Master Thesis

Design and Implementation of an Ultra Low Power UHF Transponder for the EPC Class-1 protocol

Martin Zechleitner

Institute of Microwave and Photonic Engineering
Graz University of Technology
Head: Univ.-Prof. Dipl.-Ing. Dr.techn Wolfgang Bösch

Supervisor: Ao.Univ.-Prof. Dipl.-Ing. Dr.techn. Erich Leitgeb
External Supervisor: Dipl.-Ing. Johannes Schweighofer

Graz, September 2013

Kurzfassung

Die Radiofrequenz-Identifikation (RFID) Technologie zählt zu einen der am schnellsten wachsenden Märkten. Eine der beiden Realisierungsarten dieser Technologie, die Fernfeldübertragung, wird in dieser Diplomarbeit behandelt. Ein Anwendungsgebiet findet diese Art der RFID Chips in Kaufhäusern oder Lagern. Durch das Electronic Product Code (EPC) Class-1 Generation 2 Protokoll wird für die Fernfeldübertragung die physischen und logischen Anforderungen spezifiziert und standardisiert.

Die Zielsetzung dieser Arbeit war die Entwicklung eines prozessorbasierten digitalen ASICs (Application-Specific Integrated Circuit) welcher mit dem EPC Class-1 Generation 2 Protokoll kompatibel ist. Um einen Einblick in den Ablauf des Protokolls zu geben werden die Zustände und Kommandos sowie deren Anwendung durch eine Beispielsequenz dargestellt. Für die Realisierung der Architektur standen zwei unterschiedliche Prozessoren zur Auswahl welche analysiert und evaluiert wurden. Um einen möglichst effizienten Chip zu entwerfen wurden verschiedene leistungssparende Techniken angewandt.
Neben der in Assembler programmierten Firmware wurden periphere Hardware Module entworfen, welche eine Schnittstelle zwischen dem Prozessor und den zu verarbeitenden Daten bilden und desweiteren Rechenarbeit vom Prozessor auslagern. Auf die entwickelte Firmware wird mittels einer Aufbaubeschreibung sowie durch graphische Beispiele eingegangen. Die Besonderheit der Firmware besteht darin, dass der Prozessor speziell für den Anwendungsbereich von RFID Chips entworfen wurde. Dadurch ergibt sich ein reduzierter Befehlssatz was dem Prozessor ein äußerst energieeffizientes Arbeiten ermöglicht.

Durch Simulationsergebnisse wird dargestellt wie das entworfene System mit den EPC Class-1 Generation 2 Protokoll interagiert. Das implementierte System wurde für einen $130\,nm$ Complementary Metal Oxide Semiconductor (CMOS) Prozess synthetisiert.

Abstract

The Radio Frequency Identification (RFID) technology is one of the fastest growing industries. One of the main components of this technology is the far field propagation, which is issued in this thesis. A typical application scenario for this kind of RFID technology is a department store or a warehouse. With the Electronic Product Code (EPC) Class-1 Generation 2 protocol the far propagation of far field is specified and standardized.

The objective of this thesis is to develop a processor based digital ASIC (Application-Specific Integrated Circuit) compatible to the EPC Class-1 Generation 2 protocol. For an insight to the flow of events of the protocol the states and commands are explained. Furthermore an example sequence is given.
For the realization of the architecture two different processors were available that had to be analyzed. To develop an efficient chip, miscellaneous low-power techniques were applied.
In addition to the in assembler programmed firmware, peripheral hardware units were developed which act as interface between the processor and the data to be processed. Furthermore computational load from the processor is outsourced in the hardware units. The composition of developed firmware is described and graphical examples of the firmware are given. The special feature of the firmware is the processor dedicated for RFID systems. By this the instruction set is reduced whereby the processor can work very energy-efficient.

Simulation results show how the implemented system interacts with the EPC Class-1 Generation 2 protocol. The implemented system had been synthesized for a $130\,nm$ Complementary Metal Oxide Semiconductor (CMOS) process.

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Place	Date	Signature

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Ort	Datum	Unterschrift

Acknowledgement

First I would like to thank Prof. Erich Leitgeb, at the Institute of Microwave and Photonic Engineering at the Graz University of Technology, for supervising my thesis as well as for his outstanding support.

Furthermore, I would like to thank Dipl.-Ing. Gerald Holweg, head of the Contactless and Radio Frequency Exploration (CRE) Department of Infineon Technologies Austria AG, for the opportunity to design and write my thesis in an industrial and practical environment.

I truly appreciate the supervision of Dipl.Ing. Johannes Schweighofer at Infineon Technologies Austria AG in Graz. His excellent guidance and constructive reviews were a great benefit and personal gain during the course of my whole thesis. I would also like to thank all my colleagues of the CRE department of Infineon Technologies Austria AG for the positive and creative work atmosphere.

Finally, I would like to express my gratitude to all of my family and especially to my parents Cäcilia and Dieter Zechleitner for their wonderful support during my whole life.

Contents

List of Abbreviations

ALU	Arithmetic and Logic Unit
ASIC	Application-Specific Integrated Circuit
ASIG	Autonomous Sense and Identification Grain
ASIP	Application-Specific Instruction-set Processor
ASK	Amplitude Shift Keying
CMOS	Complementary Metal Oxide Semiconductor
CRC	Cyclic Redundancy Check
CW	Continuous Wave
DFE	Digital Front End
DR	Divide Ratio
EBV	Extensible Bit Vectors
EEPROM	Electrically Erasable Programmable
EPC	Electronic Product Code
FSM	Finite State Machine
GE	Gate Equivalents
GS1	Global Standards One
HF	High Frequency
HW	Hardware
LF	Link Frequency
LFSR	Linear Feedback Shift Register
LUT	Look Up Table
MSB	Most Significant Bit
NFC	Near Field Communication
NVM	Non Volatile Memory
PIE	Pulse Interval Encoding
PRNG	Pseudo Random Number Generator
PSK	Phase Shift Keying
PW	Pulse Width
RAM	Random Access Memory
RF	Radio Frequency
RFID	Radio Frequency Identification
ROM	Read Only Memory
RTcal	Reader Tag calibration time

RTL	Register Transfer Level
SFR	Special Function Register
SPI	Serial Peripheral Interface Bus
TID	Tag Identifier
TRcal	Tag Reader calibration time
UHF	Ultra High Frequency
VHDL	Very High Speed Integrated Circuit Hardware Description Language

List of Figures

List of Tables

Chapter 1: Introduction

This chapter provides an overview of Radio Frequency Identification (RFID) systems, their different technologies, techniques and application areas. Furthermore the goal and the motivation for this project are given as an outline of the thesis.

1.1 RFID Systems

The RFID technology is based on integrated circuits which are able to communicate contactless with a reader. In contrast to contact based chips like these on a cash card are very prevalent in the modern environment, the contactless chips give a new scope to these applications. New cellular phones already provide an RFID interface, which can be seen as an enabler for contactless technology.
According to [Finkenzeller 2002] the RFID market is one of the fastest growing markets. Due to the shared communication medium, a strong anti-collision system has to be provided, in addition to a cryptographic system providing the security of personal data.

The most simple RFID system consists of two components, a reader and a tag. The reader provides the energy for the tag and queries the tag. Since a single reader can communicate with a bunch of tags, the efficient design for a low-power and small area is more important for the tag than for the reader [Finkenzeller 2002].

Furthermore RFID tags can be subdivided in two classes: active RFID chips and passive RFID chips. On the one hand active tags have an own power supply in form of on-chip batteries or a direct connected supply. On the other hand passive RFID chips gain their energy out of the field from reader. A combination between these classes is a semi passive RFID tag. These tags gain their energy out of the field from the reader as passive RFID tags. For the receiving mechanism a semi passive chip uses a power source on the chip. By this technique the range of the tag can be increased. The most interesting type of RFID tags are the passive tags since they are independent to any power supply and their lifetime is not limited by any battery [Finkenzeller 2002].

The data exchange of RFID tags depends on the used frequency, more in detail whether the tag is in the near field or in the far field of the reader. The technique how the chip sends

data back to the reader is defined by the difference between near field and far field. In the case of a near field tag, the chip gains the power supply out of the magnetic field of the reader. By doing a load modulation, the reader can detect a difference in the used current for generating the field. The technology for a far field chip is called backscattering. Since the chip is in the far field, the weak magnetic field cannot supply the chip. Therefore the electro-magnetical field is gained by an dipole antenna. In the backscattering technique the chip changes the antenna coefficient by switching a load onto the antenna. This change in the antenna coefficient reflects the electro-magnetic wave towards the reader. The reader can detect this small change in the sent signal and generates the data from the tag out of it [Want 2006] [Finkenzeller 2002].

The near field tags and far field tags are used in different application scenarios. Since the working distance of the reader to the tag in the near field is very small compared to the far field, the near field supports usually weak anticollision routine. Contrary to the near field, at the far field many chips can respond to the reader. This calls for a strong anticollision scheme. In the near field more power can be transmitted to the tag. Due to this fact, and the short anticollision scheme, a higher data rate can be achieved in comparison to the far field. The benefit of the far field tags is the long reading range [Finkenzeller 2002].
By this basic principles the near field and far field tags find different applications. The near field technology is used for personal identification, ticketing systems and even as an electronic wallet. Generally, the near field technology is used in case of objects near to the reader. Contrarily, the far field technology is predestinated to detach the printed bar code. Far field chips are very interesting for storage management. One of the most prominent example for the utilization of the far field tags is a supermarket. Here the consumer just pass a reader which computes the prices of all purchased tagged products without any queues. Dependent on the application a near field or far field chip can be used. Commonly the NFC (Near Field Communication) protocol is used for the near field communication. Contrarily to the far field main protocol is the EPC (Electronic Product Code). This thesis issues the implementation of the EPC protocol [Want 2006] [Finkenzeller 2002].

1.2 Motivation and Project Goals

The goal of the project was the implementation of a processor based solution for the EPC Global Class-1 Generation-2 Ultra High Frequency (UHF) RFID Standard. The state before the thesis was a small and fast hardware based chip with a huge finite state machine as core unit. Due to its complexity this state machine became unmaintainable. Therefore a slower but more flexible and adaptable implementation was desired. This solution is a processor based approach which guarantees a higher hardware independence.

1.3 Thesis Outline

In chapter 2 the EPC Global Class-1 Generation-2 UHF RFID Standard is introduced and explained.
Chapter 3 gives an overview of the ASIG (Autonomous Sense and Identification Grain) project where the work of this thesis is included for one of the two communication protocols.
The planning of the implementation and design criteria is the topic of chapter 4.
The register transfer level design and every implemented unit are issued in chapter 5.
Chapter 6 deals with the firmware, the flow of the commands and gives an analysis of the implemented code. Furthermore the timing of the firmware is analyzed and the firmware is explained with two examples.
The simulation, the test bench and some examples are shown in chapter 7.
The output of the synthesis is given in chapter 8.
Finally in chapter 9 concluding remarks are done.

Chapter 2: EPC Global Class-1 Generation-2 UHF RFID Standard

This chapter summarizes and explains the most important information of the Global Class-1 Generation-2 UHF RFID Standard in respect of this thesis.

2.1 General

The EPC intends to generate a universal identifier for any objects. Developed by the EPC-global, a part of the Global Standards One (GS1), the EPC is the state of the art standard for UHF RFID systems. Compared to High Frequency (HF) systems the increased operating range enables UHF systems to operate in a new field of applications. Just one example of many is a warehouse inventory management system.
Every RFID system consists of at least one reader, also known as interrogator, and at least one tag, also known as label or transponder. These components communicate contactless with each other [Chawla and Ha 2007] [EPCGlobal 2005].

Class-1 tags are transponders which backscatter their information passively to the reader. An integrated memory stores data like the EPC and a Tag Identifier (TID). The Class-1 tags support a kill function to disable a tag. A password protected access control and a user memory is declared as optional in the standard. Compared to the other classes like Class-2 tags or Class-4 tags, the Class-1 tags support just the basic functions of a transponder. As defined by the protocol the reader has to initiate the communication. Since the tag is in range of a reader, it is powered by the Radio Frequency (RF) field of the reader. With the backscattering mechanism data is sent from the tag to the interrogator. While the tag is backscattering data, the reader sends a Continuous Wave (CW) RF signal [EPCGlobal 2005].

According to [Finkenzeller 2002] the backscattering mechanism is described as follows, where P1' is the power received from the tag and P2 is the power sent from the tag:

> "A proportion of the incoming power P1' is reflected by the antenna and returned as power P2. The reflection characteristics (= reflection cross-section) of the antenna can be influenced by altering the load connected to the antenna. In order to transmit data from the transponder to the reader, a load resistor R_L connected in parallel with the antenna is switched on and off in time with the stream to be transmitted. The amplitude of the power P2 reflected from the transponder can thus be modulated (modulated backscatter)." [Finkenzeller 2002]

Since the tags needs to be powered by the reader, the communication is half-duplex. Interrogators modulate their data by double-sideband amplitude shift keying, single-sideband amplitude shift keying or phase-reversal shift keying. The data from the reader to the tag is pulse interval encoded (PIE). Tags encode their data in FM0 or Miller and modulate the data by amplitude shift keying (ASK) or phase shift keying (PSK) [Chawla and Ha 2007] [EPCGlobal 2005].

2.2 Encoding

The different encoding schemes from the interrogator and the tag are shown here.

2.2.1 Reader to Tag

The pulse interval encoding is shown in figure 2.1. A Tari defines the duration of a data-0 sequence and is in the range between 6.25 μs and 25 μs. The data-1 sequence is defined by the Tari time value multiplied with 1.5 to 2. The Pulse Width (PW) is given by the Tari time value multiplied with 0.265 to 0.525, but must exceed at least 2 μs [EPCGlobal 2005].

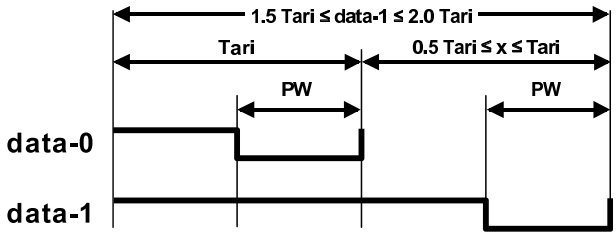

Figure 2.1: PIE symbols, [EPCGlobal 2005]

At the beginning of every transmission from the interrogator to the tag the frame-sync is sent. This sequence contains a delimiter, a data-0 field and the Reader to Tag calibration time (RTcal), see figure 2.2. At the Query command a special sequence is sent instead of the frame-sync. This sequence is called preamble and equals the frame sync plus Tag to Reader calibration time (TRcal) [EPCGlobal 2005].
The intent for the symbols RTcal and TRcal are further explained in the timing section 2.3.

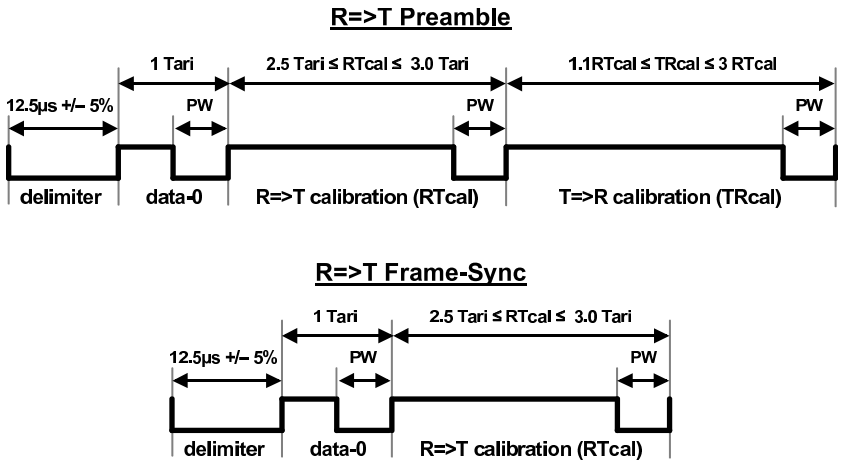

Figure 2.2: RT preamble and frame-sync, [EPCGlobal 2005]

2.2.2 Tag to Reader

Tags can backscatter their data in four different encoding schemes: FM0, Miller2, Miller4 and Miller8. The interrogator selects one of these four encoding schemes in the Query command, declared in section 2.6.2 [EPCGlobal 2005].

FM0

In the FM0 encoding a data-0 is defined by inverting the signal in the middle of the symbol. A data-1 is defined as a zero or one for the whole symbol duration. After every symbol the base band phase is inverted. The duty cycle of two identical bits encoded should have a nominal value of $50\,\% \pm 5\,\%$ [EPCGlobal 2005].

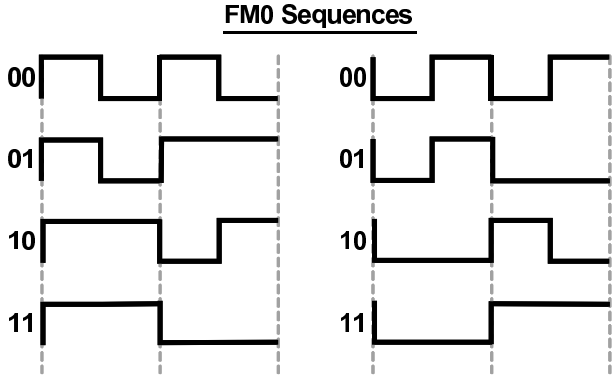

Figure 2.3: FM0 sequences, [EPCGlobal 2005]

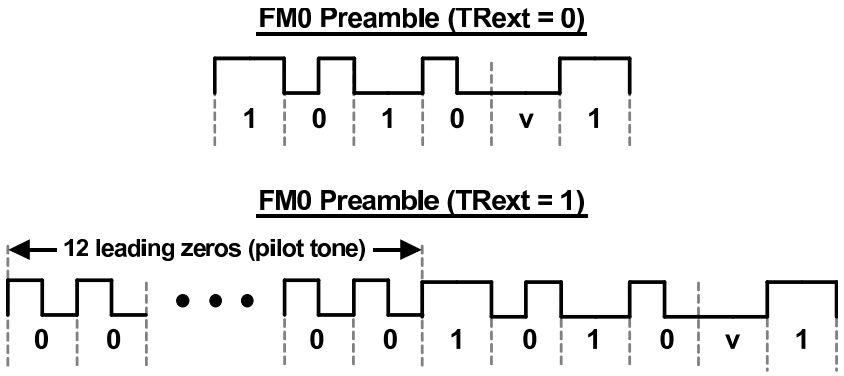

Figure 2.4: FM0 preamble, [EPCGlobal 2005]

Before the data transition starts the FM0 preamble (see figure 2.4) has to be sent. This preamble can be extended by the reader with 12 leading encoded data-0 bits. The "v" in figure 2.4 indicates a FM0 violation. At the end of a FM0 sequence the FM0 End-Of-Signaling is added, a data-1 bit encoded [EPCGlobal 2005].

Miller

The Miller encoding scheme uses phase inversion for transferring data. A data-0 bit is encoded by no phase inversion. A data-1 bit is encoded by inverting the phase in the middle of the symbol. Between two data-0 bits the phase is also inverted [EPCGlobal 2005].

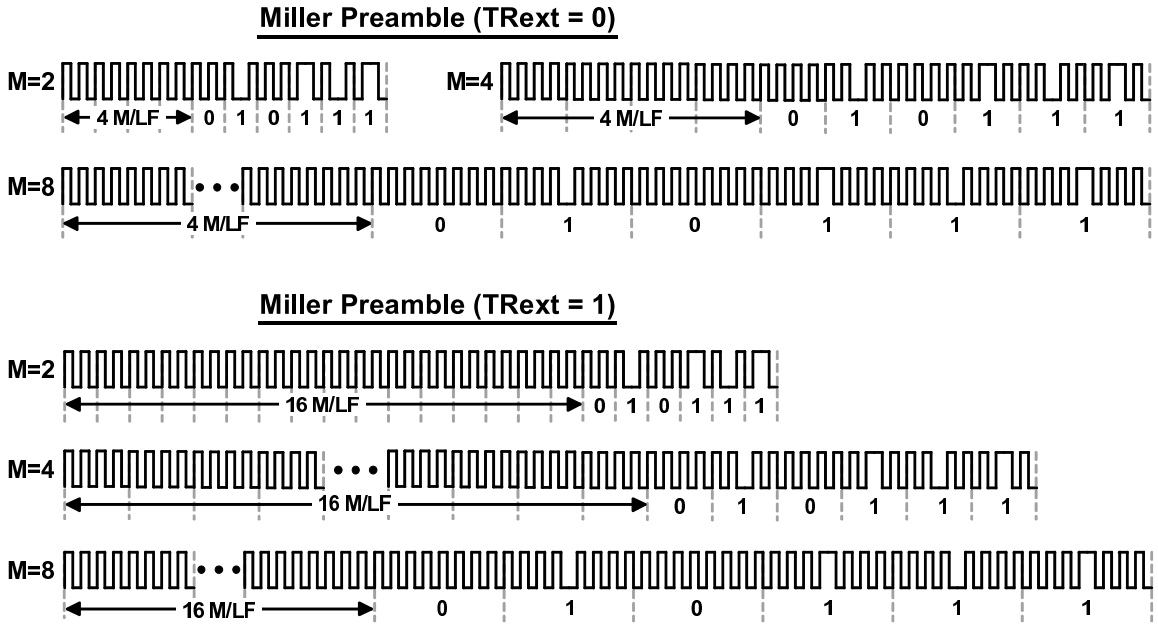

Figure 2.5: Miller preamble, [EPCGlobal 2005]

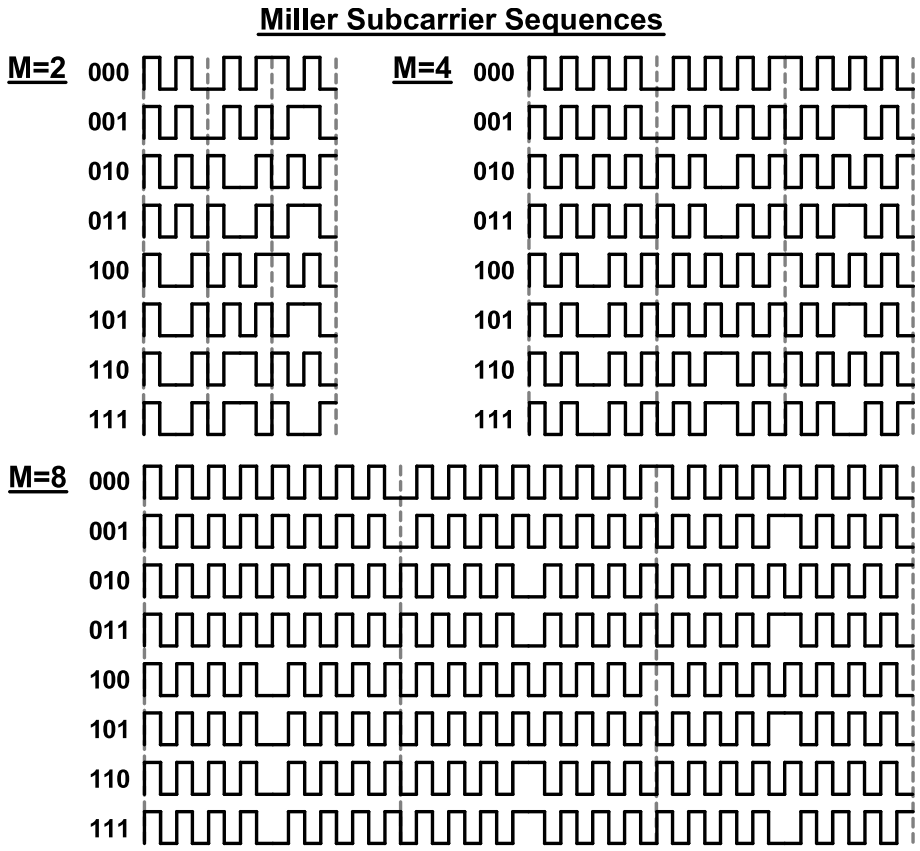

Figure 2.6: Miller sequences, [EPCGlobal 2005]

Depending on the modulation index determined by the Query command, declared in section 2.6.2, the Miller sequence contains two, four or eight sub carrier cycles per bit. The duty cycle of two identical bits encoded should have a nominal value of $50\% \pm 5\%$. In figure 2.6 the Miller preamble is depicted. The reader can extend the preamble by sending a TRext = 1 at the Query command. The extended preamble starts with 12 leading not encoded symbols. After the extension the preamble as shown in figure 2.6 with TRext = 0 is sent. At the end of a Miller encoded sequence an encoded data-1 bit is added [EPCGlobal 2005].

2.3 Timing

In the standard four times for the communication are defined by $t1$, $t2$, $t3$ and $t4$.

t1 is the most important time for a tag. It defines the time when a tag has to respond to a command sent from the reader. To meet the requirements of the standard the maximal value of $t1$ must not be exceeded as well as the minimal value of $t1$ must not be undercut. $t2$ is the time a reader has to wait to send a new command after a response from the tag has been received. In case of a non responding tag, the reader has to wait for the time $t3$ as long as there is no Lock, Kill or Write command. For these commands the $t3$ is extended since they write to non volatile memory (NVM) and this operation takes more time than $t3$. More details for this case can be seen in the section 2.6.3. $t4$ is the minimum time a reader needs to wait between two commands.

Equation 2.1 shows the composition of $t1_{max}$. RT_{cal} is the Receiver to Tag calibration time and T_{pri} is the link period, defined by equation 2.3 where LF is the link frequency, declared at equation 2.2. The term FT is the frequency tolerance which depends on the used frequency and varies from $\pm 4\,\%$ to $\pm 22\,\%$. To reduce the variables to the initial values RT_{cal} and TR_{cal} the link frequency is defined by equation 2.2 where DR is the divide ratio. The divide ratio is set by the Query command, declared in section 2.6.2, to 8 or 64/3. Equation 2.2 shows that the link frequency depends on the divide ratio and on TR_{cal} [EPCGlobal 2005].

$$t1_{max} = max(RT_{cal}, 10 \times T_{pri} \times (1 + FT) + 2\,\mu s) \tag{2.1}$$

$$LF = \frac{DR}{TR_{cal}} \tag{2.2}$$

$$T_{pri} = \frac{1}{LF} \tag{2.3}$$

Assumed a tag has a response time of $40\,\mu s$, the resulting maximum link frequency is $200\,kHz$ for a divide ratio of 8 (see equation 2.4) and $533.\overline{3}\,kHz$ for a divide ratio of 64/3 (see equation 2.5) [EPCGlobal 2005].

$$LF = \frac{DR}{TR_{cal}} = \frac{8}{40\,\mu s} = 200\,kHz \tag{2.4}$$

$$LF = \frac{DR}{TR_{cal}} = \frac{\frac{64}{3}}{40\,\mu s} = 533.\overline{3}\,kHz \tag{2.5}$$

Since the protocol has 12 mandatory commands, the response time varies for every command. The longest response time has to be taken into account to set the maximal RT_{cal}. The maximal link frequency is calculated in equation 2.2 and depends also on RT_{cal} since TR_{cal} is a variable multiplication of RT_{cal}. The data rate depends beside this on the encoding of the data. At the FM0 encoding scheme the bit rate equals the link frequency, the bit rate at Miller2 is half the link frequency, Miller4 uses $\frac{LF}{4}$ as bit rate and the bit rate at Miller8 is $\frac{LF}{8}$ [EPCGlobal 2005].

2.4 Memory

The standard describes no maximal or minimal memory amount, only the format of the memory is given. The tag memory is divided into four memory banks.[EPCGlobal 2005]

Starting at the address 00_h the memory bank $RESERVED$ includes a 16 bit password for the kill command and a 16 bit access password. An explanation for the use of the passwords is given in the commands section 2.6. The $RESERVED$ memory bank can be expanded optionally.
The second memory bank is the EPC bank. Here the 16 bit Cyclic Redundancy Check (CRC-16), Protocol Control (PC) and EPC is stored. The CRC-16 is the actual checksum. The PC bits contain a definition of the length of the EPC and a numbering system identifier (NSI). The NSI is an EPCglobal header as defined in the EPC Tag standard [EPCGlobal 2011]. Optional the NSI bits can be set zero. The EPC consists on the one hand of up to 31 words, where one word equals 16 bits. On the other hand the minimum size of the EPC is zero words.
In the TID memory bank an 8 bit class identifier, a 12 bit tag mask designer identifier and a 12 bit tag model number is stored. The TID memory bank can be expanded optionally.
As fourth and last memory bank the $USER$ bank contains user or application specific data with no constraints from the protocol [EPCGlobal 2005].

All data is stored with the most significant bit (MSB) first. Logically every memory bank begins at address zero (00_h). Figure 2.7 depicts the memory map of the tag memory.

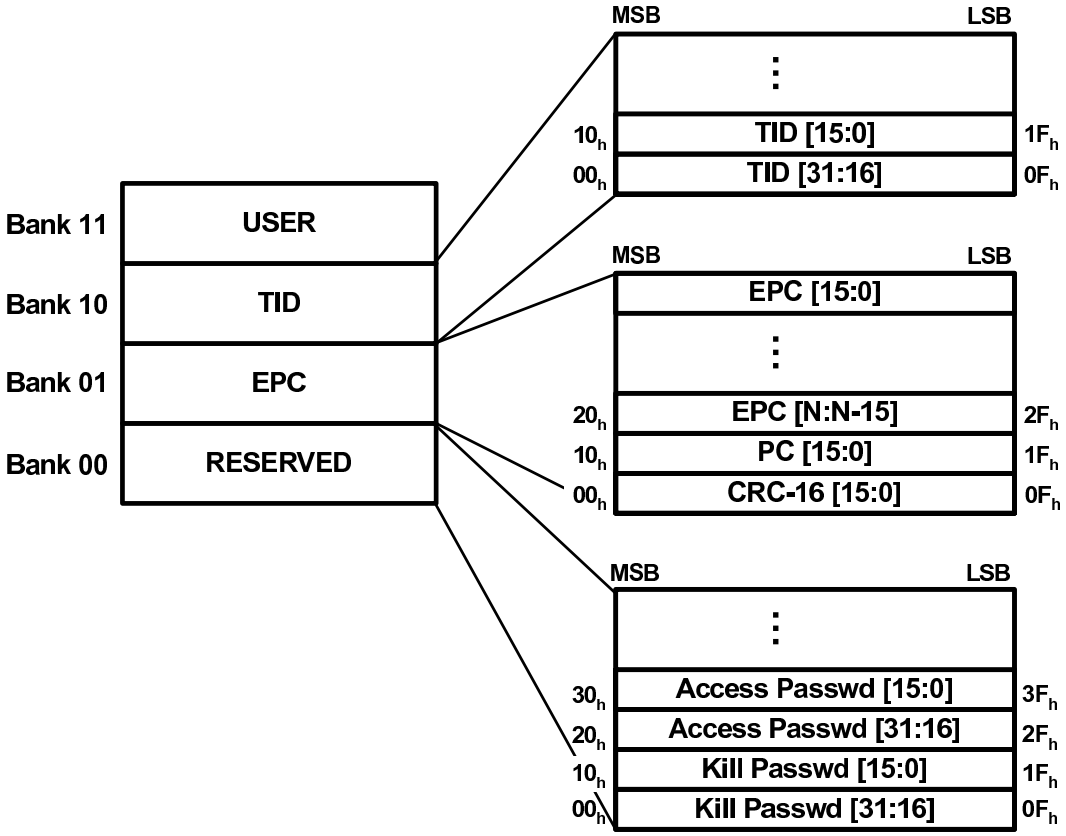

Figure 2.7: Memory map, [EPCGlobal 2005]

Every memory bank except the *RESERVED* bank can be locked by the Lock command 2.6.3 which cause the tag backscattering an error code whether a data operation of a locked memory bank is requested. For the *RESERVED* memory bank the Lock command can lock each of the two passwords separately [EPCGlobal 2005].

Beside the tag memory some other memory elements are needed for the protocol adherence. Tags can be in four logical sessions, S0, S1, S2 and S3 with an inventoried flag each. Every session's inventoried flag can be set to session A or session B.
Another memory element is the selected flag. This flag is binary, so the reader can set or reset a tag's selected flag. Afterward the reader chooses all selected or all not selected tags. For the anti collision routine a 16 bit slot counter affects the decision whether a tag should reply to the reader or not. The slot counter is loaded with a 16 bit random number. To make the anti collision more adaptable just 2^Q bits of the random number get loaded into the slot counter. The Q value consists of four bits which are set at the Query command 2.6.2. A Pseudo Random Number Generator (PRNG) is needed for the slot counter, the handle and for replying random numbers. The data denoted as handle is the random number

which was generated at the last state transition from Acknowledged to Open or Secured [EPCGlobal 2005].

2.5 States

The EPC protocol consists of seven states: Ready, Arbitrate, Reply, Acknowledged, Open, Secured and Killed.
Ready is the first state and also the initial state which is entered when the tag is energized. A tag state transition can only occur whether a valid command has been received successfully. For anti collision in the inventory round the slot counter has to equal zero to enter a higher state. A Query command 2.6.2 with matching parameters is needed to change the state to Arbitrate or Reply. In case of a change to the state Arbitrate the internal slot counter exceeds zero. Otherwise, in case of the internal slot counter equals zero, the state changes to Reply. A state change from Arbitrate to Reply occurs again in case that a slot counter equals zero. This can be achieved by one of the commands Query, Query Reply or Query Adjust [EPCGlobal 2005].

In the Reply state the interrogator can send a valid Acknowledge command 2.6.2 to enter the state Acknowledged. In case of an invalid Acknowledge command the state changes back to Arbitrate. From Acknowledged every state except Killed is reachable. Whether an Inventory command (see 2.6.2) is sent from the reader the tag cannot reach any higher state than Acknowledged. The state of the tag will fall back to the inventory states Ready, Arbitrate or Reply [EPCGlobal 2005].

With the command Request Random Number 2.6.3 the states Open or Secured can be reached in case of the actual state is Acknowledged. In both cases the reader has to send a valid random number for a valid command. By this precondition, the state changes to Open whether the access password is not zero, also denoted as not empty. In case of an empty access password the state changes to Secured. The states Open and Secured enable the reader to send a Kill, Read or Write command to interact with the memory of the tag. Only in the Secured state the lock bits of the internal memory can be modified. Except this the tag acts in the Secured state the same as in the Open state [EPCGlobal 2005].

The last state is the Killed state. This state can be reached from the states Open and Secured. To enter the Kill state the reader has to send the 32 bit kill password to the tag. In the case a tag enters the Kill state the state remains the same and the tag does not interact with the environment any more [EPCGlobal 2005].

2.6 Commands

Additionally to the 11 in the standard defined commands two custom commands were realized, Test Read and Test Write. Every command has an individual command code. In the standard four classes of encoded commands exist: 2 bit commands, 4 bit commands and 8 bit commands. According to the standard also 16 bit commands are possible whether they are needed but these are not further used here [EPCGlobal 2005].
All the implemented commands are listed in table 2.1 with their specific command code and command length.

Command	**Code (binary)**	**Length (bits)**
Query Reply	00	4
Acknowledge	01	18
Query	1000	22
Query Adjust	1001	9
Select	1010	>44
Reserved for future use	1011	-
Not Acknowledge	11000000	8
Request Random Number	11000001	40
Read	11000010	>57
Write	11000011	>58
Kill	11000100	59
Lock	11000101	60
Test Read	11100001	>57
Test Write	11100010	>58

Table 2.1: Command overview, [EPCGlobal 2005]

2.6.1 Select Command

The purpose of the Select command is to pick a certain amount of desired tags out of a bunch of tags. After the Select command the reader can interact with the selected tag(s). All other tag(s) ignore following inventory or access commands [EPCGlobal 2005].

Select

Before any inventory round starts the Select command can choose tags to respond. With the Select command the session of the tag can be set to one of the four possible sessions. The inventoried flag of the tag can be set or reset by the Select command. The Select command sends a certain amount of data which is compared to the data in the internal memory of the tag. In case of the sent data equals the data in the memory the action matches. By this the above mentioned flags are set or reset in respect of table 2.3. In case of the truncate bit is set the Acknowledge command 2.6.2 should respond shorter or individually. Beside a set truncate bit, the target bits of the Select command need to be binary 100 and the compared sent data has to end in the *EPC* memory bank to generate a valid truncate. The whole Select command, as shown in table 2.2, is protected by a 16 bit CRC checksum. This checksum starts at the first command bit and ends at the truncate bit [EPCGlobal 2005]. The Select command is listed in table 2.2 and table 2.3.

	Command	**Target**	**Action**	**MemBank**	**Pointer**	**Length**	**Mask**	**Truncate**	**CRC**
# of bits	4	3	3	2	EBV	8	Variable	1	16
description	1010	000: Inventoried (S0) 001: Inventoried (S1) 010: Inventoried (S2) 011: Inventoried (S3) 100: SL 101: RFU 110: RFU 111: RFU	See Table 2.3	00: RFU 01: EPC 10: TID 11: User	Starting Mask address	Mask length (bits)	Mask value	0: Disable truncate 1: Enable truncate	

Table 2.2: The Select command in detail, [EPCGlobal 2005]

Action	**Matching**	Non-Matching
000	assert SL or inventoried -> A	deassert SL or inventoried -> B
001	assert SL or inventoried -> A	do nothing
010	do nothing	deassert SL or inventoried -> B
011	negate SL or (A -> B, B -> A)	do nothing
100	deassert SL or inventoried -> B	assert SL or inventoried -> A
101	deassert SL or inventoried -> B	do nothing
110	do nothing	assert SL or inventoried -> A
111	do nothing	negate SL or (A -> B, B -> A)

Table 2.3: Tag response to Action parameter, [EPCGlobal 2005]

2.6.2 Inventory Commands

In the inventory round the reader identifies tags which were selected by the Select command. The inventory round is the period between successive Query commands. Since more than one tag can be selected, the reader may detect and resolve a collision. In case of a non resolvable collision the reader issues a Query Adjust command, a Query Reply command or a Not Acknowledge command. With this commands the tags return to the Arbitrate state and wait for a new command to reach the Reply state. By this procedure the reader can communicate with the desired tags [EPCGlobal 2005].

Query

The Query command defines the data encoding type of the tag by the parameters DR, M and TRext. For ensuring to modify the selected tag(s), the Select, Session and Target bits, sent from the reader, has to match the internal flags of the tag. The Q value is for the anti collision mechanism. The reader can send a Q value from 0 up to 15. This value is set to the power of two and defines how many bits of the random number are loaded into the slot counter. In case that the slot counter equals zero a random number is transmitted to the reader and the state changes to Reply. The slot counter can be zero whether the Q value is zero, the random number is zero or the valid bits of the random number are zero. By this the tag response probability can be set from $2^0(1)$ to $2^{-15}(0.000031)$. Depending on the tags in the field of the reader the Q value can vary. Whether the reader assumes only a few tags, a low Q value gives a faster response from the tags. On the contrary, in case of a huge amount of tags is assumed in the field, a high Q value prevents collisions.
In the states Acknowledged, Open and Secured the Query command causes a tag to invert the inventoried flag in case of the session parameter matches the session in the inventory round [EPCGlobal 2005].
The Query command is listed in table 2.4 and table 2.5.

	Command	**DR**	**M**	**TRext**	**Sel**	**Session**	**Target**	**Q**	**CRC-5**
# of bits	4	1	2	1	2	2	1	4	5
description	1000	0: DR = 8 1: DR = $\frac{64}{3}$	00: M = 1 01: M = 2 10: M = 4 11: M = 8	0: No pilot tone 1: Use pilot tone	00: All 01: All 10: ~SL 11: SL	00: S0 01: S1 10: S2 11: S3	0: A 1: B	0-15	

Table 2.4: Query command in detail, [EPCGlobal 2005]

	Response
# of bits	16
description	RN16

Table 2.5: Query command response, [EPCGlobal 2005]

Query Adjust

The main purpose of the Query Adjust command is to adjust the Q value. The reader can increment or decrement the number of valid bits of the random number in the slot counter. At every new Query Adjust command the valid bits of a new random number are loaded into the slot counter. Only whether the session the reader sent matches the session of the tag the Query Adjust command is valid and evaluates the Up/Down bits. In case of the slot counter equals zero in the inventory round, the tag changes its state to Reply and backscatters a random number. On the other hand, the state remains the same whether the slot counter does not equal zero.
In the states Acknowledged, Open and Secured the Query Adjust command causes a tag to invert the inventoried flag in case of the session parameter matches the session in the inventory round [EPCGlobal 2005].
The Query Adjust command is listed in table 2.6 and table 2.7.

	Command	**Session**	**UpDn**
# of bits	4	2	3
description	1001	00: S0 01: S1 10: S2 11: S3	110: Q = Q + 1 000: No change to Q 011: Q = Q - 1

Table 2.6: Query Adjust command in detail, [EPCGlobal 2005]

	Response
# of bits	16
description	RN16

Table 2.7: Query Adjust command response, [EPCGlobal 2005]

Query Reply

The Query Reply command decrements the slot counter. While the Query Adjust modifies the Q value, the Query Reply modifies the slot counter directly. By this the slot counter gets affected bitwise. As at the Query Adjust command, the session sent from the reader has to match the session bits of the tag to perform an action. In case of a slot counter which equals zero after the decrementation in the inventory round, the tag backscatters a random number and enters the state Reply. On the contrary the tag gives no response and does not change its state.
In the states Acknowledged, Open and Secured the tag inverts the inventoried flags whether the sessions parameter matches the session in the inventory round [EPCGlobal 2005].

	Command	**Session**
# of bits	2	2
description	00	00: S0 01: S1 10: S2 11: S3

Table 2.8: Query Reply command in detail, [EPCGlobal 2005]

	Response
# of bits	16
description	RN16

Table 2.9: Query Reply command response, [EPCGlobal 2005]

The Query Reply command is listed in table 2.8 and table **??**.

Acknowledge

An Acknowledge command forces a tag to identify itself to the reader in case of it is selected. To perform this the tag has to be in the Reply state or any state above (Acknowledged, Open or Secured). The reader selects the tag by sending the random number which was backscattered when the tag changed the state from Arbitrate to Reply. Since this random number should be unique, only one tag shall reply to the Acknowledge command. In the states Ready and Arbitrate an Acknowledge command has no effect on the tag. The Acknowledge command leads from the inventory round to the Access commands. Whether a tag is already in the Open or Secured state the sent 16 bit value of the command has to be the handle instead of the random number.
The backscattered data depends on the truncate bit from the Select command. In case of the reader has sent a valid truncate the tag will respond with a truncated EPC. In this case the first five bits of the reply data are defined as five zero bits. Then the EPC in the memory after the truncate EPC is sent back. The response is protected by a CRC-16 checksum. In case of a non truncated response the whole EPC with the PC bits is backscattered to the reader. The length of the EPC is defined with the PC bits [EPCGlobal 2005].
The Acknowledge command is listed in table 2.10 and table 2.11.

Not Acknowledge

The command Not Acknowledge sets the tag back into the inventory round. At the states Ready and Killed the state remains the same. In any other state the next state becomes Arbitrate. By this the tag needs again a slot counter of zero to reach the Reply state which is a precondition for any access commands [EPCGlobal 2005].

	Command	**RN**
# of bits	2	16
description	01	Echoed RN16 or handle

Table 2.10: Acknowledge command in detail, [EPCGlobal 2005]

	Response
# of bits	21 to 528
description	{PC, EPC, CRC16} or {00000_2, truncated EPC, CRC16 }

Table 2.11: Acknowledge command response, [EPCGlobal 2005]

The Not Acknowledge command is listed in table 2.12.

	Command
# of bits	8
description	11000000

Table 2.12: Not Acknowledge command in detail, [EPCGlobal 2005]

2.6.3 Access Commands

The access commands are executed by tags in the Secured or Open state. The Request Random Number command can also be performed by tags in the Acknowledge state to reach the states Open or Secured. Except Request Random Number all access commands have to reply with a zero header in case of a successful command. In case of an error, the tag backscatters a data-1 header followed by an error code, the handle and a CRC-16 checksum. For the commands Write, Kill and Lock a tag replies with the extended preamble (TRext = 1) even whether the TRext parameter at the Query command is not set [EPCGlobal 2005].

Request Random Number

Tags in the Acknowledged state need a Request Random Number command with the last backscattered random number to enter the Open or Secured state. Depending on the password in the memory of the tag, the next state is Secured in case of the access password equals zero or Open whether the password does not equal zero. Since the Access command is not implemented in the firmware, the only way to enter the Secured state is a zero access password. In the case that the random number equals the last from the tag sent random number and the CRC-16 matches, the tag loads a new random number from the random number generator, saves this random number as handle and backscatters the generated handle.
In the states Open and Secured the reader sends the handle instead of the last random number from the inventory round. The new random number, which is backscattered from the tag, will not be saved as new handle. This random number is needed for the commands Kill and Write. In case of a Request Random Number command in the Open or Secured state the state will not be modified [EPCGlobal 2005].
The Request Random Number command is listed in table 2.13 and table 2.14.

	Command	**RN**	**CRC**16
# of bits	8	16	16
description	11000001	Prior RN16 or handle	

Table 2.13: Request Random Number command in detail, [EPCGlobal 2005]

	RN	**CRC**16
# of bits	16	16
description	New RN16 or handle	

Table 2.14: Request Random Number command response, [EPCGlobal 2005]

Read

The Read command sends the read data back to the interrogator as long as it executes successfully. The command reads from all memory banks of the NVM. A valid Read command

needs the actual handle of the tag and a valid CRC-16 checksum. Since all memory banks start virtually at address zero, the MemBank parameter gives the offset for the physical memory address. The parameter WordPtr is the word pointer which gives the exact start address of the first word which should be read. To know how many words the command should read out of the memory the WordCount value gives the quantity of required words. The tag only replies to the command whether the Lock bits of the read memory are not set to read-locked. An error is backscattered in case of the address of the required memory words does not exist.
Since the Read command is an access command it can only be executed from tags in the Open or Secured states [EPCGlobal 2005].
The Read command is listed in table 2.15 and table 2.16.

	Command	**MemBank**	**WordPtr**	**WordCount**	**RN**	**CRC**16
# of bits	8	2	EBV	8	16	16
description	11000010	00: Reserved 01: EPC 10: TID 11: User	Starting address pointer	Number of words to read	handle	

Table 2.15: Read command in detail, [EPCGlobal 2005]

	Header	**Memory Words**	**RN**	**CRC**16
# of bits	1	Variable	16	16
description	0	Data	handle	

Table 2.16: Read command response, [EPCGlobal 2005]

Write

Tags can write to all memory banks with the Write command. Whether the lock bits of the memory indicate a write locked memory bank which should be written an error code is backscattered. A valid Write command needs a preceding Request Random Number command. The data which should be written to the memory has to be xor-ed with the new random number. Since writing data to the memory needs more time than T1, the maximal response time for a Write command is defined by $20\,ms$. Whether this time is exceeded, the interrogator expects the Write command as not successful and continues with any command. Since the Write command sends just one word to write, there is no word counter needed.
The Write command response has to start with the extended preamble. Since the Write command is an access command it can only be executed from tags in the Open or Secured states [EPCGlobal 2005].
The Write command is listed in table 2.17 and table 2.18.

	Command	**MemBank**	**WordPtr**	**Data**	**RN**	**CRC**16
# of bits	8	2	EBV	16	16	16
description	11000011	00: Reserved 01: EPC 10: TID 11: User	Address pointer	RN16 xor word to be written	handle	

Table 2.17: Write command in detail, [EPCGlobal 2005]

	Header	**RN**	**CRC**16
# of bits	1	16	16
description	0	handle	

Table 2.18: Write command response, [EPCGlobal 2005]

Kill

A successful Kill command sets the state of the tag to the Killed state where no more commands get executed by the tag. By this the tag is permanently disabled. The whole kill procedure consists of a Request Random Number command followed by the first Kill command. After this again a Request Random Number command has to be sent followed by the second Kill command. In case of all these commands succeed the tag is killed. Since the kill password has 32 bit, the first Kill sends the most significant 16 bit of the password and the second Kill command sends the least significant bits. The password bits get xor-ed with the requested random number. Whether the Kill procedure is interrupted by any other command, except a Query, the tag falls back into the Arbitrate state.
As response to the first Kill command the tag sends no header bit since the Kill procedure has not been finished. At the second Kill command a logically zero header is the start of the backscattered data. Tags respond to the first Kill command within the limit of T1 and with a TRext as set from Query. At the second Kill command the response has to be sent within $20\,ms$ and with the extended preamble. Only tags with a nonzero kill password can be killed. In case of a kill password which equals zero the tag backscatters an error code as response to the Kill command [EPCGlobal 2005].
The Kill command is listed in table 2.19 table 2.20 and table 2.21.

	Command	**Password**	**RFU**	**RN**	**CRC**16
# of bits	8	16	3	16	16
description	11000100	($\frac{1}{2}$ kill password) xor RN16	000_2	handle	

Table 2.19: Kill command in detail, [EPCGlobal 2005]

	RN	**CRC**16
# of bits	16	16
description	handle	

Table 2.20: Kill command response to the first Kill command, [EPCGlobal 2005]

	Header	**RN**	**CRC**16
# of bits	1	16	16
description	0	handle	

Table 2.21: Kill command response to a successful Kill command, [EPCGlobal 2005]

Lock

Every memory bank can be locked individually by the lock bits. These lock bits are set by the Lock command. The lock bits consist of two bits for the memory banks EPC, TID and $USER$. Furthermore the kill password and the access password in the $RESERVED$ memory bank have two lock bits each. These bits added together result in ten lock bits. The Lock command consists of the payload followed by the handle and the CRC-16 checksum. The payload has a mask and an action field. By the mask the Lock command can modify each bit separately and leave all other bits unchanged. The action field describes the action which should be taken in case of the mask is set. The action bits are listed in table 2.24.
The lock bits have different actions for the passwords and the other memory banks. Since the other memory banks should be readable all the time, the action bits just configure the write ability of these memory banks. For the passwords in the $RESERVED$ memory bank also reading can be prohibited by the lock bits.
The combination of the two action bits, pwd-write and permalock, gives four different configurations (see table 2.24). Since the action bits can be skipped by the mask bits the resulting lock bits can also be a combination of the existing bits of the tag with the payload bits. The permalock bits cannot be changed whether they once were set to a logically one. In case of a Lock command attempts to reset a permalock bits which is logically one to a logically zero the tag backscatters an error.
The response of a tag to a Lock command starts with an extended preamble. Tags should respond to a Lock command within $20\,ms$. In case that this time has been exceeded the interrogator can send any command [EPCGlobal 2005].
The Lock command is listed in table 2.22 table 2.23 and table 2.24.

	Command	**Payload**	**RN**	**CRC**16
# of bits	8	20	16	16
description	11000101	Mask and Action Fields	handle	

Table 2.22: Kill command in detail, [EPCGlobal 2005]

	Header	**RN**	**CRC**16
# of bits	1	16	16
description	0	handle	

Table 2.23: Lock command response, [EPCGlobal 2005]

pwd-write	**permalock**	**Description**
0	0	Associated memory bank is writable from either the open or secured states.
0	1	Associated memory bank is permanently writable from either the open or secured states and may never be locked.
1	0	Associated memory bank is writable from secured state but not from the open state.
1	1	Associated memory bank is not writable from any states.
pwd read/write	**permalock**	**Description**
0	0	Associated password location is readable and writable from either the open or secured states.
0	1	Associated password location is permanently readable and writable from either the open or secured states and may never be locked.
1	0	Associated password location is readable and writable from the secured state but not from the open state.
1	1	Associated password location is not readable from any state.

Table 2.24: Lock command Action-field functionality, [EPCGlobal 2005]

2.6.4 Custom Commands

The EPC standard gives the developer the possibility to define custom commands. Since the designed chip is still in the prototype status, the custom commands Test Read and Test Write were implemented. These commands have the same payload as the in the standard defined commands Read and Write. Nevertheless the lock bits are not checked at the Test Read and Test Write commands and the handle does not need to match the tags handle. The main advantage compared to the Read and Write commands defined by the standard is the independence of the actual state of the tag. This enables the interrogator to read and write to the memory at any time with no additional proceeding commands to reach the states Open or Secured.

Test Read

Test Read has the same parameters as Read. The backscattered random number is generated from the random number generator. The tag backscatters also a valid CRC-16 checksum. Since Test Read can be executed from any state the handle is not compared to the internal handle. There might be no internal handle in case that the tag is in the inventory round. To be able to read and write all memory regions, the lock bits are ignored by the tag.
The Test Read command is listed in table 2.25 and table 2.26.

	Command	**MemBank**	**WordPtr**	**WordCount**	**RN**	**CRC**16
# of bits	8	2	EBV	8	16	16
description	11000010	00: Reserved 01: EPC 10: TID 11: User	Starting address pointer	Number of words to read	handle	

Table 2.25: Test Read command in detail

	Header	**Memory Words**	**RN**	**CRC**16
# of bits	1	Variable	16	16
description	0	Data	handle	

Table 2.26: Test Read command response

Test Write

Test Write has the same parameters as Write. The difference between Write and Test Write is that the lock bits are not checked, the handle is not checked, the data is not xor-ed with a random number and there is no need for a proceeding Random Number command.
The Test Write command is listed in table 2.27 and table 2.28.

	Command	**MemBank**	**WordPtr**	**Data**	**RN**	**CRC**16
# of bits	8	2	EBV	16	16	16
description	11000011	00: Reserved 01: EPC 10: TID 11: User	Address pointer	Word to be written	handle	

Table 2.27: Test Write command in detail

	Header	**RN**	**CRC**16
# of bits	1	16	16
description	0	handle	

Table 2.28: Test Write command response

2.7 Examples for a Communication Sequences

To show how the protocol works in practice a simple communication example sequence is provided here. All listed parameters in this example are decimal as long as there is no $0x$ notation.

The first command in the example is a Query command with the parameters

DR = 0, M = 2, TRext = 1, SEL = 2, SESSION = 0, TARGET = 0, Q = 10

Since the parameters match a tag in the initial state, the slot counter is loaded with 10 bits from the random number generator. The probability is about 0.1 % that the slot counter is loaded with all zeros because the random number generator has the same probability of 50 % for a zero and a one bit. Therefore, the tag might enter the Reply state or the Arbitrate state. In the case that the slot counter was loaded with zero, the tag changes its state to Reply. All transmitted data is saved in the internal memory of the tag.

Select is the next command sent in the example. Since the tag executes a Select in any state except Killed the actual state is regardless. The Select parameters are:

TARGET = 4, ACTION = 0, MEMBANK = 1, POINTER = 2, LENGTH = 16,

MASK = 0xFEDC, TRUNCATE = 1

The EPC memory of the tag is filled up with the sequence 0xFEDCBA9876543210FEDC and so on for 31 words. By the Select command a valid truncate is generated and the tag gets selected although it would be selected anyway. The tag changes the state to Ready since a new Select command has been sent.

To start the inventory round again a new Query command is sent. This command has the parameters:

$$\text{DR} = 0,\ \text{M} = 0,\ \text{TRext} = 1,\ \text{SEL} = 2,\ \text{SESSION} = 0,\ \text{TARGET} = 0,\ \text{Q} = 0$$

Since the Q value is zero, the tag loads its slot counter with zero and replies with a random number. The internal state of the tag changes to Reply. All parameters are saved again in the internal registers and the Random Access Memory (RAM) of the tag.

To demonstrate a longer inventory round, two Query Adjust commands are sent next. The session has to be zero to match the session of the tag. To force the tag to increment the Q value, the updown field holds the data $0x06$. By this the Q value is incremented from zero to one. Since the first generated random number is assumed for the first Query Adjust as $0xCE3D$ the slot counter is loaded with $0x01$ and the tags state changes from Reply to Arbitrate. After the second Query Adjust the Q value is increased to two. This has the consequence that the last two bits of a new random number are loaded into the slot counter. Since the generated random number is $0x2ADE$ the slot counter is loaded with $0x02$. The state of the tag remains in the Arbitrate state since the Query Adjust command did not load the slot counter with zero.

The next step is decrementing the slot counter until it equals zero to enter again the Reply state. Therefore the easiest way would be decrementing the Q value with Query Adjust commands until it equals zero. By this the tag would be in the Reply state at least after two Query Adjust commands. The Query Adjust command is executable from the states Arbitrate and Reply and decrements the Q value, not the slot counter.
The other method is the more fine grained modification of the slot counter by the Query Reply command. Query Reply decrements the slot counter only when the tag is in the Arbitrate state. If the tag is already in the Reply state the internal state changes to Arbitrate and the slot counter is not modified. As a result of decrementing the slot counter the command Query Reply can be executed 2^{15} times in the worst case to reach a slot counter loaded with zero. On the other hand the Query Adjust command can be executed 15 times until a slot counter of zero is reached. In this example in the worst case after 2^Q $(Q = 2) = 4$ Query Reply commands the tag reaches the Reply state and backscatters again a random number. Since the slot counter is loaded with $0x02$ just two Query Reply commands are required to reach the Reply state. After the first Query Reply command the slot counter is decremented to $0x01$ and the tag stays in the Arbitrate state. At the second Query Reply command the slot counter is decremented again and equals finally zero. The state changes to Reply and a random number is backscattered.

For the next step towards the access commands the command Acknowledge is needed. The Acknowledge command sends the random number which the tag backscattered when

its state changed from Arbitrate to Reply. Whether this random number does not match the tag falls back to the state Arbitrate. In case of a matching random number the tag replies with the for an Acknowledge command defined response. This response depends on the truncate from the Select command. The tag changes the internal state from Reply to Acknowledged.

The last step for reaching the access commands is a Request Random Number command. To get a valid command the sent random number has to be equal to the last sent random number from the tag. The tag stays in the state Acknowledged whether the random number does not match. Depending on the access password the tag enters the Open or Secured state.

Until now no data from the memory has been modified by the reader. Since the states Open or Secured are reached the Read and Write commands are now executable by the tag. Furthermore, the lock bits can be modified and the tag can be killed. All these commands are listed at the tables 2.15 to 2.24.

For a read the addressed memory has to be available and the correct handle needs to be sent to the tag to achieve a valid command. The tag backscatters the read data to the interrogator only in case of the command was valid, the memory was accessible and the memory was not locked.
The Write command needs a previous Request Random Number command. The data which should be written to the memory has to be xor-ed with the responded random number from the tag. Here also the handle has to match the tags handle, the memory must not be locked and the memory has to be available.
The Read and Write commands do not change the state of the tag as long as the tag is in the Open or Secured state.

To modify the lock bits the Lock command has to be sent with valid parameters. Therefore the Lock command has to send the handle and a valid payload protected by a CRC-16 checksum to the tag. Whether all these requirements are met the tag modifies the lock bits and responds to the reader.

The last command is the Kill command. To Kill a tag the sequence of Request Random Number - Kill - Request Random Number - Kill has to be sent with valid parameters. Therefore, the Request Random Number command and the Kill command send the handle to the tag. Furthermore, the Kill command has to send the valid kill password xor-ed with the random number the tag backscattered to the tag. After this procedure the tag responds a second time to the reader, changes its internal state to Kill and will not interact with any reader any more.

Chapter 3: System Overview

This chapter provides an overview of the ASIG project. The main challenge of the digital part of the ASIG is the combination of both HF and UHF communication paths provided by the NFC and EPC block in figure 3.1, respectively.

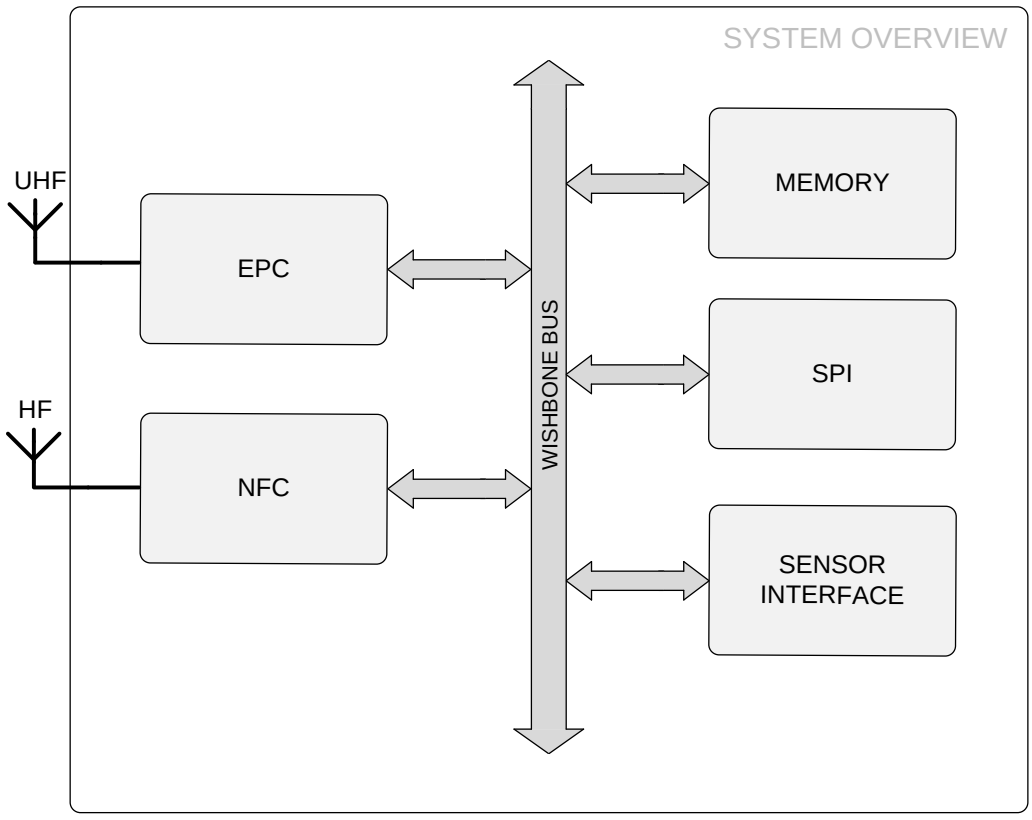

Figure 3.1: System overview

3.1 EPC

The EPC block is responsible for the UHF communication and interacts with the MEMORY block, the SPI block and the SENSOR INTERFACE block. There is no direct interaction between the NFC and the EPC block because both protocols operate alternatively at different frequencies.
The EPC block consists of a low power Reduced Instruction Set Computing (RISC) controller with peripheral hardware units and a Read Only Memory (ROM). The purpose of the peripheral hardware units is on one hand to support the RISC controller with peripheral functions (e.g. interrupts).
On the other hand some operations can be implemented much more efficiently directly in hardware rather than in firmware. The program memory for the RISC controller is located in the ROM and consists of 2048 words with a word size of 16 bits.

The most important hardware unit is the EPC Digital Frontend (DFE). Here the incoming data bits are collected. A desired amount of bits is stored in a send buffer and an interrupt is generated for the firmware to load the data. For encoding several Special Function Registers (SFRs) can configure the different settings for the encoding mode of the EPC DFE. Overall the EPC DFE is the interface between the firmware and the incoming and outgoing data. The EPC DFE uses the TIMER unit as counter for enumerating the amount of incoming bits and for the encoding scheme.

The INTERRUPT CONTROLLER unit handles the hardware interrupts and the internal halt signal for the RISC controller.

The purpose of the CRC Linear Feedback Shift Register (LFSR) unit is generating a 5 bit CRC checksum, a 16 bit CRC checksum and a pseudo random number. These modes can be switched by special function registers.

3.2 NFC

The NFC block is responsible for the ISO 15693 and ISO/IEC 14443 protocols. All details used in this system can be seen at [Odobasic 2012].

3.3 Memory

There are two memory blocks in the system, the NVM and the RAM. The NVM is the tag memory with the logical structure as defined in figure 2.7.
The RAM is used as temporary buffer for data occurring during the execution of the EPC protocol.
For simplicity behavioral models of the RAM and NVM were used in the simulations.

3.3.1 NVM

Since the used model of the NVM is a behavioral model, the read and write time can be set by the VHDL code. To get realistic simulation results the read time is set to $1\,\mu s$ and the write time is set to $3.6\,ms$. The NVM consists of 80 data words with a size of 16 bits each. The NVM should be realized as Electrically Erasable Programmable Read Only Memory (EEPROM).

3.3.2 RAM

Since the RISC controller has insufficient memory to save internally all the data required for the EPC protocol, the RAM provides an additional data storage. The RAM consists of 80 words, of which five words are used by the firmware. Assuming the RAM to be much faster than the NVM the read and write times are set to $1\,ns$ each.

3.4 SPI

The Serial Peripheral Interface (SPI) provides an interface for miscellaneous master or slave SPI devices. By the wires SLCK (synchronous clock), MOSI (master out slave in), MISO (master in slave out) and SS (slave select) the four wire bus enables a fast full duplex communication [Leens 2009].

3.5 Sensor Interface

The SENSOR INTERFACE acts as bridge between the internal WISHBONE BUS and the analog senor bus. The analog sensor is able to acquire data like temperature.

3.6 Wishbone Bus

The internal bus is realized as WISHBONE BUS. By this a multi master bus connects all the above mentioned components. The used WISHBONE BUS is a simpler, modified version of the Wishbone B4 [OpenCores 2010].

Chapter 4: Controller Evaluation

The starting point for this thesis was an existing implementation of an EPC capable transponder Application-specific Integrated Circuit (ASIC) realized by a Finite State Machine (FSM) controller.

One major drawback of the FSM-based approach is that the FSM became quite inflexible in terms of maintenance and expandability. Therefore a firmware-based solution with a programmable controller as substitution for the FSM is required.
Basically, there are two potential processor architectures available, namely a lightweight 8 bit Application-specific Instruction-set Processor (ASIP) and more powerful 16 bit ASIP.

In the following the EPC-FSM and the two potential controller approaches for a firmware-based solution are compared and evaluated in detail in terms of area, code density and power consumption.

4.1 FSM-based EPC Implementation

Since the EPC protocol is relatively complex, an implementation of the protocol in a hardware ASIC exhibits also a certain complexity. The FSM ASIC realizes the EPC protocol in the area of about 9000 Gate Equivalents (GE) according to [Heyszl 2007].

As mentioned above the main disadvantage of the hardware implementation is the inflexibility in terms of maintenance and expandability. In the design the FSM is the main controlling unit for the whole transponder. The FSM connected to its subunits by a broad interface of control and data lines. Thus an adoption of its underlying VHDL code including all side-effects can be very difficult and affect more than just a single unit. In the worst case it could result in a change of the complete design. Due to the complexity of the whole ASIC an overview of the system is hard to manage.
A benefit of the hardware FSM ASIC is the parallel data processing, which enables the ASIC to operate faster than any firmware solution.
Overall the hardware ASIC is very fast and small but a special purpose hardware is hard to expand and maintain.

For the further development and extensions of the ASIG project an easy adaptable ASIC is needed. To fulfill this requirements the EPC standard is implemented on a RISC controller.

4.2 RISC Controller

A RISC controller is a small, fast and simple processor. To reach these benefits the instruction set is reduced to the necessary operations of a specific application. The reduced instruction set allows a faster decoding procedure and additionally leads to a simpler Arithmetic and Logic Unit (ALU). With the simplicity the hardware effort decreases and the power consumption drops in comparison to a usual controller.

> "The main feature of the RISC processor is its ability to support single cycle operation, meaning that the instruction is fetched from the instruction memory at the maximum speed of the memory. [Sakthikumaran *et al.* 2011]"

With the single cycle operations an instruction is executed in one clock cycle after loading the code from the memory. Processors with no single cycle mechanism preload their instructions usually in a pipeline. In case of a jump in the program code, the pipe content is not valid anymore and the processor has to wait until new instructions are loaded into the pipe. This is the main benefit of a single cycle RISC controller compared to a pipelining processor.

4.3 Analysis of the Applicable Controller

The analysis is in respect to the area of the ASIPs and their instruction sets. For the comparison a sample code was implemented on both ASIPs. In this code the most important operations for the EPC protocol were used. Furthermore the power, the coding efficiency, the tooling chain, the maintenance, the flexibility and the reuse of the code were taken into account. Both ASIPs are realized in a Harvard architecture.

> "A microprocessor utilizing Harvard Architecture is disclosed with a data space for data storage, an instruction space for instruction storage, and a common space implemented as a high-speed on-chip RAM that functions as a continuous extension of both the data space and the instruction space, for the simultaneous and flexible storage of data and instructions. [Yasui and Shimazu 1991]"

4.3.1 8 Bit Controller

The 8 bit RISC controller has an 8 bit data path, a 16 bit instruction word and a register based architecture. For storing data in the core 16 internal 8 bit registers are provided. The synthesis of the 8 bit processor gives an area of about 2400 GE. Due to the reduced instructions of the 8 bit ASIP some useful commands are missing in the instruction set. Complex projects might get unmanageable since the 8 bit processor is programmed with a reduced set of assembler instructions.

4.3.2 16 Bit Controller

The 16 bit RISC controller has a 16 bit data path, a 12 bit instruction word and a stack based architecture. For storing data two internal 16 bit working registers and a configurable data stack is provided. Since this ASIP is highly configurable the synthesis gives an area ranging from 2000 to 6000 GE. The 16 bit controller can be programmed in both C and assembler. This gives the benefit of a clearly arranged code also for complex projects. On the other hand the C instructions and the assembler instructions are not as optimized as in case of the 8 bit controller. Therefore some benefits like a single cycle load and save are lost.

4.4 Summary

Especially for the EPC protocol the actual state, the parameters from the Query and Select command, the handle and the random number are often used for comparing and evaluating. Therefore these data should be in an internal memory to increase the speed of the operations.

Due to the stack architecture of the 16 bit processor the internal memory is reduced to two 16 bit registers, which sums up to 32 bit. Compared to the 16 bit ASIP, the 8 bit ASIP has 16 internal registers with 8 bit each which result in 128 bit internal memory, thus four times the memory of the 16 bit ASIP. Since the memory of the 16 bit controller is less than the memory of the 8 bit controller, the 16 bit ASIP needs to swap out the data mentioned above. This leads to an additional effort of swapping data out and loading data back in again.
The 8 bit controller has the benefit of more internal memory and is able to save a majority of the EPC data internally.

> "One significant advantage of register-based computation models is that register scheduling algorithms can place temporary variables in registers instead of memory-based variables to improve code efficiency. A common criticisms of stack-based execution is that variables can't be kept on the stack without a lot of wasted stack manipulation. [Koopman and Jr. 1992]"

In respect to the area, the comparison is not directly possible since the 16 bit controller can be configured in different variations. Only an 8 bit version of the denoted 16 bit ASIP would be smaller than the 8 bit microcontroller. Any other configuration of the 16 bit ASIP is larger than the 8 bit ASIP in terms of area. Due to the doubled internal data path this result is predictable. The single cycle mechanism gives the 8 bit controller an advantage to the 16 bit ASIP with regard to efficiency and power consumption.

The fact that the 8 bit RISC controller is faster than the 16 bit ASIP gives an benefit for the 8 bit ASIP in respect of timing.

The efficiency of the code is higher in the 8 bit RISC controller since here few very fast assembler commands are provided. On the other hand the 16 bit RISC controller uses the C libraries with lots of overhead and in assembler a larger instruction set than the 8 bit ASIP. The usability, tooling and maintenance is better with the C compatible 16 bit controller since the high level language C provides predefined libraries and enables a compact programming style. Concerning flexibility, the 16 bit controller is in advantage over the 8 bit controller, since the C code is more adaptive. The reuse factor is kind of the same for both controllers.

Overall the 8 bit controller has benefits in area, power, timing and code efficiency. On the other hand the tooling, the maintenance and flexibility are advantages for the 16 bit controller.

Table 4.1 sums up the comparison of the two controllers.

Benchmark	8 **bit Controller**	16 **bit Controller**
Area	++	+
Power	+	
Timing	++	-
Code Efficiency	++	-
Tooling	-	++
Performance Scalability		++
Firmware Maintenance	-	++
Firmware Flexibility	-	++
Hardware Flexibility		++
Reuse	+	+

Table 4.1: Comparison of the two controllers

The comparison of some typical commands of the EPC, like loading and storing data, show, that the 8 bit ASIP needs less operations than the 16 bit controller. The 8 bit ASIP is very efficient at the examination of a single bit because a dedicated command is provided. Since the evaluated operations occur very often in the implementation of the EPC standard, already a difference of one or two instructions for an operation has a consequence. This affects on the one hand the overall speed of the firmware and by this the minimal clock frequencies and minimal power consumption. On the other hand the critical timing of the EPC protocol can be easier met with less instructions.

The above mentioned facts resulted to the choice of the 8 bit controller. This was a tradeoff between usability and efficiency. The elected solution provides a lower usability with a benefit in efficiency.

Chapter 5: RTL Hardware Design

In this chapter the hardware units are explained in detail. Furthermore the applied techniques to reduce power consumption are explained. The hardware units are peripheral for the ASIP in order to expand the processor in terms of functionality. The combination of the hardware units and the ASIP can be seen as an application specific microcontroller with special features like hardware interrupts a CRC unit and a random number generator.

5.1 Low Power Design Principles

To reduce the power consumption of a digital system specific low power design techniques can be applied. In this thesis especially the methods of dynamically scaling the operating frequency as well as gating the clocks of registers have been used.

Basically the power consumption can be separated into two categories: static power consumption and dynamic power consumption.
The static power consumption is generated by the leakage of every single transistor and is therefore dependent on the used technology. In case of a used operating voltage in the sub threshold region the static power consumption increases since the transistors cannot be disabled properly. [Soeser 2012]

The dynamic power consumption adds together out of the cross current during the transistor switches and the current for loading the load capacitors. Since the cross current is much smaller than the load current for the load capacitors, the main formula for the power consumption relates on the load current for the load capacitors. [Soeser 2012]

$$P = f_{CLK} \times C_L \times V_{DD}^2 \times \alpha \quad \text{[Soeser 2012]} \tag{5.1}$$

In equation 5.1 f_{CLK} stands for the clock frequency, V_{DD}^2 represents the used operating voltage, C_L is the load capacitance and α is the activity factor. Since the voltage and the capacitance of the used transistors is given by the used technology the frequency and the activity factor can be decreased for saving power. This is done by the frequency scaling and clock gating mechanism, described below.

5.1.1 Frequency scaling

The frequency scaling is realized by an integer divide of the clock signal. This generated signal is the enable signal for the hardware units. According to the necessity of a high speed module or a low speed module a different divided clock signal is used as enable signal for the hardware units.
With the frequency scaling every unit has the same clock which avoids synchronization problems.

5.1.2 Clock Gating

In the traditional VHDL design the system clock is connected to every register. Since the input value of a register may not change every clock cycle, the clock should be gated. By this only an enabled register changes the output data. In any other case the clock pin of the register is set to logically zero, thus it does not work and consumes no power. [Emnett and Biegel 2000]
The clock gating is a tool based mechanism which is added automatically by the Design Compiler from Synopsys [Design Compiler].

> "Clock signals that are passed through some gate other than buffer and inverters are called gated clocks. These clock signals will be under the control of gated logic. Clock gating is used to turn off clock to some sections of design to save power." [Sadrusham]

The local clock gating mechanism can be realized in two different designs, the latch free clock gating, figure 5.1 and the latch based clock gating, figure 5.2. The simple latch free clock gating uses an AND or OR gate to generate the gated clock. With this design the enable signal must not change during a clock cycle to generate a stable gated clock.

In case of a non constant enable signal during one clock cycle the latch based clock gating resolves the problem with the enable signal. The latch based clock gating uses a latch to synchronize the enable signal with the clock. With the latch the enable signal is stored at the positive edge of the clock. [Emnett and Biegel 2000]
To guarantee that the enable signal of the positive edge of the clock is used, the latch based clock gating is applied in this work.

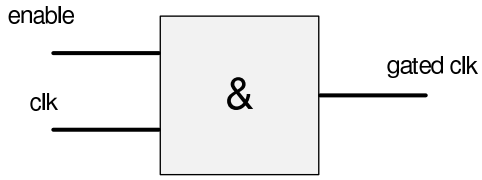

Figure 5.1: Latch free clock gating

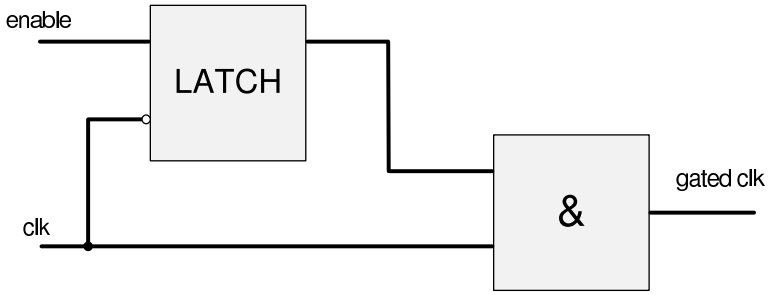

Figure 5.2: Latch based clock gating

5.2 Overview

An overview of the implemented system is given at figure 5.3. On the top side of the internal wishbone bus all peripheral hardware units are placed. The 8 bit controller is the core unit which processes the data provided by the hardware units. On the left side of the illustration the digital data stream from and to the analog part of the system is indicated. On the right side the internal wishbone bus connects the EPC part to the whole ASIG (see 3.1).

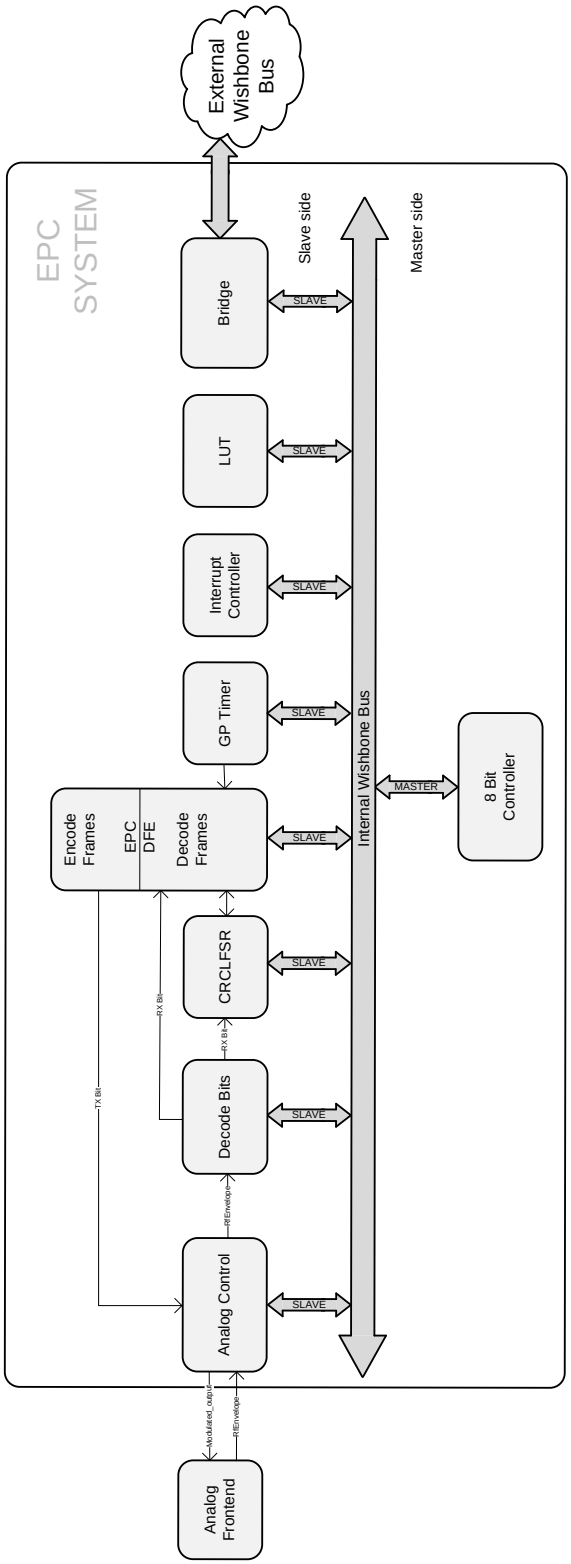

Figure 5.3: EPC system overview

5.3 Component Description

Every hardware unit depicted at figure 5.3 is described in detail below. The firmware code of the 8 bit controller is declared in the next chapter.

5.3.1 EPC DFE

The EPC DFE module, depicted in figure 5.4, decodes all received data bits and encodes all sent data. Special function registers, set from the 8 bit controller, control the EPC DFE.

The input data is delivered by Decode Bits in terms of decoded bits. These bits are collected in a shift register which is reused for the sending mechanism. The decision how many bits should be collected is made by the controller.

After the demanded amount of bits is received, a wake-up event for the ASIP is generated. Since the controller does not immediately load the received data, the data is stored in an 8 bit register until the next frame is decoded. The special case of a Query Reply, the only command with less than 8 bits, is indicated with a special interrupt which is declared below. By this procedure a wake-up event for the controller is generated when the output register is filled up with already 4 bits.

For the encoding the ASIP sends the current encoding parameters to the EPC DFE. The EPC DFE has a send buffer with 3x8 bit. Since the NVM and the RAM are built out of 16 bit blocks, two 8 bit buffers are needed for one block of data. The third 8 bit buffer ensures that the EPC DFE unit does not run out of data to send.

Generally there are three counters in the EPC DFE: timer0, timer1 and the counter. Timer1 is used for decoding because this counter has the ability of an external clock. The external clock is needed since the input data is PIE encoded and has no fixed interval thus. All the other counters are used for encoding, the major part of the EPC DFE. There are several helpers to fulfill all the demanded functions like sending a preamble and the different encoding schemes.

The link frequency is generated by counting the clock cycles with timer0 until a certain value is reached. The calculation for the amount of required clock cycles is a formula based on a Mat-Lab® [MATLAB 2010] calculation, done in [Klamminger 2013]. In case of an odd amount of needed clock cycles, an additional clock cycle is added to reach the desired link frequency.

The timer1 counts the number of link frequency periods. For the different encoding schemes, FM0, Miller2, Miller4, Miller8, the counter is used. This counter enumerates the

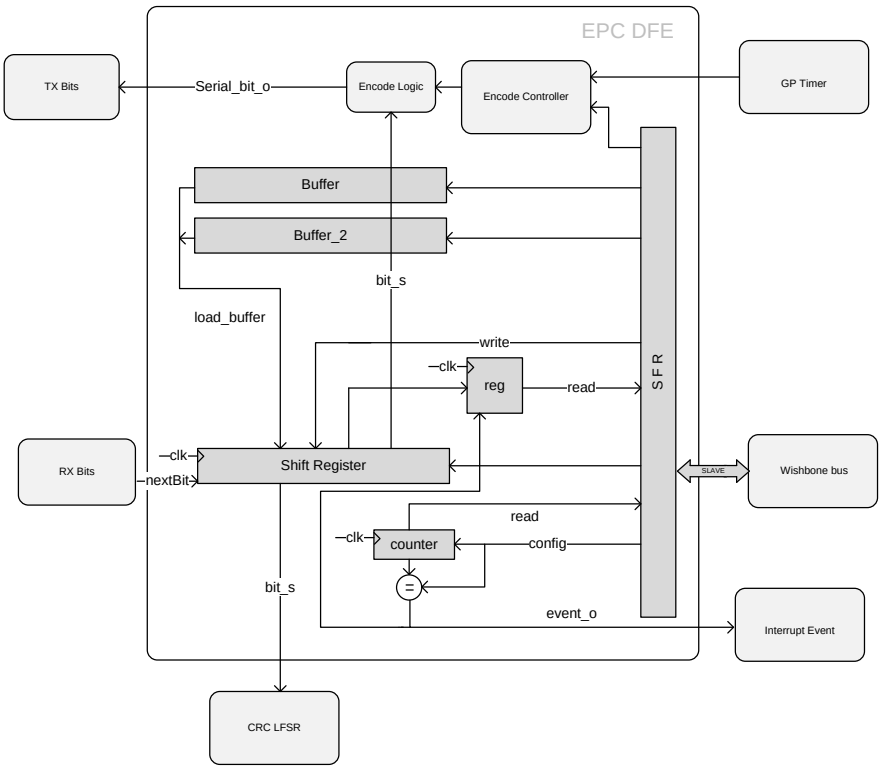

Figure 5.4: EPC DFE

link frequency periods needed for the generation of a symbol of the different encoding functions.

The extended preamble for encoded data is the same for FM0, Miller2, Miller4 and Miller8. The standard preamble is hard-coded in the hardware since the FM0 preamble includes an violation which cannot be created with the FM0 encoding scheme declared below.

For the encoding, an FM0 symbol duration equals the duration of one period of the link frequency.
An FM0 encoded output signal can be represented by a zero, a one, the link frequency or the negated link frequency for the duration of one symbol. The decision for the output is a combination of the bit which has to be encoded and the status of the sending signal (inverted or not inverted).

To encode a logically zero bit, the link frequency or the negated link frequency is sent. The encoding of a logically one is realized by sending a one or zero for the whole symbol duration. The status of the signal before the bit is encoded decides which of the two

mentioned cases are chosen. At the transition of two bits the FM0 encoded signal has to toggle. Since the preamble as well as the extended preamble is a constant sequence the first bit after the preamble and every further bit is defined by this rules. An encoded dummy one has to be sent at the end of every End-of-Signaling.

The symbol duration for the Miller encoding scheme depends on the modulation index. For Miller2 two link frequency cycles result in one symbol duration. In Miller4 four link frequency periods are used for one symbol duration. Miller8 uses eight link frequency periods for one symbol.
For the encoding mechanism a signal toggles in the middle of an encoded bit because at the Miller encoding a phase inversion can occur at the start of a symbol or in the middle of a symbol. With the toggle signal it is known whether there is a transition of two bits or the encoding is in the middle of a symbol. To encode the signal in Miller, the link frequency has to be inverted at a transition of two logical zero bits.

Otherwise the link frequency stays the same at a transition of two bits. In the middle of a bit the link frequency is inverted if the bit to encode is a logical one. In any other case the link frequency, inverted or not inverted, remains the same for the encoded signal.
After an inversion of the link frequency the link frequency stays inverted until the next inversion occurs which results again to a non-inverted link frequency. An encoded dummy one has to be sent at the end of every End-of-Signaling.

There are several interrupt signals in the EPC DFE to wake up the controller.
Before a new command is sent, the EPC DFE gets a synchronous reset. After a synchronous reset for the EPC DFE, the ASIP is woken up to save the random number because the CRC registers will be used for calculating the CRC over the incoming bits.

Wake-up decode and wake-up encode are set in case of the decoding of a frame has finished or the encoding needs more data. Since all commands except Query Reply are longer than 8 bit the decoder collects as default 8 bit and sends an interrupt to the ASIP. The firmware decides depending on the received data after how many bits the decoder should generate the next interrupt.

In the special case of a Query Reply an additional interrupt is generated since the 8 bit are not reached to wake up the firmware. The hardware compares after 2 bits the received bits against the Query Reply sequence. In case of a match the decode interrupt is set after the whole 4 bits of the Query Reply command are received. In the other case the default 8 bit data has to be collected before an interrupt is generated.

For the wake-up encode signal a toggle value toggles at every 8 bit sent. A wake-up event is generated when the toggle signal equals one. Since the initial value of the toggle signal is zero, the first wake-up is generated after the first 8 bits were encoded.

For the encoding mechanism the first 16 bit data to encode are loaded from the firmware into the shift register and the buffer2. When the EPC DFE wakes up the controller, the data of the shift register has been encoded and the data from buffer2 is loaded into the shift register. Both buffers, buffer and buffer2, are empty, so the controller can fill up the buffers.

From now on after each 16 bits sent the controller is woken up to refill buffer and buffer2. While the EPC DFE generates the interrupt, still 8 bits are left in the shift register as a buffer. With this procedure there is always data in the shift registers to ensure that the EPC DFE delivers a continuous data stream.

The wake-up encode interrupt is also set in case of all the transmission has finished. This interrupt is generated by internal hardware flags.

The last wake-up signal from the EPC DFE is for the Query command to change the CRC unit. In case of a detected TRcal, which indicates a Query command, the ASIP needs to be notified to change the settings of the CRC LSFR unit. The CRC calculation is set by default to calculate a CRC16. In case of a received Query command the CRC consists of 5 bits and the CRC LSFR has to be notified thus.

5.3.2 Interrupt Controller

Because of the 8 bit ASIP does not support interrupts directly, a dedicated interrupt controller unit was designed to handle wake-up events. The Interrupt mechanism consists of two parts: the Interrupt Controller and the RISC Control.

The Interrupt Controller is connected with all the interrupt generating modules. Whether an interrupt occurs, it will be saved in the Interrupt Controller. The Interrupt Controller has a internal register with a capacity of eight interrupt sources. Each interrupt source can be enabled or disabled individually via SFRs controlled by firmware. A wake-up signal is sent to the RISC Control unit in case of an activated interrupt sources fires an interrupt event. An interrupt which is not selected is ignored by the Interrupt Controller until it is enabled by firmware.

An occurred interrupt has to be cleared by the firmware by means of SFRs. The Interrupt Controller can handle four interrupts simultaneously. This feature is required when the 8 bit ASIP is woken up after executing an EPC command. In this case a new command can be sent, a TRcal can be detected, the random number could be saved or a delimiter is sent.

At every particular of the just mentioned possibilities an interrupt is generated to wake up the ASIP.

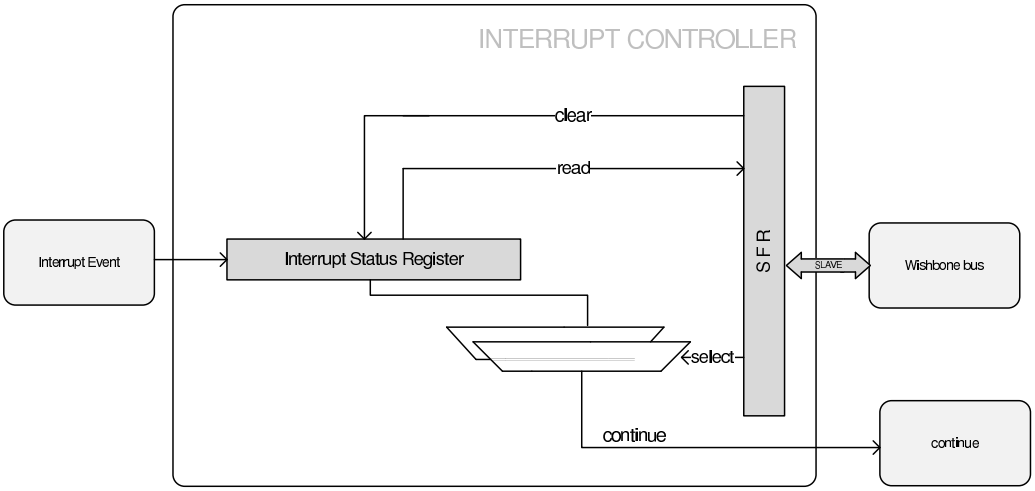

Figure 5.5: Interrupt Controller

The RISC control interacts with the Interrupt Controller and the controller. The halt signal of the ASIP is controlled by the RISC Control unit. In case of the controller is halted, the RISC control waits for an interrupt from the Interrupt Controller. The controller sends a request to halt to the RISC Control unit to wait for an interrupt.

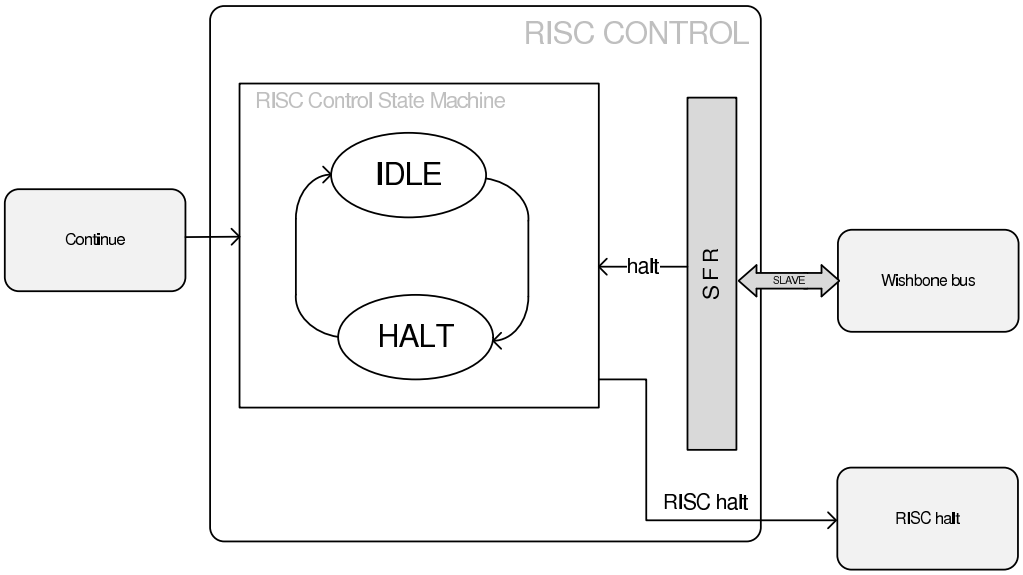

Figure 5.6: RISC Control

5.3.3 CRC LFSR

To generate a pseudo random number a linear feedback shift register is used. Since the CRC16 and the PRNG needs a 16 bit shift register, these units were combined in the CRC LFSR module.

In a linear feedback shift register the data is shifted through the registers every clock cycle. As input for the first register the data of the output from the last register is provided. Whether the polynomial of the LFSR is set for a register, the input data of a register is xor-ed with the output bit of the LFSR.

In case of the polynomial parameter of a register is not set, the register shifts the data forward. The complexity is created by the XOR function with the output. Every LFSR has a period after which the generated pseudo random numbers are repeated. The period of a LFSR is described as followed:

> "It is well known that a primitive polynomial of degree d produces a periodic sequence of distinct states of length (period) $2^d - 1$, whereas an irreducible (but not primitive) polynomial of degree d produces $\frac{2^d-1}{p}$ periodic sequences of p distinct states each, where p is the period of the irreducible polynomial." [Udar and Kagaris 2007]

The maximal period for a 16 bit LFSR is given with $2^{16} - 1$. With a primitive polynomial 65535 pseudo random numbers can be generated before a random number occurs twice.

The polynomial of the LFSR can be set by the firmware to generate an application specific LFSR. All calculated pseudo random numbers and CRC calculations are done in the CRC LFSR unit. For the CRC calculation the EPC DFE forwards the data to the CRC LSFR unit. The CRC LSFR unit calculates the CRC and compares the output to the results defined in the standard. In case of a match of these values, the CRC LFSR unit sends the ASIP a signal that indicates a valid CRC.

Since a LFSR is deterministic, the generated number cannot be classified as random number, they are pseudo random numbers. The pseudo random numbers are generated in between the CRC calculations.

Figure 5.7 shows the internal structure of the CRC LFSR. The main design looks like a CRC16 en-/decoder. However the CRC LFSR has additionally an XOR and a multiplexer at the data input of every register. This additional hardware enables the CRC LFSR to switch between the different modes: CRC16, CRC5 and LFSR. Furthermore the polynomial for the CRC and LSFR can be set individually since the multiplexer can select the XOR or the predecessor signal as input of the register. In case of a CRC5 checksum just the first 5 bits of the CRC LFSR are valid.

The different settings for the possibilities of the CRC LFSR are set from the firmware, the EPC DFE and the CRC LFSR unit. Therefore some additional hardware controls the CRC LFSR to set the needed input values and to deliver the desired output.

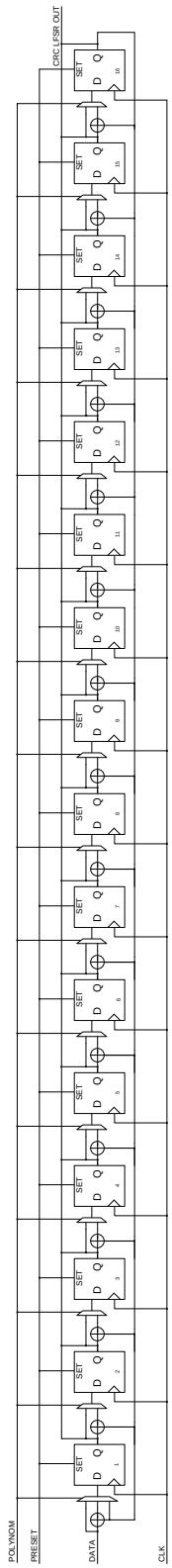

Figure 5.7: CRC LFSR Control

5.3.4 General Purpose Timer

The General Purpose Timer provides two timer which are used at the EPC DFE. Each timer can be configured as a single or a continuous counter. The clock of both timers can be prescaled by the factor 2 to 256 in power of two steps. Furthermore Timer1 can count an external event or the overflow of Timer0. The timers can be controlled by the ASIP with SFRs and are hardwired to the EPC DFE.

5.3.5 Decode Bits

The unit Decode Bits is reused from the former implementation of [Heyszl 2007] since the code is very efficient. Furthermore the decoding of a bit is quite timing critical, so Decode Bits should be realized in hardware. Decode Bits counts the duration of the high phase and the low phase of the data-0 sequence in clock cycles.
The values RTcal and TRcal which occur in the preamble are also determined in clock cycles. By comparing the incoming data to the duration of the data-0 sequence and the RTcal sequence, Decode Bits is able to estimate the value of a bit.

The delimiter is the first element of the preamble from the reader to the tag. To handle the firmware properly, an interrupt is generated after every received delimiter.
In case of any wrong sent data from the reader, the firmware aborts the actual command and waits for the next delimiter. With the delimiter interrupt the firmware is aware that a new command has been sent. The purpose of the delimiter interrupt is to avoid the 8 bit ASIP to reach any undesired state.

5.3.6 LUT

The look-up table (LUT) unit is required to calculate the operation $2^x - 1$ directly in hardware. In consideration of the computational effort for this operation directly in firmware (ASIP does not support barrel-shifter commands), a hardware LUT was chosen to implement this functionality more efficiently.

5.3.7 Bridge

The hardware unit Bridge manages the access to the NVM and the RAM. Since these components have a word size of 16 bit the Bridge translates the 2 times 8 bit values from the controller to the required devices.

The Bridge provides an indirect addressing and a memory map mode. For the indirect addressing the address has to be sent for every access. Since the read and especially the write operations may take more time than a single cycle, the Bridge unit generates an interrupt when the access has finished.

5.3.8 Analog Control

The module Analog Control builds the interface between the digital and analog part of the ASIC. For the EPC DFE the Analog Control switches the analog modulator and demodulator on and off. The Analog Control units is controlled by the firmware via SFRs.

Chapter 6: Firmware

In this chapter the firmware of the 8 bit controller is explained. The general structure of the code and the internal 16 registers is given. Also the handling of the different interrupts is declared. To show how the firmware is programmed a full Read command is explained as an example. At the end of this chapter the assembler code is analyzed.

6.1 General Structure

The firmware is written with a reduced instruction set of assembler commands. At the beginning of the firmware file, constants are defined for the readability of the code. The command decoding is the first programmed part of the firmware. Afterwards all commands are implemented and in the last part of the firmware some helper functions are located.

The internal registers (16 x 8 bit) are used for calculating and storing the input data. To save the actual EPC command, register r0 stores the command. Since the EPC state has to be proven several times, the register r1 holds the state of the firmware. The registers r2 to r11 are registers with no specific purpose and are used as temporary registers during the whole code. In the registers r12 and r13 the data from a Query command and the SELECT command is stored. In the last two registers r14 and r15, the random number is saved.

Register	**Bit 7**	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
r12	DR	M		TRext	Q			
r13	Target0	Target1	Target2	Target3	Error	Select	Session	

Table 6.1: Register 12 and register 13

6.1.1 Interrupts

The 8 bit controller can be halted by the hardware and woken up again by an interrupt. Therefore several interrupts sources are provided:

EPC DFE, TRcal, Save RN, Delimiter, Bridge

All interrupts are generated by hardware units, namely by the EPC DFE, the Bridge and Decode Bits, as described at section 5.3.

The interrupts for encoding are used for sending additional data to the EPC DFE unit or for finishing the encoding procedure. One of the four possible interrupts to wake up the 8 bit ASIP after finishing a command is the decoding interrupt. This interrupt signals the firmware that a new frame is ready to load.

Besides this, the other three interrupts are: the TRcal interrupt for changing the CRC LSFR module to CRC5, the Save RN interrupt to save the random number before the CRC is calculated and the Delimiter interrupt to be aware of a new command. To indicate a finished read or write operation, an interrupt is generated from the Bridge.

6.1.2 Application Flow of a Command

In this section the decoding of a command is declared. Furthermore the flow of an executed command is shown with the example of a Read command.

Decoding

The decoding of a command starts with a hardware interrupt since the ASIP is in the halt state. For every command the Save RN interrupt is generated before the actual command code is received. The generated random number needs to be saved because the CRC unit has to calculate the CRC over all received bits.

Afterward the decoding unit of the EPC DFE is enabled. In the case of a TRcal interrupt, the CRC unit is set to calculate the CRC5, else the CRC16 is calculated. The firmware listens to a delimiter interrupt whether the execution of a command aborts although the reader is still sending data. In this case only a new delimiter can wake up the ASIP to decode a new command. After all these possible interrupts the controller gets into the halt state and waits for the EPC DFE decoder to collect the incoming bits.

At the next wake-up from the hardware the new data is loaded from the shift register of the EPC DFE. The loaded data can be a 2 bit, a 4 bit or an 8 bit command. Certain bits of the data are compared to evaluate the command code. After the command decoding, a

very fast conclusion whether a response should be sent or not is required. This is done in the code for every command.

Execute Command

Roughly there are a few steps which are done for every command:

- Prove the exit states
- Load and format the data
- Evaluate the data, load additional data
- Change the internal data when required
- Send a response when required
- Wait for the next command

As an example for this steps, the sequence of a Read command is declared in detail below.

Prove the exit states

The read command proves first of all exit conditions since the long command code provides enough computation time until the next received data is decoded by the EPC DFE. In the states Ready, Arbitrate, Reply, Acknowledge and Killed the Read command is aborted.

Load the data

The received data consists of the data field membank inclusive the first the Extensible Bit Vectors (EBV). Since the first 8 bits after the command code are no coherent data field, the bits are separated by logical OR and AND functions. Furthermore, the separated data is stored in different registers. There is no other effective way to handle this problem since the decoding of a command takes a certain amount of time.

After this time the decoder in the EPC DFE unit might have already decoded more bits than the first data field requires. In this case the decoder in the EPC DFE unit cannot deliver only the first data field since additional bits are already added. To solve this problem the bits are separated in firmware since the commands have no uniformly ordered data fields. The separation in firmware means further to leave the frame size for the decoder in the EPC DFE unit unchanged at the default value of 8. At the first possible time the offset of the data is removed by ordering a certain amount of bits from the decoder in the EPC DFE.

In the case of a Read command, after loading the first 8 bits the decoder is instructed to deliver 2 bits to compensate the offset generated by the 2 bit data field membank. Every

parameter of the Read command after the offset compensation is at least integral dividable by 8 and the EPC DFE decoder can deliver 8 bit frames to the firmware.

Evaluate the data, load additional data

After loading the first EBV of the word pointer, the data has to be evaluated whether another EVB is sent. Since the used memory is addressable with 7 bit, there is no need of an additional EBV. Nevertheless the code supports up to two EBV values. The relevant EBV is stored in r7 if there is only one EBV, otherwise r7 and r6 are used.
All the other data is loaded after the EBV, namely word count and handle. Before the CRC is evaluated, the handle is loaded from the RAM and compared to the received handle value. The command exits in case of a non-matching handle. After the handle the lock bits in the NVM are evaluated. The lock bits are located at the address 0x4F, the last word of the NVM. These lock bits are evaluated according to the readability of the requested memory region.
The data loading and data evaluating sequences are often interleaved as described above. This is done with the purpose of a faster response since the time t1 starts when the last bit has been sent from the reader. Therefore all calculations which can be done during the data is being received shortens the response time.
From now on also the Test Read command has the same code than the Read command.

The offset of the membank is added to the received word pointer. Now the physical address of the data to read has been figured out. To achieve a valid Read command the CRC from the command has to be checked. The CRC LFSR has been set at the beginning of the command to the CRC16 mode to compute the CRC over the sent data. Assuming the sent CRC to be valid, the CRC LSFR unit sends a CRC-OK signal.

Change the internal data when required

Since the Read command just reads data, no internal data change is required.

Send a response when required

In the valid Read command the first data word is loaded from the NVM. The parameters of the encoder in the EPC DFE are set to the values which are stored in the register r12, without the Q value. Now the loaded data is sent to the shift register and buffer2 of the encoder in the EPC DFE unit. To start the encoder, the enable signal, load shift register signal and the add header signal are set. All interrupts except the EPC DFE interrupt are turned off. To be prepared for the following interrupt the next data of the NVM is loaded. After this the firmware is halted to wait for the encoder to finish the preamble. The generated interrupt is cleared and the read loop begins.

The read loop starts by setting the state of the controller to halt because the first 16 bits have not been encoded yet. At the wake-up the first 8 bit data has been encoded and 8 bits are still in the buffer of EPC DFE. Since these 8 bit are already loaded into the internal shift register of the encoder, the registers buffer and buffer2 can be filled up again. To perform this the preloaded data is stored in the SFR of the buffers. Before the firmware is set to halt again, the next data is loaded from the NVM. Whether the word counter, which is decreased in every loop iteration, equals to zero, the read loop exits. In the other case of a non-zero word counter the read loop starts again.

After finishing the loop the EPC DFE interrupt is cleared and the handle is loaded from the RAM. Then the encode buffers get filled with the handle and the interrupt is set to EPC DFE. The interrupt is cleared again and the controller is halted. After waking up the controller, the interrupt has to be cleared and the parameter for the CRC and a flag to signal a finished encoding are sent to the encoder. The ASIP is halted again to waits for the CRC to be sent.

Wait for the next command

The interrupt is cleared and at the next wake-up the CRC has been encoded. Now the decoder in the EPC DFE is activated for decoding the next command. The random number generator is activated and the controller can wait now for the next command from the EPC DFE.

6.2 Code Analysis

In table 6.2 all used commands are listed with their absolute quantity and the percental occurrence.
Figure 6.1 illustrates graphically the amount of the used commands. It can be seen that the *mov* and *st* commands are nearly 50 % of the whole firmware code. The store commands are mostly used for controlling the hardware units by their SFRs. For the protocol adherence the data has to be compared. To achieve the comparison the data has to be in the correct format which is generated with the *mov* command. Furthermore the controlling of the hardware is triggered by the *mov* command.

The machine code instructions for the firmware consist of 2055 lines of code. The hardware is controlled by 538 commands affecting the SFRs, which equals to 26,1 % of the whole firmware code.
These 538 SFR commands consist of: 202 commands for the Bridge, 209 commands for the Interrupt Controller, 12 commands for the Look Up Table, 171 commands for the EPC DFE and 55 commands for the CRC LFSR. This shows that the Bridge, the Interrupt Controller and the EPC DFE are the most used hardware units.

Command	**Numeric occurrence**	**Percental occurrence**
add	33	1,61
and	111	5,40
call	108	5,26
jeq	169	8,22
jmp	122	5,94
jne	59	2,87
ld	126	6,13
mov	537	26,13
movc	0	0
not	11	0,53
or	40	1,95
ret	8	0,39
rol	69	3,36
seqb	8	0,39
seq	212	10,32
sl	0	0
slt	7	0,34
st	412	20,05
sub	17	0,83
xor	6	0,29

Table 6.2: Command analysis

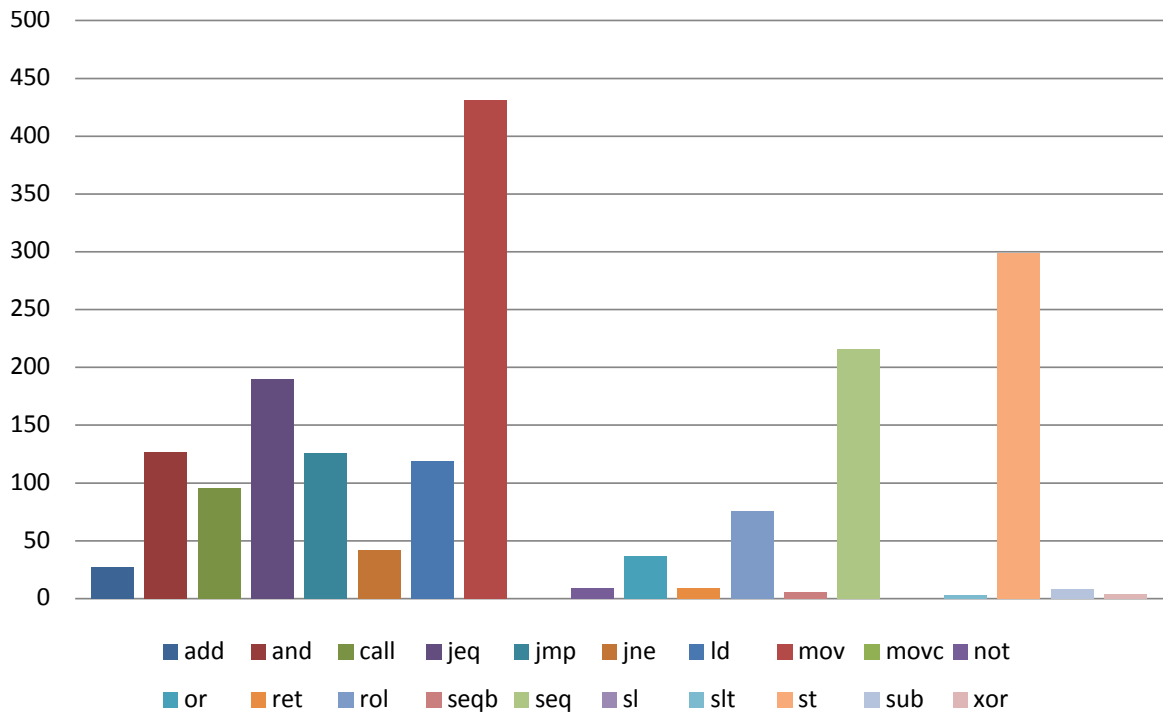

Figure 6.1: Command occurrence

6.3 Time Analysis

Here the processing time of a read cycle, as described in 6.1.2, is analyzed. In this example 16 bit are loaded from the NVM and backscattered to the interrogator. The tag has a clock frequency of 2.4 MHz and transmission values as follows:

$Tari$: 12.5 μs

$Data1$: 1.5 $\times$ $Tari$

RT_{cal} : 2.5 $\times$ $Tari$

TR_{cal} : 4 $\times$ $Tari$

PW : 0.5

The clock period is given with the reciprocal value of clock frequency which results in 0.416$\dot{6}$ μs. Since the controller is enabled at every second clock cycle, an assembler command is executed in 0.8$\dot{3}$ μs. This would correspond to clock frequency of 1.2 MHz. In this example the divide ratio is zero, the encoding is set to Miller4 and TRext is activated.

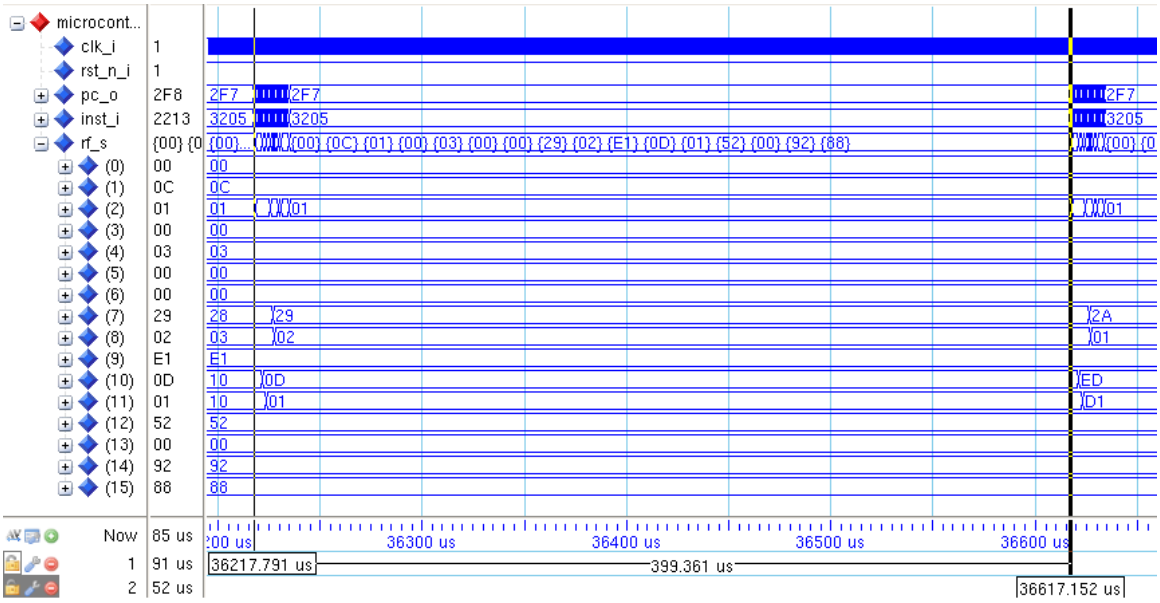

Figure 6.2: Time analyses for encoding 16 bit

Figure 6.2 shows the overall time which is needed to encode 16 bit, 399.36 μs. Divided by 16, the time for encoding one bit in Miller4 with the above mentioned parameters takes 24.96 μs. This gives an output bandwidth of 40.064 $\frac{Kbit}{sec}$ and corresponds to a link frequency of 160.256 KHz. Since the Miller4 encoding is a very slow encoding scheme, the ASIP is most of the time waiting for the EPC DFE to encode the data.

This can also be seen in figure 6.2. The backscattered data for this example can be seen at r10 and r11 with the values $0x0D$, $0x01$. In the register r8 the length of the read data is saved in words, register r7 holds the NMV address of the read data.
Figure 6.3 shows the time when the controller is busy during one iteration of the read loop. As figure 6.2 shows, most of the time of a read iteration the controller is idling. The execution of these 21 commands takes 16.64μs. Divided by 21, one command needs 0.79μs, as already calculated above. During one read loop, where 16 bit are encoded, the ASIP is busy 4.16̇ % of the time, else it stays idle.

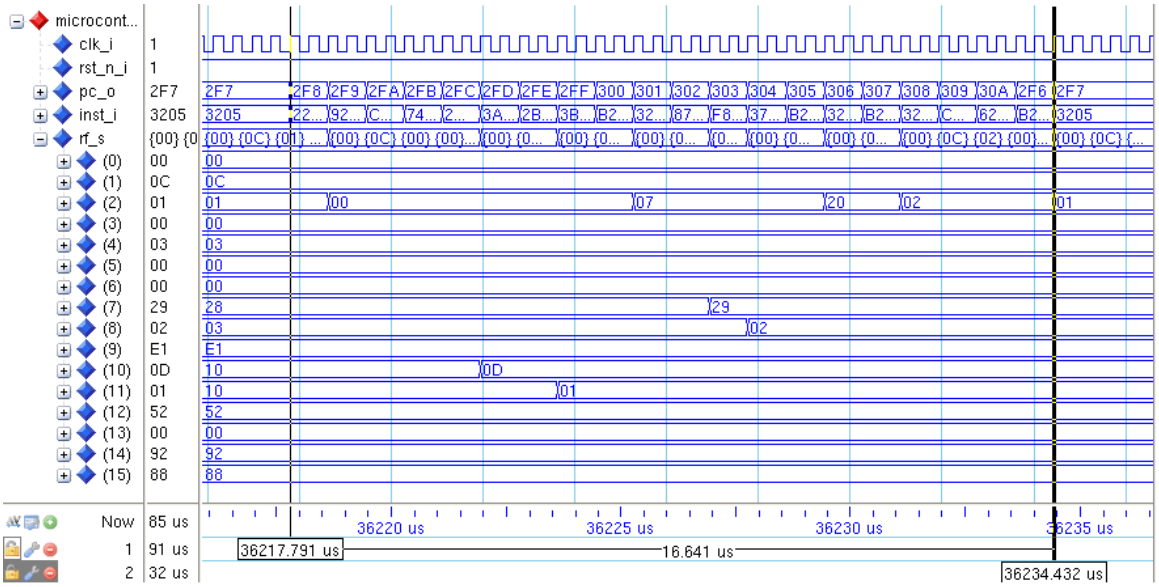

Figure 6.3: Time analysis for the read loop

Command	**Response time [μs]**
Query Reply	40.046
Query Adjust	34.71
Acknowledge	35.293
Request Random Number	24.017
Test Read	39.969
Read	39.989
Query	34.373

Table 6.3: Command response times

This example shows that the peripheral components can easily achieve the data rates required by the EPC standard. The bottleneck of the system is the response time for the first response.

With the custom commands Test Read and Test Write 13 commands were implemented. To achieve the minimal value for t1, the command with the maximal response time has to be figured out. Not all commands send a response, so the response for 7 commands where a response is sent is listed in 6.3. The times for Test Write, Write, Kill and Lock are not listed since the interrogator waits anyway 20.000 μs for a response.

The command Query Reply determines the $t1_{max}$ with $40.046\,\mu s$ since it has the longest response time. For every time beyond the $40.046\,\mu s$ for $t1$, the firmware is able to respond to the interrogator.
The 2 bit command Query Reply with a payload of 4 bit lacks of calculation time during the data is received. All the other commands have a longer payload providing more calculation time. The response of the Read command is nearly as long as the Query Reply response since the Read command is not optimized in respect of time. This is not needed because the Query Reply command has the longest response time and any other time does not influence the maximal value for $t1$.

The formula for $t1_{max}$ is shown in equation 2.1. In equation 6.2 the maximal value for the time $t1$ is calculated for the values given from 6.1.

$$\begin{aligned} &Tari = 12.5\,\mu s,\ FT = 7\,\%,\ DR = 8 \\ &RT_{cal} = 2.5 \times Tari = 31.25\,\mu s,\ TR_{cal} = 4 \times Tari = 50\,\mu s \\ &T_{pri} = \frac{1}{LF} = \frac{TR_{cal}}{DR} = \frac{50\,\mu s}{8} = 6.25\,\mu s \end{aligned} \tag{6.1}$$

$$\begin{aligned} t1_{max} &= max(RT_{cal}, 10 \times T_{pri} \times (1 + FT) + 2\,\mu s) \\ &= max(31.25\,\mu s, 10 \times 6.25\,\mu s \times (1 + 0.07) + 2\,\mu s) \\ &= 68.875\,\mu s \end{aligned} \tag{6.2}$$

The values $Tari$, RT_{cal}, TR_{cal} and DR define the time $t1_{max}$. With any variation of these values resulting in a $t1_{max}$ bigger than $40.046\,\mu s$ the firmware is able to respond fast enough to fulfill the requirements of the EPC standard at a clock frequency of $2.4\,MHz$.

Chapter 7: Simulation

In this chapter the simulation environment and the used tools are explained. The test bench is introduced and an example for a read and a write sequence is given.

7.1 Simulation Environment and Test Bench

The VHDL code for the peripherals and the firmware code were simulated with Mentor Graphics Questa Sim [QuestaSim]. Eclipse [EclipseSDK] was used as editor and a company internal test bench was provided. The synthesis was done with the Synopsys Design Compiler [Design Compiler].
To test the firmware and the peripherals a test bench was provided, done in another thesis [Moritsch 2012]. The test bench is written in System Verilog and uses a Universal Verification Methodology (UVM) model. The UVM enables the test bench to be easily extendable. Furthermore, random tests can be applied to the tested device. The used test bench generates valid EPC commands and compares the answer from the tested units to an internal scoreboard. Therefore the test bench needs to be aware of the content of the NVM to verify commands like Read.
Since random tests might take a long time to get the right parameters incidentally, a dedicated test had to be written. This test generates every command at least once and prolongs the inventory round. To achieve the protocol adherence the test bench needs to know the last sent random number, the last random number which led from the state Arbitrate to Reply and the handle. All these data is casted and sent manually since a wrong value makes the tested system fall back to a previous, not desired state.

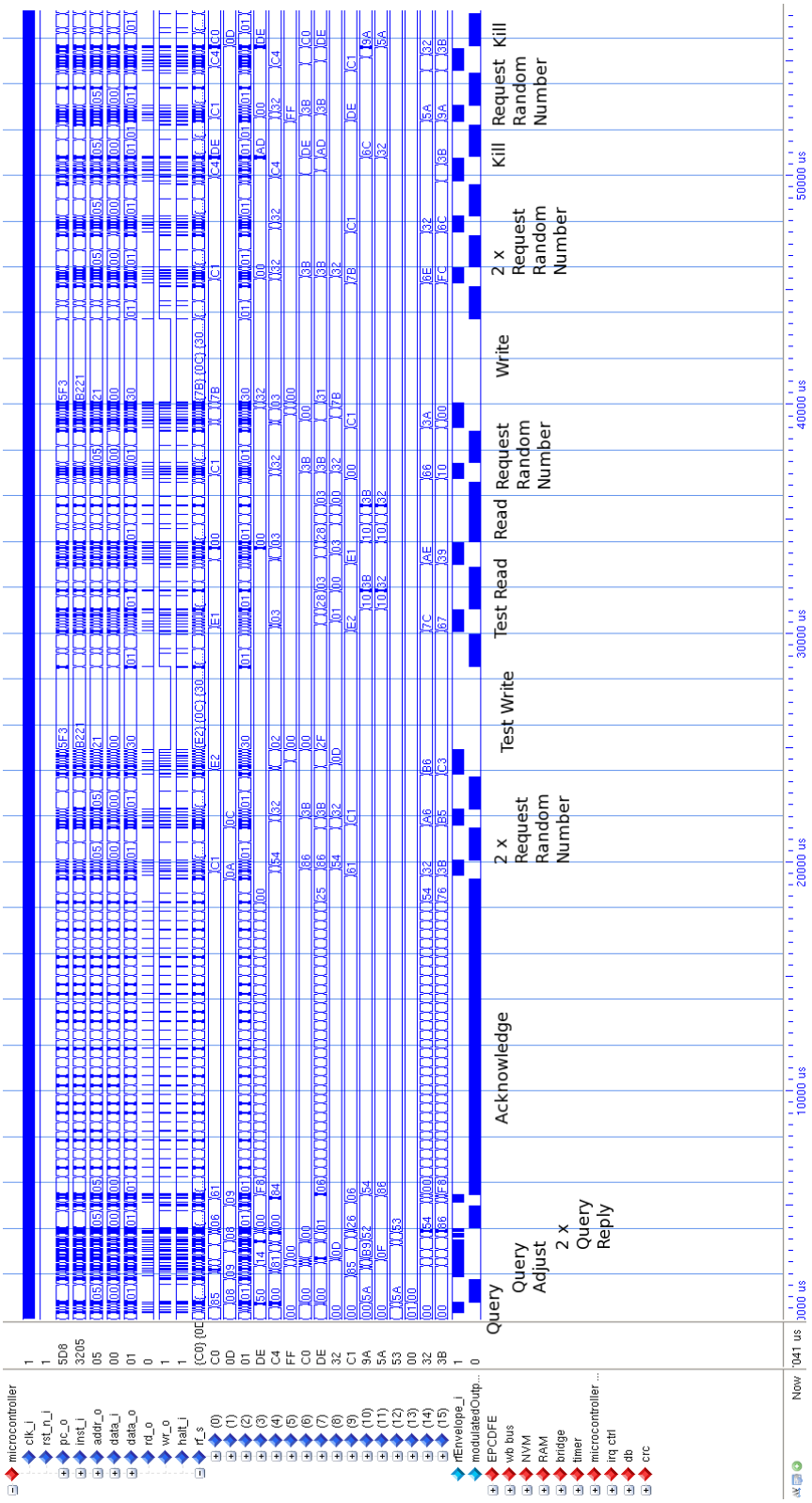

Figure 7.1: All tested sequences

In figure 7.1 the tested sequence is depicted. The tested system receives an Acknowledge command after the inventory round. The response to the Acknowledge is set by the Select command to the whole EPC data in the NVM. Therefore a long response can be seen at figure 7.1. After the valid Acknowledge the commands Test Write, Test Read, Request Random Number, Write, Read, Lock and Kill are sent.

7.2 Read Example

In figure 7.2 the firmware signals from a Read sequence is depicted. At the time when the test bench is still sending data, the firmware starts already working to get a faster response. The activity of the firmware is higher at the receiving part than at the sending part. This is caused by using Miller4 encoding and the fact that all comparison and evaluation has already been done during the time the command was received. Other encoding schemes like the fastest possible, FM0, would just scale the response phase. At figure 7.2 the long idle times during the encoding scheme can be seen as precalculation whilst the command is received.

Figure 7.3 shows a short sequence of the Read command in figure 7.2. Here the data from the EPC DFE is loaded and new data is requested. The signal pc_o gives the line of the hex code in the ROM. With a generated *.lst* file the instruction can be disassembled to see the assembler code. To describe how the firmware works this short sequence is declared below:

The start is at the program counter value of $0x5D8$ where the firmware is in the halt routine. Here the firmware is halted and waits to be woken up. This wake-up happens when the hardware has finished collecting the incoming data. In the following instructions from $0x5D9$ to $0x5DA$ the generated interrupt is cleared. The instruction $0x5DB$ is the return to the read sequence.

Back in the read sequence the collected data is stored and the next 8 bits are requested in the instructions at the program counter $0x284$ to $0x286$. The firmware jumps again with the instruction from program counter $0x287$ to the halt routine.

In the halt routine the instructions from the program counter $0x5D3$ to $0x5D6$ select again the valid interrupts which might have changed. This is necessary since the halt routine is often called and the interrupt should not be edited every time before the routine is called. Therefore the routine sets the interrupts by itself. In this case the setting of the interrupt would not be necessary but to provide a safe general usage of the function the interrupts are set in any case. The last instruction read at program counter $0x5D8$ is the command to halt the firmware to wait for the next incoming 8 bit data.

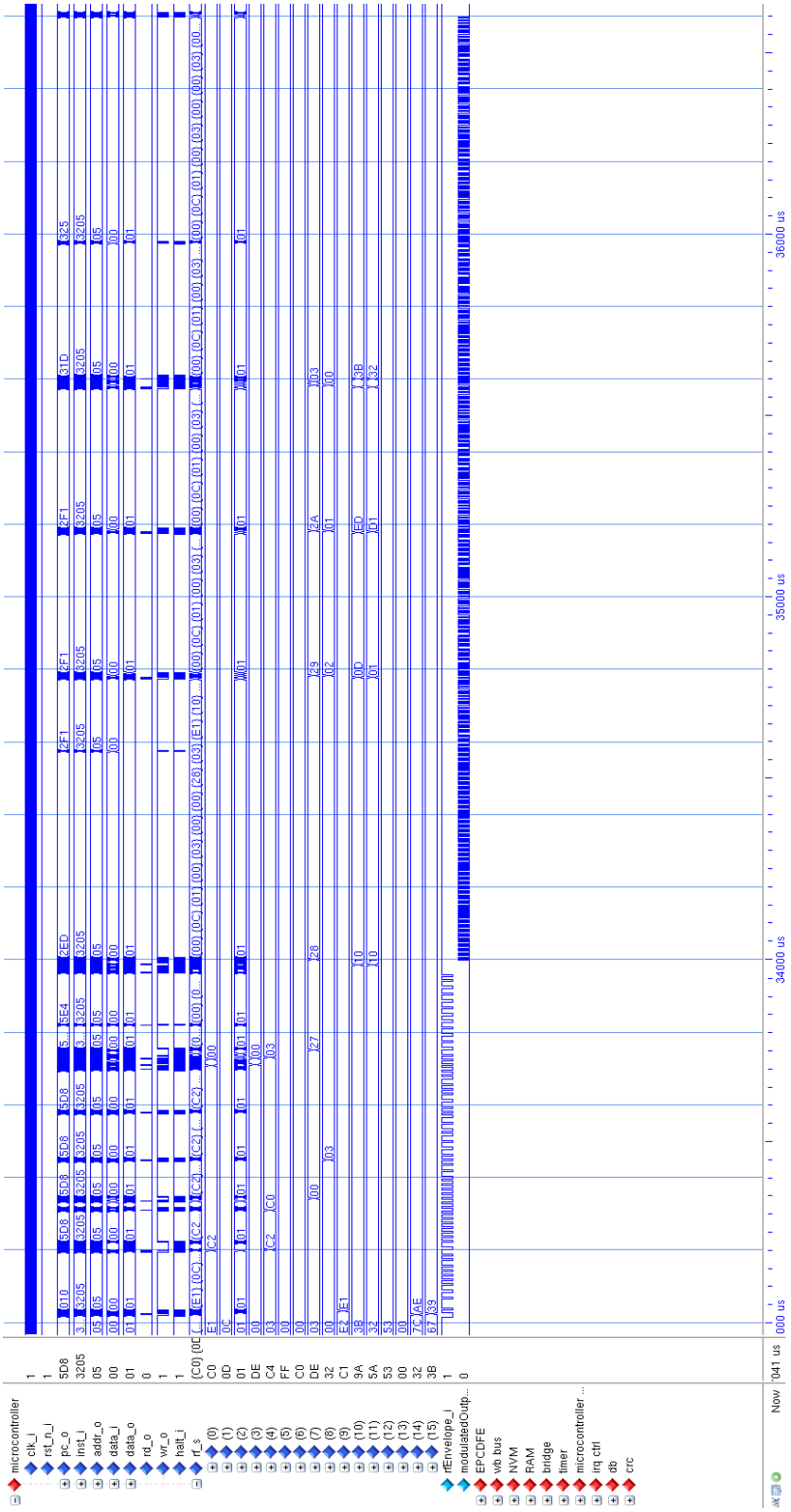

Figure 7.2: The Read command

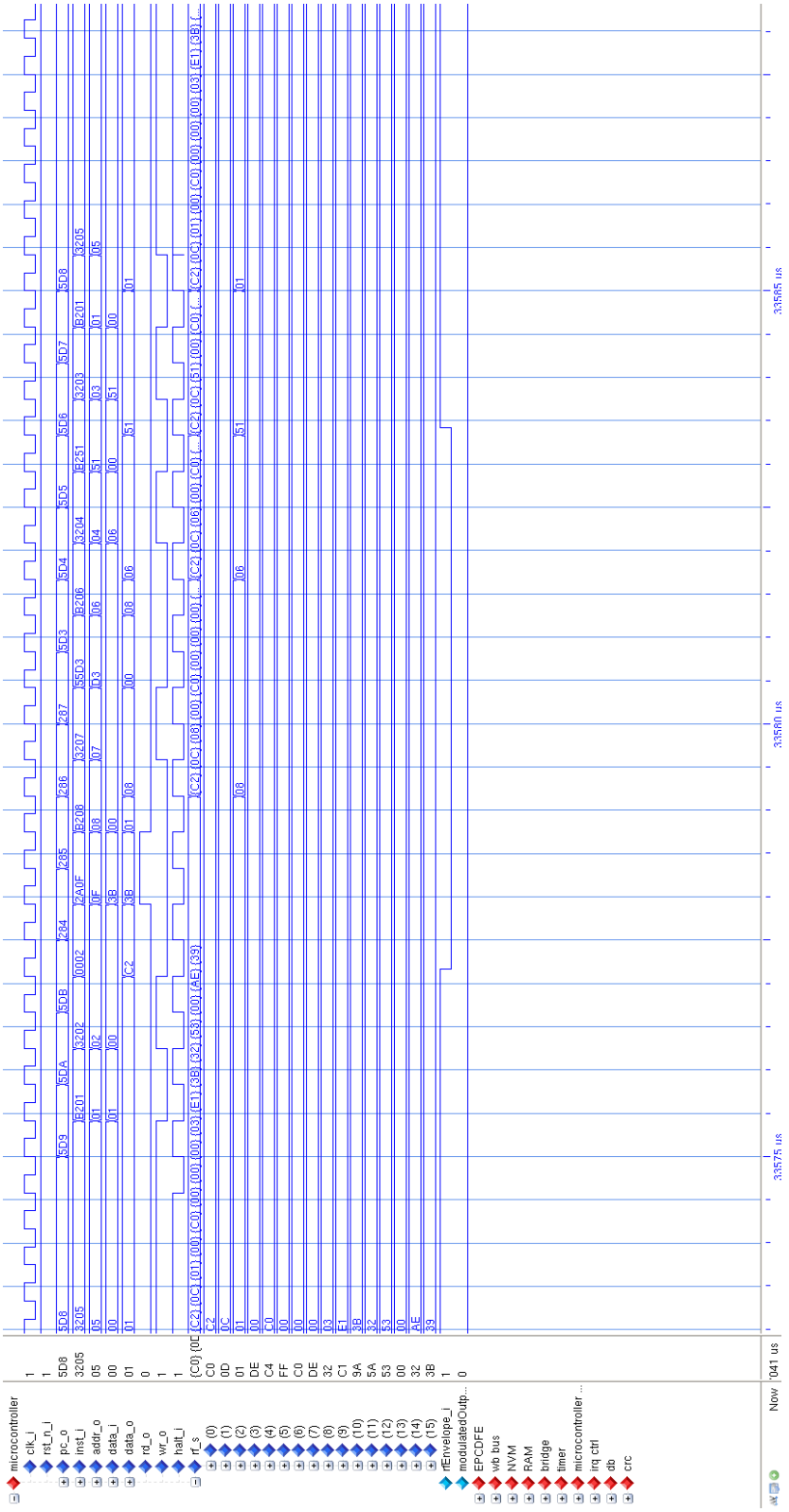

Figure 7.3: The Read command in detail

7.3 Write Example

In figure 7.4 the firmware signals for a Write command are depicted. The signal $rfEnvelope\_i$ is the input signal of the simulated tag, $ModulatedOutput\_o$ is the output. The huge gap between these two signals is caused by the simulated time for a write operation of 16 bit to the NVM. Similar to the Read command the data is already computed after the first 8 bits are received. The receiving sequence is similar to the sequence depicted in figure 7.3.

After all data is received and the CRC16 is valid, the firmware configures the Bridge to write to the NVM. The write data is forwarded to the Bridge and the firmware waits for the NVM to finish the operation. Therefore the firmware halts itself and waits for an interrupt from the Bridge. After the Bridge sends the wake-up signal to the firmware, the handle is loaded from the RAM. Before the firmware waits for the next command, the handle with a CRC16 is backscattered.

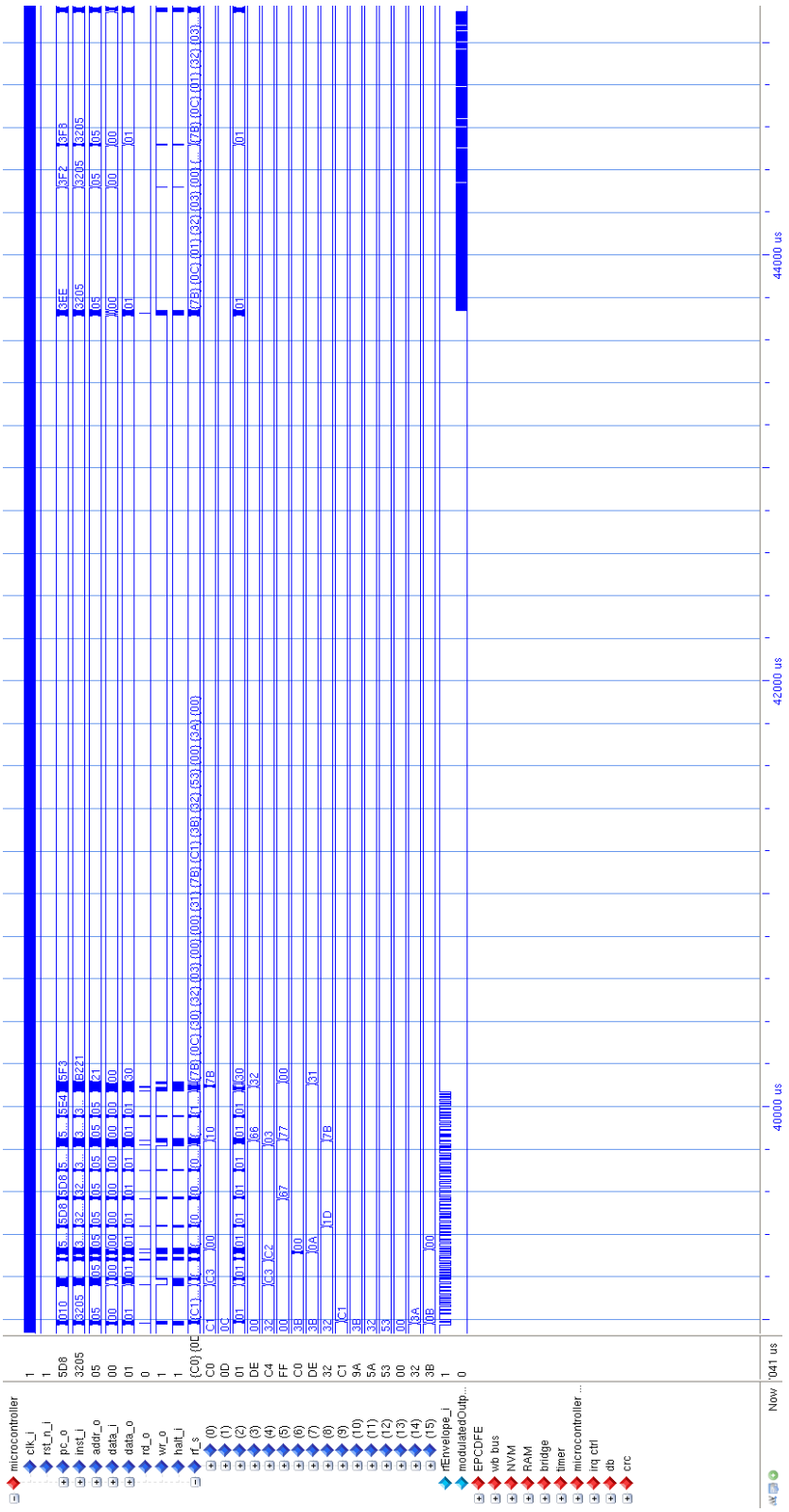

Figure 7.4: The Write command

Chapter 8: Synthesis

The project was synthesized based on a $130\,nm$ Complementary Metal Oxide Semiconductor (CMOS) fabrication technology with the Design Compiler from Synopsys [Design Compiler]. Since the used synthesis environment only supported $VHDL\,87$ code, the whole hardware was synthesized in $VHDL\,87$. The synthesis was done for an over-constrained clock with the period of $300\,ns$. A duty cycle of $50\,\%$ was used. Therefore the clock frequency results in $3.3\dot{3}MHz$. The actual operating frequency of the system is given with $2.4MHz$ which is covered by the synthesis with a $3.3\dot{3}MHz$ clock.

8.1 Area

Here the area results are given and analyzed. In general there is a difference between combinational area and noncombinational area. The combinational part is only sensitive to certain signals while the noncombinational part contains registers which are sensitive to every clock cycle. To reduce the activity of the noncombinational part clock gating was implemented, as declared in 5.1.2. The results of the clock gating are listed in table 8.2.

Combinational area	25217.280231 μm^2
Noncombinational area	23600.159870 μm^2
Total cell area	48817.440102 μm^2

Table 8.1: Synthesis result for the area

The area of chips are usually measured in gate equivalents (GE). This measurement method implies the process technology and provides a process independent comparison in terms of complexity thus [Feldhofer 2011]. To get the amount of gate equivalents the total cell area is divided by the area of a 2 input NAND2 gate which corresponds to $5.76\,\mu m^2$. This results in 8475.25 NAND2 gates or gate equivalents.

8.2 Clock gating

Since all noncombinational part is dependent on the clock, these cells should be gated to reduce the power as mentioned in 5.1.2.

Number of Clock gating elements	55	
Number of Gated registers	488	90.71% of all registers
Number of Ungated registers	50	9.29% of all registers
Total number of registers	538	

Table 8.2: Synthesis result for the clock gated registers

Table 8.2 shows that most of the registers are gated. Since some processes like a clock divider cannot be gated, it is nearly impossible to implement clock gating to all registers. Furthermore the clock gating is only used at a register width bigger than four due to the fact that it is not effective to use an additional latch for saving power for only one register. The factor of four is an empirical value. By looking more into the clock gating issue, a latch, which is used for clock gating, occupies $24.75\,\mu m^2$ whilst a flip-flop needs $37,035\,\mu m^2$. Besides this most of the registers are clock gated and save power thus.

8.3 ASIG Synthesis

The last step of the project was the integration of the EPC subsystem into the ASIG project. Due to design decisions and availableness problems the NVM was replaced by a RAM. Furthermore a pseudo ROM was added which enables the user to load new or modified firmware into the processor. This data can be loaded for reasons of debugging or for further projects as firmware code for the controller. The internal Wishbone bus was demangified from 16 bit to 8 bit for the reason of simpleness. By this the hardware unit Bridge got obsolete. Thereby the EPC firmware also had to be modified since some SFRs were abandoned. All these changes were done for the tape out of the ASIG test chip on July $31^{st}, 2013$.

Overall the synthesis results mentioned in section 8.1 are still valid since just minor changes were done. Nevertheless the results of the ASIG project are listed below in the tables 8.3 and 8.4.

Combinational area	54134.400281 μm^2
Noncombinational area	114606.398668 μm^2
Total cell area	168740.798949 μm^2

Table 8.3: Synthesis result for the area for the ASIG

To get the amount of gate equivalents the total cell area is again divided by the area of a 2 input NAND2 gate which corresponds to $5.76\,\mu m$. This results in 29295.28 NAND2 gates or gate equivalents.

Although the ROM was not synthesized for the results of table 8.1, the hardware can still be compared. The in this thesis developed system holds 28,93 % of all area of the digital

Number of Clock gating elements	218	
Number of Gated registers	1879	96.11% of all registers
Number of Ungated registers	76	3.89% of all registers
Total number of registers	1955	

Table 8.4: Synthesis result for the clock gated registers for the ASIG

part of the ASIG chip. In figure 8.1 the layout of the whole ASIG chip can be seen. This layout is for the tape out at July 31^{st}.

The analog part of the ASIG is documented in the work of Steffan Christoph in [Steffan 2013], Phillip Greiner in [Greiner 2013] and Wiessflecker Marin [Wiessflecker 2013]. The digital part of the chip is shown in figure 8.2

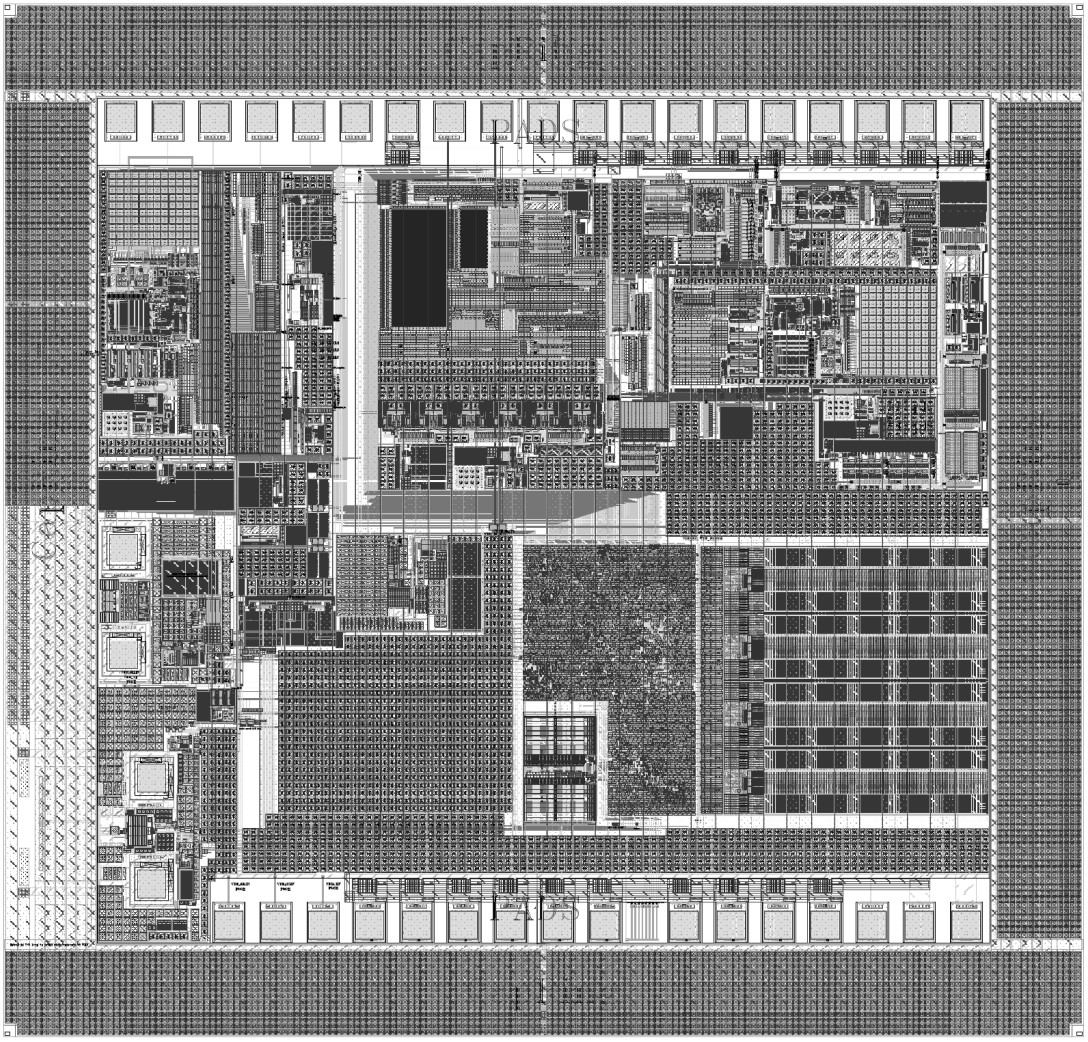

Figure 8.1: The layout of the ASIG chip

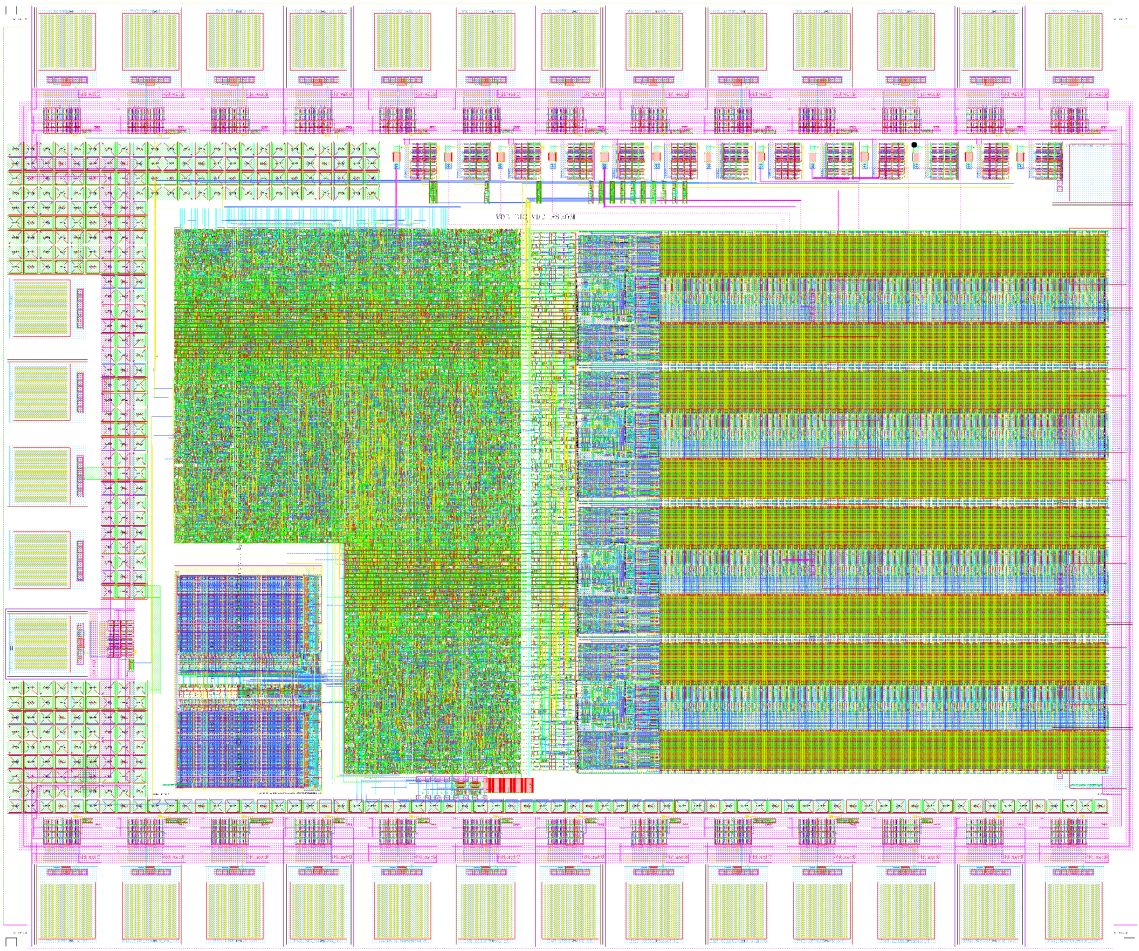

Figure 8.2: The digital layout of the ASIG chip

Chapter 9: Conclusion

The aim of this project is the development of a flexible and modular system, implementing the EPC Global Class-1 Generation-2 UHF RFID Standard. In comparison to the former hardware based system the new system offers a flexible setup for different applications.

9.1 Summary and Results

The first step towards the project was analyzing code for a decision whether to use an 8 bit controller or a 16 bit controller. With the use of a simple firmware the controller were analyzed. After the decision for the controller the main part of the work was started. Thereby the first very simple version of the firmware was developed with the hardware units of the old system which was based on a finite state machine. The idea behind this approach was to keep the existing hardware setup and just replacing the finite state machine with the processor. As it turned out the newly introduced controller also required a redesign of the hardware units. Since the finite state machine had no limitations like 8 bit registers and a sequential mode of operation in comparison to the controller. This demanded the data and the communication to be formatted in a different way. Due to the newly introduced controller system, dedicated hardware units had to be developed.

The development of the dedicated hardware units for the firmware and the realization of the EPC global standard in firmware was the main challenge of this thesis. Since the EPC standard is, compared to the NFC communication, a rather complex standard, the development was laboriously.

The functionality of the overall system depends on the firmware which can be altered flexibly in the form of a ROM mask. The drawback of the solution is the reduced data throughput since the processor oriented architecture causes a lot of overhead. Overall an expandable EPC Global Class-1 Generation-2 compatible platform was designed.

After the main task for the EPC was accomplished the integration into the ASIG project was done. Therefore both systems should use the same processor since the ASIG tag only communicates either with the EPC protocol or the NFC protocol. Some changes had to be made in order to accomplish the EPC system compatible with the other system. A huge

benefit during the development phase is the pseudo ROM where a firmware code can be loaded into the chip.

9.2 Further Work

During the composition of this thesis the tape out was still in progress. By this the chip has to be tested whether the implemented and simulated results can also be achieved with the produced chip.

The pseudo ROM offers the opportunity to execute a different firmware with the same hardware units. By this the ASIG test chip can be modified to execute a simpler, not EPC conform protocol, or a even more advanced protocol as long as the hardware units support these.

A new published version of the EPC standard might cause some changes in the firmware. Since nearly all hardware units are designed as general purpose units, an adoption would not be necessary. Just the Decode Bit unit and the EPC DFE unit are dedicated to the EPC protocol version 1.0.9 and the implemented firmware.

For the next tape out the code of the firmware and the hardware units might be adopted and extended in respect to future challenges.

Bibliography

[Chawla and Ha 2007] V. Chawla and Dong Sam Ha. An overview of passive RFID. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4342873, 2007. [online; accessed September 17, 2013].

[Design Compiler] Synopsis Design Compiler. http://www.synopsys.com.

[EclipseSDK] EclipseSDK. version 3.7.1. http://www.eclipse.org.

[Emnett and Biegel 2000] Frank Emnett and Mark Biegel. Power Reduction Through RTL Clock Gating. http://www.aiec.com/Publications/snug2000.pdf, 2000. [online; accessed September 17, 2013].

[EPCGlobal 2005] EPCGlobal. EPC Radio-Frequency Identity Protocols Class-1 Generation-2 UHF RFID Protocol for Communications at 860 MHz - 960 MHz. http://www.gs1.org/gsmp/kc/epcglobal/uhfc1g2/uhfc1g2_1_0_9-standard-20050126.pdf, January 2005. [online; accessed September 17, 2013].

[EPCGlobal 2011] EPCGlobal. GS1 EPC Tag Data Standard 1.6. http://www.gs1.org/gsmp/kc/epcglobal/tds/tds_1_6-RatifiedStd-20110922.pdf, September 2011. [online; accessed September 17, 2013].

[Feldhofer 2011] Martin Feldhofer. Lecture notes VLSI-Design, 2011.

[Finkenzeller 2002] K. Finkenzeller. *Rfid Handbuch*. Hanser, 2002.

[Greiner 2013] Phillip Greiner. Design of an NFC and EPC-HF compatible Analog-Front-End with Load-Regulator. Master's thesis, TU Graz, 2013.

[Heyszl 2007] Johann Heyszl. System Research and RTL Design of a combined passive HF / UHF RFID Tag. Master's thesis, TU Graz, 2007.

[Klamminger 2013] Michael Klamminger. Improving the energy–consumption of a passive RFID–tag. Master's thesis, Graz University of Technology, 2013.

[Koopman and Jr. 1992] Philip Koopman and Jr. A preliminary exploration of optimized stack code generation. http://www.ece.cmu.edu/~koopman/stack_compiler/stack_co.pdf, 1992. [online; accessed September 17, 2013].

[Leens 2009] F. Leens. An Introduction to I2C and SPI Protocols. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4762946, 2009. [online; accessed September 17, 2013].

[MATLAB 2010] MATLAB. version 7.10.0 (r2010a), 2010.

[Moritsch 2012] Michael Moritsch. Verification of a combined passive HF/UHF RFID Tag with Universal Verification. Master's thesis, TU Graz, 2012.

[Odobasic 2012] Dino Odobasic. Digital RFID Frontend for a Sensor Chip. Master's thesis, TU Graz, 2012.

[OpenCores 2010] OpenCores. Wishbone B4. http://cdn.opencores.org/downloads/wbspec_b4.pdf, 2010. [online; accessed September 17, 2013].

[QuestaSim] Mentor Graphics QuestaSim. version 10.1a. http://www.mentor.com.

[Sadrusham] Nahi Jnanena Sadrusham. Clock Definitions. http://asic-soc.blogspot.de/2009/01/clock-definitions.html. [online; accessed September 17, 2013].

[Sakthikumaran *et al.* 2011] S. Sakthikumaran, S. Salivahanan, and V.S.K. Bhaaskaran. 16-bit RISC Processor Design for Convolution Application. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5972425, june 2011. [online; accessed September 17, 2013].

[Soeser 2012] Peter Soeser. Skriptum und Unterlagen zur Vorlesung aus Integrierte Schaltungen, 2012. Version 1.1.4.

[Steffan 2013] Christoph Steffan. Concept and Design of an integrated dc/dc Voltagecontroller a for on-chip Charge Supply. Master's thesis, TU Graz, 2013.

[Udar and Kagaris 2007] S. Udar and D. Kagaris. LFSR Reseeding with Irreducible Polynomials. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4274869, july 2007. [online; accessed September 17, 2013].

[Want 2006] R. Want. An Introduction to RFID Technology. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1593568, 2006. [online; accessed September 17, 2013].

[Wiessflecker 2013] Martin Wiessflecker. *Design of a Generic Low Voltage, Ultra-Low Power Sensor Interface for Wirelessly Powered IC.* PhD thesis, TU Graz, 2013.

[Yasui and Shimazu 1991] I. Yasui and Y. Shimazu. Microprocessor with Harvard Architecture. http://www.google.de/patents/US5034887.pdf, July 23 1991. [online; accessed September 17, 2013].