that the sorted section will still be binary searchable even after the remove operation has taken place. If a deleted number is still found, it is simply regarded as not found. Numbers 7, 15, 16 and 45 indicate this condition. A new number that fits, may be added to such a deleted slot, e.g. number 9 as in Table 19 below, where also 210 and 220 have been added, to make the list full once again.

2				
4	30	48	90	160
(7)	32	49	100	170
(15)	34	50	110	180
(16)	40	60	120	190
18	42	70	130	200
20	(45)	73	140	
25	47	80	150	

Table 18: List-Management Example

2				
4	30	48	90	160
9	32	49	100	170
(15)	34	50	110	180
(16)	40	60	120	190
18	42	70	130	200
20	(45)	73	140	210
25	47	80	150	220

Table 19: List-Management Example

Defragmenting the List of User-Data Configurations

When the list has become full as in Table 19, but new numbers still need to be added, a defragmentation procedure gets initiated, examining whether there are any deleted numbers residing in the sorted section. Starting from the first sorted index, all the numbers marked as deleted get cleaned out, as in Table 20 and Table 21 below, before a new number, 230, can be added.

2				
4	30	48	90	160
9	32	49	100	170
18	34	50	110	180
20	40	60	120	190
	42	70	130	200
	(45)	73	140	210
25	47	80	150	220

Table 20: List-Management Example

As soon as the defragmentation procedure has finished, a new (successful) attempt to add 230 is made, as seen in Table 21. Again, the list is fully searchable during this operation.

2				
4	34	60	120	190
9	40	70	130	200
18	42	73	140	210
20	47	80	150	220
25	48	90	160	230
30	49	100	170	
32	50	110	180	

Table 21: List-Management Example

The Quicksort and Binary-Search Algorithms and Their Usage

In order to be able to sort a list of user-numbers, the well-known Quicksort algorithm was used [44]. Quicksort is considered very fast with regards to the number of comparisons needed for sorting a list. On average it makes $O(n \cdot \log(n))$ comparisons, and in the worst case, it makes $O(n^2)$ comparisons in order to sort a list of n elements [45]. It works by first selecting a pivot element, and then it compares all other elements to that pivot element, sorting all elements that are less than the pivot element to one side of it, and all elements that are larger than the pivot element to the other side of it. This proceeding assures that the pivot element must then be in the correct final position, dividing the list into two subsets. Each of these two subsets can then be recursively sorted by once again employing the Quicksort algorithm onto the two subsets. The procedure continues until it can be concluded that all elements must be in the correct position, leaving the list completely sorted.

Because of the manner in which numbers are placed to begin with, the Quicksort algorithm is only utilized (page-wise) when sorting the unsorted numbers of either the first page (the smallest numbers in the list), or the numbers, whose values are in between those of the sorted numbers (i.e. the numbers in the rows in the right-hand section of Table 12). This approach means that n is never going to be very large (on the order of maximum a few dozen elements), implying that the difference between the Quicksort algorithm and a possibly simpler approach is likely to be rather small. Furthermore, because an entire non-volatile memory page is read into RAM-memory all at once, for the matter of timing and power issues regarding serial-interface operations, the simpler approach might have been employed almost as well. Still, a sorting algorithm had to be chosen, and Quicksort was still deemed appropriate. As soon as a memory page has been sorted in RAM, the result is written back over the original page on non-volatile memory.

The Binary-Search algorithm was used in order to swiftly be able to find a user-number without having to search through all the elements in the list. In order to be effective, the Binary-Search algorithm presumes that the list has been sorted previously, and works by selecting the middle element of the set, determining whether that number is larger or smaller than the number sought. If the number is larger than the number sought, then the sought element must be located below the middle number, and if the number is smaller than the number sought, the sought element must be located above the middle number. The procedure is then repeated on the subset of interest until, eventually, the sought number is either found, or can be concluded to not exist in the sorted list. On average, Binary Search makes about $(\log_2(n) - 1)$ probes, and in the worst case $\log_2(n)$ probes to find an item in a list of n elements [45].

A combination of Binary Search and simple look-through-all-the-elements-in-the-set method has been employed to find sought UIDs. Basically, all sorted elements are searched using Binary Search to find a number, and if the number cannot be found in the sorted section, then the simpler approach is employed on all the remaining places where the number could possibly reside. This approach does not pose a timing problem, as the remaining slots are always limited to residing within one memory page on non-volatile memory.

List-Handling Verification

As previously mentioned, the user-data handling algorithm was first developed in MATLAB and then recreated in C, with the addition of then using state machines. Different test-cases trying to cover certain error-prone situations, were first run in MATLAB. For instance, lots of numbers were added, flooding the list, then numbers were removed, numbers added again, trying to provoke and remedy flawed behaviour while also running the binary-search algorithm simultaneously. A test-list was created and finally filled up this way, then all the add and remove-number operations were accounted for, and noted, to see if any discrepancies against the generated test list were present. The same test case was then run in the C-code version, again looking for discrepancies against the result from the MATLAB version. None were found.

Log-Data Handling

Once an event has occurred, for instance a user has opened the door using a key-card, a log entry, consisting of the user-number causing the event, the current time and date, and a status byte indicating which kind of event has occurred, is stored onto non-volatile memory. The entry is stored in a special area of the non-volatile memory, dedicated to this purpose. Log entries are slightly compressed by cleaning out all abundant data bits from the byte values for the date entry. This approach allows for one extra log entry per memory page to be written onto the standard EEPROM variant. Log entries are stored into slots, as mentioned before, and once the memory has been consumed, the oldest entries are overwritten in a cyclical fashion. It is thus up to the PC gateway to retrieve the log entries before they are overwritten. One could also imagine a scenario where the log entries get sent back to the gateway node, as soon as possible after an event has occurred, or as soon as the memory is starting to fill up. This functionality, however, has not been implemented.

Time-Schedule Data Handling

Management of time schedules is a rather straightforward affair. Time schedules are stored into numbered slots onto the designated portion of the non-volatile memory. Time schedules may be updated or removed. Removing a time schedule simply means to un-mask it on memory, since it is not really necessary to actually delete data that cannot be accessed anyway. Note that a time schedule may be active, though remain un-referenced from the user-data section.

Information-Section Data Handling

As mentioned previously, this section stores the values of important system variables that need to be kept safe during a power down, normal reset event, or a system hang causing a watchdog-triggered reset. This section is updated every time the list changes. It uses some extra memory and a cyclic wear-levelling algorithm to prevent the non-volatile memory from malfunctioning due to an excessive number of write cycles at the same spot over longer periods of time. The wear-levelling algorithm works by simply spreading out commonly stored data onto a customizable number of slots, using a 16-bit sequence-number to determine which slot was written to last. A higher sequence-number indicates newer data, and as the number is about to wrap around back to zero, the entire area is first formatted to ensure no old data may be interpreted as new. At

startup, this section is always searched, looking for the data with the highest sequence-number. If data is found, then it is loaded back into RAM memory.

7.5 Card-Polling and Select Procedure

A state machine was set up and made responsible for the card-polling and sleeping behaviour of the system. If approximately 750 ms has passed since the last poll, a new poll gets initiated. In the meantime, the system will try to sleep, unless it has some important activity scheduled. Such an activity might for example be defragmentation of user-data, or in case it has been awakened by an external interrupt that needs to be dealt with.

When a poll is initiated, the MiFare interrogator is turned on and configured for operation. In order to conserve energy, it is imperative that the electromagnetic field of the interrogator is turned on for as short time as possible. For this reason, the field is not turned on until 6 ms prior to the transmission of the REQA-pulse, as this is the minimum-allowed time for proper energization of a proximity card, according to part three of ISO14443 [11]. If no card is detected, the field is immediately turned off.

Only after a card has been detected, selected, and authenticated, a search is initiated to determine whether that card-number is stored in the user-data section. If the number is found, the associated time schedule is retrieved and compared to the current time read out from the real-time clock. Access is then either granted, whereby the door-lock is opened for a specified amount of time, or denied, on basis of that comparison. A log entry is also written to the log-entry section.

MiFare Classic authentication is implemented on the system. However, since the MiFare-Classic encryption algorithm is considered broken [14] [15] [16], a more advanced authentication procedure is desirable. Such a procedure will become available with the introduction of the MiFare Plus line of cards, which uses AES 128-bit encryption. In the meantime, since it is really only of interest to determine whether a card belongs to the system, because access-rights are already stored inside the door-node, another approach may be taken. By hashing the unique MiFare card UID using a secure cryptographic hash function, with a corresponding secret-key sequence, and then saving the result onto the user-configurable data area of the card, the system would be able to determine whether a card presented to the reader actually belongs to the system or not. It would do this by calculating the expected hash value, using the secret key and the card number, and then check that against the previously stored, pre-calculated hash value on the card. If the results match the results stored in the user-configurable area of the card, then the card is considered authentic [46].

7.6 Remote Operation

In order to set up and properly operate a door-node over a wireless network, a network configuration needs to be agreed upon, a command interface needs to be implemented, and the functionality then needs to be thoroughly verified.

ZigBee Setup

A ZC forming a network, initially performs an energy-detect scan, deciding on, and selecting the channel determined to be the quietest one. It then proceeds to make an active scan, sending out a Beacon request and waiting for a response. These activities are performed in order to determine what other 802.15.4 EPIDs are using that channel. For a node only interested in joining an existing network, only the active scan is performed [4]. A 4-byte (32-bit) channel mask is used to signal which channels to use when forming or joining a network. For the sake of this project, all ZigBee-specific channels, numbered 11–26, were deemed appropriate, though in the case compatibility problems with other networking protocols are experienced, some channels may be masked out later, as required.

A random 8-byte EPID range was selected to represent Decta-specific networks, and the process of carefully selecting a MAC address for each node was omitted at this stage because Decta has not yet purchased an OUI from IEEE. The 8-byte MAC address, unique for each node, should otherwise consist of a 3-byte OUI representing Decta, and five extra bytes identifying each Decta node.

ZCL was not used in this project, as it offered a lot of functionality deemed unnecessary. Instead, a custom command protocol was developed, and a private ZigBee profile ID, in the private profile-ID range 0xBF00–0xFFFF, was selected to house the protocol.

The developed system uses standard-security mode, which means all nodes trust each other internally, thus any device that manages to join a network with the correct EPID using the correct network keys, will automatically be considered a trusted door-node belonging to the system. Because the number of device IDs used in the system, is limited to just one on a single end-point, there is no need for the provided binding and device-discovery mechanisms that could otherwise have been implemented. Any node on the network is simply implied to be of the correct type, and to understand the command protocol. Addressing parameters, such as on which end-point the application should reside, were of course specified, but are not particularly meaningful in a system only implementing one application per ZigBee node, using its own command protocol, given of course, that they coincide.

In order to conserve battery power, a sleepy ZED polls its parent for data as infrequently as possible. The default setting for the parameter that specifies the time interval between which a door-node should poll its parent, is as seldom as once per every twenty seconds, and was considered suitable for this type of application.

All parameters specified in this section may easily be changed later at compile time, if determined necessary.

Command Protocol

A ZigBee-command protocol was developed in order to be able to communicate with a PC-gateway node and execute received commands. The following commands are available to the gateway operator:

Command:	Byte Value:	Arguments:	Action:
FORMAT_ENABLE	128	None	Command required prior to formatting the entire memory
FORMAT_MEMORY	127	None	Formats the entire memory
FORMAT_AND_SET_MEMORY_AREAS	126	Number of pages allocated to the different memory sections	Formats the entire memory, and regulates memory usage
GET_MEMORY_ALLOCATION	125	None	Retrieves the number of memory pages allocated to the different memory sections
ADD_NUMBER	1	The MiFare user-number to add	Adds a MiFare user-number
REMOVE_NUMBER	2	The MiFare user-number to remove	Removes a MiFare user-number
UPDATE_TIMESCHEDULE	3	The time-schedule slot to update and the time schedule itself	Updates a time schedule
REMOVE_TIMESCHEDULE	4	The time-schedule slot to remove	Removes a time schedule, making it inactive
RETRIEVE_LOGS	5	None	Retrieves all available log entries
ERASE_LOGS	6	None	Formats the log section
RETRIEVE_INFO_PAGE_DATA	124	None	Retrieves the contents of the info-page section. Not normally used
SET_GROUP_ID	123	A new ZigBee group ID	Sets a new ZigBee group ID
GET_GROUP_ID	122	None	Gets the current ZigBee group ID
SET_REAL_TIME_CLOCK	121	Year, month, date, day-of-week, hour, minute, second	Sets the time of the real-time clock
GET_REAL_TIME_CLOCK	120	None	Gets the time of the real-time clock
MESSAGE_FOLLOWS	119	None	Keeps the end-node polling its parent more frequently for a few seconds
GET_BATTERY_VOLTAGES	118	None	Returns the present values of VDD_PADS, VDD_CORE, and VDD_LOCK
SET_MIFARE_CLASSIC_KEY	117	A new MiFare-Classic key sequence	Sets the Mifare-Classic Key
UNLOCK_DOOR	116	Duration to be open in seconds, 255 interprets as always open	Unlocks the door for a specified duration
LOCK_DOOR	115	None	Locks the door
IDENTIFY	114	None	Makes remote door-node draw attention to itself

Table 22: ZigBee Gateway Commands

Command arguments are normally 8-bit numbers, though time-schedule numbers, ZigBee group IDs, and memory-page allocations are 16-bit. Time schedules may be up to [PageSize-1] bytes big (i.e. 63 bytes using the 24AA256 EEPROM), and the number of bytes used for a UID may be set at compile time. A MiFare-Classic key sequence always consists of six bytes.

In order to send a command to the door-module, a message must be formatted with the command-bytes as payload. The payload may consist of multiple commands in one message, as many as can fit, or of just a single command per message. If multiple messages are to be sent in consecutive order, it is recommended to send a *[MESSAGE_FOLLOWS]* command somewhere in each message, to keep the end-node

polling its parent more frequently after execution of the first command. Otherwise, sending a bulk of commands may be a slow process.

Messages received at the door-node are validated to make sure they contain a legal sequence of commands. In case validation is successful, an attempt to execute the command(s) is made. Upon completion of a command, a message is returned back to the gateway, containing the sequence-number of the command-message, as well as the American-Standard-Code-for-Information-Interchange (ASCII) byte value for either A or N (Acknowledged or Not acknowledged). In this way the ACKnowledgement (ACK) message can be associated with a specific sent command. Each command in a message receives one separate acknowledgment message. A Non-ACKnowledgement (NACK) message is also sent in case validation failed.

Time-Schedule Format

Time schedules are interpreted according to the scheme found in Table 23 below:

Token:	Byte Value:	Interpretation:
EMPTY	255	Token used to terminate a time schedule
OR	254	Token implying further time intervals
YEAR	253	Token implying one or many year intervals
MONTH	252	Token implying one or many month intervals
DATE	251	Token implying one or many date intervals
DAY	249	Token implying one or many day intervals
TIME	248	Token implying one or many intra-day intervals

Table 23: Time-Schedule Tokens

Time schedules should be written in the form: *[Interval Token][Number of Intervals][Start Interval Value(s)][End Interval Value(s)]*. A time schedule should also end with an *[EMPTY]* token, unless the time-schedule area is completely filled up.

Example:
Monday–Thursday, between 8.00–9.10 a.m. and 3.00–5.30 p.m., or the entire year of 2009, would be written as:

[DAY][01][00][03][TIME][02][08][00][09][10][15][00][17][30][OR]
[YEAR][1][09][09][EMPTY]

Any user-data configuration referring to this time-schedule number during this period of time will be permitted access. Access attempts outside of the valid intervals will be denied, as expected.

In order to allow a user access during a different time-interval, the user-data configuration in question may be removed and then added again, this time referring to a new time schedule. Or, the time schedule may be changed, though this has implications for all other references to that time schedule.

Testing of the Remote Command Interface

The Telegesis EAP-E was used to form a ZC gateway node, making it possible to send different commands to the prototype boards. Results were visualized on-screen with the PuTTY Terminal software, over the SIF interface via the InSight Adapters, and the Lantronix Com-Port–Redirector software was used to route the EAP-E to a virtual Com-port that could be accessed through the Bray Terminal software.

The EAP-E, which is based on the same Telegesis ETRX2 module that was used in the demonstration-board, uses firmware that supports the proprietary Telegesis AT command-set [47]. This firmware makes setting up a network a rather simple task because no custom firmware needs to be developed in order to have it form a ZigBee network. Common network parameters can easily be manipulated using the pre-defined registers dedicated to the purpose, and there are commands for issuing broadcasts, unicasts and the like. For example, after having configured all the common network parameters and formed a network, an add-number command may be sent. Adding card number 0x{04 8B AD 11 12 7A 00} associated with time schedule 0x0000, in Bray Terminal using the AT command-set, the following text needs to be entered:

AT+UCASTB:0A,D9F6#013#001$04$8BAD11$12$7A$00#000#000#013

The command-line above should be interpreted as follows: Send a unicast with binary data of length 0x0A (10 bytes), to node with network address 0xD9F6. The ‘#013’ token is the ASCII carriage-return symbol, and is necessary for the command to execute properly. By specifying the number of bytes to be sent, even special symbols like carriage return may be sent, refraining from limiting the system by disallowing certain binary combinations. When using Bray, the ‘#’ prefix interprets the following characters as a decimal number, while the ‘$’ symbol represents hexadecimal notation. The ‘#001’ token after the first carriage-return symbol, is the add-number command itself. After that comes the card-number and the time schedule to be associated with this new user-data configuration. A final carriage-return symbol is then used to signal the end of the entire command string.

Note that, because a door-node polls its parent node for messages as seldom as once every twenty seconds or so, it is likely that issuing a command will be associated with a delay before reception of either an ACK or a NACK, indicating either success or failure in trying to execute the command.

Building on the example above; after also setting the real-time clock and adding a time schedule to slot number 0x0000, the MiFare card using the previously added UID may be held up towards the antenna, and access will then be either granted or denied, and a log entry will be made. All the gateway commands were tested and evaluated in this manner, and no flaws were discovered.

7.7 Peripheral Devices

Management of the peripheral devices running over the serial buses, such as memory and the real-time clock, was abstracted into two levels. The lowest layer of abstraction provides handling for the low-level manipulation of memory-mapped hardware-registers controlling the integrated serial controllers. It also takes care of timing issues,

such as for example: ensuring that a device is not written to again, before a previous write-cycle has finished, and it is once again ready to accept new data.

The second, and higher layer of abstraction, provides simplified read and write functions intended to be used by the main program. This approach hides the fundamental hardware-specifics of the different memories and clock types from the program using those devices. With the exception of memory size, an application running the higher-level functions does not need to consider the differences between an EEPROM running over an I^2C bus, or a Flash memory running over an SPI bus. Likewise, it does not need to consider what precise clock hardware is used to keep track of time.

RFID Interface

The MFRC523 is controlled on a low level by the process of manipulating several of its internal registers over the I^2C bus, reading received data from, and writing data to be transmitted to, its internal FIFO register, and so on. Its numerous error flags, as well as its internal CRC Co-processor used to ensure the validity of all data transceived, also needs to be considered. The BCC added at the end of the UID during anti-collision and card selection, is verified by software.

To handle timing-requirements for the response, and read/write timings for the MiFare cards, the internal timer of the device was used, as to avoid using any of the EM250 internal counters. In order to handle communication with multiple cards present in the field at the same time, the full anti-collision algorithm was implemented in software.

Miscellaneous

A function capable of generating a square-wave output on the MFRC523 interrupt pin was employed for facilitating measurements and frequency trimming of the 27.12 MHz oscillator. This functionality is convenient for both automated In-Circuit–Test (ICT) routines, and to avoid skewing measurements by loading the oscillator with the capacitance associated with external test probes.

The real-time clock has the power to wake the system from sleep by issuing an interrupt. This feature is prepared for, in both hardware and software, by an interrupt handler, but since no system function requires this functionality at the current stage, the system is left doing nothing more than going back to sleep if such an interrupt were to occur.

8. Production prototype

In this chapter, the development of the production prototype is described.

8.1 Overview

For the final design, the choice of real-time clock was reconsidered, and another unit was selected. Furthermore, the lock interface was moved to a separate I^2C bus constructed in software using two unused GPIOs. This action was undertaken in order to avoid exposure of internal communication and the contents of the EEPROM onto an external bus, as well as containing the relatively fast falling edges of the internal I^2C interface to within the board. This separation of buses also allowed for the communication speed of the lock interface to be lowered, in order to improve EMC behaviour without impeding the speed of the other communication.

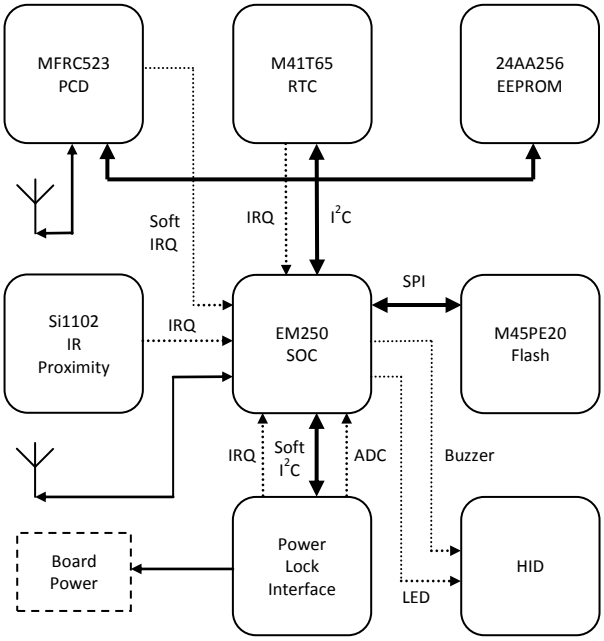


Figure 24: Conceptual System Diagram

8.2 Parts Re-Selection

As work with the demonstration-board ended and proof of concept was attained, a chance to re-evaluate and reconsider the previously selected main constituent parts presented itself.

Real-Time Clock

Since the smallest package the S-35390A is manufactured in, turned out to be unavailable, the choice was made to replace the previously selected real-time clock with the M41T65 from STMicroelectronics [48]. The main characteristics of this chip are:

- Maximum timekeeping current consumption of 0.7 µA at 3.6 V
- I^2C compatible serial bus interface
- Built-in clock-correction functionality
- Calendar with automatic leap-year compensation
- Watchdog functionality
- Programmable alarm with interrupt functionality

The M41T65 comes in a QFN16 casing, which is much smaller than the SOP-8 casing used for the S-35390A. Furthermore, it contains integrated load capacitances for the 32.768 kHz crystal. The M41T65 can tolerate a supply-voltage of at most 4.4 V, is fully operational down to 1.3 V, and timekeeping down to 1.0 V. The features of the chip are otherwise very similar to those of the S-35390A, though the registers for reading out data are slightly more detailed. Another difference is that the M41T65 uses an explicit control byte for issuing commands. Because of the hierarchical and modular firmware structure, only the bottom driver-layer needed to be replaced in order to swap the S-35390A towards the M41T65.

Flash Memory

The M45PE20 from Numonyx offers very similar characteristics to that of the M25PE10, except that it is 2-Mibit, and was selected because of a more suitable pin layout [49]. Its main features are:

- Power-down current consumption of maximum 10 µA over the entire operational voltage range
- SPI-compatible serial bus interface running at speeds of up to 75 MHz
- Page erasable
- More than 100,000 guaranteed write cycles
- Data retention time greater than 20 years

The M45PE20 can operate in a supply-voltage range from 3.6 V down to 2.7 V. The memory has a page size of 256 bytes, and an internal buffer that is capable of holding one page prior to it being read or written. Data can be written to the M45PE20 byte-wise, but can only be erased page-wise. Writing data requires that the data-bits have previously been erased, that is, reset to a high state. For this chip, the SO8N package version was used.

Proximity Detection

In order to lower the power consumption, the Si1102 IR proximity-detector from SiliconLabs was added to the production prototype [38]. Its key features are:

- Proximity-sensing detection with a sensing range of up to 50 cm
- Power consumption typically less than 10 µA in the operational voltage range
- Adjustable detection threshold and strobe frequency
- Detection status-latch to prevent controlling devices from missing a detection

The device operates by pulsing current through an external IR LED. Any light reflected back, and received by the device, is converted to a voltage and then compared to an externally defined threshold. If the voltage is higher than the pre-selected level, the

device latches an output signal that can be used as a wake-on-interrupt signal for a sleeping microprocessor. The current pulsating through the LED can be quite large, so in order to protect both the battery from excessive load changes, and improve the EMC behaviour of the device, extensive decoupling is used by recommendation of the manufacturer [50]. Efforts were also made to reduce the loop-area for the LED drive-current as much as possible. The Si1102 comes in the 8-pin ODFN package, and the operational voltage range is between 2.0 V and 5.25 V.

Since there is no use waking the system for PICCs that are not close enough to be communicated to, or detecting cards at a more rapid rate than what is needed in order for end-users not to perceive the system as sluggish, the Si1102 was configured to poll at only 2 Hz using minimum drive strength.

A caveat on using IR-proximity detectors is that they have only a limited dynamic range, and thus may latch permanently on in bright ambient-light conditions. To guard against this possibility, the firmware should revert to polling the MiFare transceiver at the regular interval, in case permanent proximity is indicated by the Si1102.

Human-Interface–Device Peripherals

The buttons used on the initial demonstration-board were removed from the production prototype, since no such interaction with system end-users is expected, nor desired. The through-hole–mounted buzzer was replaced with a smaller surface-mounted one, and the LEDs were changed to a single integrated RGB-multicolor surface-mounted LED.

8.3 Board Design Considerations

To ease manufacturing of the board, thereby reducing manufacturing costs, an effort was made to use simple PCB design-rules to the greatest extent possible. The design-rules are stated as follows:

1. 7-mil annular ring
2. 8-mil minimum trace-width
3. 6-mil spacing
4. No blind vias

These design-rules impose some restrictions, however. For instance, the ground-plane must have a clearance for every via not connected to ground. Also, the impedance requirements of some traces will be hard to meet, but since the manufacturing tolerances of such thin lines are pretty poor anyway, Ember Corporation recommends simply just minimizing trace lengths as far as possible, thereby minimizing the problem that way instead [51].

Board and Layer Stack

The final PCB is made out of FR4 material with 0.8 mm total thickness. The layer-stack of the board is based on the classical four-layer stack:

1. Signal Plane
2. Ground Plane
3. Power Plane
4. Signal Plane

Modified to look like this:

1. Signal Layer & RF ground
2. Ground Plane & EMC Shield
3. Power, Signal & Mifare antenna
4. Signal Layer & EMC shield

This configuration enables the efficient use of all four layers, as well as allowing for placement of the EMC shields in a symmetrical fashion around the MiFare antenna, even though an undisturbed ground must be provided for the ZigBee antenna.

MiFare Antenna

The same considerations that were taken for the demonstration-board antenna carries over here, and since the antenna used on the demonstration-board exceeded the achievable performance according to the MFRC523 data sheet [1], it was brought over to the production prototype with only minimum changes. Those changes were: maximizing the loop area onto the size of the actual board, and adding EMC shielding on its top and bottom sides. These shields mainly affect the electric field radiated from the antenna, and are made in the form of cover sheets. They may not form a closed circle in cross-section, since that would constitute a short-circuit path for the antenna field. Furthermore, the addition of EMC shields increases the antenna's parasitic capacitance, but since additional parallel capacitance is needed anyway, this does not necessarily pose a drawback.

The EMC filter layout and the TX decoupling of the MFRC523 is of great importance, and efforts were made to reduce the loop area of the filter and the decoupling pin as much as possible. Since both the antenna and the drivers are differential, the return currents from the filter should ideally sum to a constant, and are thus routed together before they are joined to the ground plane.

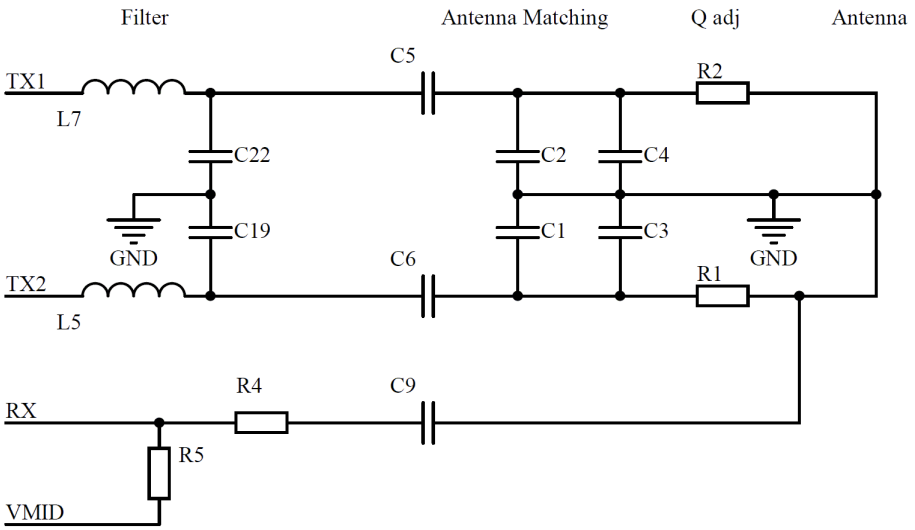


Figure 25: MiFare Antenna Schematic

Additional components R4, R5 and C9 are used to condition the received signal for the MFRC523's internal receiver. This conditioning only consists of attenuation and level-shifting as to avoid exceeding the receiver's dynamic range.

A ferrite-bead–capacitor filter depicted in Figure 26, was also added to the TX-pin power supply. This filter is intended to be tuned during EMC testing, in order to provide the most beneficial attenuation. The filter layout is shown in Figure 27.

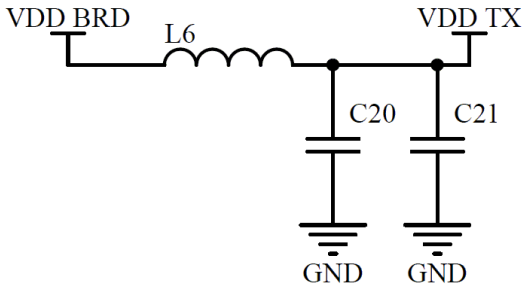


Figure 26: MiFare TX-supply EMC-Filter Schematic

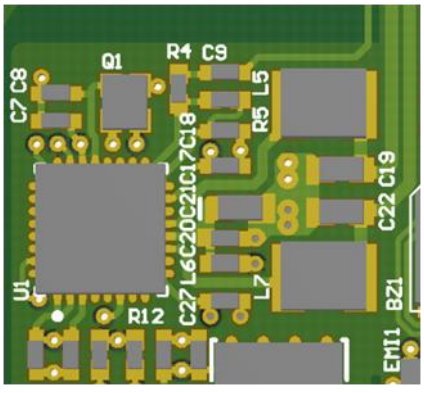


Figure 27: Close-Up of MiFare TX EMC-Filter Layout

Controlled Impedance-Traces and Decoupling

With the removal of the pre-fabricated Telegesis ETRX2 module, the EM250 SOC, and all its associated components, had to be implemented directly on the board. Ember does provide comprehensive recommendations, as well as reference designs to facilitate this process [51] [52]. However, because of specific board-shape demands, they could not be strictly followed, though all the stated rules have been adhered to.

Of great concern is the RF-decoupling of the transmit Power Amplifier (PA), as well as the Voltage-Controlled Oscillator (VCO), and the Low-Noise Amplifier (LNA) in the receiving circuitry. Decoupling for these circuits was laid out first, and every ground-return path was considered in detail. Every effort was made for decoupling capacitors to be laid out with minimum trace-lengths, and ground via placements were made as to minimize parasitic inductance, which would otherwise reduce the effectiveness of the decoupling capacitors.

ZigBee Antenna, BALUN and Filter

A ceramic chip-antenna from Johanson Technology was selected for the ZigBee transceiver. Johanson Technology does not reveal the internal structure of their antennas, but a ceramic chip-antenna is usually a quarter-wave monopole, internally constructed as a spiral wound around a material with suitable dielectric properties, selected as to shorten the wavelength as much as possible, in order to make the antenna both compact and efficient.

Even if the antenna is designed by the manufacturer, to present a 50 Ω impedance at 2450 MHz, it will be affected by the dielectric properties of the board, the thickness, shape, and fill of the ground plane, as well as the influence from any other object in its near-field. The antenna, as specified in the datasheet, is very omni-directional, with an average gain of only -0.5 dBi. In the datasheet, Johanson Technology suggests that the antenna is matched with a shunt and a series inductor [53]. This proposal seems to imply that an unmatched antenna can be expected to present a capacitive load to the feed-line. A grounded Co-Planar–WaveGuide (CPWG) structure with a 50 Ω impedance, was used to feed the antenna after the BALUN and the EMC filter. A co-planar waveguide was used instead of the more common micro-strip line, since it requires less space and produces comparatively less leakage fields.

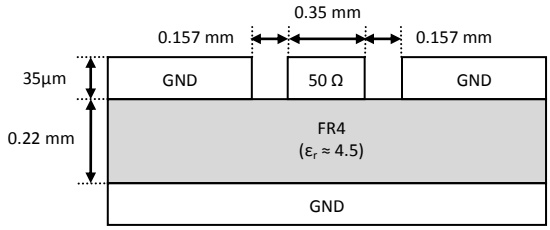


Figure 28: 50 Ω Grounded Co-Planar Waveguide

The antenna drivers on the EM250 are differential for the same reason as those on the MFRC523: to use the available supply-voltage with more efficiency. However, since the operating frequency is 2.45 GHz, transmission-line effects must be considered, and the chip requires an external BALUN to match the 200 Ω differential outputs to the

50 Ω antenna feed-line. The receiving circuitry is connected internally, and does not require further external components for signal conditioning. The selected BALUN, which is manufactured by Murata, has a maximum insertion loss of 1.1 dB at its center frequency of 2450 MHz, and was recommended in the Ember reference design. The EMC filter was selected to give good attenuation of unwanted sidebands, without having too high insertion loss, as to not degrade the link-budget too much.

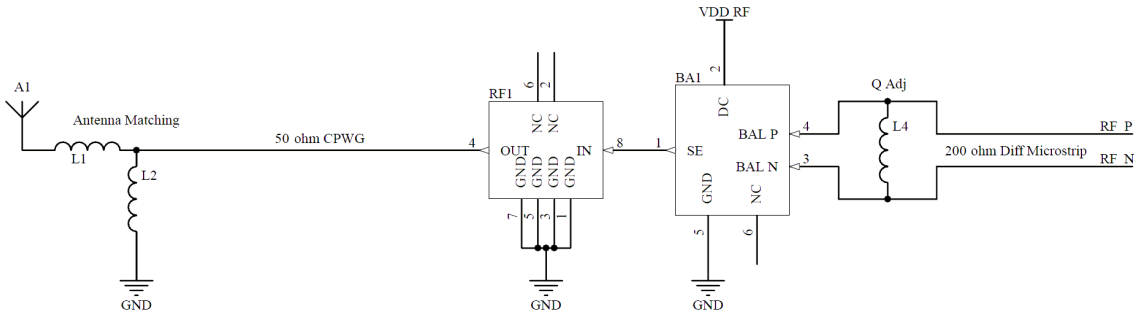


Figure 29: ZigBee Antenna Schematic

In accordance with datasheet recommendations for the antenna [53], the antenna was placed at the edge of the board, adhering to the specified ground-clearance rules.

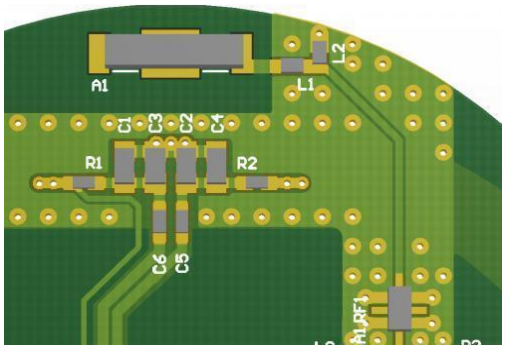


Figure 30: ZigBee Antenna Layout

Lock Connector

EMC filters from Murata were added to the lock-interface connectors [54]. These filters consist of a special type of capacitor with midpoint terminals. Because the I^2C data-lines are incapable of handling large capacitive loads, they use the same type of filters, but with much smaller capacitances, and hence, a higher cut-off frequency. A layout view of the lock-interface connectors, including the EMC filters, is shown in Figure 31 below:

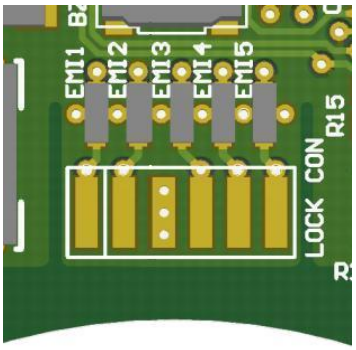


Figure 31: Close-Up View of Lock-Connectors with EMC Filters

Overvoltage transient-protection diodes were also added to the interface on the bottom side of the board.

8.4 Layout of Production-Prototype

The top and bottom layers of the production-prototype board layout are shown in Figure 32 below. The board was drawn in Altium Designer [55], since this is the software tool preferred by Decta.

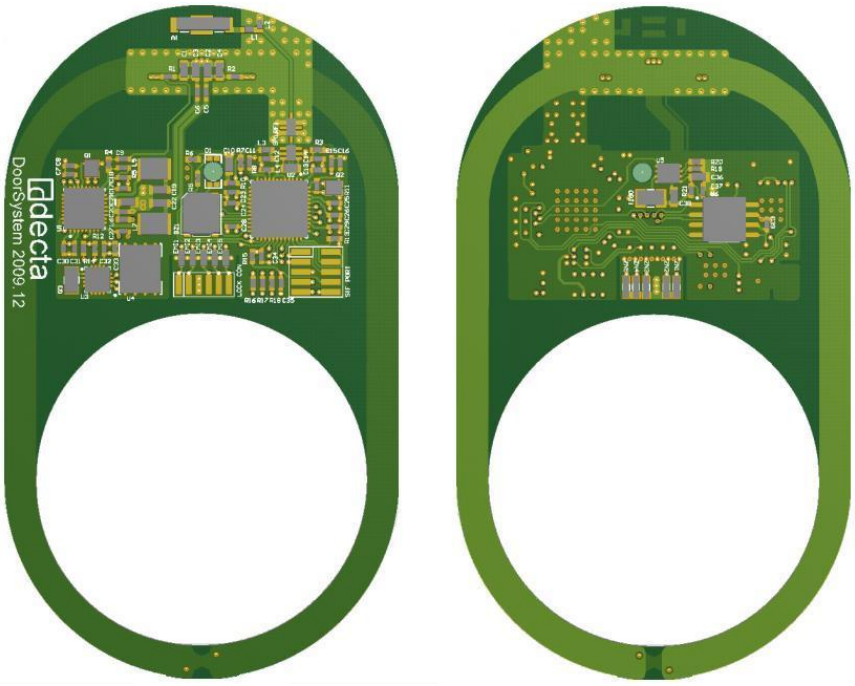


Figure 32: Production-Prototype Top and Bottom-Layer Layout

For a complete schematic, and view of all four layers, consult Appendix B.

8.5 Component Tuning

The antenna matching, the exact oscillating frequency of the crystals, and the pulse-rate of the infrared detector, as well as the impedance of certain sensitive decoupling capacitors, must all be verified empirically by measurements. Some of this work is left for Decta to attend to.

Zigbee Antenna-matching

Since the ZigBee antenna operates in a frequency roughly two-hundred times higher than the ISO14443 MiFare antenna, it cannot easily be matched using only an oscilloscope, but must rather be matched using a Vector Network Analyzer (VNA).

The goal of matching the antenna can be stated as achieving an impedance transformation of the actual antenna impedance, at the frequencies of interest, into that of the feed-line, in this case into 50 Ω. This transformation can be accomplished by the addition of impedances in series, or in parallel with the antenna. The effect of these additions is best illustrated as moving along the circles of constant resistance, or constant conductance, in a Smith chart:

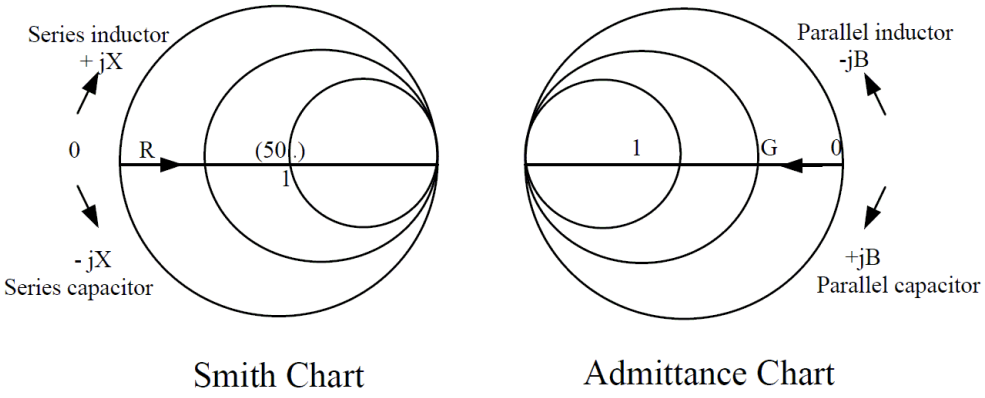


Figure 33: Impedance Matching Paths [59]

In order to match the antenna, an SMA connector, with as short a coaxial cable as practically possible, was soldered directly onto the PCB, and was used to connect an Agilent E5071C VNA, calibrated to display the impedance at the point of the antenna. A size-0402 RF-grade 50 Ω resistor [56], soldered between the antenna feed point and ground, with the antenna disconnected, was used as a 50 Ω reference for the three-step VNA calibration procedure.

The VNA was set to measure over a frequency span of 1–4 GHz, although ZigBee only uses the small 2.405–2.480 GHz subset of this band. The results of measuring the 50 Ω resistor using the calibrated VNA is shown in Figure 34 below:

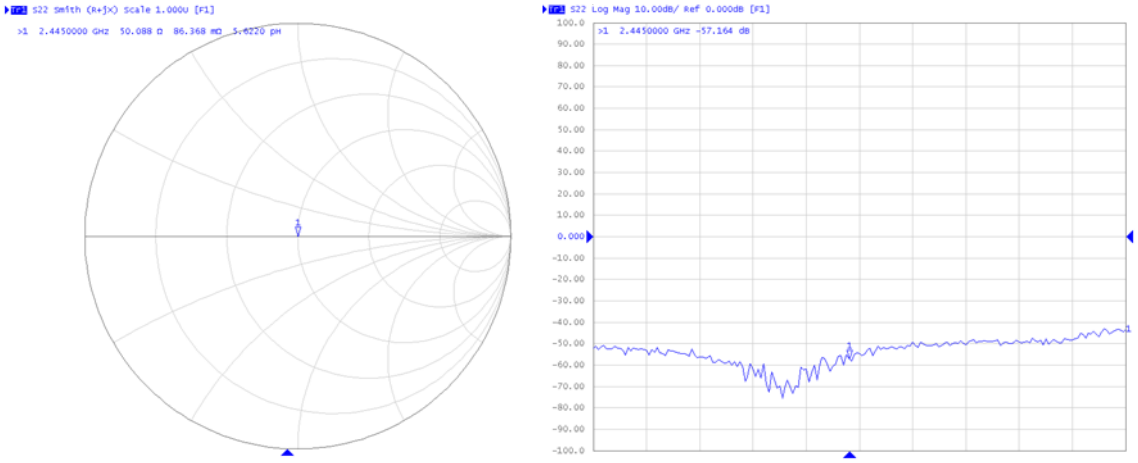


Figure 34: Calibration-Resistor Impedance and Return Loss

As soon as the VNA was calibrated, and the 50 Ω resistor removed, the impedance of the antenna could be measured correctly. As can be seen from Figure 35 below, the antenna, although the manufacturer-specified ground-clearance requirements and layout recommendations were followed, offers only a very poor fit at the target frequency. Furthermore, the recommended matching-component values from the antenna datasheet, did not improve things significantly [53].

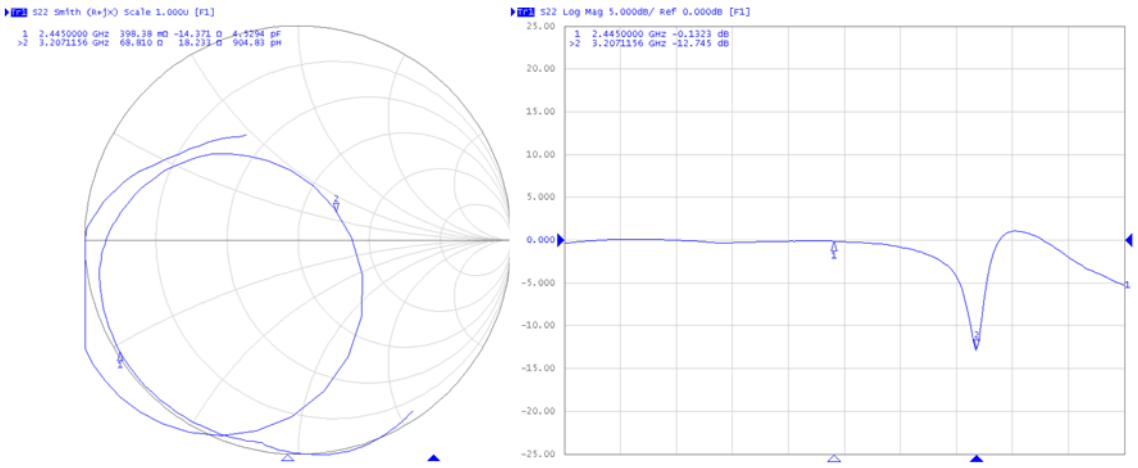


Figure 35: Un-Matched Antenna Impedance and Return Loss

Since the layout was based on datasheet recommendations, footprints were only provided for one shunt and one series element, as can be seen in Figure 29. This restriction limited the amount of available paths in the Smith diagram. This fact, combined with the effects of inherent parasitics in the matching components [57] [58] and the overall antenna circuit, at these high frequencies, meant that some manual tweaking was required.

The best match, across the frequency band used by ZigBee, was found by using a 4.7 nH inductor in series, and a 2.2 pF capacitor shunted to ground. Matching, using these components, resulted in a return loss of -15.2 dB at the target center frequency. The impedance and the return loss of the matched antenna are visualized in Figure 36 below:

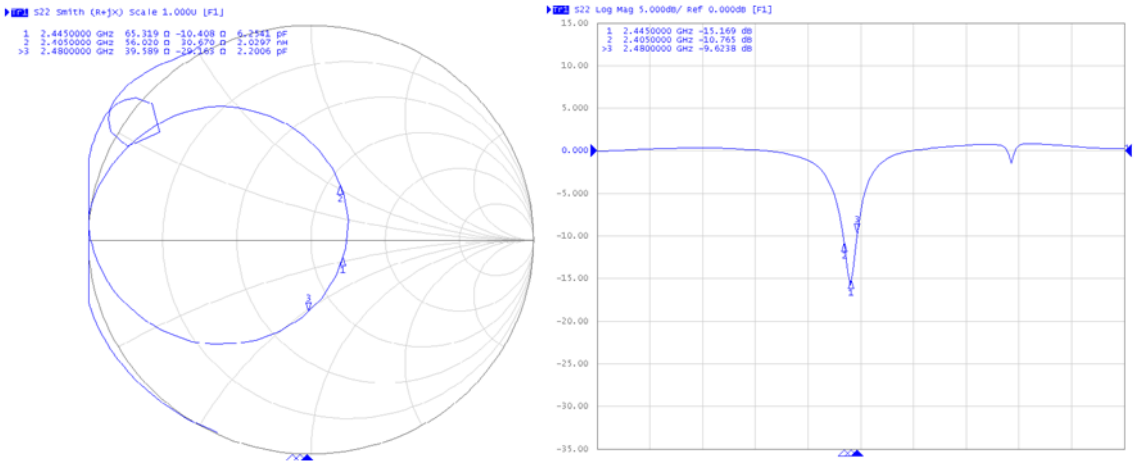


Figure 36: Matched-Antenna Impedance and Return Loss

Oscillator Tuning

There are three oscillators onboard, neither of which can be measured directly, since the oscilloscope probe capacitances would interfere with the load capacitances of the oscillator circuitry, thereby changing the oscillation frequency. Instead, frequency test functionality is implemented, whereby a divided-down frequency output based on the oscillator under test is presented, and can be evaluated with an external frequency counter of high-enough accuracy. Proper verification of oscillator frequencies has not been done at this stage, however.

EMC Verification

EMC compliance is to be verified by an external, certified test-house. Test protocol pending.

8.6 Firmware Modifications

The firmware developed for the demonstration-board required only a few minor changes and additions in order for it to work on the new production-prototype hardware. In addition to these alterations, some of the GPIO sleep-states were tweaked further, in order to provide the lowest overall power dissipation possible.

Real-Time Clock

The lower layer of the clock functions had to be rewritten for the new real-time clock, though no changes at the application layer had to be made because of this.

Proximity Detector

An interrupt handler for the proximity detector was added. If the board contains a proximity-detection device, it is possible to make the board sleep for as long as the specified ZigBee parent-poll interval allows. When the proximity-detector signals an interrupt by having the interrupt pin turn low, the device resumes normal polling-behaviour until the proximity interrupt, once again, returns to a high state.

9. Results

In this chapter, the project results are summarized, discussed, and reflected upon.

9.1 Production-Prototype Findings

Five production prototype PCBs were ordered, three of which were assembled and used for functional verification. One of the assembled prototypes can be viewed in Figure 37 below:

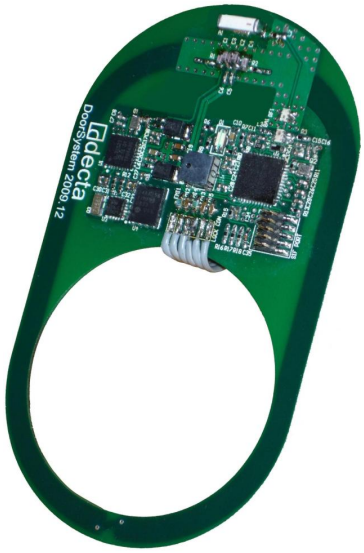


Figure 37: Assembled Production Prototype

The sleep-mode current consumption for the production prototype was only measured for a few configurations, since the peripheral ICs are soldered directly onto the board, and not socketed as on the demonstration-board. When sleeping, the production prototype only consumed around 1 µA, as shown in Table 24 below. The measurement setup was the same as for the demonstration-board.

EM250:	M41T65:	MFRC523:	24AA256:	Si1102 Active:	M45PE20 Sleeping:	Measured Supply Current (µA):
•	•	•	•			1
•	•	•	•	•		4–9
•	•	•	•	•	•	10–15

Table 24: Production-Prototype Current-Measurement Results

Notice that the current figure for the Si1102 is stated for when the device is actively polling for objects within proximity, and not while sleeping.

As for the demonstration-board, the production-prototype MiFare antenna-matching components were trimmed to achieve maximum peak-to-peak voltages across the antenna terminals. Figure 38 below, shows the necessary parallel capacitance values:

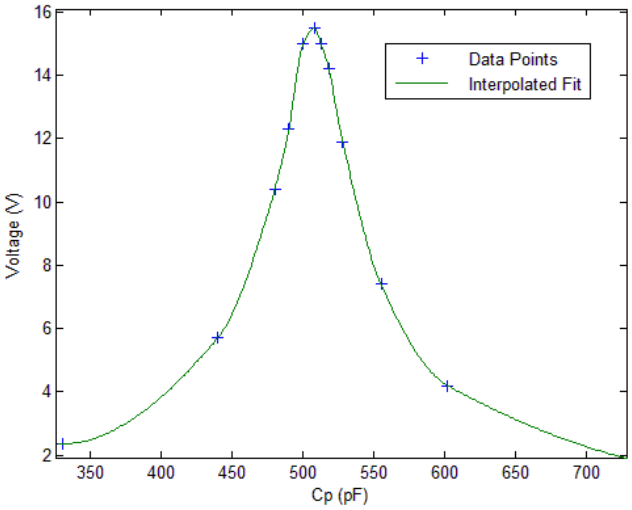


Figure 38: Single-Ended Antenna Voltage versus C_p (Mounted with Lock-Cylinder)

After the card-reader had been trimmed in, the reading-range was also measured for differing supply-voltages, as depicted in Figure 39 below:

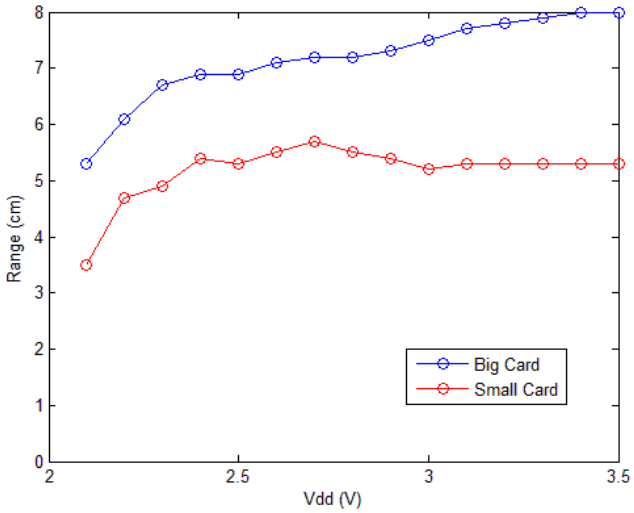


Figure 39: Reading-Range versus Supply-Voltage (Mounted with Lock-Cylinder)

As can be seen from Figure 39, the range for the large card is affected more by a diminishing supply-voltage, than is the range for the small card. It should also be noted that the MFRC523 mounted on the production prototype, remains operational down to 2.3 V, which is lower than the minimum voltage of 2.5 V, as stated by NXP in the datasheet.

Furthermore, the supply current was measured against the supply-voltage, with the transmitter put into a loop condition, continuously transmitting:

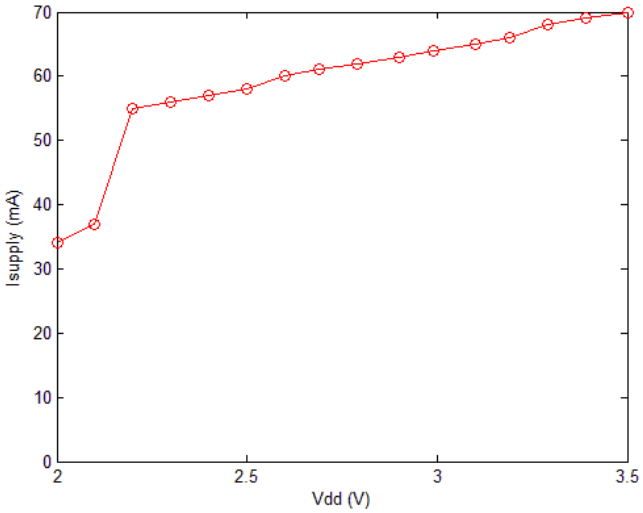


Figure 40: Supply-Current versus Supply-Voltage (Mounted with Lock-Cylinder)

As can be expected, less current is drawn at a lower voltage, though this also has some implications for the range, as seen previously in Figure 39. Beware that the values in Figure 40 include the current drawn by the EM250 while it is operating the MFRC523.

The average current-consumption estimate for the system, using an IR-proximity detector, can be re-calculated with the most significant current-consuming events factored out. The total current consumption for the system can thus be primarily viewed as the superposition of five different current-consuming activities. These activities are: MiFare communication, proximity detection, sleep-currents, ZigBee parent-polling, and house-holding tasks such as sorting the list or updating time schedules etc.

Regarding the ZED only polling its parent once every twenty seconds or so, and using the rough approximation that one complete message poll takes around 10 ms [4], the estimated mean average current for this activity becomes:

$$35\ \text{mA} \cdot 10\ \text{ms} \cdot \frac{1}{20\ \text{s}} = 17.5\ \mu\text{A} \qquad (34)$$

In addition, the current consumed by PCD-PICC transactions, for the given use-case of one-hundred expected card-encounters per day, should be added to the total value. With an average poll current draw calculated to 61 mA, using the values used to generate Figure 40 above, the total average current draw can, assuming a worst-case approximated 100 ms MiFare transaction time, then be estimated as:

$$61\ \text{mA} \cdot 100\ \text{ms} \cdot \frac{100}{1\ \text{day} \cdot 24\frac{\text{h}}{\text{day}} \cdot 60\frac{\text{min}}{\text{h}} \cdot 60\ \frac{\text{s}}{\text{min}}} = 7.1\ \mu\text{A} \qquad (35)$$

Thus, disregarding house-holding tasks, the total average current consumption for the system, using the IR-proximity detector and the standard EEPROM, sums up to around 31 µA.

9.2 Discussion

Using the MFRC523 to poll, in accordance with ISO14443-3 [11], while at the same time meeting the current-budget associated with a two-year operating time, proved to be excessively difficult, even when considering manipulating the poll rate, or reducing the read-out range. However, with the IR detection device detecting users, and the very low current consumption associated with that, it seems that the system is likely to achieve a very long operating-time, exceeding that of the design target of two years.

The quality of the IR-detection process seems viable, though needs to be investigated further, with special concern taken to the effect of sunlight and accumulated dirt on the display window. It should also be noted that the infrared detector needs proper light insulation from the other light sources on the board, especially the infrared emitter itself. This insulation could be achieved by using a small pipe, outwards, towards the casing.

Having metal in close vicinity of the RFID antenna did affect communication slightly, but was determined not to be an impediment, because the system was still able to read out tags directly in front of the lock-cylinder, albeit at a slightly reduced maximum distance, on the order of about one centimeter less.

A few areas of improvement still exist regarding the developed firmware. For instance, the time-schedule coding mechanisms could have been more intricate, allowing for more advanced time-interval descriptions, with for example, repeating intervals and so on, that also better utilizes the designated storage space. There is, however, nothing preventing Decta from either building on the existing solution, or replacing it with an entirely new one.

The way in which storage space was partitioned could have been a little more dynamic, with memory sections growing towards each other until they meet. This concept was considered in the initial planning phase, though the idea was dropped because of its moderate usefulness and high complexity. Another idea that was considered, was to have user-data configurations have their time schedules not directly mapped from the user-data configurations, but mapped to an intermediate step before using pointers to map to the starting points of the time schedules, instead. This approach would have allowed for time schedules of arbitrary maximum length, though this idea was also dropped as it became apparent that the defragmentation procedure and extra work associated with such a routine simply did not justify its implementation. If time schedules prove to be too short using the current implementation, the recommendation is instead to either rewrite or extend the time-interval coding mechanisms, or use a memory with a larger page size (such as the M45PE20), as this actually sets the limit for the time-schedule size.

Even though the ratio between the different memory areas can be adjusted by issuing ZigBee commands over the air, the total available memory space may in some cases not be sufficient to handle the desired number of users, log entries and time schedules. In

such cases larger memories may be used instead. If the memory is larger than 1 Mibit, remote boot-loading capability may also be implemented.

The hardware was designed with offline capability in mind. However, other conceivable operating modes, like for example, full online operation, where the system goes online and queries for information whenever it needs to, might also be employed, in which case other hardware configurations may be used. For instance, the real-time clock and the on-board memory might be removed, reducing the hardware cost somewhat. These changes are also likely, however, to cause a different average current consumption, which, along with its implications, would then of course also need to be examined.

The use of hardware abstraction layers in software, makes it easy to implement page-wise encryption at memory-level without affecting any of the higher-level functionality. However, this would only improve security slightly, since a correct application of the card-authentication procedure, using hashed UIDs, makes the system reject unauthorized cards anyway, even if present on memory.

A vulnerability that is left unattended for, is the exposed, un-authenticated, communication over the soft-I^2C bus connecting the outside board with the lock-controller PCB on the inside of the door. This feature, however, was deliberately left for Decta to resolve, because the feature remains unimplemented on the controller-card itself.

When available, firmware may be extended to support the MiFare Plus Standard, if desired. However, if the employed hash-authentication algorithm is sufficiently secure, no greater security gain is achieved by using MiFare Plus.

9.3 Conclusions

Because the current consumption using the MFRC523 to detect cards, proved to exceed the current-budget, a decision was made to implement an alternative infrared detection scheme. The resulting estimated device operating time, from using two Duracell MX1500 AA-cells, is expected to be well in excess of the battery shelf-life. In cases where an IR-detection device is undesirable, the operating time may instead be extended by increasing the total battery capacity.

Using the default configuration with the standard EEPROM, the memory is sufficient for holding 3010 user-data configurations, 59 unique time schedules, and 100 log entries, though the internal ratio between these numbers may be changed, if necessary.

Optional Flash memory can be used to increase storage space, but system administrators must then consider the consequences from the increased minimum supply-voltage level and the consequent reduction in system operating time.

The list-sorting algorithm was implemented to reduce the amount of sorting needed, in order to improve battery life; to decrease search times, and to ensure system accessibility at all times without noticeable latencies, even in such cases where lists containing thousands of user-numbers are being managed.

In cases where an excessive amount of ZigBee end-nodes are communicating with one gateway, source-routing might need to be implemented. This issue has not been

considered in this work, but support for this is available in the ZigBee-Pro stack that was used in this project.

Having a chromed-brass lock-cylinder inside of the RFID antenna-loop, did affect the range slightly, though not prohibitively so. The performance of the developed solution still proved to be fully adequate.

Although functionality has been verified at both block and system-level, it is also recommended that verification is carried out in a live trial-installation, before marketing of the complete system.

Finally, because of its small size, low cost and the combination of attractive main features, it seems reasonable to assume that this product offers an edge on the otherwise highly competitive market for door-entry systems.

9.4 Reflections

The partitioning of the firmware into abstraction layers turned out to be useful, since it greatly simplified the code changes needed when replacing the real-time clock. It also facilitated quickly switching between different types of memories, by hiding hardware-specific properties.

On a memory-constrained platform such as the EM250, it would have been beneficial to encapsulate the memory buffers used to store temporary data, with control functions, so that a buffer is not usable until it has been flagged as released and ready-for-use again by other parts of the firmware. This approach is likely to prevent errors, such as accidental writes, while at the same time simplifying the verification procedure, as those kinds of errors can then be ruled out.

The matching of the ZigBee antenna could have been simplified with the addition of a third component, enabling the shunt component to be placed either before, or after the series element, as this would have given a higher degree of freedom in choosing suitable matching components.

Finally, the development-time of the firmware was initially underestimated. In total, more than eight-thousand lines of C-code were written, excluding pre-made Ember stack code.

References

[1] NXP, "MFRC532 Contactless Reader IC," *NXP web page*. [Online]. Available: http://www.nxp.com/documents/data_sheet/MFRC523.pdf. [Accessed: Mar. 20, 2010].

[2] Ember Corporation, "EM250 Single-Chip ZigBee/802.15.4 Solution," *Ember Corporation web page*. [Online]. Available: http://www.ember.com/pdf/120-0082-000_EM250_Datasheet.pdf. [Accessed: Mar. 20, 2010].

[3] ZigBee Alliance. (2010, Aug.) ZigBee Alliance web page. [Online]. http://www.zigbee.org/About/OurMission.aspx

[4] Drew Gislason, *ZigBee Wireless Networking*. Burlington, Massachussets, United States of America: Newnes, 2008.

[5] IEEE. (2010, May) IEEE 802.15 WPAN Task Group 4 (TG4). [Online]. http://www.ieee802.org/15/pub/TG4.html

[6] ZigBee Alliance, "ZigBee Specification," *ZigBee Alliance web page*. [Online]. Available: http://www.zigbee.org/Products/DownloadZigBeeTechnicalDocuments.aspx. [Accessed: May 09, 2010].

[7] NIST, "ADVANCED ENCRYPTION STANDARD (AES)," *NIST web page*. [Online]. Available: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf. [Accessed: May 09, 2010].

[8] NXP Semiconductors. (2010, Apr.) MIFARE.net. [Online]. http://mifare.net/

[9] International Standards for Business, Government and Society, "Identification cards -- Contactless integrated circuit cards -- Proximity cards -- Part 1: Physical characteristics," *ISO/IEC 14443-1:2008*. [Online]. Available: http://www.iso.org/iso/catalogue_detail.htm?csnumber=39693. [Accessed: Apr. 20, 2010].

[10] International Standards for Business, Government and Society, "Identification cards -- Contactless integrated circuit(s) cards -- Proximity cards -- Part 2: Radio frequency power and signal interface," *ISO/IEC 14443-2:2001*. [Online]. Available: http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=28729. [Accessed: Apr. 20, 2010].

[11] International Standards for Business, Government and Society, "Identification cards -- Contactless integrated circuit(s) cards -- Proximity cards -- Part 3: Initialization and anticollision," *ISO/IEC 14443-3:2001*. [Online]. Available: http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=28730. [Accessed: Apr. 20, 2010].

[12] International Standards for Business, Government and Society, "Identification cards -- Contactless integrated circuit cards -- Proximity cards -- Part 4: Transmission protocol," *ISO/IEC 14443-4:2008*. [Online]. Available: http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=50648. [Accessed: Apr. 20, 2010].

[13] Philips. (2000, May) MIFARE Design of MFRC500 Matching Circuit and Antennas. Document.

[14] G. De Koning Gans, J. H. Hoepman, and F. D. Garcia, "A Practical Attack on the MIFARE Classic," *Lecture Notes in Computer Science*, vol. 5189, pp. 267-282, 2008.

[15] F. D. Garcia, P. Van Rossum, R. Verdult, and R. Wichers Schreur, "Wirelessly Pickpocketing a Mifare Classic Card," in *30th IEEE Symposium on Security and Privacy*, Oakland, 2009, pp. 3-15.

[16] F. D. Garcia et al., "Dismantling MIFARE Classic," *Lecture Notes in Computer Science*, vol. 5283, pp. 97-114, 2008.

[17] NXP Semiconductors. (2010, Apr.) MIFARE.net. [Online]. http://mifare.net/security/mifare_classic.asp

[18] NXP, "NXP MIFARE PLUS - benchmark security for mainstream applications," *MIFARE.net*. [Online]. Available: http://mifare.net/downloads/NXP_Mifare_Plus_leaflet.pdf. [Accessed: Apr. 20, 2010].

[19] Warren L Stutzman and Gary A Thiele, *Antenna Theory and Design*. United States of America: John Wiley & Sons, Inc, 1981.

[20] Klaus Finkenzeller, *RFID handbook: fundamentals and Applications in Contactless Smart Cards and Identification*, Second edition ed. Munich, Germany: Carl Hanser Verlag, 2003.

[21] W. Aerts, E. De Mulder, B. Preneel, G. A.E. Vandenbosch, and I. Verbauwhede, "Dependence of RFID Reader Antenna Design on Read Out Distance," *IEEE Transactions on Antennas and Propagation*, vol. 56, no. 12, pp. 3829-3837, Dec. 2008.

[22] Microchip, "AN678: RFID Coil Design," *Microchip web page*. [Online]. Available: http://ww1.microchip.com/downloads/en/appnotes/00678b.pdf. [Accessed: Apr. 20, 2010].

[23] Microchip, "AN710: Antenna Circuit Design for RFID Applications," *Microchip web page*. [Online]. Available: http://ww1.microchip.com/downloads/en/AppNotes/00710c.pdf. [Accessed: Apr. 20, 2010].

[24] Texas Instruments, "HF Antenna Design Notes Technical Application Report," *Texas Instrument web page*. [Online]. Available: http://www.ti.com/rfid/docs/manuals/appNotes/HFAntennaDesignNotes.pdf. [Accessed: Apr. 21, 2010].

[25] M. Gebhart, M. Wienand, J. Bruckbauer, and S. Birnstingl, "Automatic Analysis of 13.56 MHz Reader Command Modulation Pulses," University of Applied Sciences Hagenberg, Austria,.

[26] Gilles Cerede, "Understanding the antenna design Challenge," *rfdesign.com*, vol. EMERGING WIRELESS TECHNOLOGY –A Supplement to RF Design, pp. 10-13, Sep. 2006.

[27] NXP Semiconductors, "UM10204: I2C bus specification and user manual," *NXP web page*. [Online]. Available: http://ics.nxp.com/support/documents/interface/pdf/i2c.bus.specification.pdf. [Accessed: Apr. 21, 2010].

[28] Duracell, "MX1500 Alkaline-Manganese Dioxide Battery," *Duracell web page*. [Online]. Available: http://www1.duracell.com/Procell/pdf/MX1500_US_UL.pdf. [Accessed: Mar. 20, 2010].

[29] S. Castillo, N. K. Samala, K. Manwaring, B. Izadi, and D. Radhakrishnan, "Experimental Analysis of Batteries under Continuous and Intermittent Operations," in *Proceedings of the International Conference on Embedded Systems and Applications*, Las Vegas, 2004, pp. 18-24.

[30] Texas Instruments, "TPS61097: LOW INPUT VOLTAGE SYNCHRONOUS BOOST CONVERTER," *Texas Instruments web page*. [Online]. Available: http://focus.ti.com/lit/ds/symlink/tps61097-33.pdf. [Accessed: Apr. 21, 2010].

[31] NXP Semiconductors, "MF0ICU1: Functional specification," *NXP web page*. [Online]. Available: http://www.nxp.com/acrobat./M028634_MF0ICU1_Functional_Spec_V3.4.pdf. [Accessed: Apr. 21, 2010].

[32] Telegesis Ltd, "ETRX2 and ETRX2HR ZIGBEE® MODULES PRODUCT MANUAL," *Telegesis web page*. [Online]. Available: http://www.telegesis.com/downloads/general/TG-ETRX2-PM-001-109.pdf. [Accessed: Aug. 04, 2010].

[33] Microchip, "256K I2C CMOS Serial EEPROM," *Microchip web page*. [Online]. Available: http://ww1.microchip.com/downloads/en/devicedoc/21203M.pdf. [Accessed: Apr. 20, 2010].

[34] Seiko, "S-35390A 2-WIRE REAL-TIME CLOCK," *Seiko web page*. [Online]. Available: http://datasheet.sii-ic.com/en/real_time_clock/S35390A_E.pdf. [Accessed: Apr. 20, 2010].

[35] Numonyx, "M25PE20 M25PE10: 1 and 2 Mbit, page-erasable serial Flash memories," *Numonyx web page*. [Online]. Available: http://www.numonyx.com/Documents/Datasheets/M25PE20_10.pdf. [Accessed: Apr. 21, 2010].

[36] CadSoft Computer GmbH. (2010, Aug.) CadSoft online web page. [Online]. http://www.cadsoft.de/

[37] Kyoritsu, "KM Series datasheet," *Kyoritsu web page*. [Online]. Available: http://www.kew-ltd.co.jp/en/index.html. [Accessed: May 09, 2010].

[38] SILICON LABS, "Si1102 Optical Proximity Detector," *SILICON LABS web page*. [Online]. Available: http://www.silabs.com/pages/DownloadDoc.aspx?FILEURL=Support%20Documents/TechnicalDocs/Si1102.pdf. [Accessed: Mar. 20, 2010].

[39] Telegesis Ltd, "EAP-E and EAP-E-PA Ethernet Access Point PRODUCT MANUAL," *Telegesis web page*. [Online]. Available: http://www.telegesis.com/downloads/general/TG-EAPE-PM-006-104.pdf. [Accessed: Aug. 04, 2010].

[40] MathWorks Inc. (2010, Aug.) MathWorks web page. [Online]. http://www.mathworks.com/products/matlab/

[41] Simon Tatham. (2010, Aug.) PuTTY web page. [Online]. http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html

[42] Br@y++. (2010, Aug.) Bray Terminal Google web page. [Online]. http://sites.google.com/site/terminalbpp/

[43] Lantronix Inc. (2010, Aug.) Lantronix web page. [Online]. http://www.lantronix.com/

[44] C.A.R. Hoare, "Quicksort," *The Computer Journal*, vol. 5, no. 1, pp. 10-16, 1962.

[45] Cormen H. Thomas, Charles E. Leiserson, and Rivest L. Ronald, *Introduction to Algorithms*, 1st ed. Cambridge, Massachusetts, United States of America: MIT Press, 2009.

[46] NXP, "AN 073120 mifare Ultralight Features and Hints," *NXP web page*. [Online]. Available: http://www.nxp.com/documents/application_note/M073120.pdf. [Accessed: Apr. 20, 2010].

[47] Telegesis Ltd, "TG-ETRXn-R302-AT-Commands," *Telegesis web page*. [Online]. Available: http://www.telegesis.com/downloads/general/TG-ETRXn-R302-Commands.pdf. [Accessed: May 09, 2010].

[48] STMicroelectronics, "M41T62 M41T63 M41T64 M41T65: Serial real-time clock with alarms," *STMicroelectronics web page*. [Online]. Available: http://www.st.com/stonline/products/literature/ds/10397/m41t64.pdf. [Accessed: Apr. 20, 2010].

[49] Numonyx, "M45PE20: 2-Mbit, page-erasable serial flash memory with byte alterability and a 75 MHz SPI bus interface," *Numonyx web page*. [Online]. Available: http://www.numonyx.com/Documents/Datasheets/M45PE20.pdf. [Accessed: Apr. 20, 2010].

[50] SILICON LABS, "AN442: Si1102 AND Si1120 DESIGNER'S GUIDE," *SILICON LABS web page*. [Online]. Available: http://www.silabs.com/pages/DownloadDoc.aspx?FILEURL=Support%20Documents/TechnicalDocs/AN442.pdf. [Accessed: Mar. 19, 2010].

[51] Ember Corporation, "PCB Design with an EM250," *Ember Corporation web page*. [Online]. Available: http://www.ember.com/pdf/120-5026-000_Designing_with_an_EM250.pdf. [Accessed: Mar. 20, 2010].

[52] Ember Corporation, "EM250 Reference Design," *Ember Corporation's web page*. [Online]. Available: http://www.ember.com/zip/EM250_REF_DES_CER.zip. [Accessed: Apr. 21, 2010].

[53] JOHANSON TECHNOLOGY, "2450 Mhz Antenna P/N 2450AT43B100," *Johanson technology web page*. [Online]. Available: http://cht.johansontechnology.com/products/rfc/ant/JTI_Antenna-2450AT43B100_2006-05.pdf. [Accessed: Mar. 20, 2010].

[54] Murata, "SMD/BLOCK Type EMI Suppression filters EMIFIL," *Murata web page*. [Online]. Available: http://www.murata.com/products/catalog/pdf/c31e.pdf. [Accessed: Apr. 21, 2010].

[55] Altium Limited. (2010, Aug.) Altium web page. [Online]. http://www.altium.com/products/altium-designer/

[56] Vishay Intertechnology, Inc., "High Frequency (up to 20 GHz) Resistor, Thin Film Surface Mount Chip," *Vishay web page*. [Online]. Available: http://www.vishay.com/docs/60093/fcseries.pdf. [Accessed: Apr. 20, 2010].

[57] JOHANSON TECHNOLOGY, "MULTI-LAYER HIGH-Q CAPACITORS," *JOHANSON TECHNOLOGY web page*. [Online]. Available: http://www.johansontechnology.com/images/stories/catalog/JTI_MLCC_HighQ_2010_17.pdf. [Accessed: Aug. 25, 2010].

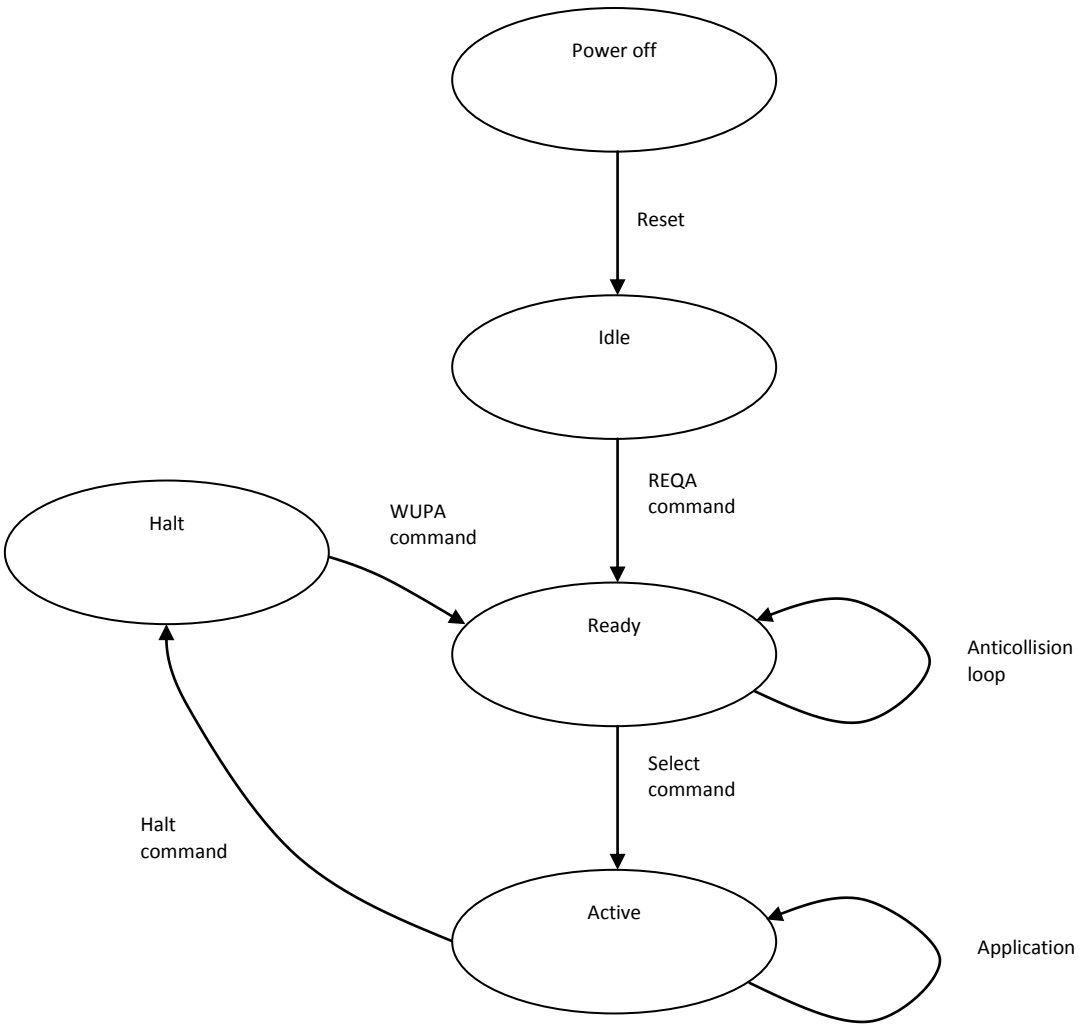
[58] JOHANSON TECHNOLOGY, "RF CERAMIC CHIP INDUCTORS," *JOHANSON TECHNOLOGY web page*. [Online]. Available: http://www.johansontechnology.com/images/stories/rf-inductors/cci/JTI_Inductors_Chip_2010-02.pdf. [Accessed: Aug. 25, 2010].

[59] RF Monolithics, Inc., "ASH Transceiver Antenna Impedance Matching," *RFM web page*. [Online]. Available: http://www.rfm.com/products/apnotes/antennamatch.pdf. [Accessed: Apr. 20, 2010].

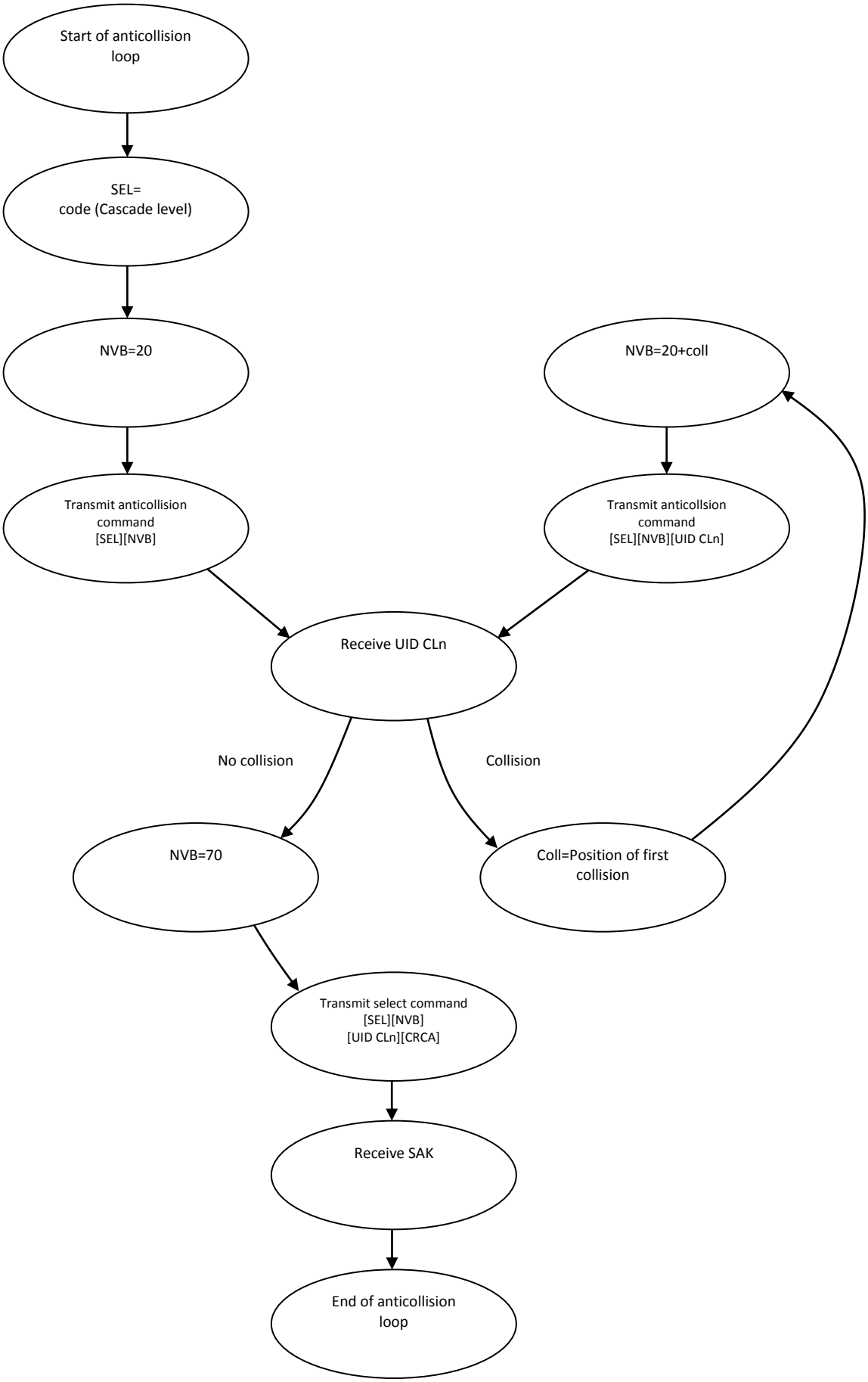
[60] Ember Corporation. (2010, Mar.) Ember Corporation web page. [Online]. http://www.ember.com/products_zigbee_development_tools_kits.html

Appendix A: System Documentation

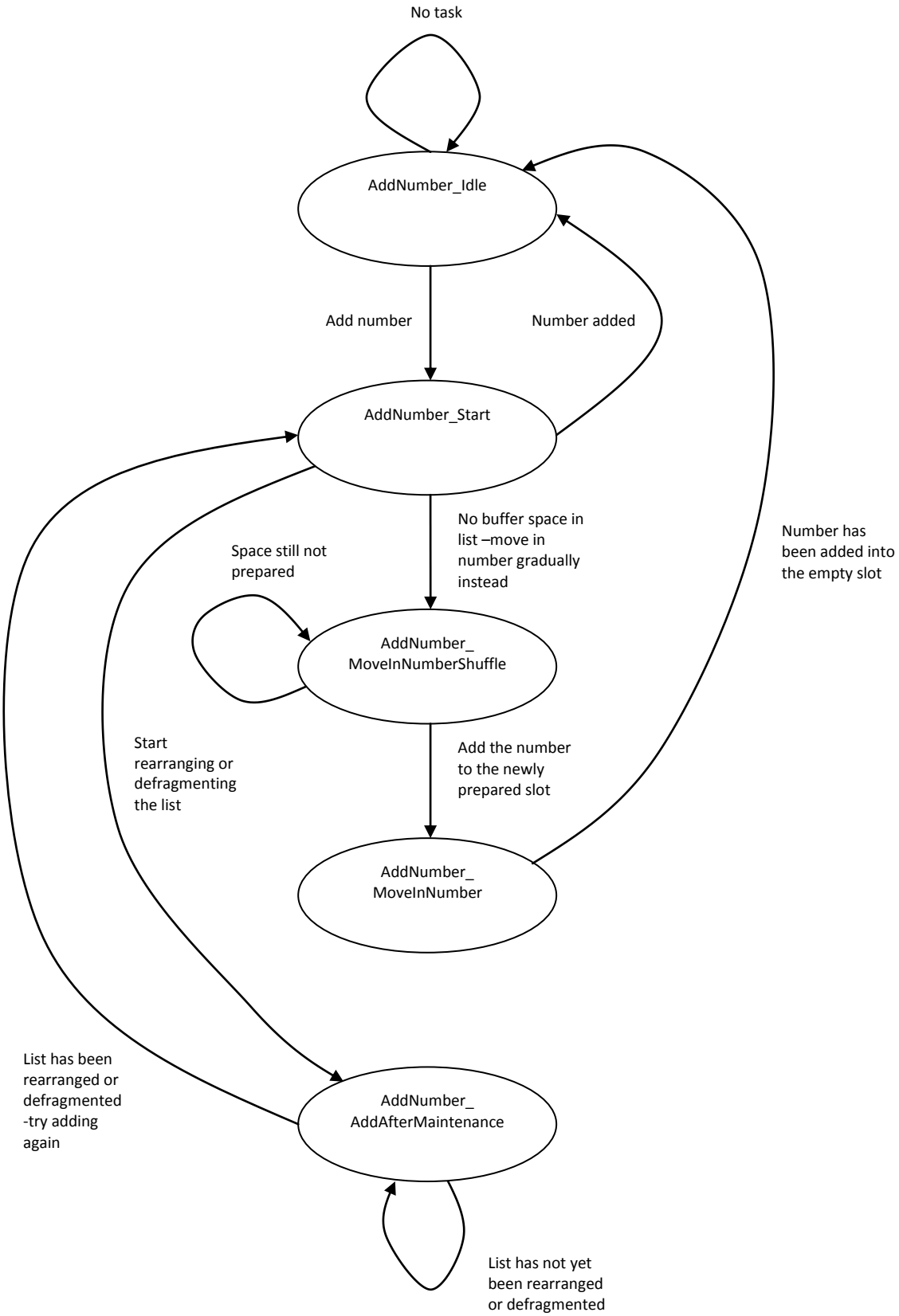
PICC Type A, State Diagram [11]



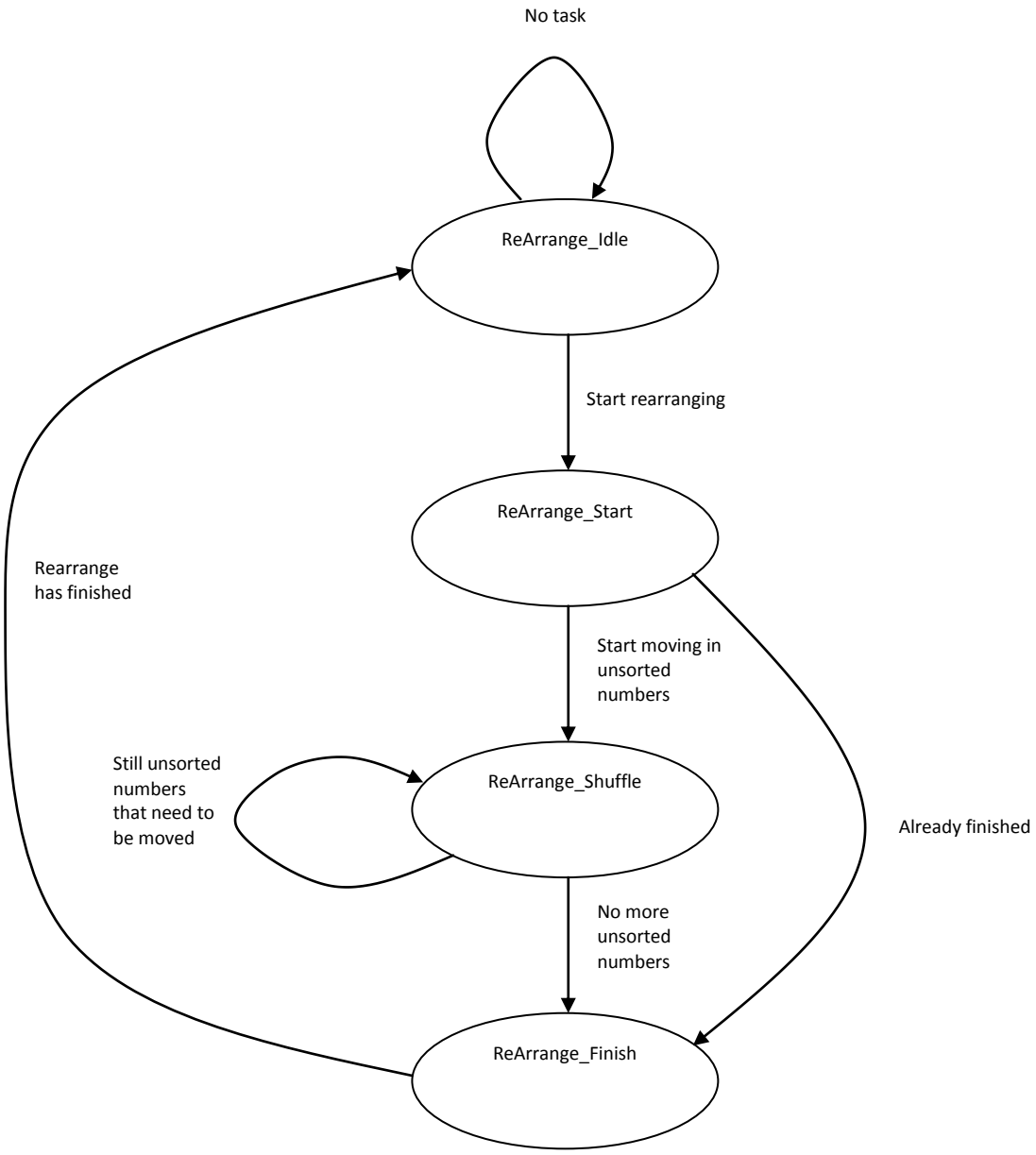
Anti-Collision Loop, Flow-Chart for PCD [11]



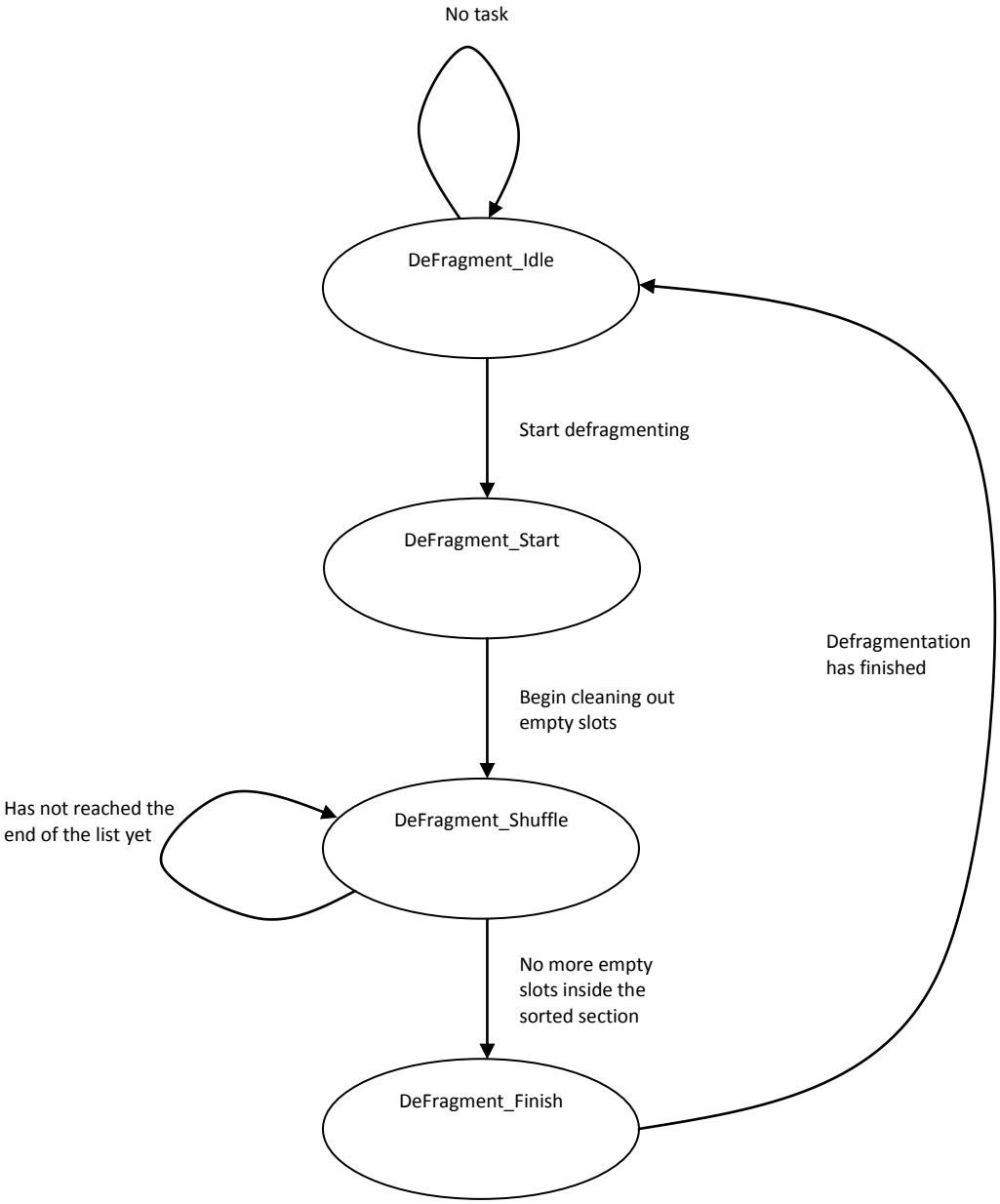
Add-Number State Machine



Rearrange State Machine



Defragmentation State Machine



List of Firmware Source Files

decta-Defines.h
Contains general definitions applicable to the entire firmware.

decta-demoboard.h
Contains system state configurations for the demonstration-board.

decta-DoorSystemConfiguration.h
Contains general ZigBee stack settings.

decta-DoorSystemMain.c
Contains the main program loop.

decta-hal-i2c.c
Implementations for the corresponding header file.

decta-hal-i2c.h
Contains low-level hardware abstractions for the I^2C bus.

decta-hal-spi.c
Implementations for the corresponding header file.

decta-hal-spi.h
Contains low-level–hardware abstractions for the SPI interface.

decta-LockDriver.c
Implementations for the corresponding header file.

decta-LockDriver.h
Contains functions required to manage the doorlock.

decta-NumberManagement.c
Implementations for the corresponding header file.

decta-NumberManagement.h
Contains functions necessary in order to manage and sort data on non-volatile memory.

decta-prodboard.h
Contains system state configurations for the production prototype board.

decta-RfidDriver.c
Implementations for the corresponding header file.

decta-RfidDriver.h
Contains functions to manage and operate contactless cards.

decta-soft-i2c.c
Implementations for the corresponding header file.

decta-soft-i2c.h

Contains functions necessary for emulating an I^2C bus interface using GPIOs.

decta-StorageDriver.c
Implementations for the corresponding header file.

decta-StorageDriver.h
Contains functions for abstracting low-level memory management.

decta-TimeDriver.c
Implementations for the corresponding header file.

decta-TimeDriver.h
Contains functions for abstracting real-time clock management.

Firmware Header Files

This section contains the contents of the firmware header files. Please note however, that the corresponding implementations are classified and can be found in (the secret) Appendix B.

decta-Defines.h

```
//*****************************************************************************
// File: decta-Defines.h
//
// Description:
// Contains general defines applicable to the entire firmware
//
// Authors: Mikael Högrud and Johan Riisberg-Jensen (2010)
//*****************************************************************************



//*****************************************************************************
// General defines:
//*****************************************************************************

// Success or failure definitions for Dectas commands:
#define DECTA_SUCCESS   0x00
#define DECTA_FAILURE   0xff

// The serial ports used for debugging:
#define APP_SERIAL   1
#define DEBUG_SERIAL 0



//*****************************************************************************
// Door lock status-byte defines:
//*****************************************************************************

// Door status byte bitmasks:
#define LATCH_INTERRUPT        0x80
#define LATCH_MOVE_DIRECTION   0x40
#define KEY                    0x20
#define DEADLOCK               0x10
#define BOLT                   0x08
#define LAST_ACTION_BITS       0x07



//*****************************************************************************
// Default Door open time
//*****************************************************************************

// Standard open door interval in seconds:
#define STD_OPEN_TIME     5



//*****************************************************************************
//  I2C Defines
//*****************************************************************************

#define Write          0xFE
#define Read           0x01



//*****************************************************************************
// Default MiFare Classic key:
//*****************************************************************************

#define DEFAULT_MIFARE_CLASSIC_KEY    {0xff, 0xff, 0xff, 0xff, 0xff, 0xff}



//*****************************************************************************
```

```
// ZigBee Network defines:
//*****************************************************************************

// Extended PAN ID unique for Decta (if they purchased an OUI):
// Format:                      [Decta OUI]      , [intra-OUI PAN ID identifier]
#define DECTA_EXTENDED_PAN_ID  {0x3a, 0x42, 0xfd, 0x03, 0x04, 0x05, 0x06, 0x07}

// (0x07fff800 <-> {0000 0111 1111 1111 1111 1000 0000 0000} <-> 11-26)
#define CHANNEL_MASK      0x07fff800

// Interval (in milliseconds) between a sleepy end device's polls for data at
// the router. Note: if this is set for too long and there is no router that
// can store the incoming packets in the meantime, then the packets may get
// dropped! 20000 should be the default value when using a router.
// (Note: ZigBee Pro recommends a value no greater than 7500, but this should
// be OK for this application because we send our own command acks and nacks)
#define STANDARD_PARENT_POLL_INTERVAL            20000

// Standard sleep duration in quarter seconds:
#define STANDARD_SLEEP_DURATION                  3

// Just some delay in milliseconds when signalling open or close:
#define SIGNAL_DELAY_TIME                        1000

// Number of extra fast polls after a MESSAGE_FOLLOWS command:
#define MESSAGE_FOLLOWS_NUMBER_OF_FAST_POLLS     5

// Profile id for a manufacturer specific profile:
// (0xbf00-0xffff are manufacturer specific profiles!)
#define DECTA_DOORSYSTEM_PROFILE_ID              0xbf00



//*****************************************************************************
//Endpoints etc:
//*****************************************************************************

// Numeric index for the first endpoint. Applications can use any
// number of endpoints between 1 and 238. Endpoint 0 is used by the
// ZDO, endpoint 239 is used by the SPDO, and endpoint 240 is used by
// the EDO. This application uses only one endpoint. The constant
// makes the code easier to read.
// (Data received on endpoint 2 is printed by the ETRX2 Telegesis
// Ethernet bridge)
#define DECTA_DOORSYSTEM_ENDPOINT                  2

// Total number of endpoints:
#define DECTA_DOORSYSTEM_NUMBER_OF_ENDPOINTS       1

//Define clusters supported by the endpoints:
#define DECTA_DOORSYSTEM_CLUSTER_ID                1

// Number of input and output clusters:
// (Note: Binding is currently not used)
#define DECTA_DOORSYSTEM_INPUT_CLUSTER_COUNT       1
#define DECTA_DOORSYSTEM_OUTPUT_CLUSTER_COUNT      1

// Identifies the doorsystem application from other possible endpoints:
#define DECTA_DOORSYSTEM_DEVICE_ID                 1

// Device version:
#define DECTA_DOORSYSTEM_DEVICE_VERSION            1

// Initial NWK and LINK key settings:
#define DECTA_DOORSYSTEM_DEFAULT_NWK_KEY   {5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5}
#define DECTA_DOORSYSTEM_DEFAULT_LINK_KEY  {7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7}



//*****************************************************************************
// Command byte defines:
//*****************************************************************************

// Byte commands used for instructing the sleepy end door node to perform
// various activities.
// Commmands come in the form of [Command], [Data], [Next command], [Data] and
// so on...
// Note: A sent message must start with a byte describing how long
```

```
// the total message is.

// Command byte that makes formatting of memory possible. See comment below.
#define FORMAT_ENABLE                   128

// Formats the entire non-volatile memory and sets the default page values.
// Note: This command requires that the FORMAT_ENABLE byte has been sent
// immediately prior to this byte.
#define FORMAT_MEMORY                   127

// Sets the number of pages used for various for numbers and logs etc. This
// command must be preceeded by a FORMAT_ENABLE command.
#define FORMAT_AND_SET_MEMORY_AREAS     126

// Gets the number of pages used for various for numbers and logs etc.
#define GET_MEMORY_ALLOCATION           125

// Adds a user number of size userNumberSize bytes.
#define ADD_NUMBER                      1

// Searches for and removes a user number of size userNumberSize bytes.
#define REMOVE_NUMBER                   2

// Updates a timeschedule slot, which has place for timeScheduleSize number
// of bytes.
#define UPDATE_TIMESCHEDULE             3

// Erases a timeschedule, making it inactive.
#define REMOVE_TIMESCHEDULE             4

// Sends back all the available logs back to the coordinator.
#define RETRIEVE_LOGS                   5

// Formats the log pages.
#define ERASE_LOGS                      6

// Provides some information about the number of cards in the system etc.
// Not normally used.
#define RETRIEVE_INFO_PAGE_DATA         124

// Set this door node's group id.
#define SET_GROUP_ID                    123

// Gets this door node's group id.
#defineGET_GROUP_ID                     122

// Sets the real time clock.
#define SET_REAL_TIME_CLOCK             121

// Gets the latest real time clock data.
#define GET_REAL_TIME_CLOCK             120

// Keeps the system fast polling its parent a
// MESSAGE_FOLLOWS_NUMBER_OF_FAST_POLLS extra times. If no new message is
// received. If this command is omitted, then the system goes to sleep
// immediately after reception of a message.
#define MESSAGE_FOLLOWS                 119

// Returns the present value of VDD_PADS, VDD_CORE and VDD_LOCK
#define GET_BATTERY_VOLTAGES            118

// Sets the Mifare Classic Key
#define SET_MIFARE_CLASSIC_KEY          117

// Unlocks door for a specified number of secounds. 0xff means always open.
#define UNLOCK_DOOR                     116

// Locks door
#define LOCK_DOOR                       115

// Plays a tune on a remote door node
#define IDENTIFY                        114


//*****************************************************************************
// The following codes are returned to the coordinator upon reception or
// failure of commands and are preceeded by the sender's message sequence
// number:
```

```
//*****************************************************************************

// The sleepy door node responds with this number when it has received and
// successfully interpreted a message of commands and executed them.
#define COMMAND_ACK                     65

// The sleepy door node responds with this number if it has problems
// interpreting a received message.
#define COMMAND_NACK                    78



//*****************************************************************************
// Number management timeschedule interpretation defines:
//*****************************************************************************

// Time codes must be in the form of:
// [Code][Number of intervals][Time specification]
// and finish with an EMPTY code if not the last byte.
// Example: Monday-Thursday between 8-9 am and 3-5 pm, or the entire year of
// 2009, would be written as:
// [DAY][01][00][03][TIME][02][08][00][09][00][15][00][17][00][OR]
// [YEAR][1][09][09][EMPTY]

#define EMPTY        255    // Code for: Time code string has finished
#define OR           254    // Code for: Further coming interval
#define YEAR         253    // Time code for: Year (00-99)
#define MONTH        252    // Time code for: Month (01-12)
#define DATE         251    // Time code for: Date (01-31)
#define DAY          249    // Time code for: Day (00-06)
#define TIME         248    // Time code for: Time (00-23, 00-59)
```

decta-demoboard.h

```
//*****************************************************************************
// File: decta-demoboard.h
//
// Description:
// Contains system state configurations for the demonstration board.
//
// Authors: Mikael Högrud and Johan Riisberg-Jensen (2010), based on
// code from Ember Corporation
//*****************************************************************************



//*****************************************************************************
// Board configuration
//*****************************************************************************

#ifndef __BOARD_H__
#define __BOARD_H__
#define EMBER_SERIAL_BAUD_CUSTOM  13



//*****************************************************************************
//  GPIO Configuration Definitions
//*****************************************************************************
/*
 * The following are used to specify the GPIO configuration to establish
 * when Powered (POWERUP_), and when Deep Sleeping (POWERDN_).  The reason
 * for separate Deep Sleep settings is to allow for a slightly different
 * configuration that minimizes power consumption during Deep Sleep.  For
 * example, inputs that might float could be pulled up or down, and output
 * states chosen with care, e.g. to turn off LEDs or other devices that might
 * consume power or be unnecessary when Deep Sleeping.
 *
 * A convenience macro to convert GPIO# into bit mask for the
 * GPIO registers.
 */
#define GPIO(n) (1u << ((n) & 0xF))



//*****************************************************************************
// GPIOS by connection
//*****************************************************************************

// GPIO(0)  MOSI
// GPIO(1)  MISO
// GPIO(2)  MCLK
// GPIO(3)  nRSTPD
// GPIO(4)  PTI_EN
// GPIO(5)  PTI_DATA
// GPIO(6)  ADC
// GPIO(7)  NC
// GPIO(8)  LOCK IRQ
// GPIO(9)  SDA
// GPIO(10) SCL
// GPIO(11) RED_LED
// GPIO(12) nCHIPSELECT
// GPIO(13) NC
// GPIO(14) RTC IRQ
// GPIO(15) MIF IRQ
// GPIO(16) BUZZER



//*****************************************************************************
// GPIOS by functional grouping
//*****************************************************************************

// SPI buss
#define  MOSI           GPIO(0)
#define  MISO           GPIO(1)
#define  MCLK           GPIO(2)
#define  nCHIPSELECT    GPIO(12)
```

```
// Lock connector
#define  LOCK_ADC       GPIO(6)
#define  LOCK_SDA       GPIO(9)
#define  LOCK_IRQ       GPIO(8)
#define  LOCK_SCL       GPIO(10)

// Main I2C bus
#define SDA             GPIO(9)
#define SCL             GPIO(10)

// Mifare Interrupt & control
#define  MIF_IRQ        GPIO(15)
#define  nRSTPD         GPIO(3)

// Real time clock interrupt
#define  RTC_IRQ        GPIO(14)

// SIF ZigBee Network DEBUG
#define  PTI_EN         GPIO(4)
#define  PTI_DATA       GPIO(5)

// Human interface devices
#define  RED_LED        GPIO(11)
#define  BUZZER         GPIO(16)



//*****************************************************************************
//  LED Definitions
//*****************************************************************************
/*
 * The following are used to aid in the abstraction with the LED
 * connections.  The microcontroller-specific sources use these
 * definitions so they are able to work across a variety of boards
 * which could have different connections.  The names and ports/pins
 * used below are intended to match with a schematic of the system to
 * provide the abstraction.
 *
 * The BOARDLEDn enum values should always be used when manipulating the
 * state of LEDs, as they directly refer to the GPIOs to which the LEDs are
 * connected.
 */

// remove later

enum HalBoardLedPins {
  BOARDLED0 = 4,
  BOARDLED1 = 5,

  BOARD_ACTIVITY_LED  = BOARDLED0,
  BOARD_HEARTBEAT_LED = BOARDLED1

};

/**
 * @brief This mask indicates which GPIO the LEDs are connected to.
 * A bit is set for each GPIO to which an LED is connected.
 */
#define BOARDLED_MASK       0



//*****************************************************************************
// Button Definitions
//*****************************************************************************
/*
 * The following are used to aid in the abstraction with the Button
 * connections.  The microcontroller-specific sources use these
 * definitions so they are able to work across a variety of boards
 * which could have different connections.  The names and ports/pins
 * used below are intended to match with a schematic of the system to
 * provide the abstraction.
 *
 * The BUTTONn macros should always be used with manipulating the buttons
 * as they directly refer to the GPIOs to which the buttons are connected.
 *
 * @note The GPIO number must match the IRQ letter
 *
```

```
 * @brief The actual GPIO BUTTON0 is connected to.  This define should
 * be used whenever referencing BUTTON0.
 */



//*****************************************************************************
//  Debounces interrupt pins
//*****************************************************************************
// #define DEBOUNCE 0



//*****************************************************************************
//  LOCK IRQ
//*****************************************************************************

#define BUTTON0           8
/**
 * The interrupt configuration register for BUTTON0.
 */
#define BUTTON0_INTCFG    GPIO_INTCFGA
/**
 * The filter bit for BUTTON0.
 */
#define BUTTON0_FLT_BIT   GPIO_INTFILT_BIT
/**
 * The interrupt trigger selection for BUTTON0.
 */
#define BUTTON0_MOD_BITS  GPIO_INTMOD_BIT
/**
 * The interrupt bit for BUTTON0.
 */
#define BUTTON0_INT_BIT   INT_GPIOA



//*****************************************************************************
//  RTC IRQ
//*****************************************************************************

#define BUTTON1           14
/**
 * The interrupt configuration register for BUTTON1.
 */
#define BUTTON1_INTCFG    GPIO_INTCFGB
/**
 * The filter bit for BUTTON1.
 */
#define BUTTON1_FLT_BIT   GPIO_INTFILT_BIT
/**
 * The interrupt trigger selection for BUTTON1.
 */
#define BUTTON1_MOD_BITS  GPIO_INTMOD_BIT
/**
 * The interrupt bit for BUTTON1.
 */
#define BUTTON1_INT_BIT   INT_GPIOB



//*****************************************************************************
// Packet Trace
//*****************************************************************************
/*
 * When PACKET_TRACE is defined, GPIO_CFG will automatically be setup by
 * halInit() to enable Packet Trace support on GPIO Pins 4 and 5,
 * in addition to the configuration specified below.
 *
 * @note This define will override any settings for GPIO 4 and 5.
 */
//@{
/**
 * This define does not equate to anything.  It is used as a
 * trigger to enable Packet Trace support on the breakout board (dev0455).
 */
//#define PACKET_TRACE  // We do have PACKET_TRACE support
//@} //END OF PACKET TRACE DEFINITIONS
```

```
//*****************************************************************************
// Powered up configurations
//*****************************************************************************
/**
 * Powered setting of GPIO_DBG debug configuration register.
 */
#define POWERUP_GPIO_DBG     0
/**
 *  Powered setting of GPIO_CFG configuration register
 **/
#define POWERUP_GPIO_CFG 0x0498
/**
 * Powered setting of GPIO_DIRH GPIO16 output-enable register.
 */
#define POWERUP_GPIO_DIRH    0
/**
 * Powered setting of GPIO_DIRL GPIO15..0 output-enable register.
 */
#define POWERUP_GPIO_DIRL   (nCHIPSELECT|nRSTPD)
/**
 * Powered setting of GPIO_CLRH clear GPIO16 output register.
 */
#define POWERUP_GPIO_CLRH    0                 // No output state change
/**
 *  Powered setting of GPIO_CLRL clear GPIO15..0 outputs register.
 */
#define POWERUP_GPIO_CLRL    0                 //
/**
 *  Powered setting of GPIO_SETH set GPIO16 output register.
 */
#define POWERUP_GPIO_SETH    0                 // No output state change
/**
 *  Powered setting of GPIO_SETL set GPIO15..0 outputs register.
 */
#define POWERUP_GPIO_SETL   (BOARDLED_MASK)
/**
 *  Powered setting of GPIO_PUH GPIO16 pullup resistor enable
 * register.
 */
#define POWERUP_GPIO_PUH     0                 // No pullup on GPIO16
/**
 *  Powered setting of GPIO_PUL GPIO15..0 pullup resistors enable
 * register.
 */
#define POWERUP_GPIO_PUL    (LOCK_IRQ|MIF_IRQ|RTC_IRQ|PROX_IRQ|nCHIPSELECT)
/**
 *  Powered setting of GPIO_PDH GPIO16 pulldown resistor enable
 * register.
 */
#define POWERUP_GPIO_PDH     0                 // No pulldown on GPIO16
/**
 *  Powered setting of GPIO_PDL GPIO15..0 pulldown resistors enable
 * register.
 */
#define POWERUP_GPIO_PDL   (MCLK|nRSTPD)

//*****************************************************************************
//  Power Down configurations
//*****************************************************************************
/**
 *  Deep Sleep setting of GPIO_DBG debug configuration register.
 */
#define POWERDN_GPIO_DBG     POWERUP_GPIO_DBG
/**
 *  Deep Sleep setting of GPIO_CFG configuration register.
 */
// Disable analog I/O switches
#define POWERDN_GPIO_CFG    (POWERUP_GPIO_CFG & ~0x4F00)
/**
 *  Deep Sleep setting of GPIO_DIRH GPIO16 output-enable register.
 */
#define POWERDN_GPIO_DIRH   0xff// POWERUP_GPIO_DIRH//0xFF
/**
 *  Deep Sleep setting of GPIO_DIRL GPIO15..0 output-enable register.
 */
```

```
#define POWERDN_GPIO_DIRL    0xff//POWERUP_GPIO_DIRL// 0x01
/**
 *  Deep Sleep setting of GPIO_CLRH clear GPIO16 output register.
 */
#define POWERDN_GPIO_CLRH    0                 // No output state change
/**
 *  Deep Sleep setting of GPIO_CLRL clear GPIO15..0 outputs register.
 */
#define POWERDN_GPIO_CLRL    0x0000            // No output state change
/**
 *  Deep Sleep setting of GPIO_SETH set GPIO16 output register.
 */
#define POWERDN_GPIO_SETH    0                 // No output state change
/**
 *  Deep Sleep setting of GPIO_SETL set GPIO15..0 outputs register.
 */
#define POWERDN_GPIO_SETL   0xf7// BOARDLED_MASK     // LEDs off (high)
/**
 *  Deep Sleep setting of GPIO_PUH GPIO16 pullup resistor
 * enable register.
 */
#define POWERDN_GPIO_PUH    0// POWERUP_GPIO_PUH
/**
 *  Deep Sleep setting of GPIO_PUL GPIO15..0 pullup resistors
 * enable register.
 */
#define POWERDN_GPIO_PUL    0//POWERUP_GPIO_PUL//0
/**
 *  Deep Sleep setting of GPIO_PDH GPIO16 pulldown resistor
 * enable register.
 */
#define POWERDN_GPIO_PDH    0
/**
 *  Deep Sleep setting of GPIO_PDL GPIO15..0 pulldown resistors
 * enable register.
 */
#define POWERDN_GPIO_PDL    0//POWERUP_GPIO_PDL//0

//****************************************************************************
// Packet trace
//****************************************************************************
#ifdef  PACKET_TRACE
// Take care of PTI_EN and PTI_DATA
// |GPIO(5)            /* PTI  DATA pull up   */ \                           )
#endif//PACKET_TRACE

#ifdef  PACKET_TRACE
// |GPIO(4)            /* PTI    EN pull down */ \                 )
#endif//PACKET_TRACE

//****************************************************************************
// Wake on interrupt
//****************************************************************************
/**
 *  A convenient define that chooses if this external signal can
 * be used as source to wake from deep sleep.  Any change in the state of the
 * signal will wake up the CPU.
 */
#define WAKE_ON_GPIO0   FALSE
#define WAKE_ON_GPIO1   FALSE
#define WAKE_ON_GPIO2   FALSE
#define WAKE_ON_GPIO3   FALSE
#define WAKE_ON_GPIO4   FALSE
#define WAKE_ON_GPIO5   FALSE
#define WAKE_ON_GPIO6   FALSE
#define WAKE_ON_GPIO7   FALSE
#define WAKE_ON_GPIO8   FALSE         //BUTTON0
#define WAKE_ON_GPIO9   FALSE
#define WAKE_ON_GPIO10  FALSE
#define WAKE_ON_GPIO11  FALSE
#define WAKE_ON_GPIO12  FALSE
#define WAKE_ON_GPIO13  FALSE//TRUE
#define WAKE_ON_GPIO14  FALSE//TRUE   //BUTTON1
#define WAKE_ON_GPIO15  FALSE//TRUE   //BUTTON2
#define WAKE_ON_GPIO16  FALSE
//@}
```

```
//@} //END OF GPIO Configuration Definitions


/** @name Board Specific Functions
 *
 * The following macros exist to aid in the initialization, power up from sleep,
 * and power down to sleep operations.  These macros are responsible for
 * either initializing directly, or calling initialization functions for any
 * peripherals that are specific to this board implementation.  These
 * macros are called from halInit, halPowerDown, and halPowerUp respectively.
 */
 //@{
/**
 * @brief Initialize the board.  This function is called from halInit().
 */
#ifndef EZSP_UART
  #define halInternalInitBoard()                                    \
          do {                                                      \
             halInternalPowerUpBoard();                             \
             halInternalRestartUart();                              \
             halInternalInitButton();                               \
          } while(0)
#else
  #define halInternalInitBoard()                                    \
          do {                                                      \
            halInternalPowerUpBoard();                              \
          } while(0)
#endif

/**
 * @brief Power down the board.  This function is called from
 * halPowerDown().
 */
#define halInternalPowerDownBoard()                                 \
        do {                                                        \
          /* GPIO Output configuration */                           \
          GPIO_CLRH  = POWERDN_GPIO_CLRH; /* output states  */      \
          GPIO_CLRL  = POWERDN_GPIO_CLRL; /* output states  */      \
          GPIO_SETH  = POWERDN_GPIO_SETH; /* output states  */      \
          GPIO_SETL  = POWERDN_GPIO_SETL; /* output states  */      \
          GPIO_DIRH  = POWERDN_GPIO_DIRH; /* output enables */      \
          GPIO_DIRL  = POWERDN_GPIO_DIRL; /* output enables */      \
          /* GPIO Mode Config (see above) */                        \
          GPIO_DBG   = POWERDN_GPIO_DBG;                            \
          GPIO_CFG   = POWERDN_GPIO_CFG;                            \
          /* GPIO Pullup/Pulldown Config */                         \
          GPIO_PDH   = POWERDN_GPIO_PDH;                            \
          GPIO_PDL   = POWERDN_GPIO_PDL;                            \
          GPIO_PUH   = POWERDN_GPIO_PUH;                            \
          GPIO_PUL   = POWERDN_GPIO_PUL;                            \
        } while(0)

/**
 * @brief Power up the board.  This function is called from
 * halPowerUp().
 */

#define halInternalPowerUpBoard()                                   \
        do {                                                        \
          /* GPIO Pullup/Pulldown Config */                         \
          GPIO_PUH   = POWERUP_GPIO_PUH;                            \
          GPIO_PUL   = POWERUP_GPIO_PUL;                            \
          GPIO_PDH   = POWERUP_GPIO_PDH;                            \
          GPIO_PDL   = POWERUP_GPIO_PDL;                            \
          /* GPIO Mode Config (see above) */                        \
          GPIO_DBG   = POWERUP_GPIO_DBG;                            \
          GPIO_CFG   = POWERUP_GPIO_CFG;                            \
          /* GPIO Output configuration */                           \
          GPIO_CLRH  = POWERUP_GPIO_CLRH; /* output states  */      \
          GPIO_CLRL  = POWERUP_GPIO_CLRL; /* output states  */      \
          GPIO_SETH  = POWERUP_GPIO_SETH; /* output states  */      \
          GPIO_SETL  = POWERUP_GPIO_SETL; /* output states  */      \
          GPIO_DIRH  = POWERUP_GPIO_DIRH; /* output enables */      \
          GPIO_DIRL  = POWERUP_GPIO_DIRL; /* output enables */      \
        } while(0)
//@} //END OF BOARD SPECIFIC FUNCTIONS

#endif //__BOARD_H__
```

decta-DoorSystemConfiguration.h

```
//******************************************************************************
// File: decta-DoorSystemConfiguration.h
//
// Description:
// Contains general ZigBee stack settings
//
// Authors: Mikael Högrud and Johan Riisberg-Jensen (2010), based on
// code from Ember Corporation
//******************************************************************************



//******************************************************************************
// This template configuration file contains some sample application
//  configuration options.  Please see:
//    stack/config/ember-configuration/defaults.h  and
//    hal/ember-configuration.c
//  for further options that can be specified.



//******************************************************************************
// Stack Profile Parameters
//******************************************************************************

// (ZigBee Pro -enables many to one source routing etc)
#define EMBER_STACK_PROFILE 2



//******************************************************************************
// Ember static memory requirements
//******************************************************************************

// There are constants that define the amount of static memory
// required by the stack. If the application does not set them then
// the default values from ember-configuration-defaults.h are used.
//
// for example, this changes the default number of buffers to 8
// #define EMBER_PACKET_BUFFER_COUNT 8
// #define EMBER_BINDING_TABLE_SIZE 8



//******************************************************************************
// Application Handlers
//
// By default, a number of stub handlers are automatically provided
// that have no effect.  If the application would like to implement any
// of these handlers itself, it needs to define the appropriate macro
//******************************************************************************

// needs to be done to use the scan utilities
#define EMBER_APPLICATION_HAS_ENERGY_SCAN_RESULT_HANDLER

// Some additional examples
//#define EMBER_APPLICATION_HAS_BUTTON_HANDLER
//#define EMBER_APPLICATION_HAS_REMOTE_BINDING_HANDLER

// For debugging in the IDE
// #define DISABLE_WATCHDOG

// If the following timeout is exceeded, then the parent-child connection will
// be regarded as lost: (used on both sides)
// Value in seconds (left shifted by the value below)
#define EMBER_END_DEVICE_POLL_TIMEOUT 70
#define EMBER_END_DEVICE_POLL_TIMEOUT_SHIFT 0

// Define fragmented messages buffer size to 8, which is the maximum number:
#define EMBER_FRAGMENT_WINDOW_SIZE 1

//#define EMBER_PACKET_BUFFER_COUNT 24       // Default value is 24

// We have our own parent poll callback handler
#define EMBER_APPLICATION_HAS_POLL_COMPLETE_HANDLER
```

decta-hal-i2c.h

```
//*****************************************************************************
// File: decta-hal-i2c.h
//
// Description:
// Contains low-level hardware abstractions for the I2C bus
//
// Authors: Mikael Högrud and Johan Riisberg-Jensen (2010)
//*****************************************************************************



// I2C initialization
void halSC1I2cInit(void);

// I2C routine for the Eeprom
int8u halI2cEe(int8u control, int16u address, int8u *data, int8u length);

// I2C routine for the Real time clock
int8u halI2cRtc(int8u control,                int8u *data, int8u length);

// I2C routine for the Mifare interrogator
int8u halI2cMif(int8u control, int8u address,  int8u *data, int8u length);

// I2C routine for the Lock
int8u halI2cLock(int8u address, int8u *data, int8u length);
```

decta-hal-spi.h

```
//*****************************************************************************
// File: decta-hal-spi.h
//
// Description:
// Contains low level hardware abstractions for the SPI interface
//
// Authors: Mikael Högrud and Johan Riisberg-Jensen (2010)
//*****************************************************************************



// Needed since Ember SPI hardware requires a byte to be sent for every byte
// received.
#define dummyByte  0xFF

// SPI initialization:
void halSC2SpiInit(void);

// Sends a data byte:
void  halSpiWrite(int8u data);

// Receives a data byte:
int8u halSpiRead(void);

// Send and receive a data byte simultaneously:
int8u halSpiReadWrite(int8u data);

// Auxiliary functions:
void halSpiCsEnable(void);
void halSpiCsDisable(void);
```

decta-LockDriver.h

```
//*****************************************************************************
// File: decta-LockDriver.h
//
// Description:
// Contains functions to manage the doorlock
//
// Authors: Mikael Högrud and Johan Riisberg-Jensen (2010)
//*****************************************************************************



// Lock's physical I2C address:
#define LOCK_DEVICE_ID  0xD0

// Lock door:
int8u closeLock(void);

// Open door for time Specified with time, 1-254sec, 0xFF = open forever:
int8u openLock(int8u time);

// Gets the status of the last operation this also resets the interrupt flag:
int8u getLockStatus(void);
```

decta-NumberManagement.h

```
//*****************************************************************************
// File: decta-NumberManagement.h
//
// Description:
// Contains functions necessary in order to manage and sort data
// on non-volatile memory
//
// Authors: Mikael Högrud and Johan Riisberg-Jensen (2010)
//*****************************************************************************



//*****************************************************************************
// Defines:
//*****************************************************************************

// Size of card usernumber
#define userNumberSize                7

// Size of timeschedule code
#define codeDataSize                  2

// Size of timeschedule
#define timeScheduleSize              63

// Size of compressed log entry on non-volatile
// memory (contains: [userNumber, YY-MM-DD HH:MM, Operation])
#define logSize                       (userNumberSize+4+1)

// Size of uncompressed log entry
#define uncompressedLogSize           (logSize+1)

// Size of storing slot for important variables on non-volatile memory
#define infoSectionSize               32



//*****************************************************************************
// Standard settings for 24AA256 type EEPROM:
//*****************************************************************************

#ifdef MEM_TYPE_24AA256

#define userDataPerPage               7
#define useMaskByteSize               1
#define timeSchedulesPerPage          1
#define timeScheduleUseMaskByteSize   1
#define logsPerPage                   5
#define logUseMaskByteSize            1
#define infoSectionsPerPage           2

// Default non-volatile memory area section division:

// Number of pages available for the 24AA256
//#define MaxPages                    512

// Default number of pages for number storing.
#define defaultAvailablePages         430

// Default number of pages for log data
#define defaultLogPages               20

// Default number of pages for timeschedule data
#define defaultTimeSchedulePages      59

// Default number of pages for various info that needs to be stored onto flash.
#define fixedNumberOfInfoPages        3

#endif



//*****************************************************************************
// Standard settings for 24AA1024 type EEPROM:
//*****************************************************************************
```

```
#ifdef MEM_TYPE_24AA1025

#define userDataPerPage               14
#define useMaskByteSize               2
#define timeSchedulesPerPage          2
#define timeScheduleUseMaskByteSize   1
#define logsPerPage                   10
#define logUseMaskByteSize            2
#define infoSectionsPerPage           4

// Default non-volatile memory area section division:

// Number of pages available for the 24AA1025
//#define MaxPages                    1024

// Default number of pages for number storing.
#define defaultAvailablePages         863

// Default number of pages for log data
#define defaultLogPages               40

// Default number of pages for timeschedule data
#define defaultTimeSchedulePages      118

// Default number of pages for various info that needs to be stored onto flash.
#define fixedNumberOfInfoPages        3

#endif



//****************************************************************************
// Standard settings for M45PE20 (and M25PE10-A) type Flash memory:
//****************************************************************************

#ifdef MEM_TYPE_M45PE20

#define userDataPerPage               28
#define useMaskByteSize               4
#define timeSchedulesPerPage          4
#define timeScheduleUseMaskByteSize   1
#define logsPerPage                   21
#define logUseMaskByteSize            3
#define infoSectionsPerPage           8

// Default non-volatile memory area section division:

// Number of pages available for the M45PE20
//#define MaxPages                    1024

// Default number of pages for number storing.
#define defaultAvailablePages         863

// Default number of pages for log data
#define defaultLogPages               40

// Default number of pages for timeschedule data
#define defaultTimeSchedulePages      118

// Default number of pages for various info that needs to be stored onto flash.
#define fixedNumberOfInfoPages        3

#endif



//****************************************************************************
// Time code defines:
//****************************************************************************

// Time codes should be in the form of
// [Code][Number of intervals][Time specification] and finish with
// an EMPTY code if not the last byte.
// Example: Monday-Thursday between 8-9 am and 3-5 pm, or the entire year
// of 2009, would be written as:
// [DAY][01][00][03][TIME][02][08][00][09][00][15][00][17][00][OR]
// [YEAR][1][09][09][EMPTY]
```

```
//#define EMPTY      255    // Code for: Time code string has finished
//#define OR         254    // Code for: Further coming interval
//#define YEAR       253    // Time code for: Year (00-99)
//#define MONTH      252    // Time code for: Month (01-12)
//#define DATE       251    // Time code for: Date (00-31)
//#define DAY        249    // Time code for: Day (01-07)
//#define TIME       248    // Time code for: Time (00-24, 00-59)


//***************************************************************************
// Add, remove, defragmentation defines:
//***************************************************************************

// Number of "first" unsorted rows (before the sorted section).
// The routines only support 1.
#define skipRows            1

// The index number where the sorted section starts.
#define firstSortedIndex    (skipRows*userDataPerPage)



//***************************************************************************
// General functions forward declarations:
//***************************************************************************

// Compare 2 numbers:
// Result used for AgreaterThanB():
//     FALSE               0
//     TRUE                1
#define EQUAL               2

int8u AgreaterThanB(int8u *userNumberA, int8u *userNumberB);

// Index/Page/Column conversions (Be careful with these!):
int32u toIndex(int16u page, int8u col);
int16u GetPage(int32u index);
int8u GetCol(int32u index);

// Help function for converting and storing an index
// into page and col variables;
void toPageCol(int32u index, int16u *pageNumber, int8u *colNumber);



//***************************************************************************
// Ram Quicksort procedure:
//***************************************************************************

// Note: Addresses are RAM addresses!
void QuickSort(int8u *startAddress, int8u *stopAddress);



//***************************************************************************
// Quicksort forward subroutine declarations (private):
//***************************************************************************

int8u *sortPivot(int8u *startAddress, int8u *stopAddress, int8u  *pivotAddress);

boolean AgreaterThanB_switched(int8u *addressA, int8u *addressB);



//***************************************************************************
// Add, remove, find, defragmentation routines:
//***************************************************************************

// Function for setting useMask value for a pointer
// to a RAM page byte array for a certain column number:
void SetUseMask(int8u *pageBuffer, int8u col, boolean useValue);

// Function for getting useMask value for a pointer to a RAM page
// byte array for a certain column number:
boolean GetUseMask(int8u *pageBuffer, int8u col);
```

```
// Help function for getting a pointer to the start position of a
// userNumber at a certain column in a RAM page byte array:
int8u *GetUserData(int8u *pageBuffer, int8u col);

// Help function for setting a userNumber at a certain column in a
// RAM page byte array:
void SetUserData(int8u *pageBuffer, int8u col, int8u *userData);

// Help function for calculating the corresponding EEPROM or FLASH
// start address for a certain page value:
int32u GetNonVolatileAddress(int16u page);

// Internal function that deletes 1 userData from RAM page byte
// array and fills up with userData from the right.
void DeleteAndMoveLeft(int8u *pageBuffer, int8u col);

// Returns the column in which a userNumber exists in a RAM page
// byte array.
// If the number doesn't exist, then it returns userDataPerPage
// which is an illegal column value.
int8u PageFind(int8u *pageBuffer, int8u *userNumber);

// This function performs a binary search on the entire sorted
// section on the non volatile memory.
// Number found if number(resultindex)==number.
// Returns number index or closest lower number index, except
// when searching for number below firstSortedIndex, then it
// returns the closest higher number index.
// Note: this function will deliberately find numbers with useMask==0 !!!
int32u BinSearch(int8u *userNumber, int32u startIndex, int32u stopIndex);

// This function searches for a number both in sorted and unsorted
// sections of the list. The search is limited to in between startIndex
// and stopIndex with the exception of startindex=firstSortedIndex, then
// first page numbers are also included in the search.
// A startIndex<firstSortedIndex is not legal.
// Results are stored in resultPage and resultCol.
// If the number wasn't found then it returns illegal page and
// column codes: availablePages and userDataPerPage.
void GlobalSearch( int8u *userData, int32u startIndex, int32u stopIndex, \
                   int16u *resultPage, int8u *resultCol);

// Function for Determining if there is enough buffer space left
// to add a new unsorted or firstPage number before a rearrange:
boolean StillBufferSpace();



// Function for removing a userData with a certain userNumber:
int8u RemoveNumber(int8u *userData);



// Start AddNumber state machine:
int8u AddNumber(int8u *userData);

// Tick the state machine:
void AddNumberTick();



// Start ReArrange state machine:
void ReArrange();

// Tick the state machine:
void ReArrangeTick();



// Start DeFragment state machine:
void DeFragment();

// Tick the state machine:
void DeFragmentTick();



// Check if any of the number managing state machines are currently in use:
```

```
boolean NumberStateMachinesIdle();



//*****************************************************************************
// Time schedule forward function declarations:
//*****************************************************************************

// Function for adding a timeSchedule to a location on non-volatile memory
// determined by its timeScheduleNumber:
// IMPORTANT: Do not update timeschedule
// unless NumberStateMachinesIdle()==TRUE!!!
int8u UpdateTimeSchedule(int16u timeScheduleNumber, int8u *timeSchedule);

// Function for removing a timeSchedule to a location on non-volatile
// memory determined by its timeScheduleNumber:
int8u RemoveTimeSchedule(int16u timeScheduleNumber);

// Get and set useMask functions can be used for setting and getting useMasks
// also for timeschedules!

// Function for getting a pointer to the first element in a timeSchedule
// at a certain col position in a RAM byte array:
int8u *GetTimeSchedule(int8u *pageBuffer, int8u col);

// Function for setting a timeschedule at a certain col position in
// a RAM byte array:
void SetTimeSchedule(int8u *pageBuffer, int8u col, int8u *timeSchedule);

// Help function for converting a timeScheduleNumber to page and col values:
void TimeScheduleNumberToPageCol(int16u timeScheduleNumber,\
                                 int16u *pageNumber, int8u *colNumber);

// Function that takes a timeScheduleNumber, retrieves the timeSchedule
// from non-volatile memory and compares it with the real time clock.
// The function returns TRUE if the time is inside the time interval,
// and FALSE if it's outside:
boolean CompareTimeToTimeSchedule(int16u timeScheduleNumber);



//*****************************************************************************
// Memory boundary function declarations:
//*****************************************************************************

// Functions for getting page boundaries on non-volatile memory:
int16u GetInfoEndPage();
int16u GetInfoStartPage();
int16u GetTimeScheduleEndPage();
int16u GetTimeScheduleStartPage();
int16u GetLogEndPage();
int16u GetLogStartPage();
int16u GetUserDataEndPage();
int16u GetUserDataStartPage();

// Function for setting memory boundaries:
// Note: this functions should only be used right after
// the memory has been formatted!
int8u SetMemoryPageBoundaries( int16u newNumberOfinfoPages,     \
                               int16u newNumberOftimeSchedulePages,  \
                               int16u newNumberOflogPages,         \
                               int16u newNumberOfuserDataPages);



//*****************************************************************************
// Log function declarations:
//*****************************************************************************

// Function that returns the total number of log slots that the system can hold:
int16u NumberOfLogSlots();

// Adds a log entry in chronological order: (FIFO wrap around style when
// the memory is full. Note: this function uses the temporary page buffer!)
// Make sure the log entry is in the format of
// [userNumber, Year, Month, Date, Hour, Minute, Operation] as bytes.
int8u AddLogEntry(int8u *uncompressedLogEntry);
```

```
// Help function for copying in the data to an uncompressed log entry
// byte array in the correct format:
void CopyLogDataToLogEntry( int8u *userNumber, int8u year, int8u month,     \
                            int8u date, int8u hour, int8u minute,      \
                            int8u operation, int8u *uncompressedLogEntry);

// Formats the non-volatile log memory area:
int8u EraseEntireLogArea();

// Get and set useMask functions can be used for setting and getting useMasks
// also for log entries!

// Function for retrieving the n:th latest log entry.
// FIFO wrap around style when the memory is full, and n must be less than
// NumberOfLogSlots!
// Note: this function uses the temporary page buffer.
int8u GetNthLatestLogEntry(int16u n, int8u *uncompressedLogEntry);

// Function for getting a pointer to the first element in a log entry at a
// certain col position in a RAM byte array:
int8u *GetLog(int8u *pageBuffer, int8u col);

// Function for setting a log entry at a certain col position
// in a RAM byte array:
void SetLog(int8u *pageBuffer, int8u col, int8u *logEntry);

// Help function for converting a log entry number to page and col values:
void logNumberToPageCol(  int16u logNumber, int16u *pageNumber, \
                          int8u *colNumber);

// Function to compress the date portion of the log entry in
// order to save non-volatile memory space:
void CompressLogEntry(int8u *logEntry);

// Function to decompress the date portion of the log entry in
// order to save non-volatile memory space:
void DeCompressLogEntry(int8u *logEntry);



//***************************************************************************
// Variable management function declarations:
//***************************************************************************

// This function saves the most important variables to the info
// area on non-volatile memory:
// Doesn't use tempPageBuffer -so always safe to use!
void SaveVariables();

// This function loads the most important variables from the info area
// on non-volatile memory:
// This function is normally only called after a reset.
// Doesn't use tempPageBuffer -so always safe to use!
void LoadVariables();

// Help function getting the number of available info section slots:
int16u GetNumberOfInfoSectionSlots();

// Function for getting the non-volatile address for a certain page
// and col info section area:
int32u GetInfoSectionNonVolatileAddress(int16u page, int8u col);

// This function retrieves the corresponding page and col values that are
// associated with a certain cycle sequence number:
void SequenceNumberToPageCol(  int16u sequenceNumber, int16u *pageNumber, \
                               int8u *colNumber);

// Function for formatting all the info pages:
void FormatInfoPages();


//***************************************************************************
// General function declarations:
//***************************************************************************

// If the memory is to be formatted, then this function is to be used as
// it also resets the variables to default values:
boolean FormatAndSetDefaultValues();
```

decta-prodboard.h

```
//*****************************************************************************
// File: decta-prodboard.h
//
// Description:
// Contains system state configurations for the production prototype board
//
// Authors: Mikael Högrud and Johan Riisberg-Jensen (2010), based on
// code from Ember Corporation
//*****************************************************************************



//*****************************************************************************
// Board configuration
//*****************************************************************************

#ifndef __BOARD_H__
#define __BOARD_H__
#define EMBER_SERIAL_BAUD_CUSTOM  13



//*****************************************************************************
//  GPIO Configuration Definitions
//*****************************************************************************
/*
 * The following are used to specify the GPIO configuration to establish
 * when Powered (POWERUP_), and when Deep Sleeping (POWERDN_).  The reason
 * for separate Deep Sleep settings is to allow for a slightly different
 * configuration that minimizes power consumption during Deep Sleep.  For
 * example, inputs that might float could be pulled up or down, and output
 * states chosen with care, e.g. to turn off LEDs or other devices that might
 * consume power or be unnecessary when Deep Sleeping.
 *
 * A convenience macro to convert GPIO# into bit mask for the
 * GPIO registers.
 */
#define GPIO(n) (1u << ((n) & 0xF))

//*****************************************************************************
// GPIOS by connection
//*****************************************************************************

// Pin:     Name:          I/O:   Sleepstate:
// GPIO(0)  MOSI           out    low
// GPIO(1)  MISO           in     low
// GPIO(2)  MCLK           out    low, pulldown
// GPIO(3)  LOCK_SCL       out    high, pullup
// GPIO(4)  RED_LED        out    high
// GPIO(5)  GREEN_LED      out    high
// GPIO(6)  LOCK_ADC       in     low
// GPIO(7)  LOCK_SDA       in/out high, pullup
// GPIO(8)  LOCK_IRQ       in     input, pulldown
// GPIO(9)  SDA            in/out high, external pullup
// GPIO(10) SCL            out    high, external pullup
// GPIO(11) nCHIPSELECT    out    high, pullup
// GPIO(12) nRSTPD         out    low, pulldown
// GPIO(13) MIF_IRQ        in     high, pullup
// GPIO(14) RTC_IRQ        in     high, pullup
// GPIO(15) PROX_IRQ       in     high, pullup
// GPIO(16) BUZZER         out    low

// NOTE: The sleepstate values of LOCK_ADC and PROX_IRQ must be set with
// consderation to if their (external) functional circuits are present in
// the system.



//*****************************************************************************
// GPIOS by functional grouping
//*****************************************************************************

// SPI bus
#define  MOSI           GPIO(0)
#define  MISO           GPIO(1)
```

```
#define  MCLK           GPIO(2)
#define  nCHIPSELECT    GPIO(11)

// Lock connector
#define  LOCK_ADC       GPIO(6)
#define  LOCK_SDA       GPIO(7)
#define  LOCK_IRQ       GPIO(8)
#define  LOCK_SCL       GPIO(3)

// Main I2C bus
#define SDA             GPIO(9)
#define SCL             GPIO(10)

// Mifare Interrupt & control
#define  MIF_IRQ        GPIO(13)
#define  nRSTPD         GPIO(12)

// Real time clock interrupt
#define  RTC_IRQ        GPIO(14)

// Proximity interrupt
#define  PROX_IRQ       GPIO(15)

// Human interface devices
#define  RED_LED        GPIO(4)
#define  GREEN_LED      GPIO(5)
#define  BUZZER         GPIO(16)

//*****************************************************************************
//  LED Definitions
//*****************************************************************************
/*
 * The following are used to aid in the abstraction with the LED
 * connections.  The microcontroller-specific sources use these
 * definitions so they are able to work across a variety of boards
 * which could have different connections.  The names and ports/pins
 * used below are intended to match with a schematic of the system to
 * provide the abstraction.
 *
 * The BOARDLEDn enum values should always be used when manipulating the
 * state of LEDs, as they directly refer to the GPIOs to which the LEDs are
 * connected.
 */

enum HalBoardLedPins {
  REDLED = 4,
  GREENLED = 5,

  #ifdef DEBUG
  BOARD_ACTIVITY_LED=GREENLED // Signals network activity
  #else
  BOARD_ACTIVITY_LED=18 // NOTE: Undefined GPIO
  #endif
};
/**
 * @brief This mask indicates which GPIO the LEDs are connected to.
 * A bit is set for each GPIO to which an LED is connected.
 */
#define BOARDLED_MASK       (RED_LED | GREEN_LED)



//*****************************************************************************
// Button Definitions
//*****************************************************************************
/*
 * The following are used to aid in the abstraction with the Button
 * connections.  The microcontroller-specific sources use these
 * definitions so they are able to work across a variety of boards
 * which could have different connections.  The names and ports/pins
 * used below are intended to match with a schematic of the system to
 * provide the abstraction.
 *
 * The BUTTONn macros should always be used with manipulating the buttons
 * as they directly refer to the GPIOs to which the buttons are connected.
 *
 * @note The GPIO number must match the IRQ letter
 *
```

```
 * @brief The actual GPIO BUTTON0 is connected to.  This define should
 * be used whenever referencing BUTTON0.
 */

//*****************************************************************************
//  Debounces interrupt pins
//*****************************************************************************
//#define DEBOUNCE 0



//*****************************************************************************
//  LOCK IRQ
//*****************************************************************************

#define BUTTON0           8
/**
 * The interrupt configuration register for BUTTON0.
 */
#define BUTTON0_INTCFG    GPIO_INTCFGA
/**
 * The filter bit for BUTTON0.
 */
#define BUTTON0_FLT_BIT   GPIO_INTFILT_BIT
/**
 * The interrupt trigger selection for BUTTON0.
 */
#define BUTTON0_MOD_BITS  GPIO_INTMOD_BIT
/**
 * The interrupt bit for BUTTON0.
 */
#define BUTTON0_INT_BIT   INT_GPIOA



//*****************************************************************************
//  RTC IRQ
//*****************************************************************************

#define BUTTON1    14
/**
 * The interrupt configuration register for BUTTON1.
 */
#define BUTTON1_INTCFG    GPIO_INTCFGB
/**
 * The filter bit for BUTTON1.
 */
#define BUTTON1_FLT_BIT   GPIO_INTFILT_BIT
/**
 * The interrupt trigger selection for BUTTON1.
 */
#define BUTTON1_MOD_BITS  GPIO_INTMOD_BIT
/**
 * The interrupt bit for BUTTON1.
 */
#define BUTTON1_INT_BIT   INT_GPIOB



//*****************************************************************************
//  PROX IRQ
//*****************************************************************************

#define BUTTON2           15
/**
 * The interrupt configuration register for BUTTON0.
 */
#define BUTTON2_INTCFG    GPIO_INTCFGC
/**
 * The filter bit for BUTTON0.
 */
#define BUTTON2_FLT_BIT   GPIO_INTFILT_BIT
/**
 * The interrupt trigger selection for BUTTON0.
 */
#define BUTTON2_MOD_BITS  GPIO_INTMOD_BIT
/**
 * The interrupt bit for BUTTON0.
```

```
 */
#define BUTTON2_INT_BIT   INT_GPIOC


//*****************************************************************************
// Packet Trace
//*****************************************************************************
/*
 * When PACKET_TRACE is defined, GPIO_CFG will automatically be setup by
 * halInit() to enable Packet Trace support on GPIO Pins 4 and 5,
 * in addition to the configuration specified below.
 *
 * @note This define will override any settings for GPIO 4 and 5.
 */
//@{
/**
 * This define does not equate to anything.  It is used as a
 * trigger to enable Packet Trace support on the breakout board (dev0455).
 */
//#define PACKET_TRACE  // We do have PACKET_TRACE support
//@} //END OF PACKET TRACE DEFINITIONS



//*****************************************************************************
// Powered upp configurations
//*****************************************************************************
/**
 * Powered setting of GPIO_DBG debug configuration register.
 */
#define POWERUP_GPIO_DBG    0
/**
 *  Powered setting of GPIO_CFG configuration register
 **/
#define POWERUP_GPIO_CFG    0x0498
/**
 * Powered setting of GPIO_DIRH GPIO16 output-enable register.
 */
#define POWERUP_GPIO_DIRH   0                 // GPIO16 = Input
/**
 * Powered setting of GPIO_DIRL GPIO15..0 output-enable register.
 */
#define POWERUP_GPIO_DIRL   (RED_LED|GREEN_LED|nCHIPSELECT|nRSTPD)
/**
 * Powered setting of GPIO_CLRH clear GPIO16 output register.
 */
#define POWERUP_GPIO_CLRH    0                 // No output state change
/**
 *  Powered setting of GPIO_CLRL clear GPIO15..0 outputs register.
 */
#define POWERUP_GPIO_CLRL    0                 //
/**
 *  Powered setting of GPIO_SETH set GPIO16 output register.
 */
#define POWERUP_GPIO_SETH    0                 // No output state change
/**
 *  Powered setting of GPIO_SETL set GPIO15..0 outputs register.
 */
#define POWERUP_GPIO_SETL   (RED_LED|GREEN_LED)
/**
 *  Powered setting of GPIO_PUH GPIO16 pullup resistor enable
 * register.
 */
#define POWERUP_GPIO_PUH     0                 // No pullup on GPIO16
/**
 *  Powered setting of GPIO_PUL GPIO15..0 pullup resistors enable
 * register.
 */
#define POWERUP_GPIO_PUL    (  /*LOCK_IRQ|*/    \
                               MIF_IRQ       \
                               |RTC_IRQ      \
                               /*|PROX_IRQ*/    \
                               |LOCK_SDA     \
                               |LOCK_SCL     \
                               |nCHIPSELECT)
/**
 *  Powered setting of GPIO_PDH GPIO16 pulldown resistor enable
```

```
 * register.
 */
#define POWERUP_GPIO_PDH     0                 // No pulldown on GPIO16
/**
 *  Powered setting of GPIO_PDL GPIO15..0 pulldown resistors enable
 * register.
 */
#define POWERUP_GPIO_PDL     (LOCK_IRQ|nRSTPD|MCLK)



//*****************************************************************************
//  Power Down configurations
//*****************************************************************************
/**
 *  Deep Sleep setting of GPIO_DBG debug configuration register.
 */
#define POWERDN_GPIO_DBG     POWERUP_GPIO_DBG
/**
 *  Deep Sleep setting of GPIO_CFG configuration register.
 */
// Disable analog I/O switches
#define POWERDN_GPIO_CFG    (POWERUP_GPIO_CFG & ~0x4F00)
/**
 *  Deep Sleep setting of GPIO_DIRH GPIO16 output-enable register.
 */
#define POWERDN_GPIO_DIRH    0x0001
/**
 *  Deep Sleep setting of GPIO_DIRL GPIO15..0 output-enable register.
 */
#define POWERDN_GPIO_DIRL    0x7EFF
/**
 *  Deep Sleep setting of GPIO_CLRH clear GPIO16 output register.
 */
#define POWERDN_GPIO_CLRH    (BUZZER)                 // No output state change
/**
 *  Deep Sleep setting of GPIO_CLRL clear GPIO15..0 outputs register.
 */
#define POWERDN_GPIO_CLRL    (MOSI|MISO|MCLK|LOCK_ADC|nRSTPD)
/**
 *  Deep Sleep setting of GPIO_SETH set GPIO16 output register.
 */
#define POWERDN_GPIO_SETH    0x0000                // No output state change
/**
 *  Deep Sleep setting of GPIO_SETL set GPIO15..0 outputs register.
 */
#define POWERDN_GPIO_SETL (  LOCK_SCL    \
                            |RED_LED    \
                            |GREEN_LED    \
                            |LOCK_SDA   \
                            /*|LOCK_IRQ*/ \
                            |SDA      \
                            |SCL      \
                            |nCHIPSELECT  \
                            |MIF_IRQ    \
                            |RTC_IRQ    \
                            /*|PROX_IRQ*/)

/**
 *  Deep Sleep setting of GPIO_PUH GPIO16 pullup resistor
 * enable register.
 */
#define POWERDN_GPIO_PUH     POWERUP_GPIO_PUH
/**
 *  Deep Sleep setting of GPIO_PUL GPIO15..0 pullup resistors
 * enable register.
 */



//*****************************************************************************
// Packet trace
//*****************************************************************************

#ifdef  PACKET_TRACE
// Take care of PTI_EN and PTI_DATA
// |GPIO(5)            /* PTI  DATA pull up   */ \                            )
#endif//PACKET_TRACE
```

```
#define POWERDN_GPIO_PUL    POWERUP_GPIO_PUL

/**
 *  Deep Sleep setting of GPIO_PDH GPIO16 pulldown resistor
 * enable register.
 */
#define POWERDN_GPIO_PDH    POWERUP_GPIO_PDH // TMR1 pull down
/**
 *  Deep Sleep setting of GPIO_PDL GPIO15..0 pulldown resistors
 * enable register.
 */
#ifdef  PACKET_TRACE
// |GPIO(4)            /* PTI    EN pull down */ \                  )
#endif//PACKET_TRACE
#define POWERDN_GPIO_PDL    POWERUP_GPIO_PDL



//******************************************************************************
// Wake on interrupt
//******************************************************************************
/**
 *  A convenient define that chooses if this external signal can
 * be used as source to wake from deep sleep.  Any change in the state of the
 * signal will wake up the CPU.
 */
#define WAKE_ON_GPIO0   FALSE
#define WAKE_ON_GPIO1   FALSE
#define WAKE_ON_GPIO2   FALSE
#define WAKE_ON_GPIO3   FALSE
#define WAKE_ON_GPIO4   FALSE
#define WAKE_ON_GPIO5   FALSE
#define WAKE_ON_GPIO6   FALSE
#define WAKE_ON_GPIO7   FALSE
#define WAKE_ON_GPIO8   TRUE   // Lock IRQ
#define WAKE_ON_GPIO9   FALSE
#define WAKE_ON_GPIO10  FALSE
#define WAKE_ON_GPIO11  FALSE
#define WAKE_ON_GPIO12  FALSE
#define WAKE_ON_GPIO13  FALSE
#define WAKE_ON_GPIO14  FALSE   // BUTTON1
#define WAKE_ON_GPIO15  FALSE   //TRUE    // Prox IRQ
#define WAKE_ON_GPIO16  FALSE
//@}

//@} //END OF GPIO Configuration Definitions

/** @name Board Specific Functions
 *
 * The following macros exist to aid in the initialization, power up from sleep,
 * and power down to sleep operations.  These macros are responsible for
 * either initializing directly, or calling initialization functions for any
 * peripherals that are specific to this board implementation.  These
 * macros are called from halInit, halPowerDown, and halPowerUp respectively.
 */
 //@{
/**
 * @brief Initialize the board.  This function is called from halInit().
 */
#ifndef EZSP_UART
  #define halInternalInitBoard()                                    \
         do {                                                       \
            halInternalPowerUpBoard();                              \
            halInternalRestartUart();                               \
            halInternalInitButton();                                \
         } while(0)
#else
  #define halInternalInitBoard()                                    \
          do {                                                      \
            halInternalPowerUpBoard();                              \
         } while(0)
#endif

/**
 * @brief Power down the board.  This function is called from
 * halPowerDown().
 */
#define halInternalPowerDownBoard()                             \
```

```
        do {                                                    \
          /* GPIO Output configuration */                       \
          GPIO_CLRH  = POWERDN_GPIO_CLRH; /* output states  */  \
          GPIO_CLRL  = POWERDN_GPIO_CLRL; /* output states  */  \
          GPIO_SETH  = POWERDN_GPIO_SETH; /* output states  */  \
          GPIO_SETL  = POWERDN_GPIO_SETL; /* output states  */  \
          GPIO_DIRH  = POWERDN_GPIO_DIRH; /* output enables */  \
          GPIO_DIRL  = POWERDN_GPIO_DIRL; /* output enables */  \
          /* GPIO Mode Config (see above) */                    \
          GPIO_DBG   = POWERDN_GPIO_DBG;                        \
          GPIO_CFG   = POWERDN_GPIO_CFG;                        \
          /* GPIO Pullup/Pulldown Config */                     \
          GPIO_PDH   = POWERDN_GPIO_PDH;                        \
          GPIO_PDL   = POWERDN_GPIO_PDL;                        \
          GPIO_PUH   = POWERDN_GPIO_PUH;                        \
          GPIO_PUL   = POWERDN_GPIO_PUL;                        \
        } while(0)

/**
 * @brief Power up the board.  This function is called from
 * halPowerUp().
 */

#define halInternalPowerUpBoard()                               \
        do {                                                    \
          /* GPIO Pullup/Pulldown Config */                     \
          GPIO_PUH   = POWERUP_GPIO_PUH;                        \
          GPIO_PUL   = POWERUP_GPIO_PUL;                        \
          GPIO_PDH   = POWERUP_GPIO_PDH;                        \
          GPIO_PDL   = POWERUP_GPIO_PDL;                        \
          /* GPIO Mode Config (see above) */                    \
          GPIO_DBG   = POWERUP_GPIO_DBG;                        \
          GPIO_CFG   = POWERUP_GPIO_CFG;                        \
          /* GPIO Output configuration */                       \
          GPIO_CLRH  = POWERUP_GPIO_CLRH; /* output states  */  \
          GPIO_CLRL  = POWERUP_GPIO_CLRL; /* output states  */  \
          GPIO_SETH  = POWERUP_GPIO_SETH; /* output states  */  \
          GPIO_SETL  = POWERUP_GPIO_SETL; /* output states  */  \
          GPIO_DIRH  = POWERUP_GPIO_DIRH; /* output enables */  \
          GPIO_DIRL  = POWERUP_GPIO_DIRL; /* output enables */  \
        } while(0)
//@} //END OF BOARD SPECIFIC FUNCTIONS

#endif //__BOARD_H__
```

decta-RfidDriver.h

```
//*****************************************************************************
// File: decta-RfidDriver.h
//
// Description:
// Contains functions to manage and operate contactless cards
//
// Authors: Mikael Högrud and Johan Riisberg-Jensen (2010)
//*****************************************************************************



//*****************************************************************************
// I2C bus ID:
//*****************************************************************************

#define RFID_DEVICE_ID (0x50)



//*****************************************************************************
//Commands:
//*****************************************************************************

//      Command       Code      Action
#define Idle          0x00      // No action; cancels current command
                                // execution
#define Mem           0x01      // Stores 25 byte into the internal buffer
#define GenRandId     0x02      // Generates a 10 byte random ID number
#define CalcCRC       0x03      // Activates the CRC co-processor or performs
                                // a selftest.
#define Transmit      0x04      // Transmits data from the FIFO buffer
#define NoCmdCHG      0x07      // No command change. This command can be used
                                // to modify different bits in the command
                                // register without touching the command
                                // E.g. Power-down.
#define Receive       0x08      // Activates reciever circuitry
#define Transceive    0x0C      // Transmits data from FIFO buffer to the
                                // antenna and activates
                                // automaticaly the receiver after transmission.
//     Reserved       0x0D      // Reserved for future use
#define MFAuthent     0x0E      // Performs the MIFARE standard autenthication
                                // as a reader
#define SoftReset     0x0F      // Resets the MFRC523



//*****************************************************************************
// Error codes:
//*****************************************************************************

#define SELECT_ERROR            0xFA
#define ANTICOLL_ERROR          0xFB
#define NOT_MIFARE_COMPLIANT    0xFC
#define NO_CARD_DETECTED        0xFD



//*****************************************************************************
// Page 0:  Command and Status:
//*****************************************************************************

//     Reg name       Adr       Function
//     Reserved       0x00      // Reserved for future use
#defineCommandReg     0x01      // Starts amd stops command execution
#defineCommlEnReg     0x02      // Controls bits to enable and disable the
                                // passing of interrupt requests
#defineDivlEnReg      0x03      // Controls bits to enable and disable the
                                // passing of interrupt requests
#defineCommIRqReg     0x04      // Contains interrupt request bits
#defineDivIRqReg      0x05      // Contains interrupt request bits
#defineErrorReg       0x06      // Error bits showing the error status of the
                                // last command executed
#defineStatus1Reg     0x07      // Contains status bits for communication
#defineStatus2Reg     0x08      // Contains status bits of the reciever and
                                // transmitter
```

```
#defineFIFODataReg    0x09      // In- and output 64 byte FIFO buffer
#defineFIFOLevelReg   0x0A      // Indicates the number of bytes stored
                                // in the FIFO
#defineWaterLevelReg 0x0B       // Defines the level for FIFO under- and
                                // overflow warning
#defineControlReg     0x0C      // Contains miscellaneous Control Registers
#define BitFramingReg 0x0D      // Adjustments for bit oriented frames
#defineCollReg        0x0E      // Bit position of the first bit collision
                                // detected on hte RF-interface
//   Reserved         0x0F      // Reserved for future use


//*****************************************************************************
// Page 1:  Command:
//*****************************************************************************

//     Reg name         Adr      Function
//     Reserved         0x10     // Reserved for future use
#defineModeReg          0x11     // Defines the modes for transmitting and
                                 // recieving
#defineTxModeReg        0x12     // Defines the transmission data rate
                                 // and framing
#defineRxModeReg        0x13     // Defines the receive data rate and framing
#defineTxControlReg     0x14     // Controls the logical behavior of the
                                 // antenna driver pins TX1 and TX2
#defineTxASKReg         0x15     // Controls the setting of the TX modulation
#defineTxSelReg         0x16     // Selects the internal sources for
                                 // the antenna driver
#defineRxSelReg         0x17     // Selects the internal receiver settings
#defineRxThresholdReg   0x18     // Selects thresholds for the bit decoder
#defineDemodReg         0x19     // Defines demodulator settings
//   Reserved           0x1A     // Reserved for future use
//   Reserved           0x1B     // Reserved for future use
#defineMfTxReg          0x1C     // Controls some MIFARE communication
                                 // transmit parameters
#defineMfRxReg          0x1D     // Controls some MIFARE communication recieve
                                 // parameters
#defineTypeBReg         0x1E     // Configure the ISO/UEC 14443 B functionality
#defineSerialSpeedReg   0x1F     // Selects the speed of the serial
                                 // UART interface


//*****************************************************************************
// Page 2:  CFG:
//*****************************************************************************

//     Reg name         Adr      Function
//     Reserved         0x20     // Reserved for future use
#defineCRCResultRegL    0x21     // Shows the actual MSB and LSB values of
                                 // the CRC calculation
#define CRCResultRegH   0x22     // Upper half of CRC Result reg
//     Reserved         0x23     // Reserved for future use
#defineModWidthReg      0x24     // Controls the settings of the ModWidth
//   Reserved           0x25     // Reserved for future use
#defineRFCfgReg         0x26     // Configures the receiver gain
#defineGsNReg           0x27     // Selects the conductance of the antenna
                                 // driver pins TX1 and TX2
#defineCWGsPReg         0x28     // for modulation
#defineModGsPReg        0x29
#defineTModeReg         0x2A     // Defines settigns for the internal timer
#defineTPrescalerReg    0x2B     // Describes the 16-bit timer reload value
#defineTReloadRegH      0x2C
#defineTReloadRegL      0x2D
#defineTCounterValRegH  0x2E     // Shows the 16-bit actual timer value
#define TCounterValRegL 0x2F


//*****************************************************************************
// Page 3:  TestRegister:
//*****************************************************************************

//     Reg name         Adr      Function
//     Reserved         0x30     // Reserved for future use
#defineTestSel1Reg      0x31     // General test signal configuration
#defineTestSel2Reg      0x32     // General test signal configuration
```

```
                                // and PRBS control
#defineTestPinEnReg     0x33    // Enables pin output driver on D1-D7
#defineTestPinValueReg  0x34    // Defines the vaules for D1-D7 when
                                // it is used as I/O bus
#defineTestBusReg       0x35    // Shows the status of the internal testbus
#defineAutoTestReg      0x36    // Controls the digital selftest
#defineVersionReg       0x37    // Shows the version
#defineAnalogTestReg    0x38    // Controls the pins AUX1 and AUX2
#defineTestDac1Reg      0x39    // Defines the test value for the TestDAC1
#defineTestDaC2Reg      0x3A    // Defines the test value for the TestDaC2
#defineTestADCReg       0x3B    // Shows the actual value of ADC I and Q
//      Reserved        0x3C    // Reserved for future use
//      Reserved        0x3D    // Reserved for future use
//      Reserved        0x3E    // Reserved for future use
//      Reserved        0x3F    // Reserved for future use


//*****************************************************************************
// Modem states:
//*****************************************************************************

#define Idle                0x00 // Ready to serve
#define WaitForStartSend    0x01 // in BitFramingReg
#define TxWait              0x02 // Waits for RF field to
                                 // be present before transmitting
#define Transmitting        0x03 // Transmission in progress
#define RxWait              0x04 // Waits for RF field to be
                                 // present before receiving
#define WaitForData         0x05 // Wait for data
#define Recieving           0x06 // Reception in progress

// ATQA responses
#define MIFARE_ULTRA_LIGHT_64   0x0044
#define MIFARE_1K               0x0004
#define MIFARE_4K               0x0002
#define MIFARE_PLUS1            0x0042
#define MIFARE_PLUS2            0x0044
#define DESFIRE                 0x0344



//*****************************************************************************
// Main functions:
//*****************************************************************************

// Playground function for testing
void qtest(void);

// Sends the reqa byte (Asks for any card in proximity that
// are awake to make their precense known)
int8u reqa(int8u *cardType);

// Runs the anticolision sequence for the selected level.
// Returns 5 valid bytes regardless of no of cards
int8u antiLoop(int8u *uID,int8u level);

// Selects a card at any partiular level of anticollision
int8u select(int8u *uID , int8u level);

// Runs the full anticollision over loop multiple levels,
// and returns the uID of the selected card.
int8u antiCollision(int8u *uID);

// Sends the halt command a card must be selected for it to work.
int8u haltCard(void);

// Authenticates communication with a selected card
int8u authenticate(  int8u AuthCommand,
                     int8u *uID,
                     int8u *sectorKey,
                     int8u blockAddress );

// Reads a sector card must furst be selected/authenticated
int8u readSectorBlock(int8u *Data, int8u sectorBlock);

// Writes a secctor card must furst be selected/authenticated
int8u writeSectorBlock(int8u *Data, int8u sectorBlock);
```

```
// Reads a secctor on a ultralight card, card must furst be selected
int8u writeUltraLightPage(int8u *Data, int8u sectorBlock);


//*****************************************************************************
// Auxillary functions:
//*****************************************************************************

// Initiates the rfid interface to its active state,
// must be run before the rfid interface is used
int8u rfidInit(void);

// Puts the rfid interface device to deep sleep in order
// to conserve battery life
void rfidHardPowerDown(void);

// Runs the self test procedure
int8u selfTest(int8u *responseData);
```

decta-soft-i2c.h

```
//*****************************************************************************
// File: decta-soft-i2c.h
//
// Description:
// Contains functions necessary for emulating an I2C bus interface using GPIO's
//
// Authors: Mikael Högrud and Johan Riisberg-Jensen (2010)
//*****************************************************************************



//*****************************************************************************
// Defines the approximate bus speed  BUS SPEED = 12Mhz / ((LIN/2) * 2^EXP )
// but cycle times for doing the register writes and checks and so on lowers
// the actual speed
//*****************************************************************************

#define EXP   3  //5
#define LIN   150 //60



// Initates the software i2c interface, mind GPIO_CFG must
// be set to allow GPIO on selected pins
void  softI2cInit (void);

// eeprom software i2c routines
int8u softI2cEe   (int8u control, int16u address, int8u *data, int8u length);

// DoorLock software i2c routine
int8u softI2cLock (int8u address ,int8u *data, int8u length);

// Suport function to test the frequency of the interface.
int8u freqTest(void);
```

decta-StorageDriver.h

```
//*****************************************************************************
// File: decta-StorageDriver.h
//
// Description:
// Contains functions for abstracting lower-level memory management
//
// Authors: Mikael Högrud and Johan Riisberg-Jensen (2010)
//*****************************************************************************



//*****************************************************************************
// 24AA256 type EEPROM:
//*****************************************************************************

#ifdef MEM_TYPE_24AA256

   // A2 =  high A1 and A0 connected to ground internally to package
   // if the package is MSOP
     #define EeDeviceId         0xA8

   // 64 bytes per page
     #define PageSize           64

   #define MaxPages             512

   #define StartAddress         0x0000

   #define StopAddress          0x7FFF

#endif



//*****************************************************************************
// 24AA1025 type EEPROM:
//*****************************************************************************

#ifdef MEM_TYPE_24AA1025

   // Where the 8 represents the block select bit and A1 A0 are
   // assumed to be set to zero as is done internally by the 24AA256
   #define EeDeviceId           0xA8

   // 128 bytes per page
   #define PageSize             128

   // Spread over two separate memory spaces as defined by the block select bit
   #define MaxPages             1024

   #define StartAddress         0x0000

   // In reality the range is 0-1FFFF but the 1 is handled by the Block select
   // bit mentioned above
   #define StopAddress          0xFFFF

#endif



//*****************************************************************************
// M45PE20 (and M25PE10-A) type Flash memory:
//*****************************************************************************

#ifdef MEM_TYPE_M45PE20

   // 256 bytes per page
     #define PageSize           256

     #define MaxPages           1024

     #define StartAddress       0x00000

     #define StopAddress        0x3FFFF
```

```
  // internal instructions for the M25PE10-A (and M45PE20):

    // Write enable
    #define WREN        0x06

  // Write disable
    #define WRDI        0x04

  // Read identification
    #define RDID        0x9F

  // Read status register
    #define RDSR        0x05

  // Write status register
    #define WRSR        0x01

  // Read data bytes
    #define READ        0x03

  // Read data bytes at higher speed
    #define FAST_READ   0x0B

  // Erase and Write combined
    #define PW          0x0A

  // Page program
    #define PP          0x02

  // Page erase
    #define PE          0xDB

  // Sector erase
    #define SE          0xD8

  // Bulk erase
    #define BE          0xC7

  // Deep power down
    #define DP          0xB9

  // Read identification and power up
    #define RES         0xAB



    // Flash status bitmasks:
    #define SRWD   0x80         // Status Register Write protect
    #define BP1    0x08         // Block Protect bit 1
    #define BP0    0x04         // Block Protect bit 0
    #define WEL    0x02         // Write Enable Latch bit
    #define WIP    0x01         // Write In Progress bit

#endif



//****************************************************************************
// Functions
//****************************************************************************

// Format the entire memory (writes ones everywhere)
int8u formatMemory(void);

// Formats a single page (writes ones everwhere)
int8u formatPage(int32u address);

// Writes a page worth of data to the device
int8u writePage(int32u address, int8u *data);

// Reads a page worth of data from the device
int8u readPage (int32u address, int8u *data);

// Reads a variable length string of bytes from the device
int8u readBytes(int32u address, int8u *data, int8u length);
```

```
// If flash memory is used then the space on memory to be
// written MUST previously be set to all ones
int8u writeBytes(int32u address, int8u *data, int8u length);

// Wakes the memory for active use
int8u wakeUpMemory(void);

// Sets the memory to sleep mode for energy conservation
int8u goToSleepMemory(void);

extern void halInternalResetWatchDog(void);
```

decta-TimeDriver.h

```
//*****************************************************************************
// File: decta-TimeDriver.h
//
// Description:
// Contains functions for abstracting real-time clock management
//
// Authors: Mikael Högrud and Johan Riisberg-Jensen (2010)
//*****************************************************************************




//*****************************************************************************
// Device specifics for Real time clock S-35390A:
//*****************************************************************************

// (Note: Time format for day of week 0-6 for this device)
#ifdef CLOCK_TYPE_S35390A

    // I2c buss id
    #define RTC_DEVICE_ID   0x60

    // Availble command calls
    #define  ReadRtcStatus1             0x01
    #define  SetRtcStatus1              0x00
    #define  ReadRtcStatus2             0x03
    #define  SetRtcStatus2              0x02
    #define  R_RtcDateAndTimeOfDay      0x05
    #define  S_RtcDateAndTimeOfDay      0x04
    #define  R_RtcTimeOfDay             0x07
    #define  S_RtcTimeOfDay             0x06
    #define  ReadRtcAlarm1              0x09
    #define  SetRtcAlarm1               0x08
    #define  ReadRtcAlarm2              0x0B
    #define  SetRtcAlarm2               0x0A
    #define  ReadRtcClockCorrect        0x0D
    #define  SetRtcClockCorrect         0x0C
    #define  ReadFreeReg                0x0F
    #define  WriteFreeReg               0x0E

#endif



//*****************************************************************************
// Device specifics for M41T65:
//*****************************************************************************

// (Note: Time format for day of week 1-7 for this device)
#ifdef CLOCK_TYPE_M41T65

    // I2C bus ID
    #define RTC_DEVICE_ID   0xD0

    // Register addresses
    #define TensAndHundredsReg          0x00
    #define SecondsReg                  0x01
    #define MinutesReg                  0x02
    #define HoursReg                    0x03
    #define DayofWeekReg                0x04
    #define DayofMonthReg               0x05
    #define CenturyMonthReg             0x06
    #define YearReg                     0x07
    #define CalibrationReg              0x08
    #define WatchDogReg                 0x09
    #define AlarmMonthReg               0x0A
    #define AlarmDayOfMonthReg          0x0B
    #define AlarmHourReg                0x0C
    #define AlarmMinReg                 0x0D
    #define AlarmSecReg                 0x0E
    #define FlagsReg                    0x0F
#endif
```

```
// struct to hold the clock time:
struct clockType
{
    int8u hour;
    int8u min;
    int8u sec;
};

// struct to hold the calendar as well as clock:
struct calendarType
{
    int8u year;
    int8u month;
    int8u date;
    int8u dayOfWeek;
    struct clockType clock;
};

// Starts the rtc with 24h clock notation:
int8u  rtcInit(void);

// Read or or write one byte to/from any register of the users choice:
int8u  setRtcReg ( int8u command, int8u *data );
int8u  getRtcReg ( int8u command, int8u *data );

// Set and get the clock and date month year and so on:
int8u  setCalendar ( struct calendarType *time );
int8u  getCalendar ( struct calendarType *time );

// Read or set only the time of day i.e. hours minutes seconds:
int8u  setTime ( struct clockType *time );
int8u  getTime ( struct clockType *time );
```

Appendix B: Secret Documentation

Firmware Source-File Implementations

Contents classified.

Demonstration-Board Documentation

Contents classified.

Demonstration-Board Schematic

Contents classified.

Demonstration-Board Layout

Contents classified.

Production-Prototype Documentation

Contents classified.

Production-Prototype Schematic

Contents classified.

Production-Prototype Layout

Contents classified.

Production-Prototype Bill-of-Materials

Contents classified.