We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

4,800
Open access books available

122,000
International authors and editors

135M
Downloads

Our authors are among the

154
Countries delivered to

TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities

CLARIVATE ANALYTICS BOOK CITATION INDEX INDEXED

WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com

26

Object-Oriented Solutions for Information Storage on RFID Tags

Cristina Turcu, Remus Prodan, Marius Cerlinca and Tudor Cerlinca
Stefan cel Mare University of Suceava
Romania

1. Introduction

Already moving into the real world through a wide variety of applications, Radio Frequency Identification (RFID) technology uses radio waves to uniquely identify an entity (object, animal, or person). This data collection technology uses electronic tags to store identification data and other specific information, and a reader to read and write tags. A tag is a chip with an antenna. Tags fall into three categories: are active (battery- powered), passive (the reader signal is used for activation) or semi-passive (battery-assisted, activated by a signal from the reader). In certain tag types, the information on the tag is reprogrammable. There are existing and proposed RFID standards that deal with the air interface protocol (the way tags and readers communicate), data content (the way data is organized or formatted), conformance (ways to test whether products meet the standard) and applications (how standards are used on shipping labels, for example).
RFID solutions run at several frequencies:

- Low – from 125 KHz to 134 KHz (LF)
- High – 13.56MHz (HF)
- Ultra High – 860-960 MHz (UHF)
- Micro Wave – 2.45 GHz

The cost of simple RFID tags is likely to fall to roughly $0.05/unit in the next several years (Sarma, 2001), while tags as small as 0.4mm × 0.4mm, and thin enough to be embedded in paper are already commercially available (Takaragi et al., 2001). Such improvements in cost and size will ensure a rapid proliferation of RFID tags in many new areas.
The use of RFID is becoming more and more popular in industry, logistics, retail and other branches as an alternative to the barcode. In fact RFID tags are expected to replace conventional barcode labels due to their major benefits: high data storage capacity, read-write capability, read-speed rate, multiple entity identification, information updating, no-line-of-sight scanning, durability, and environmental resistance.
But how much data should be placed on RFID tags? There are two schools of thought:

1. as little as possible (just an ID);
2. more in support of efficiency and performance (e.g. item name, security data, etc.).

In the former case, the electronic product code (EPC) is a typical example. While the EPC standard continues to be adopted in various markets and employed in a wide range of applications (e.g. the retail supply chain), many RFID users are particularly interested in high-level functionality features to meet their own requirements. Using the EPC number as

Source: Development and Implementation of RFID Technology, Book edited by: Cristina TURCU,
ISBN 978-3-902613-54-7, pp. 554, February 2009, I-Tech, Vienna, Austria

an identifier certainly provides benefits, but there are many applications that require additional memory on the tag in order to more fully meet the needs of many users (***, 2008). This paper considers the latter case and focuses on the general methods of data storage on a passive HF tag operating at 13.56 MHz. The International Organization for Standardization (ISO) has created standards that define how data is structured on the tag for specific applications. For example, ISO 11784 and 11785 describe the structure and the information content of the codes stored in the tag for RF identification of animals. But an ISO 11784 dedicated application allows only the identification of animals and cannot be used in other domains such as product identification, for instance. The current memory capacity for commercially available HF tags is typically 128 bytes, or 256 bytes. But standards-based ISO tags such as those operating at the high frequency (HF) can now provide for up to 8 Kilobits of memory. For example, the announcement by Hewlett Packard (HP) of its memory Spot technology provides for an RFID chip containing 4 Megabits of storage that can be written to and read multiple times (Greene, 2006). Anyway, by having over 128-bits of dedicated user memory, the tags allow the storage of additional item information and opens up new application areas. This available memory can be used to customize an application and allows users the flexibility that a standard EPC tag or ISO 11784 dedicated applications cannot fulfill.

Furthermore, as more applications make use of the same tag technology, memories of different capacities could become available. The proposed solution has been designed to address this particular aspect, so that tags with different memory capacities may be used in the same system.

The general rule with any memory-based system has always been that no amount of memory is ever sufficient. Invariably, the response to enlarging the memory capacity of a system is to increase the scope of the application so that it requires even more memory. But, there is a strong relationship between price and capacity, larger memory capacities directly increase the cost per tag and the price of tags with a larger storage capacity is rather high. On the other hand, the RFID application implementation costs must be as small as possible, so as the costs of the tags be low as well. Although tag prices have considerably lowered lately, the price of large-memory tags has remained fairly high because added capacity always pays off. Under the circumstances, solutions are sought in order to increase memory capacity on the limited space of small and average tags (at an affordable price). There are needed solutions that could be adapted to a variety of activity domains. We propose a solution based on templates defined according with any users' requirements. A template is used to describe the data format to be applied for writing/reading data into/from tags.

2. Data types

The memory space on tags is entirely dependent on the data type used to store information. Thus, our solution proposes the use of some fundamental data along with additional data type defined by the user. The list of fundamental data types includes eight data types that are presented in Table 1. The data types have variable length. For each data type, both the occupied space and value ranges are determined.

Let us consider the date type. As this data type allows us to store date values in the YYYY-MM-DD format, 4 bits will be employed for the data type and 15 bits for the representation of the date value.

NO.	DATA TYPE	AMOUNT OF STORAGE (BITS)		DESCRIPTION, RANGE
		TYPE	VALUE	
1	BIT	4	1	true/false or yes/no;
2	INT4	4	4	non-negative integer values between 0 and 15
3	INT8/CHAR	4	8	non-negative integer values between 0 and 255 / ASCII char
4	INT12	4	12	non-negative integer values between 0 and 4095
5	INT16	4	16	non-negative integer values between 0 and 65535
6	INT32/REAL	4	32	real values or integer values represented on 32 bits
7	DATE_TIME	4	26	date and time values between 1st of January 1969, 0:00 and 31st of December 2031, 23:59
8	DATE	4	15	date values between 1st of January 1969 and 31st of December 2031

Table 1. Fundamental data types

The date value will be represented in accordance with the following specifications:

14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
century	year					MM				DD				

where the meanings 'century' and 'year' are indicated by the following formulas:

century = YYYY/100 – 19

year = YYYY-2000, if century = 1 (e.g. 2006)
2000-YYYY, if century = 0 (e.g. 1999)

The last two formulas may be simplified as follows:

$$year = (2 * century - 1) * (YYYY - 2000).$$

Different applications have different requirements and demand different data structures. Through this fundamental data type, new data type can be defined using list and class types. Using the list type, one is able to present different other kinds of data, such as strings (considered as list of characters), bits lists (which can be used in the case of true/false or yes/no operations), lists of integers, etc.

A user can also define a data type called *class* to represent a collection of different data (properties) and functions grouped together under a single name. The instance of class can be used as data field on a tag or as member object of another class. The proposed solution enables the definitions of each class to be encoded once at the beginning of the tag; the values of each data field of class type (instance/object) are encoded subsequently. But, four bits of memory should be used to encode the data type for each different data element and they prefix the corresponding values.

The class type is very useful especially if the information to be stored on the tag corresponds to certain logically grouped information types, and the information types in question are repeated with the same meaning on the same tag.

Defined data types are meant to compact the data for a more efficient encoding within the RFID tag memory and to organize the data in the memory. Consequently, a variety of data may be encoded and some of it may remain permanently locked. Furthermore, data types should support selective read/write operations, as well tag data updates.
The data fields and data encoding discussed later in this chapter only define, as example, a class for a particular application. But other classes may be easily defined, depending on users' needs.
Each data field should be defined by *data type*, *value* and several *associated attributes*. Thus, each data field can be associated with some attribute in the following categories:

- *empty*: indicating whether the associated field has no value;
- *read only*: data elements can be read, but not updated or erased;
- *normal*: read-write field;
- *codified*: on the RFID tag some data elements need to be represented by a value with certain meaning revealed by the template associated with the tag.

Users can define more than one new class and specify any access privileges (attributes).
The introduction of data types ensures the uniform coding of data, while the use of templates increases the flexibility of the data to be coded. Furthermore, fixed-length and variable-length fields may be easily handled simultaneously.

3. Script language

A user-defined class may contain data and function members. The user can define a function member as a subroutine (sequence of statements to perform an action) called *script*. This section includes the specification of the proposed script language and its instructions syntax. The defined data types and script language allow the data and the commands to be specified on a tag in a uniform way, irrespective of any particular application.
Our script language offers 16 distinct instructions presented in Table 2.
Using these instructions, users can easily define a script which can be compiled to byte-code on a PC or PDA. The code resulting from the compilation is small sized and can be stored on a tag. Every time the tag is read on a PC/PDA or even on a low-resources embedded device (station), the code can be interpreted and executed.

4. Data template

We described a solution where the possibility to define both the data type and the script has been taken into consideration. They may be easily tailored to the template which best suits the needs of a user for a given application domain. Other templates may be adopted at the choice of the user to meet any specific requirements.
A data tag is based on a specific template, which makes it unique not only within the particular domain of an application, but also among all other domains. A template:

- provides guidelines on how data shall be written on the tag;
- defines the desired data classes, based on specific requirements;
- specifies the data fields attributes;
- specifies the commands that are supported for defining the indispensable actions that must be executed at RF tag reading.

Since all data elements can be easily defined, users are free to use standard RFID tags and, depending on tag memory capacity, choose which data elements are most appropriate for a specific application.

INSTRUCTION	PARAMETERS	BINARY	OPCODE	BYTES	LENGTH
#DEFINE	BYTE	00000	0x00	1+1+1	3
	WORD	00001	0x01	1+1+2	4
INC	FIELD_NO	00010	0x02	1+1	2
	FIELD_NO_GIVEN_BY_CONSTANT	00010	0x02	1+1	2
DEC	FIELD_NO	00011	0x03	1+1	2
	FIELD_NO_GIVEN_BY_CONSTANT	00011	0x03	1+1	2
SETVAL	FIELD_NO, FIELD_NO	00100	0x04	1+1+1	3
	FIELD_NO, CONSTANT_VALUE	00100	0x04	1+1+1	3
	FIELD_NO_GIVEN_BY_CONSTANT, FIELD_NO	00100	0x04	1+1+1	3
	FIELD_NO_GIVEN_BY_CONSTANT, CONSTANT_VALUE	00100	0x04	1+1+1	3
IF	GATE==CONSTANT	00101	0x05	1+1	2
	Variable1 == Variable2	00101	0x05	1+1	2
	Vriable1 != Variable2	00110	0x06	1+1	2
IF	GATE != CONSTANT	00110	0x06	1+1	2
	FIELD_NO==CONSTANT	00111	0x07	1+1	2
	FIELD_NO!=CONSTANT	01000	0x08	1+1	2
	FIELD_NO_GIVEN_BY_CONSTANT == CONSTANT	00111	0x07	1+1	2
	FIELD_NO_GIVEN_BY_CONSTANT != CONSTANT	01000	0x08	1+1	2
EVENTS	SERVER_EVENT	01001	0x09	1+1	2
EVENTI	INTERNAL_EVENT	01010	0x0A	1+1	2
STOP		01111	0x0F	1	1
SCRIPT	SCRIPT_NO	10000	0x10	1+1	2
RETURN	-	10001	0x11	1+1	2
CONSTRUCTOR	SCRIPT_NO	10010	0x12	1+1	2
DESTRUCTOR	SCRIPT_NO	10011	0x13	1+1	2
GOTO	LABEL	10100	0x14	1+1	2
CALL	LABEL	10101	0x15	1+1	2
	SCRIPT_NO	10110	0x16	1+1	2
SETVAL_STRUCT	FIELD_NO0.FIELD_NO1. … VAR	10111	0x17	1+1+1+ …	2+…
SETVAL_STRING	FIELD_NO [INDEX] VAR	11000	0x18	1+1+1+1	4
GETVAL_STRUCT	VAR FIELD_NO0.FIELD_NO1. …	11001	0x19	1+1+1+ …	3+…
GETVAL_STRING	VAR FIELD_NO [INDEX]	11001	0x1A	1+1+1+1	4
SETVAL	FIELD_NO, VAR	11010	0x1B	1+1+1	3
GETVAL	FIELD_NO, VAR	11011	0x1C	1+1+1	3
ADD	Variable CONSTANT	11100	0x1D	1+1+1	3
IF	VAR1 == VAR2	11101	0x1E	1+1+1	3
	VAR1 != VAR2	11111	0x1F	1+1+1	3
//	COMMENT				

Table 2. Instruction set

Once the template has been created, users can specify the desired values for every defined field. On the other hand, the template is required for a complete understanding of the data tag in its entirety.

The logical tag content is divided into three sections as shown in Figure 1. Thus, the first logical section is a header which contains all the information regarding the physical and logical organization of the data on the tag. In this first section specific fields are included (for example, the length of the data tag) as well as information regarding the classes structure on which objects are instantiated in the tag data section. A class encapsulates the properties – data of the fundamental types (previously defined), and operations/methods of the class - described by scripts. A script will be considered as a static method that should be associated with a class rather than an object. Each class can vary in length and content (e.g. the number of data members, the type of data members, etc.). At the moment the implementation of all object-oriented programming principles (i.e. encapsulation, inheritance, polymorphism) is not an option because the tag memory size needed for this operation is considerably larger than the memory size available on current tags. The purpose of this header organized as a logical memory map is to provide extensibility for future, unanticipated data requirements.

The second section defined on the tag contains all the desired values. This section is associated with information encoded on the tag, and is made up of the data members and instances of the classes defined in the previous section. Whenever data must be encoded on the tag, the class notion defined in previously allows an optimization of occupied space.

The last section on the tag represents the map of the bits associated with each field or object on the tag. Within this map, there are 2 bits allocated for each field, and they are required for the encoding of the following states: normal (read-write), empty, read-only, codified (Figure 1).

Since the template structure is memorized directly on the tag, additional memory space is required for the definitions of classes and field types. This information is necessary in order to ensure the independence of tags. Furthermore, the classes and field types could be used by a low-resources embedded device (station) to interpret and modify the tags read.

When tag independence becomes an optional feature and when an application supports a large number of templates, there are other solutions to be sought. The solution we have chosen refers to the identification of each template through a unique number and the storage of this identification number associated with the template in the header section of the tag. The stations could memorize these templates and their associated identifiers. If the hardware resources do not support high memory usage, the template identified through the identification number on the current tag will be downloaded directly from a server. Obviously, this operation takes more time because it is necessary for the station to connect to the server and download the required template. In fact, much tag space is saved if the template identifier is memorized in the tag header.

If a tag template is unknown, its content cannot be interpreted and thus data privacy is ensured.

The evolution of the tag market and the demand for RFID-based applications will dictate the appropriate choice.

5. Security

The capacity of an RFID tag to be secured against unauthorized access, theft or damage is an issue to be considered. Some users may not wish to share the information and the data types

stored on their RFID tags with their competitors. The data generated and used in our RFID system represent a valuable asset characterized by confidentiality, integrity and availability. We have devised a method to verify or authenticate whether the information read from a tag is genuine; taking into consideration the serial number of the tag, the solution proposed does not require any database to verify whether a tag is a copy or a fake.

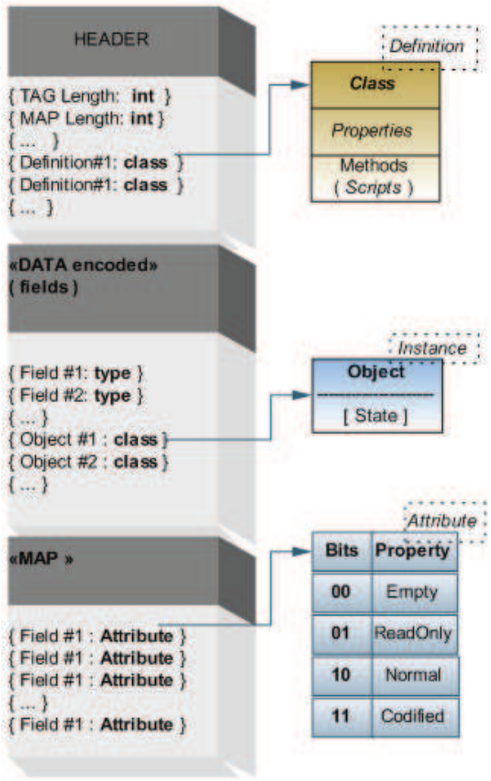

Fig. 1. The logical organization of a tag's content

One of the RFID privacy principles according to (BISG, 2004) consider that all businesses, organizations, libraries, educational institutions and non-profits that buy, sell, loan, or otherwise make available content to the public utilizing RFID technologies shall protect data by reasonable security safeguards against interpretation by any unauthorized third party. The template employed to define the data stored on the tag enables the reading of the tag content by authorized users only. Hence, it is impossible to identify the content of the tag without the corresponding template.

6. Case study

A selection of date type elements is presented below in order to help us illustrate how data encoding is performed and to estimate the amount of memory required to store this type of information. Within the adopted format, the length of these elements is either fixed or variable. Moreover, we have associated this format to a well-defined class depicted as a set of data and member functions. Data class members are stored sequentially in the tag memory.
To illustrate the implementation of our solution, we will consider an RFID system to be used in an automotive service center. The list of fields to be stored on the tag should include:

Minimum data for car identification

		FIELD	DATA TYPE
Section 1	1	License number	STRING
	2	Brand	codified
	3	Car body series	STRING
	4	Engine type	STRING
	5	Energy source	codified
	6	Cylinders [cm3]	INTEGER
	7	Owner	STRING
	8	Phone number	STRING
	9	Insurance company	codified
	10	Color code	STRING
Section 2	1	Gate 1	INT5
	2	Date and time	DATE_TIME
	3	Gate 2	INT5
	4	Date and time	DATE_TIME
	5	Gate 3	INT5
	6	Date and time	DATE_TIME
	7	Gate 4	INT5
	8	Date and time	DATE_TIME
	9	Gate 5	INT5
	10	Date and time	DATE_TIME
	11	Gate 6	INT5
	12	Date and time	DATE_TIME
	13	Gate 7	INT4
	14	Gate 8	INT4
	15	Gate 9	INT4
	16	Gate 10	INT4
	17	Gate 11	INT4
	18	Gate 12	INT4
Section 3	1	Counselor	codified
	2	Date and time of car entry	DATE_TIME
1	1	Operation type	codified
	2	Recommended operation	codified
	3	Scheduled operation	DATE
	4	Operator identity	codified
	5	Execution date	DATE
	6	Execution status	3 bits
	…		
10	1	Operation type	codified
	2	Recommended operation	codified
	3	Scheduled operation	DATE
	4	Operator identity	codified
	5	Execution date	DATE
	6	Execution status	3 bits

Since it is evident that the information contained in section two is repeated six times, a class with two data members (i.e. gate and date) should be considered. This class may be used to store more information concerning the gate number and the entry date of any identified vehicle.

Similarly, since the information contained in section three is repeated ten times, the redundant information may be eliminated if one considers the introduction of a class that contains data members (i.e. properties) corresponding to various fields of interest, namely operation type, recommended operation, scheduled operation, operator identity, execution date, execution status. However, any user may feel free to adopt other classes, templates or scripts.

If we consider the classes defined above, then we realize that the tag memory capacity required for all proposed fields is 197 bytes. If a dedicated application is used, namely an application in which the meaning of the values stored on the tag is predefined, the required tag memory capacity would be 156 bytes. Therefore, the proposed solution is certain to be able to optimize the memory space on RFID tags and to offer a high generality degree.

Let us suppose, for instance, that every driver entering the service center is given an RFID tag as an entry or parking pass. The following script has been defined in order to screen entry to secure areas in the automotive service center:

Example

```
# DEFINE  BYTE   FIELD_ENTRIES            0x01
    //First field from tag (how many entries)
# DEFINE  BYTE FORBIDDEN_BARRIER1  0x02
    //2'nd field from the tag (one forbidden entry)
# DEFINE  BYTE OPEN_BARRIER            0x0E
    //constant in embedded system (open the barrier)
# DEFINE  BYTE BEEP_ALARM              0x0D
    //also constant
IF (GATE == FORBIDDEN_BARRIER1)
        EVENTI (BEEP_ALARM)
    //beep if the car tries to enter in a forbidden area
IF (GATE == FORBIDDEN_BARRIER1)
        STOP
    //and then stop if forbidden door
IF (FIELD_ENTRIES == 0x00)
        STOP
    //no more entries allowed
DEC (FIELD_ENTRIES)
    //decrement the number of parking entries
EVENTI (OPEN_BARRIER)
    //open current barrier and let the car go inside parking area
STOP
```

The RFID tags used with this script should consider the fields presented in Table 3. The compilation process may be understood by following the information included in Table 4.

Let us consider only the values in the byte-code column for the IF instruction. The values 0x81 and 0x82 refer to the first defined and to the second defined field, respectively. This convention was adopted in order to help us distinguish between defined fields and integer values. For a maximum number of 127 of defined fields, the maximum integer number to be considered in an IF instruction is again 127.

DATA TYPE	NAME	LENGTH	EXPLICATIONS
BYTE	FIELD_ENTRIES	1	One byte that will be decremented every time the car enters to one allowed area of the parking.
BYTE	FORBIDDEN_BARRIER 1	1	One byte that contains the value of one restricted area. This value will never change.

Table 3. Allowed data types

INSTRUCTION	BYTECODES
#DEFINE BYTE FIELD_ENTRIES 0x01	0x00, 0x00, 0x01
#DEFINE BYTE FORBIDDEN_BARRIER1 0x02	0x00, 0x01, 0x02
#DEFINE BYTE OPEN_BARRIER 0x0E	0x00, 0x02, 0x0E
#DEFINE BYTE BEEP_ALARM 0x0D	0x00, 0x03, 0x0F
IF (GATE == FORBIDDEN_BARRIER1)	0x05, 0x81
EVENTI (BEEP_ALARM)	0x0A, 0x82
IF (GATE == FORBIDDEN_BARRIER1)	0x05, 0x81
STOP	0x0F
IF (FIELD_ENTRIES == 0x00)	0x07, 0x80, 0x00,
STOP	0x0F
DEC (FIELD_ENTRIES)	0x03, 0x80
EVENTI (OPEN_BARRIER)	0x0A, 0x83
STOP	0x0F

Table 4. Compiling script to byte-code

If we want higher values, then we must use the #DEFINE instruction.
Whenever an RFID-tagged car approaches a barrier with an RFID embedded system,

- the content of the tag is read;
- the script, if any, will be executed and tag values may be modified;
- the embedded device will decide upon opening the barrier, sending an event, running an internal event, etc.

We have implemented 2 versions of the *ScriptCompile()* function: one for the PC and another one for the PDA. We have also employed an *ExecuteScript()* function for every type of embedded device to be used. Our source code implementation of script language functions is compatible with ANSI C standard in order to be used with different compilers:

- MS Visual C++ and Borland C++ Builder at PC level;
- Embedded Visual C++ and .NET for Windows CE devices (PDA);
- Keil Compiler for 8051 compatible uC's;
- gcc for MicroBlaze soft processor.

The first version of the *ScriptCompile()* function source code for the PC was created using Microsoft VC++. The same source was successfully used with Borland C++ Builder. For Windows CE devices we have used either Embedded Visual C++, or EVC++ (for .NET programming).
Furthermore, the *ExecuteScript()* function was tested with Keil and gcc compilers. No malfunctions were reported at the level of the embedded system. Irrespective of the micro-controller used, the whole functioning was speedy and accurate.

7. Advantages and disadvantages

The presented specifications represent a flexible and multi-purpose solution which may be easily customized for a variety of purposes. Its major advantage, following the introduction of templates and class types, is that the amount of data on the tag may be increased upon request and there is no need to change the application. Secondly, depending on the application involved, users may select the relevant data to be encoded into the RFID tag memory.
Thirdly, the embedded devices support any type of microcontroller and no large memory capacity is required. As this solution proposes the implementation of processing logic on RFID tags, there is no need to modify the software of the embedded system to allow the same device to be used in different applications (e.g. security, parking, supply chain, etc.). Furthermore, this solution enables the provision of high-quality services with high additional values in numerous domains such as supply chain, security systems, or product tracking.
However, a series of disadvantages have been also identified:

- the execution of the script from the tag may take too long, depending on the length of the script, but also on the frequency of the processing device;
- if, through the commands of the script, a change in the value of a tag field is required, the processing time increases because several tag fields need to be written;
- after the script is stored on the tag, there is less storage space left for other interest information;
- the higher the amount of data on the tag, the longer the data reading time. Nevertheless new solutions have already been found: a high capacity, high-speed LSI for RFID tags complying with ISO/IEC15693.

8. Future developments

The most efficient encoding of tag information (e.g. the encoding of a character string presented as a field value) may be obtained by applying data compaction algorithms.
If codification and several security elements are considered, it is possible for the same value to be stored differently on two tags belonging to two different applications. In this way users may create personalized applications that will ensure a higher level of security. The implementation of such system requires a unique key for encryption/decryption, namely a combination between the tag ID and the key of some application. If the security standards are not met, there is no guarantee for data consistency when it is copied from one tag to another.
One major concern would be the design and development of a mechanism that will allow users to concatenate more tags (even with different classes) into a single one, without modifying the data class that already exists on the destination tag. This mechanism is suitable for the applications designed to assemble a small number of components into a final product. In this case, the tag associated with the final product will store information about the traceability of each component.

9. Conclusions

In this paper, we summarize our approach and our research. The discussion of different methods of data storage on a passive HF tag operating at 13.56 MHz has been the major

focus of the present paper. The HF tags features that include different form factors, different memory sizes (over 128 bits) and different read ranges allow users to design customized RFID systems according to their specific application requirements. We described a novel solution for the structural optimization of information storage on RFID tags. The method proposed is designed to reduce the cost of RFID-based applications by increasing the memory capacity on limited space of small and average tags. The solution devised and presented in this paper ensures the introduction of used-defined types to memorize any kind of information. Users may develop their own templates to describe information formatting, content and specifications after defining a set of classes to represent their own data. Furthermore, they are entitled to define a script to determine what the code should be executed in certain conditions. Through the use of user-defined templates and scripts, the presented system can be easily adapted to meet future needs of the user just as well as it meets today's needs. The created templates are used in order to write some product tags with specific information. This feature enables reading of the tag content for authorized user only. Hence, it is impossible to identify the contents of the tag without the corresponding template. Thus, these specifications define the security mechanism that deals with anti-theft. The proposed solution ensures a flexible and intelligent handling of tag information processing. This solution is expected to allow the development of an RFID-based generalized system that can be easily implemented in various domains (such as supply chain, security systems or product tracking) without any modifications in the structural level of software applications.

8. References

BISG (2004), BISG Policy Statement POL-002, Radio Frequency Identification, Available at: http://www.bisg.org/docs/BISG_Policy_002.pdf

Greene, K. (2006). Wireless Wonder Chip, *MIT Technology Review*, July 2006, Available at: http://www.technologyreview.com/read_article.aspx?id=17182&ch=infotech&a=f

Sarma, S.E. (2001). Towards the five-cent tag, *Technical Report MIT-AUTOID-WH-006*, MIT Auto ID Center, Available at: http://www.autoidcenter.org.

Takaragi, K.; Usami, M.; Imura, R.; Itsuki, R. & Satoh, T. (2001). An ultra small individual recognition security chip, *IEEE Micro*, Vol. 21, No. 6, pp. 43–49, November 2001, ISSN 0272-1732.

*** (2008), Capturing the Value of EPC Gen 2 Custom Commands, *NXP Semiconductors and Sirit Inc*, Available at: http://www.rfidproductnews.com/whitepapers/files/Custom_Commands.pdf

Development and Implementation of RFID Technology

Edited by Cristina Turcu

ISBN 978-3-902613-54-7
Hard cover, 450 pages
Publisher I-Tech Education and Publishing
Published online 01, January, 2009
Published in print edition January, 2009

The book generously covers a wide range of aspects and issues related to RFID systems, namely the design of RFID antennas, RFID readers and the variety of tags (e.g. UHF tags for sensing applications, surface acoustic wave RFID tags, smart RFID tags), complex RFID systems, security and privacy issues in RFID applications, as well as the selection of encryption algorithms. The book offers new insights, solutions and ideas for the design of efficient RFID architectures and applications. While not pretending to be comprehensive, its wide coverage may be appropriate not only for RFID novices but also for experienced technical professionals and RFID aficionados.

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen