

Implementation of Gen2 RFID Tag with Data Storage to Log and Maintain Vehicle History

Kavya Kollu, Z. Joan Delalic
Temple University

Sai Nava Patanjali Seshabhattar
University of Texas at Arlington

1947 N. 12th Street
Philadelphia, PA 19122
Email: (Joan, kavya.kollu) @ temple.edu
Tel: (215) 204 3382
Fax: (215) 204 5960

Abstract

This paper will address the application of RFID technology in areas such as supply chains like WALMART, Seven-Eleven, Vehicle Identification system, etc. The goal of this project is to develop a "RFID Tag" which is implemented using VHDL. The project deals with the creation of a Multipurpose RFID Tag following the Gen2 specification. This tag is passive, read/write enabled and can store data in the memory. It can also be connected to external memory, to support functions for different applications. This multipurpose RFID tag can be used in a Vehicle Identification System to maintain the vehicle history like accident history, violated safety rules, registration details such as owner's name, date of registration, expiration date of the registration, etc. The function of the RFID tag will be realized using VHDL code. The ASIC design can be realized using a HDL code, which will be synthesized using an RTL synthesis tool. The output after the RTL synthesis would be either a gate level net-list or a transistor level net-list. This output can now be used to either have an FPGA/ ASIC design. The antenna will be a dipole antenna coiled around the tag ASIC chip. The idea of implementing horn antenna using MEMS technology is addressed in this paper. The idea behind using MEMS based on antenna is to explore a new way of integrated antenna with the chip (Advantage of being very small). Finally, a good application for the RFID tag is to enhance police timing in recording an accident as well as buyers of older vehicles will automatically have the history of the vehicle in RFID tag attached to the vehicle.

Keywords: VHDL, RFID, GEN-2, MEMS.

1. Introduction

RFID wireless identification capabilities have resulted in automation of the identification technologies. An RFID tag is a radio frequency transponder that responds to a reader. The Gen2 protocol conforms to EPC Gen2 Specifications. There are many protocols related to UHF RFID, but the class 1 Gen 2 air interface protocol provides the features that enable deployment of RFID in UHF band.

A tag generally is expected to respond when a reader initiates a talk. In the Gen2 protocol, a tag does not initiate the communication. A tag undergoes

many state changes while communicating with a reader. Every state has its own set of associated idiosyncrasies. The command set helps in reading and manipulating tag information such as the Electronic Product Code.

The implementation and exploration of EPC Generation2 tag in VHDL is a tough proposition since aspects like the different transition states of the Gen2 tag in the RF environment need to be considered.

In this paper, the first part briefly presents the design methodology of the EPC class 1 Gen2 specification and focuses on the system level

simulation. In Section III, an overview of the GEN2 protocol is given. Section IV briefly describes an inventory process and Section V sheds light on VHDL basics. This is followed by information on state transitions and how VHDL implementation of the Gen2 tag is achieved. Then, the scope for future enhancements is presented and finally, the behavioral model of Gen2 tag is simulated and the results are documented (see Appendix).

2. Design Methodology

For the sake of simplicity the project is divided into two major sections, and under each section there are subsections. These are explained in detail after a brief overview on GEN2 RFID tag protocol and an overview on VHDL language.

The first section of the project deals with the idea of implementing GEN2 RFID tag in VHDL, adding external memory access to the GEN2 RFID tag and the RTL synthesis of the successfully written code. The second major section deals with the antenna for the RFID GEN2 tag.

The VHDL coding of GEN2 protocol is further divided into few more subsections which make the explanation easier. These subsections can be categorized into the following blocks.

- a. Command Evaluation block.
- b. Random Number Generator block.
- c. Seven state logic block. -- (Ready state, open state..... kill state)
- d. Slot counter block.
- e. CRC generator block.

These blocks implements the GEN2 tag functionality. A subprogram can be embedded in the program to implement the external memory section which will be useful in applications like Vehicle Identification systems.

3. Gen2 Protocol Overview

EPC Gen2 was developed to establish a standard for RFID tags targeted mainly for the supply chains. It has many enhancements compared to previous EPC class1, EPC class 0 protocols. The current standard for class 1 devices operates in the ultra high frequency (UHF) range (860-960 Mhz) and supports operation at long distances. It has better security when compared to class 0 and class 1 Gen1 protocol with a 32 bit kill password and access password. The Gen2 protocol has data rates starting from 40kbps to fastest rates of 640kbps. It makes the system more

rapid and robust. The EPC (electronic product code) is 96-bit in Gen2 but is 64 bit in class0 and class 1. Gen2 RFID reader is faster, reliable and time efficient as it can read more than 1000 tags per second depending on the RF environment. An EPC Gen2 tag uses digital modulation techniques for high efficiency and therefore is less affected by external or channel noise. Generally, both reader and tag use either ASK (Amplitude Shift keying) Modulation or PSK (Phase Shift Keying) Modulation or PR-ASK (Phase Reversal Amplitude Shift Keying) Modulation [1]. The modulation and demodulation techniques of a reader and a tag are same, but they use different encoding techniques (i.e., FM0 and Miller Modulation). The Gen2 tag memory is logically divided into four distinct banks. They are reserved memory, EPC Memory, TID memory and the user memory. The reserved memory contains kill and access passwords. The EPC memory contains the 5-bit PC (Protocol control bits), EPC (Electronic Product Code) and CRC-16(cyclic redundancy check). The TID memory contains an 8-bit class identifier at memory locations 00h to FFh. A user memory is created for the future use. There are fifteen input commands that the tag can receive from the reader [1]. These commands may change the state of the tag to one among the many possible states that the tag can undergo. A valid command may change the transition from current state to the next state of the tag if certain conditions are satisfied. Every state the tag undergoes is realized by a finite state machine and these are implemented using VHDL. A tag and reader works on different frequencies and timing scales. That is, a reader works faster than a tag. Therefore, timing constraints in the Gen2 specify the synchronization between the tag and the reader.

The Gen2 specification specifies the use of extensible bit vector (EBV). This EBV is an array of blocks. The EPC Gen2 uses blocks of length 8-bits i.e., EBV-8. EBV-8 can represent numerical values from 0 to 127 with a single block. The commands in which the EBV comes into existence are Read, Write, Blockerage and Blockwrite commands. When the tag fails to execute an access command, it backscatters an error code. Tag backscatters error codes only in open or secured states [1].

4. An Overview Of VHDL

VHDL stands for VHSIC (Very High Speed Integrated Circuit) Hardware Description Language. It can describe the behavior and structure of electronic systems, but is particularly suited as a language to describe the structure and behavior of

digital electronic hardware designs, such as ASICs, CPLDs and FPGAs as well as conventional digital circuits. There are three models in which a VHDL code can be implemented:

1. Behavioral model – It is a combination of parallel and sequential statements which describe the functionality of a digital logic.
2. Dataflow model - A VHDL dataflow description describes the function of a design by defining the flow of information from one input or register to another register or output.
3. Structural model –It is a modular digital hardware design methodology. Structure and behavior are complementary ways of describing systems.

VHDL supports both structural and behavioral descriptions of a system at multiple levels of abstraction. In this project, the VHDL code for the GEN2 tag simulation is realized using a behavioral model. A VHDL design consists of entities, architectures and configurations.

An entity is a specification of the design's external interface. Entity declarations specify the following:

1. The name of the entity.
 2. A set of generic declarations specifying instance specific parameters.
 3. A set of port declarations defining the inputs and outputs of hardware design.
- Generic declarations and port declarations are optional.

Entity declaration

```
ENTITY entity_name IS
  GENERIC(
    Generic_1_name : generic_1_type;
    -----
    Generic_n_name : generic_n_type;
  );
  PORT(
    Port_1_name: port_1_dir port_1_type;
    -----
    Port_n_name:port_n_dir port_n_type;
  );
END entity_name;
```

Architecture is a specification of the design's internal implementation. Multiple architectures can be created for a particular entity [2].

Architecture declaration:

```
ARCHITECTURE architecture_name OF
  entity_name IS
  BEGIN
  ---Insert VHDL statements to assign outputs to each
```

```
of the output signals
-- defined in entity declaration.
END architecture_name;
```

Advantages of using VHDL: Using a HDL language is beneficiary in implementing a digital logic. Apart from that VHDL offers several advantages to the designer:

Standard language, Readily available tools, Powerful and versatile description language[3], Multiple mechanisms to support design hierarchy, Versatile design reconfiguration support and Support for multiple levels of abstraction.

5. VHDL Implementation

Implementation using the model based design is the preferred methodology for the modeling, designing and simulating complex technical systems .The complexity of EPC class 1 GEN2 tag simulation involves special purpose state machines that are carried out as the different stages of the Gen2 protocol.

The communication process between the reader and the tag is called an inventory process. It is the reader command that make the tag to undergo various state transitions i.e., reader talks first. The inventory process is initiated by a query command issued by the reader. Based on the different tag states, the project is been bifurcated into different modules in order to help in easier simulation in VHDL.

The understanding of GEN2 specification is important before we start coding the protocol in VHDL. The basic tag functionality can be described as follows:

Reader issues a Query command which includes a Q parameter to define the number of slots (2Q-1 slots) in this inventory round. On receiving the Query command, each tag randomly selects a slot and generates a RN16 (16 bit random number). On receiving a QueryRepeat command, the tag decrements its slot value. If the slot value is zero, the tag responds to the reader by sending the RN16.

The reader identifies a particular tag and then acknowledges with an ACK command. The tag then checks for the validity of the RN16 from the reader and subsequently sends out PC (Protocol control bits), EPC (Electronic Product code) and CRC-16(Cyclic redundancy check bits) [1].

The reader issues Req_Rn command with same 16-bit random number to intimate the tag that PC, EPC

and CRC-16 are received. The tag then replies with a new 16-bit random number called “handle”. Thereafter, both the reader and the tag use the handle for the communication. The reader can access the tag with the help of this new RN16. The following blocks help understand the code appended in this document.

Command Evaluation block:

The command interpretation module receives bit sequences from the reader and decodes them to ascertain which of the many reader commands has just been received by it. The input bits entering the tag are handled in this block and are appropriately treated for suitable use in the next blocks. For example, as mentioned in table 6.1 of the EPC Gen2 specification document, each command length is realized by monitoring the initial bits known as command bits and the function of the command varies depending on the rest of the bits of that particular bit [1].

Let us consider the important command of all which is responsible for initiating an inventory process i.e., the query command. The first four bits “1000” are the command bits which specify that the incoming command is the Query command and is of 22 bit length. The command evaluation block evaluates the command that specifies the meaning of each bit in the command and performs necessary action depending on the later 18 bits of the Query command.

Random Number Generator block/ Pseudo Random Number Generator

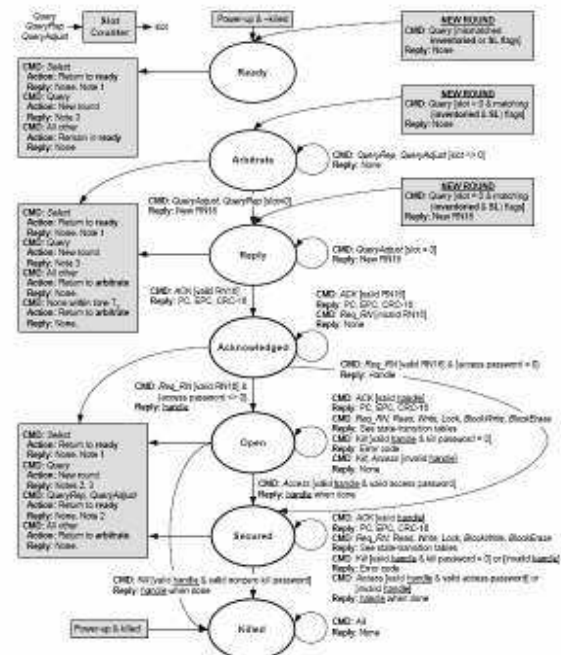
This block is responsible for generating a 16-bit random number (RN), which is useful for the slot decrement counter. Upon receiving a Query command, tags are required to pick a random value in the range $(0, 2^Q - 1)$ inclusive, and this value is used in the slot counter decrement ‘Q’ in the range which is determined from the incoming query bits. The bits 14 through 18 in the query command are the Q bits which specify the Random number range.

The random number is generated at least twice in the communication between reader and a tag. The second time random number generated is called a “handle”. The random number and / or handle serve as identification in further communication between reader and tag.

The Random number generator is implemented using shift register behavior. The random number is backscattered by the tag when reader talks first.

Tag State block

Every state defined in the Gen2 specifications is acted upon by many commands, most of them being classified as either mandatory or optional commands. Every command is characterized by a code, number of bits (length) and protection sequences wherever applicable. Query, QueryAdjust, QueryRep, ACK and NAK constitute the inventory command list. All these commands have been defined as mandatory including Req_RN, Read, Write, Kill, Lock, Access, BlockWrite and BlockErase. The last three commands are optional, which mean that they will be ignored by tags which do not support them. The individual bit details for each command are vividly explained in the EPCglobal Gen2 specification sheet. In this implementation of the Gen2 tag, all commands originating from the reader are taken as user defined inputs to the tag in the VHDL environment. The following tag state diagram depicts the tag state behavior in general.



Implementation using the model based design is the preferred methodology for the modeling, designing and simulating complex technical systems. The complexity of EPC class 1 GEN2 tag simulation involves special purpose state machines that are carried out as the different stages of the Gen2 protocol.

As mentioned before, the command interpretation module receives bit sequences from the reader and

decodes them to ascertain which of the many reader commands has just been received by it. To validate an incoming data stream, CRC bits are verified. The output of this module is given as parallel input to the state determination module which decides on the next course of action to be taken. Finally, the tag transmitter module helps serially backscatter all required signals to the reader. Details about the logic used to implement the command interpretation module and the tag transmitter module have been visited upon later.

The state determination module consists of various sub modules which when integrated elucidates the overall functioning of the module. Each state machine is depicted as a flow chart encompassing all possible conditions and situations during a reader-tag communication. The simulation flow charts based on which the VHDL code has been written for each state sub-module are explained.

1. Ready state

A tag enters into this state as and when it enters into the radio-frequency sphere of influence of the reader. Until then, it is inactive. The ready state is prevalent until the tag receives the Query command from the reader. The tag returns to this state from any other state if at all it loses power and regains it. A killed tag does not fall under these considerations.

2. Arbitrate state:

The tag comes into this state on reception of the Query command issued by the reader. The slot counter value is decremented every time a QueryRep. command is received. A non- zero tag slot value shall result in retention of the arbitrate state in the tag. For QueryRep though, the value of the slot is checked only after decrementing the slot counter. All other signals ranging from ACK to BlockErase do not affect the state under all conditions. The Select command causes state transition to Ready from the arbitrate state.

3. Reply state:

The random number generator module generates a 16 bit random number RN16 which is backscattered by the tag. If this number is acknowledged by the reader, the tag backscatters its protocol control bits (PC), Electronic Product Code (EPC) and the CRC-16. Or else, the tag returns to the previous state.

4. Acknowledge state:

The tag can jump to any state except the killed state from here. The presence of NAK signal is said to cause the acknowledged state to jump to the arbitrate condition. The QueryRep and QueryAdjust

commands cause inventory transition between two states and change to the ready state.

5. Open state:

When the access password of the tag is not equal to zero, the tag comes in to this state from the acknowledged state upon receiving Req_RN command from the reader which requests for a new 16 bit random number. The tag then forwards a new RN 16 called handle which is used in the future communication between itself and the reader. A tag in this state can have transition to any other state other than the acknowledged state. All invalid signals do not cause any state transitions. The presence of NAK signal is said to cause the open state to jump to the arbitrate condition. The QueryRep and QueryAdjust commands cause inventory transition between two states and change to the ready state.

6. Secured state:

If the tag access password is zero, then it directly transitions to this state from the acknowledged state. The same exchange of handles (as explained in the open state module) is followed here. Transitions out of this state can be to all other states other than the open or acknowledged state. The QueryRep and QueryAdjust commands cause inventory transition between two states and change the present state to the ready state.

7. Killed state:

This command disables a tag permanently (irreversible). The reader is notified of the successful operation of this command. In the VHDL implementation, the "kill" state has not been explicitly implemented as a separate state module. In fact, the kill command makes sure that the tag enters the virtual "kill" state which means cessation of all tag functionalities.

Slot counter block:

As mentioned earlier, the query command determines the 'Q' value which in turn responsible for a random number 16. This number is passed into the slot counter block which is decremented until there is an inventory time slot which can bridge the communication between the reader and tag. In a multiple tag environment a reader has to interact with many tags and so the reader assigns each tag a time slot called an inventory session. Thus when the slot counter is zero, there will be active communication setup between the present tag and the reader. Therefore a slot counter block is essential to start an inventory round.

CRC Generator block:

To validate an incoming data stream, CRC bits are verified. CRC stands for Cyclic Redundancy Check. A CRC-16 is generated for further use in commands and verification as a standard to check the incoming bits and to ensure that the communication is correct. The formula based for creating 16-bit CRC is $x^{16} + x^{12} + x^5 + 1$.

These are the five major blocks which help understand the code appended with this document. Apart from this many other commands, and functionalities specified in the GEN2 specification document are also coded.

Apart from these the following code can be written as a subprogram and embedded in the code which gives accessibility to external memory (ROM) which might be necessary in some applications which need higher memory space to write or read the contents. The following code is a sample code, the variables, signals etc may be required to be renamed while embedding in the real code.

```
entity rom_mem is port
(
  data_out: out std_logic_vector(31 downto 0);
  address: in std_logic_vector(2 downto 0)
);
end rom_mem;
--
--
architecture behavioral of rom_mem is type
  mem_array is array(0 to 7) of std_logic_vector(31
  downto 0);
  signal mem: mem_array;

begin
  read: process(address) -- index is an index into the
  array
  variable index: integer range 0 to 7 := 0; -- initialize
  index to 0

begin
  -- initialize the ROM contents
    mem(0) <= x"70000000";
    mem(1) <= x"00018000";
    mem(2) <= x"00180000";
    mem(3) <= x"01800000";
    mem(4) <= x"18000000";
    mem(5) <= x"00000001";
    mem(6) <= x"00000002";
    mem(7) <= x"00000004";
  -- The conv_integer function is in the .std_logic_arith
  package
```

```
-- Convert the address vector into an integer to access
array element
index := conv_integer( address(2 downto 0));
data_out <= mem(index);
end process;
end behavioral;
```

6. RTL SYNTHESIS OF THE VHDL CODE

The code written implementing the GEN2 code (also along with external memory for some applications) is simulated and the graphical results are examined to verify if the functionalities are implemented correctly. To realize the hardware structure for the code written we have to synthesize the code. To be able to synthesize a code, the code must be written appropriately. That means, while writing the code it must be kept in mind that the final code should be able to yield a physical realizable hardware structure.

The code appended to this paper can be synthesized. This synthesis is called RTL (Register Transfer Level) synthesis. This requires a synthesis tool which will be able to convert a VHDL code, understand and generate appropriate gate level net-list. The gate level net-list consists of the many basic logical gates interconnected in such a way that performs the function as written in the VHDL code. The input for the synthesis tool is a successfully compiled and simulated VHDL code saved as .vhd file. That is a .vhd file is an input to the simulator and a series of configuration will finally give us a gate level net-list output.

Most synthesis tools can also generate a transistor level net-list, with certain settings while synthesizing the code. Finally, the digital circuit will be the output which can be sent for further manufacturing to obtain an RFID tag chip which works as specified in GEN2 specification document [2,4].

7. Antenna

Antenna is an important device essential in every wireless communication system. Similarly, an RFID reader and tag uses specific antennas to accomplish their work. In this paper we address using a dipole antenna.

A RFID tag antenna is responsible for powering up the tag and backscatter the signal coming from the reader. The antenna in an RFID tag is a conductive element that permits the tag to exchange data with the reader. Passive RFID tags make use of a coiled

antenna that can create a magnetic field using the energy provided by the reader's carrier signal. Passive RFID tags utilize an induced antenna coil voltage for operation. This induced AC voltage is rectified to provide a voltage source for the device. As the DC voltage reaches a certain level, the device starts operating. By providing an energizing RF signal, a reader can communicate with a remotely located device that has no external power source such as a battery. Since the energizing and communication between the reader and tag is accomplished through antenna coils, it is important that the device must be equipped with a proper antenna circuit for successful RFID applications. An RF signal can be radiated effectively if the linear dimension of the antenna is comparable with the wavelength of the operating frequency. Therefore, it is difficult to form a true antenna for most RFID applications. Alternatively, a small loop antenna circuit that is resonating at the frequency is used. A current flowing into the coil radiates a near-field magnetic field that falls off with r^{-3} . This type of antenna is called a magnetic dipole antenna.

To power up a tag and input the incoming data stream we use a dipole antenna. The same antenna can be used to backscatter the output signal from the tag or a different antenna can be used for backscattering. This paper presents the idea of using the MEMS based horn antenna for backscattering the signal from tag to reader $T \Rightarrow R$ communication. The horn antenna uses MEMS technology to be created. The design of the antenna that is the physical length of the antenna and the horn will be chosen depending on the frequencies the tag deals with. The surface micromachining [7] and bulk micromachining [7] techniques will be used to for the waveguide antenna and radiate the outgoing bits. One advantage of using MEMS might be use of silicon and integrating it with the tag chip. The same silicon area can be etched and micromachined [7] to realize the chip.

But for now, the paper addresses the use of dipole antenna, for incoming as well as backscattering the signal, that is for full duplex communication. This is because the dipole antenna is widely used in such applications and are proved to be reliable.

8. Conclusion

We have presented the design methodology of a RFID tag with external memory using EPC Class-1 Gen2 protocol. The approach of realising a gen2 protocol using VHDL would make RFID technology widely used in many areas like supply chains, police department and so on. This can prove to be

interestingly useful when the next version of protocol is designed, as upgrading of the VHDL code would not be a difficult task. In the paper we have considered the application of the tag to vehicle identification which enables the police to keep track of records of the vehicle accidents automatically using the corresponding reader and storing the same in tag memory too. For future work, this code can be upgraded to serve many other related functions in the application chosen. The major considerations for this project can be to review the VHDL code and address the timing constraints and other implicit blocks of the Gen2 protocol, (implicit blocks, like serial incoming stream and serial outgoing stream of command bits. Another important upgrade would be the use of integrated antenna design which makes the whole tag device realizable in single chip.

References

- [1] EPCTM Radio-Frequency Identity Protocols Class-1 Generation-2 UHF RFID Protocol for Communications at 860 MHz – 960 MHz Version 1.0.9
<<http://www.epcglobalinc.org/standards/>>
- [2] Sudhakar Yelamanchili, "From simulation to synthesis", Prentice Hall, ISBN-10 0130809829, 2001
- [3] J. Bhasker, "A VHDL Primer", Prentice Hall Third Edition, ISBN 0-13-096575-8, 1999.
- [4] Douglas Perry "VHDL Douglas Perry", MacGraw-Hill 1993, ISBN 0-07-049434-7.
- [5] Antenna Circuit Design for RFID applications by Youbok Lee, Ph.D, Microchip Technology Inc. <http://ww1.microchip.com/downloads/en/AppNotes/00710c.pdf>.
- [6] Ying Cai, "RF MEMS and Integrated Manufacturing for miniaturized millimeter-wave systems" Presented to the Faculty of the Graduate School of The University of Texas at Arlington, August 2006.
- [7] William Trimmer "Micromechanics and MEMS", IEEE press, ISBN 0-7803-1085-3, 1997.

VHDL code and simulation results are included in the appendix. We will provide them to you **on demand**.