DESIGN AND DEVELOPMENT OF A STATE TRANSITION TABLE FOR THE EPCGLOBAL UHF CLASS1 GEN2 RFID STANDARD

by

Akram Kamrani

B.S., Sharif University of Technology, 2007

Submitted to the Graduate Faculty of

Swanson School of Engineering in partial fulfillment

of the requirements for the degree of

Master of Science

University of Pittsburgh

2009

UNIVERSITY OF PITTSBURGH

SWANSON SCHOOL OF ENGINEERING

This thesis was presented

by

Akram Kamrani

It was defended on

December 1, 2009

and approved by

Marlin Mickle, Professor, Electrical and Computer Engineering

Thesis Co-Advisor: Bryan A. Norman, Associate Professor, Industrial Engineering

Thesis Co-Advisor Jayant Rajgopal, Associate Professor, Industrial Engineering

DESIGN AND DEVELOPMENT OF A STATE TRANSITION TABLE FOR THE EPCGLOBAL UHF CLASS1 GEN2 RFID STANDARD

Akram Kamrani, M.S.

University of Pittsburgh, 2009

A Radio Frequency Identification (RFID) system is a wireless network composed of a reader which tries to communicate with and read (identify) a subset of tags from among a larger population. There are several standards defined for this type of network including the EPCGlobal UHF Class1 Gen 2 RFID standard which is the focus of this work. The increasing number of tag and reader manufacturers necessitates the design of standard conformity tests. To create conformity tests one needs to know the possible states and transitions between states that the tags may experience. In this work a tag is modeled as a state transition machine where the tag transitions from one state to another upon receiving commands from the reader. The model is Markovian; i.e., each transition only depends on the current state and the received command and is independent of the previous transitions, states and commands. The contribution of this work is to fully define the transitions of this model such that it conforms to the specification of the EPCGlobal standard. This, in turn, is necessary for the design of a conformity testing methodology.

Considering all the configurations of the internal tag parameters and command parameters yields a state transition machine with 449 unique states and 141 unique commands. For each unique command there is one unique transition out of each state; thus, there are 63,309 unique transitions that need to be considered. The transition machine has been expressed as a table which has states as rows, commands as columns and the future states as the table entries. This

table has been automatically filled using roughly 3000 lines of MATLAB code. A Graphical User Interface (GUI) has also been developed to test and verify the transitions

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1.0 INTRODUCTION

The purpose of the work described in this document is to develop a state transition machine model of EPCGlobal UHF Class 1 Gen 2 RFID tags as a basis for the development of standard conformity tests. It is important to know all of the possible tag states and transitions between states in order to develop a conformity test to verify that the tag responds appropriately in all contexts. First, some background is presented on RFID systems and the EPCGlobal standard.

An RFID (Radio Frequency Identification) system is a wireless network composed of a reader (interrogator) which tries to identify a tag (transponder)[1] from among a population of tags and read the information stored on the tag's chip. Such systems are used generally for identification purposes; e.g. to detect the presence or absence of a tag with a certain serial number in some area of interest. A series of radio-frequency signals will be transmitted between the reader and the tag until the tag is "singulated" by the reader and the reader can read the unique codes stored in the tag's memory. In this context, singulation of a tag means uniquely identifying one specific tag from a larger population of tags.

In any RFID standard there is a section regulating the tag-reader interaction, which specifies what types of commands the reader can send to the tag and how the tag should react. Henceforth, this section will be known as the tag identification protocol (tag protocol). The

[1] The terms reader/interrogator and tag/transponder will be used interchangeably throughout the text and are equivalent.

purposes of this section are to determine when tags can transmit their signals to the reader along with what they should transmit, and to give the reader a set of commands which to control the tags' behavior. In a layered protocol stack view (Tanenbaum, 2003) of the RFID system, the tag identification protocol is part of the data link layer. This work focuses solely on this part of the protocol.

A number of different RFID system designs have been proposed, implemented and even deployed and standardized; these systems differ in many areas, such as:

- *Frequency Band*: RFID systems operate in many different frequency bands using many different bandwidths. As with other short-range communication systems like WiFi and Zigbee, many of these systems operate in unlicensed frequency bands.
- *Active or Passive Tags*: Active tags carry their own power sources, while passive tags must be energized off the signal that they receive from the reader. Passive tags are cheaper but generally operate at shorter range and slower speeds than active tags.
- *Data link layer*: Each of the standards could use a different method for interference avoidance, tag identification, etc.

For the remainder of this work the discussion will focus on the EPCglobal Class1 Generation2 standard which operates in the 860 MHz – 960 MHz range using passive tags. The tag protocol of the standard will be described briefly in the following section and in more detail later.

1.1 EPCGLOBAL CLASS1 GEN2

The latest, and most widely used, of the EPCglobal tag protocols is Class 1 Generation 2; hence, it will be the focus of the remainder of this work. The EPCglobal Class 1 Gen 2 protocol improves upon previous protocol versions in the following areas:

i. Speed and Flexibility

ii. Reliability

iii. Robustness (improved by the following features):

    1. Q protocol: Designed to manage a large population of tags
    2. Dual-state symmetry (A/B): Allows each inventory cycle to be independent from inventory cycles previously undergone by the tags
    3. Mitigating reader interference
    4. Sessions: Allows multiple readers to access the same population of tags simultaneously

v. Security: Kill, access and lock passwords and the ability to control reader access to tags.

vi. Extensibility: The commands and responses are structured such that new commands can be added without fundamental changes to the standard.

The focus of this work is the data link layer of this standard. Specifically, the inventory cycle, which describes the series of commands and responses traded between the reader and tags, is of interest. In the EPCglobal Class1 Gen2 protocol, the inventory cycle starts with the transmission of a command from the reader, then there is a transition of tags between several states and changes in the tags' status bits, until each tag is singulated by, and its EPC sent to, the reader. The process can be continued leading to other stages for accessing, writing or otherwise

manipulating the information stored in the tag's memory. The details of this process will be discussed in section 3.0 .

Another aspect of interest is the structure of a tag's memory, which contains all the registers and variables that determine a tag's states. Tag memory is logically separated into four distinct banks, each of which may comprise zero or more memory words. A logical memory map is shown in Figure 1 (EPCglobal Inc., 2008).

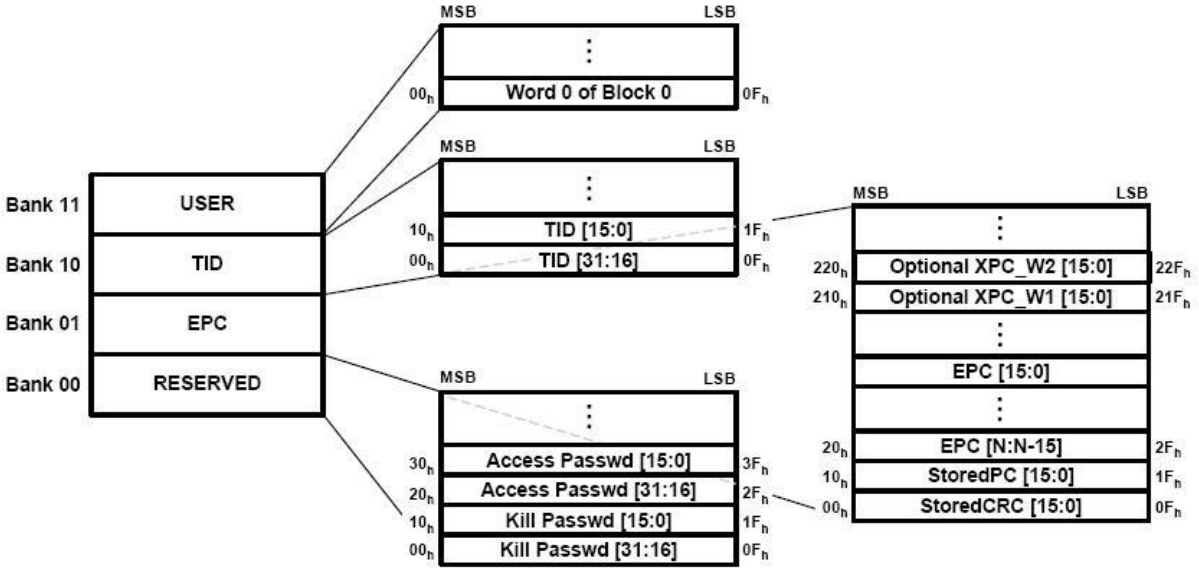

Figure 1- Logical memory map (EPCglobal Inc., 2008)

- Reserved Memory contains the kill and access passwords. If a tag does not implement the kill and/or access password, the tag shall act as though it had zero-valued password(s) that are permanently read/write locked and the corresponding memory location in Reserved memory need not exist.
- EPC memory contains a CRC-16, Protocol-Control (PC) and a code (such as an EPC) that identifies the object to which the tag is or will be attached.

- TID memory contains an 8-bit ISO/IEC allocation class identifier. It contains sufficient identifying information for an interrogator to uniquely identify the custom commands and/or optional features that a tag supports.
- User memory allows user-specific data storage. They memory organization is user-defined.

1.2 TERMS AND DEFINITIONS

Command set:

- A set of commands used to explore and modify a tag

Operating Procedure:

- Collectively, the set of functions and commands used by an interrogator to identify and modify tags (also known as the tag-identification layer)

Query:

- Query is a mandatory command that initiates and specifies an inventory round. It includes Sel, Session, Target and Q fields.

Inventory Round:

- A period initiated by a Query command and terminated by either a subsequent Query command or a Select command

Q:

- A parameter that an interrogator uses to regulate the probability of tag response. An interrogator commands tags in an inventory round to load a Q-bit random (or pseudo-random) number into their slot counters; the interrogator may also command tags to decrements their slot counters. Tags reply when the value in their slot counter is zero. Q

is an integer in the range [0, 15]; the corresponding tag-response probabilities lie in the interval [1, 0.000031= 2^{-15}]

Query Rep:

- Query Rep is a mandatory commands that instructs tags to decrement their slot counter and if slot=0 after decrementing, to backscatter an RN16 to the interrogator. Query Rep includes session fields.

Query Adjust:

- Query Adjust is a mandatory command that adjusts Q without changing any other round parameters.

Session:

- An inventory process comprising an interrogator and an associated tag population. An interrogator chooses one of four sessions and inventories tags within that session. The interrogator and associated tag population operate in one and only one session for the duration of an inventory round. For each session, tags maintain a corresponding inventoried flag. Sessions allow tags to keep track of their inventoried status separately for each of four possible time-interleaved inventory processes, using an independent inventoried flag for each process.

Random-slotted collision arbitration

- A collision-arbitration algorithm where tags load a random number into their slot counter; decrement this slot counter based on interrogator commands, and reply to the interrogator when their slot counter reaches zero.

Single-Interrogator Environment:

- An operating environment within which there is a single active interrogator at any given time.

Singulating:

- Identifying an individual tag in a multiple-tag environment

Slot:

- Slot corresponds to the point in an inventory round at which a tag may respond. Slot is the output value of a tag's slot counter; tags reply when their slot value equals zero.

Ack:

- Ack is a mandatory command. An Interrogator sends an ACK to acknowledge a single tag. Ack echoes the tag's backscattered RN16

NAK:

- NAK is a mandatory command and returns all tags to the arbitrate state unless they are in the ready or killed state, in which case the tags ignore the NAK and remain in their current state.

2.0 PROBLEM STATEMENT

As demand for RFID systems increases, various issues have led to an increasing number of manufacturers producing readers and tags; these issues include the amount of data needed to be saved on tags, new applications, specific features needed for specific functions, high demand for low-cost tags and readers, etc. Of course, the tags (and readers) produced must conform to the existing standard; as such, a method of testing said conformity is of paramount importance.

The rest of this work deals specifically with how tags transition from "state" to "state" in response to various "commands" from the reader. Whenever a reader tries to communicate with tags it transmits a signal containing some parameters which is called a command. There are 14 commands described in the standard that a reader can send to the tag to read or access data registered in the tag's memory. In this work any possible parameter configuration for each of the commands is considered an individual command. Each tag has a set of parameters (internal bits) that control a tag's behavior in any given situation (for example a tag's response to a specific command depends on these bits). Any unique combination of these parameters is considered a state. For each successful read, each tag needs to transition through several states.

In response to a specific prompt, the tag behavior that needs to be checked includes: whether the tag moves to the appropriate states, whether it follows the expected path of states, whether it sends the appropriate response(s) to the reader, whether it reacts suitably to unexpected prompts, noise, incomplete and erroneous signals, etc.

In order to test the conformity of the tag structure and design to the standard, one approach is to enumerate all of the possible combinations of states and commands and to check the tag's reaction in all of the situations and compare it with the expected behavior from the standard. This would be achieved by programming a reader to issue a certain series of commands to a tag while monitoring the tag's responses. Ideally the sequence of commands would subject the tag to every possible state-command combination and do so in as few steps as possible.

So a model of a tag as a state transition machine based on the standard is required, containing all of the tag's possible states and all of the possible reader commands and the appropriate transition and response in each situation (state/command combination). This model is necessary to design the optimal sequence of commands and also allows the reader to check the tag's responses at each stage. This state transition machine can be uniquely described by a state transition table which has states as rows and commands as columns, and the entry in row *i* and column *j* represents the next state of the tag when it is in state *i* and receives command *j* from the interrogator.

3.0 METHODOLOGY

Certain specifications and requirements have been defined by EPCglobal for the UHF Class 1 Gen 2 RFID protocols (EPCglobal Inc., 2008) (Alien Technology, 2005). There are conformity requirements expected from readers and tags that must be addressed by all of the RFID reader and tag manufacturers in order to be certified by EPCglobal.

RFID system architecture is that of a layered wireless communication network (Tanenbaum, 2003). EPCglobal defines certain requirements for different layers of the RFID system, including the physical and data link layers. In this analogy, tag identification would be the lowest sub-layer in the data link layer of an RFID network. As such, the focus of this work is on the data link layer protocols. A discussion and understanding of the physical layer and transmission schemes is beyond the scope of this work.

The tag identification layer consists of the operations through which a population of tags is addressed, identified and modified by one or more readers (also referred to as interrogators) (EPCglobal Inc., 2008; Alien Technology, 2005). In the tag identification layer a reader uses the three basic operations of Select, Inventory and Access to manage a population of tags. In Select operations, the reader sends one or more Select commands with different parameters to address a specific set of tags. In Inventory operations, the reader attempts to read (acknowledge and receive the EPC of) all of the tags in that particular set. During the Inventory operations, Query, Query Adjust, Query Rep, Ack and NAK commands will be sent by the reader; these are called

Inventory commands (See Section 1.2: "Terms and Definitions" for the definitions of the command names and acronyms used here and throughout this chapter). These commands are used to singulate tags in the Inventory operation. Access operations follow the Inventory operations, after each tag is uniquely identified and is ready for the reader's access. Through Access operations, the reader can read from and/or write to any tag's memory.

Through these three basic operations, a population of tags can be managed by one or more readers. In any of these operations, upon receiving a particular command a specific reaction is expected from the tag. Before discussing this reaction any further, the tag's structure and parameters must be detailed.

In the EPCglobal, some minimum requirements for the physical structure of Class 1 Gen 2 tags are specified as follows:

- An Electronic Product Code (EPC) identifier
- A tag identifier (tag ID)
- A Kill function which permanently disables the tag
- Access control (optional password-protection)

A tag's Electronic Product Code (EPC) will be sent to the reader during the inventory operations; if the reader receives this code successfully, the tag is singulated (i.e., uniquely identified).

Sessions: Another concept introduced in Class 1 Gen 2 tags is that of sessions. Four sessions, S0, S1, S2, S3 have been defined in this standard. The main purpose of these sessions is to allow more than one reader, independent of others, to manage the same population of tags at the same time. To make this possible, each tag should maintain four different sessions and for

each session a two-status inventoried flag, with possible values A or B. When the tags are read by the reader their inventoried flag switches from A to B or vice versa.

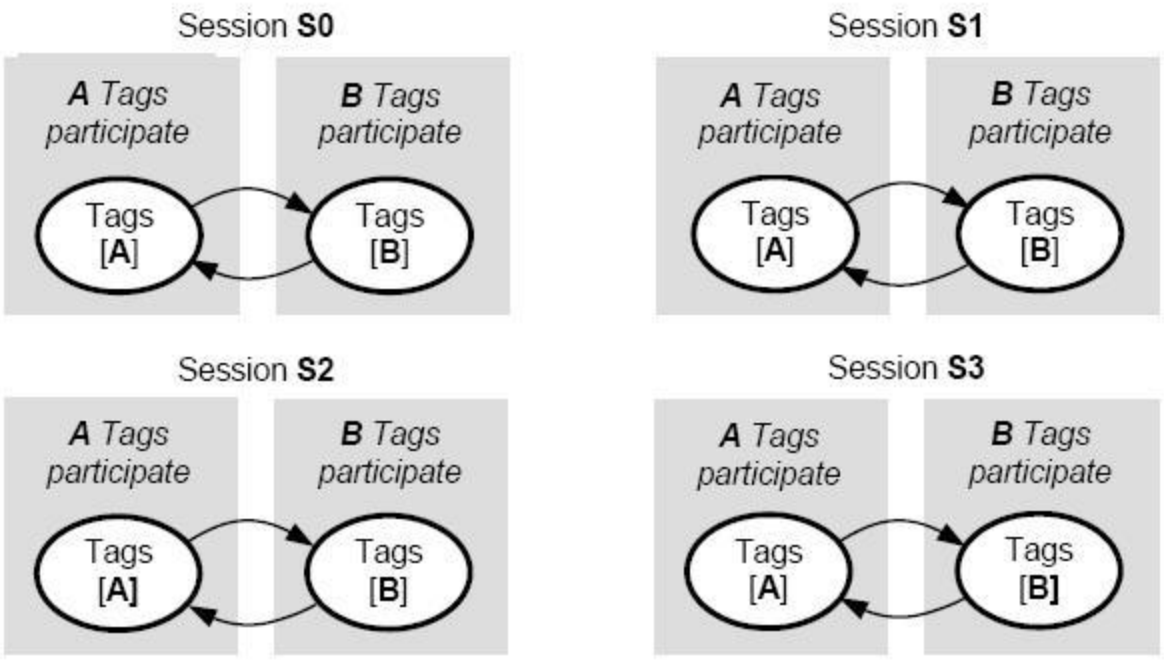

Figure 2 - Session diagram (EPCglobal Inc., 2008)

Selected flag: There is another binary flag associated with each tag, known as the "Selected flag". The possible values for the selected flag are SL (selected) or ~SL (not selected). This flag is used in the select operations, where the reader is trying to address a particular set of tags. The status of the Selected flag and/or Inventoried flag can be changed through commands.

States: According to EPCglobal Class 1 Gen 2, tags shall implement seven states. A tag goes through these seven states during the Select, Inventory and Access operations (Figure 3). These seven states are Ready, Arbitrate, Reply, Acknowledged, Open, Secured and Killed (Figure 4).

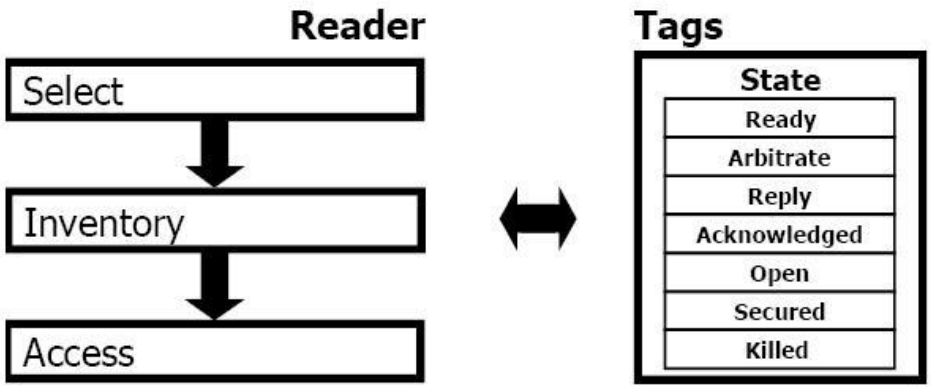

Figure 3 - Interrogator/Tag operations and tag state (EPCglobal Inc., 2008)

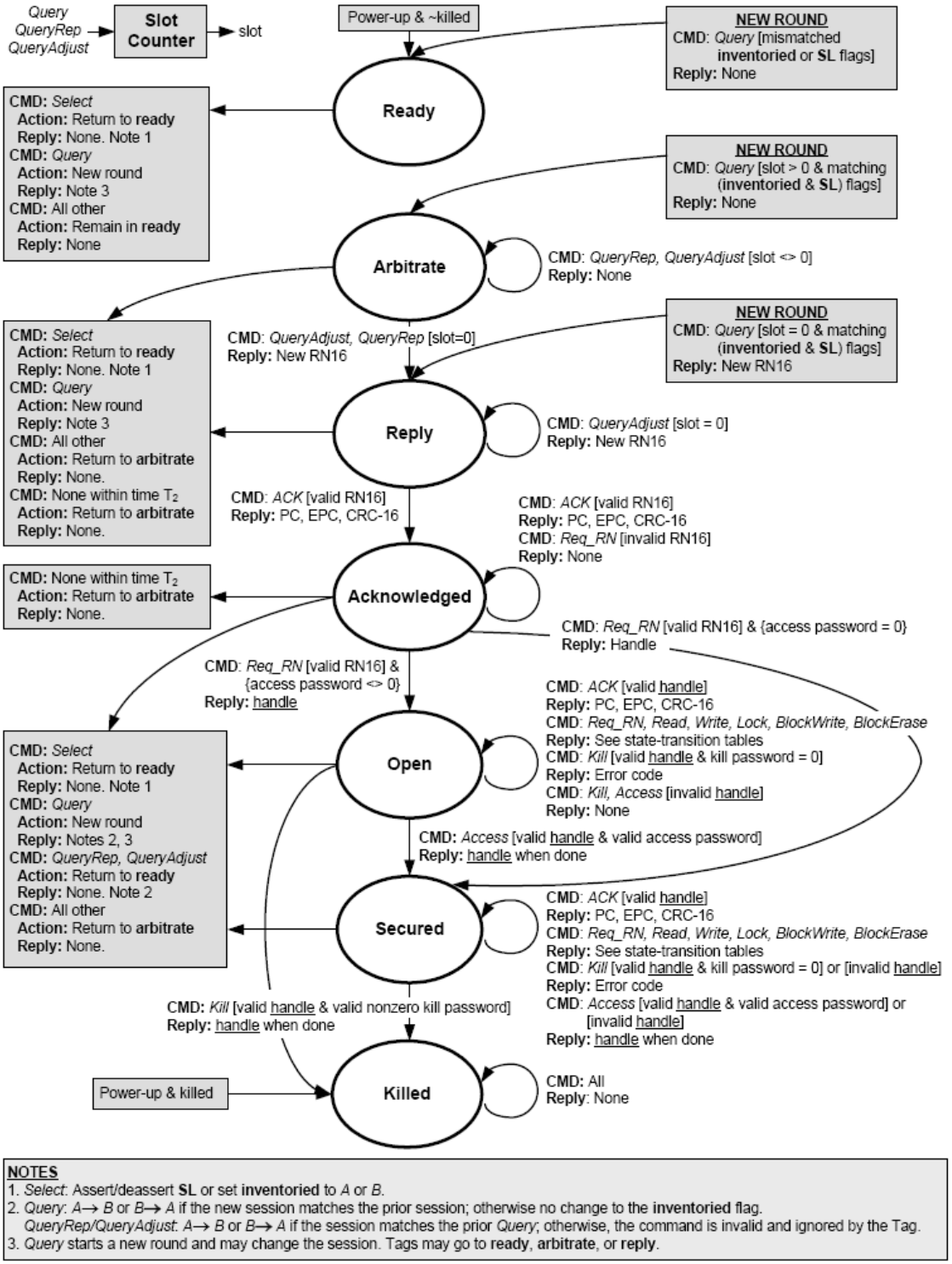

Figure 4 - Tag state diagram (EPCglobal Inc., 2008)

Killed is a hole or a dead-end state. A tag that enters the killed state will stay there forever and is permanently disabled. There is no transition from the killed state to any other state.

Ready, as can be inferred from the title, is a state where the tag is ready, hears the signals from the reader, and either goes through Inventory operations by receiving appropriate Query commands or stays in the Select procedure, depending on the received commands and other status bits of the tag.

Tags go into **Arbitrate**, **Reply** and **Acknowledged** states during Inventory operations.

Inventory operations start with a query command; all the tags which are in the ready state and receive this command will enter the inventory cycle. The Query commands send a number between 0 and 15 (called Q) to the tags, instructing them to generate a random integer in the interval $[0, 2^Q - 1]$ and initialize their slot counter's value to that number. If this random number is 0 (with a probability of $1/2^Q$) for a tag, that tag enters the Reply state and sends an RN16 number to the reader. If the tag receives a correct acknowledgement from the reader, it transits into the Acknowledged state, which means the tag has been singulated and the reader can start to read from and/or write to the tag's memory through Access operations.

In case a tag's slot counter is not zero (with probability of $1 - 1/2^Q$), it enters the Arbitrate state. Upon receiving each Query Rep command a tag in Arbitrate state decrements its slot counter value by one, until it reaches 0 where it transits into the Reply state and backscatters a RN16 number to the reader and follows the same procedure explained above.

There are various ways a tag can enter the Arbitrate state, one of which is by receiving a Query command when its slot counter is non-zero as explained above. The other situations and

conditions in which a tag will enter the Arbitrate state are detailed in Appendix B and Appendix C of the standard (EPCglobal Inc., 2008).

A Tag enters the Secured or Open state after passing through Inventory operations and being uniquely identified by the reader. There are several transition tables (table 1) in the standard, designed for each of the seven states. These tables, which are provided in Appendix B of the EPCglobal UHF Class1 Gen2 protocol, list the potential state transitions and their conditions.

Table 1 - Arbitrate state transition table (EPCglobal Inc., 2008)

Command	Condition	Action	Next State
Query [1,2]	slot=0; matching **inventoried** & **SL** flags	backscatter new RN16	reply
	slot<>0; matching **inventoried** & **SL** flags	–	arbitrate
	otherwise	–	ready
QueryRep	slot=0 after decrementing slot counter	backscatter new RN16	reply
	slot<>0 after decrementing slot counter	–	arbitrate
QueryAdjust [2]	slot=0	backscatter new RN16	reply
	slot<>0	–	arbitrate
ACK	all	–	arbitrate
NAK	all	–	arbitrate
Req_RN	all	–	arbitrate
Select	all	assert or deassert **SL**, or set **inventoried** to *A* or *B*	ready
Read	all	–	arbitrate
Write	all	–	arbitrate
Kill	all	–	arbitrate
Lock	all	–	arbitrate
Access	all	–	arbitrate
Erase	all	–	arbitrate
BlockWrite	all	–	arbitrate
BlockErase	all	–	arbitrate
BlockPermalock	all	–	arbitrate
Invalid [3]	all	–	arbitrate

1: *Query* starts a new round and may change the session.

2: *Query* and *QueryAdjust* instruct a Tag to load a new random value into its slot counter.

3: "Invalid" shall mean an erroneous command, an unsupported command, a command with invalid parameters, a command with a CRC error, a command (other than a *Query*) with a session parameter not matching that of the inventory round currently in progress, or any other command either not recognized or not executable by the Tag.

Appendix C of the same standard describes the appropriate reaction of the tags for each specific command (Table 2). It also shows what the condition for that reaction would be based on parameters associated with the tag and the command.

Table 2 - Query Command response table (EPCglobal Inc., 2008)

Starting State	Condition	Response	Next State
ready, arbitrate, reply	slot=0; matching **inventoried** & **SL** flags	backscatter new RN16	reply
	slot<>0; matching **inventoried** & **SL** flags	–	arbitrate
	otherwise	–	ready
acknowledged, open, secured	slot=0; matching **inventoried**[2] & **SL** flags	backscatter new RN16; transition **inventoried**[2] from *A*→*B* or *B*→*A* if and only if new session matches prior session	reply
	slot<>0; matching **inventoried**[2] & **SL** flags	transition **inventoried**[2] from *A*→*B* or *B*→*A* if and only if new session matches prior session	arbitrate
	otherwise	transition **inventoried** from *A*→*B* or *B*→*A* if and only if new session matches prior session	ready
killed	all	–	killed

1: *Query* (in any state other than **killed**) starts a new round and may change the session; *Query* also instructs a Tag to load a new random value into its slot counter.

2: As described in 6.3.2.8, a Tag transitions its **inventoried** flag prior to evaluating the condition.

As demonstrated, the state transition information is all contained in the standard; however, different transitions and responses are detailed in various sections, and thus, are not gathered in one reference area. As such, building a complete picture of the state transition machine requires extracting these relationships from the complicated, scattered and occasionally ambiguous text.

In order to meet the expectations and requirement of the standard for many different applications a complete transition table is devised in this thesis which considers all of the situations that can cause transitions between states. The basic and most important idea in this work is the Markovian behavior of the transitions, which will be discussed now.

3.1 MARKOVIAN MODEL FOR TAG TRANSTIONS

For the purpose of design and development of a state transition machine and also standard conformity testing, a thorough, complete and clear state transition diagram, understandable by both humans and machines, is needed. The basic objective is putting a tag in every possible situation, by applying different inputs (expected or unexpected) from the reader, observing its reaction and comparing it to the standard's requirements. In order to organize and capture all of the standard's requirements for the transitions, a set of nodes (or rows) and also a set of inputs (or column) are defined.

Each node (row) is basically representative of a unique situation (or state) in which a tag can be, while each column is representative of an input (a command with a specific set of parameters) that a tag may receive. Thus, a tag's transition between states is modeled based on a **Markovian process,** where the current state of the system essentially contains all information of the past behavior that is pertinent to predicting future behavior (Puterman, 2005). Then, the future state of a tag can be uniquely identified by knowing the current state of the tag and the command received from the reader. In other words, for each element of the table (say row *i* and column *j*) the next node should be deterministic (based on the standard).

Hence, the transition table will be the backbone of the state transition machine which specifies what the different possible states are and what commands the reader may send that could change the state of the tag. It is also possible to predict what state a tag must transition to when it is in a specific state and receives a specific command.

In order to build this machine, all possible states through which a tag transits in a tag singulating cycle must be identified. In addition, all the commands that may change the state of a tag must be recognized in order to complete the transition table.

A transition table would look like Table 3 (Figure 5) shown below. Each element *i,j* represents the future state of the tag when the current state is *i* and it receives input *j*. In the element (1,1) of the table (row one, column one), the entry "Row2" means that if a tag in state 1 receives input 1, it shall transition to state 2. After transitioning to Row2, if it receives the command in Col3, it shall transition to Row3 and so on. It is important to note that if the tag is in Row3 and receives the command in Col3, no transition will occur and the tag will stay in state 3. In this example, this case is similar to Row2 Col2 where the tag stays in Row2 and doesn't transit.

Table 3 - A sample transition table

Command / Current State	Col1	Col 2	Col 3
Row1	Row 2	Row 3	Row 1
Row2	Row 1	Row 2	Row 3
Row3	Row 3	Row 2	-

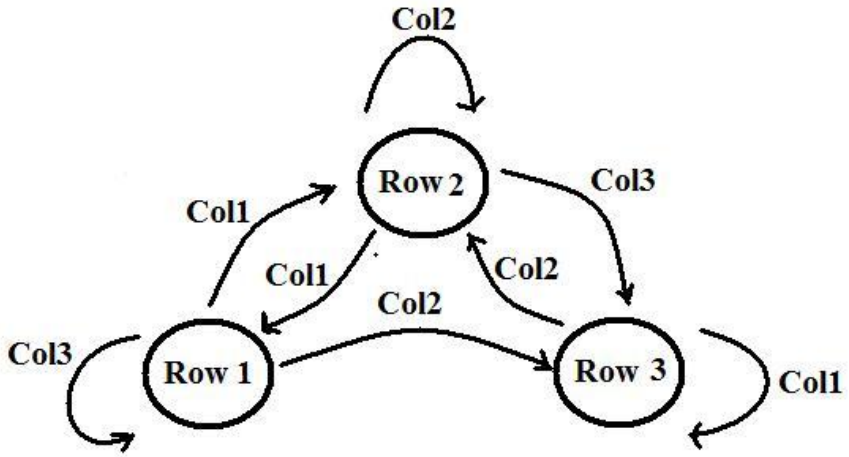

Figure 5 - State transition diagram

Now that the structure of the table is clear, the rows and columns will be discussed. As can be extracted from the standard, if the current situation of a tag (any of the seven states, the status of its flags, slot counter, and several other parameters), and also the received command and its associated parameters are known, the next situation of the tag will be determined. In other words, it is clear that the future state of the tag is solely determined by the current state and the received commands and, given that knowledge, is independent of the previous states. Preserving this independence from the past is important for defining the rows and also the columns.

Each row of the table represents a possible current situation of a tag; e.g., a tag in the Reply state, with its Selected flag set to SL, session flag set to S1, etc. Each column represents a possible input command received from the reader; e.g., a Query command, with Session set to S1 and Sel parameter set to SL.

The set of rows is defined such that each row is unique. The parameters included in each row are the minimum set of parameters that contain the important information from the tag's

history which is critical to the transition from one row to another. Therefore, each row corresponds not only to a state, as defined in the standard, but to a state and a set of values for parameters that affect the transition. Similarly, each column is also defined in a way to include the minimum parameters associated with each command that are critical for the tag's transition from one row to another.

In the following sections more details about the rows and columns are provided.

3.2 ROWS (NODES)

A Class 1 Gen2 tag is basically a small chip which consists of several banks of memory. Whenever a reader sends a command some changes may happen in the tag's status bits and the tag may backscatter a response to the reader. The parameters that play roles in this transition vary from situation to situation. For example, the following table extracted from the EPCglobal standard shows a command response:

Table 4 - Req_RN command-response table (EPCglobal Inc., 2008)

Starting State	Condition	Response	Next State
ready	all	–	ready
arbitrate, reply	all	–	arbitrate
acknowledged	valid RN16 & access password<>0	backscatter handle	open
	valid RN16 & access password=0	backscatter handle	secured
	invalid RN16	–	acknowledged
open	valid handle	backscatter new RN16	open
	invalid handle	–	open
secured	valid handle	backscatter new RN16	secured
	invalid handle	–	secured
killed	all	–	killed

This table corresponds to the tag's responses when it receives a Req_RN command (the Req_RN command is one of the Inventory commands). The first row shows that if the tag is in the Ready state and receives a Req_RN command, no matter what, it should stay in the Ready state and nothing should happen. If the tag is currently in the Acknowledged state then the tag should verify the 16-bit random number (RN 16) sent by the reader and if it is valid (and the tag access password is nonzero), the tag should backscatter a handle and change its state to the Open

state. If the access password is zero, then the tag should change its state to Secured while backscattering the handle. In case the RN 16 sent by the reader is not valid then the tag should ignore the command completely.

Therefore, as shown in this example, there are some parameters associated with a tag that affect its reaction. Considering all different possible scenarios, a minimum list of parameters related to a tag that is required (at some point, not always) for determining a tag's reaction is now prepared.

These parameters include

- The state of the tag,
- The values of the select flag (SL or ~SL),
- Inventoried flag (A or B),
- The session in which the tag is operating (S0, S1, S2 or S3)
- The information showing whether or not the kill and access passwords are zero.

Sessions are important when a tag is participating in an inventory cycle or sometimes in select operations, but do not play any role in access operations. However, in order to build a Markovian transition table, nodes must be defined in a way that holds all the information needed at some point in the table.

For the cases where a parameter does not play any role in the transitions, for all the rows that have the same value for other parameters but different values for this parameter, the reaction to the same input would be exactly the same. This may seem like a wasteful addition to the set of rows in the table, but it is the only way that all standard requirements for transitions can be mapped into one single transition table. On the other hand, it is important to keep the table as

small as possible by eliminating the parameters that are not affecting the transitions or whose effects are considered by other parameters.

The columns of Table 5 show the parameters considered in each row and their possible values.

Table 5 - Node parameters

state	sl_flag	inv_flag	session	kill_pass	access_pass
Ready	SL	A	S0	0	0
Reply	~SL	B	S1	~0	~0
Arbitrate			S2		
Acknowledged			S3		
Open					
Secured					
Killed					

The set of rows in the transition table is basically all the possible combinations of values of these 6 parameters, except for the Killed state.

Table 6 shows one of the nodes (rows) of the table. This node shows that the tag is in the ready state, its select flag is set to "SL" or it is selected, its inventoried flag is set to A, it is operating in session S0, its kill password is zero and its access password is also zero. For further information please refer to EPCglobal Inc., 2008, Section 6.2.

Table 6 - An example of one row (node)

	state	sl_flag	inv_flag	session	kill_pass	access_pass
1	'Ready'	'SL'	'A'	'S0'	'0'	'0'

3.3 COLUMNS (INPUTS)

The process of determining the set of columns is slightly more complicated. As mentioned before, all of the parameters that are important in the tag transitions and reactions need to be considered. On the other hand, the set of parameters should be kept as small as possible to avoid having a large and unmanageable table.

As mentioned briefly in the previous section, the mandatory and optional commands sent by the reader and executed by the tag are specified in the standard. There are 14 main different commands referenced in the EPCglobal UHF Class1 Gen2 standard. Two of these commands, namely the BlockWrite and BlockErase commands, are optional and the remainder are mandatory; all of these are covered in the transition table. For each command a different set of parameters affects the tag transition. All of the parameters in a command that may change the tag's state, selected flag, inventoried flag and session and may affect the tag's response signal will be considered in the columns.

In Appendix B of the standard, several tables describing different transitions from a specific state, for every possible command received by a tag, are shown. One of these tables (pertaining to the Acknowledged state) is shown below (Table 7). As shown in the table, when a tag is in the Acknowledged state and receives a Query command (first row of the table), if the tag's slot counter is zero and the inventoried flag and the select flag of the command and tag match then the tag should transition to the Reply state. It should also backscatter a new random variable and, only in the case that the command's session and the tag's session match, it inverts the inventoried flag from A to B or vice versa. This shows that the command's session, select flag and inventory flag are important and should be considered in the transition table. The tag's inventoried and SL flags as well as Session are being tracked, and therefore all these conditions

are covered in the table. The only thing left is the slot counter which will be discussed in detail later.

Table 7 - Acknowledged state-transition table (EPCglobal Inc., 2008)

Command	Condition	Action	Next State
Query[1]	slot=0; matching **inventoried**[2] & **SL** flags	backscatter new RN16; transition **inventoried**[2] from *A*→*B* or *B*→*A* if and only if new session matches prior session	reply
	slot<>0; matching **inventoried**[2] & **SL** flags	transition **inventoried**[2] from *A*→*B* or *B*→*A* if and only if new session matches prior session	arbitrate
	otherwise	transition **inventoried** from *A*→*B* or *B*→*A* if and only if new session matches prior session	ready
QueryRep	all	transition **inventoried** from *A*→*B* or *B*→*A*	ready
QueryAdjust	all	transition **inventoried** from *A*→*B* or *B*→*A*	ready
ACK	valid RN16	see Table 6.14	acknowledged
	invalid RN16	–	arbitrate
NAK	all	–	arbitrate
Req_RN	valid RN16 & access password<>0	backscatter handle	open
	valid RN16 & access password=0	backscatter handle	secured
	invalid RN16	–	acknowledged
Select	all	assert or deassert **SL**, or set **inventoried** to *A* or *B*	ready
Read	all	–	arbitrate
Write	all	–	arbitrate
Kill	all	–	arbitrate
Lock	all	–	arbitrate
Access	all	–	arbitrate
BlockWrite	all	–	arbitrate
BlockErase	all	–	arbitrate
Block-Permalock	all	–	arbitrate
T_2 timeout	See Figure 6.16 and Table 6.13	–	arbitrate
Invalid[3]	all	–	acknowledged

Let us consider another situation. In the acknowledged state, if a tag with a zero access password receives a Req_RN command, and the random number (RN16) sent by the reader with

this command is valid, the future node (row) is determined. In this case none of the parameters in the tag, with the exception of its state, will change. Therefore, if the tag is in row *k* (with specifications shown in Table 8) and receives Req_RN with a valid RN16, it will transit into row *j* as shown in Table 9. Conversely, if the Req_RN command arrives with an invalid RN-16, nothing will change and the tag will stay in the same state (Table 8). If the RN_16 is valid but the tag access password is not zero (as shown in Table 10), the tag will transition to the open state and send a handle to the reader (Table 11).

Table 8 – A sample node in the acknowledged state

#	State	Session	Select Flag	Inventoried flag	Kill password	Access Password
k	Acknowledged	S0	SL	A	0	0

Table 9 – A sample node in the secured state

#	State	Session	Select Flag	Inventoried flag	Kill password	Access Password
j	Secured	S0	SL	A	0	0

Table 10 – A sample node in the acknowledged state with access password not zero

#	State	Session	Select Flag	Inventoried flag	Kill password	Access Password
	Acknowledged	S0	SL	A	0	~0

Table 11 – A sample node in the open state

#	State	Session	Select Flag	Inventoried flag	Kill password	Access Password
	Open	S0	SL	A	0	~0

For the Req_RN command, the validity of the RN 16 determines all possible combinations of parameters. Thus, the Req_RN command comes with two columns of valid and invalid RN numbers.

Table 12 - Req_RN command

Req_RN	
Invalid RN 16	Valid RN 16

As a result of the complexities detailed above, the columns of the transition table do not end up being as straightforward as the rows. Table 13 is a complete list of the different commands and their associated parameters.

Table 13 – List of commands and their parameters

1	2	3	4	5	6	7
Select	**Query**	**QuerryRep**	**QurryAdjust**	**Ack**	**NAK**	**Req_RN**
Memory Matching	Session	Session	Session	Handle		Handle
Target	Sel					
Action	Session					
	Target (A/B)					
8	9	10	11	12	13	14
Read	**Write**	**Kill**	**Lock**	**Access**	**BlockWrite**	**BlockErase**
Handle	Handle	handle	hanlde	Handle	Handle	Handle
Memory Access	Memory Access	Kill password	Paylock	Access Password	Memory Access	Memory Access

For each command, any combination of parameters is shown in a new column of the table. For example, the Select Command has 5 different values for Target, 8 different values for Actions and 2 for memory matching; therefore there will be 80 (5 x 8 x 2)columns assigned to the 80 different flavors of the Select command.

3.4 TRANSITION TABLE STRUCTURE

Table 14 - State Transition table

141 commands

						Commands	No input slot counter = 0	No input slot counter <> 0	Query All A S0	Query All A S1	Query All A S2	Query All A S3
	+1											
449	7	2	2	4	2	2						
Nodes:	state	SL flag	invertory flag	session	kill pass	access pass						
1	Ready	SL	A	S0	0	0						
2	Ready	SL	A	S0	0	1						
3	Ready	SL	A	S0	1	0						
4	Ready	SL	A	S0	1	1						
5	Ready	SL	A	S1	0	0						
6	Ready	SL	A	S1	0	1						
7	Ready	SL	A	S1	1	0						
8	Ready	SL	A	S1	1	1						
9	Ready	SL	A	S2	0	0						
10	Ready	SL	A	S2	0	1						
11	Ready	SL	A	S2	1	0						
12	Ready	SL	A	S2	1	1						
13	Ready	SL	A	S3	0	0						
14	Ready	SL	A	S3	0	1						
15	Ready	SL	A	S3	1	0						
16	Ready	SL	A	S3	1	1						
17	Ready	SL	B	S0	0	0						
18	Ready	SL	B	S0	0	1						
19	Ready	SL	B	S0	1	0						
20	Ready	SL	B	S0	1	1						

449 states

Table 14 demonstrates a section of the transition table. The rows represent different situations of a tag. Each node represents a unique combination of the state and status bits of a tag. Each tag can be in 7 different states (Ready, Reply, Arbitrate, Acknowledged, Open, Secured, Killed). In

each of these states it reacts differently to a command based on its status bits (Session, Select flag, Inventory flag, Access password, Kill password), except when it is in the killed state. A tag in the killed state stays there no matter what command it receives or what its status bits are. Therefore, enumerating all of the possible unique combinations yields 449 unique rows. Including the intermediate states, the standard defines 8 states. 7 of these 8 states share the same parameter set. The exception is the Kill state. The Kill state is a terminating point; no matter what values the other parameters take the tag will never leave it, therefore it can be represented by a single row. For the 7 states, each has a select flag with 2 cases, an inventory flag with 2 cases, a sessions flag with 4 cases, a kill and an access password each with 2 cases, totaling 2x2x4x2x2 = 64 rows for each of the 7 states, plus one row for the Kill state, which adds up to 64x7+1 = 449 unique rows.

The columns represent different inputs (commands) a tag can receive from the reader. There are 14 different commands and each of them has a different set of parameters that affect the transition from one node to another. Enumerating these commands with their parameters yields 139 columns.

On the table there are series of intermediate states marked with a "*" and two columns (the first two columns) "No Input, Slot counter = 0" and "No Input, Slot counter <>0" that do not correspond to actual states or commands. These were added in order to model the behavior of the tag when receiving a Query or Query Rep command. In reality, when a tag receives a Query command with a Q parameter it sets its slot counter value to a random number between 0 and 2^{Q-1}. When a tag receives a Query Rep command it decrements its slot counter by one. In both of these situations, if the slot counter equals zero after the operation, the tag transits into the Reply state and if it is nonzero it transits into the Arbitrate state. In order to model these transitions in

the table, it is necessary to consider the value of the slot counter as one of the rows' parameters; clearly this is not desirable as it would increase the number of rows by a factor of 2^Q. As a less costly solution some intermediate states (marked with "*") and two columns ("No. Input Slot counter = 0" and "No. Input Slot counter <>0") have been added to the table. This is based on the assumption that the state transition machine can check the value of the slot counter and see if it is zero or nonzero. In this setting, whenever a tag receives a Query or Query Rep command, it transits into the corresponding intermediate state which has the same set of parameters. In this state when the receiving command is "No Input Slot counter = 0" it transitions to the Reply state and if it is "No Input Slot counter <>0" it transitions to the Arbitrate state. Remember that the choice between the last two commands does not actually come from the reader at all but is dependent on the value of the slot counter.

The entries of the table (the vacant, purple slots in Table 14) are the numbers of the future states (rows). Later on a snapshot of the completed table will be shown.

3.5 ASSUMPTIONS

Several assumptions are made for the tag transitions, including:

If a tag is in session S0 and receives a Query command with parameter S1, it starts the new inventory round in S1. If the tag is in session S1 and receives a Query command with session S1 it starts the new round with S1 but in this case it changes its inventory flag from A to B or vice versa.

In the case where a tag is in the Reply state and receives an ACK command with an invalid handle the standard says it transitions into the arbitrate state but it doesn't mention what would be the new value of the slot counter. In this work, when a tag in the Reply state receives an ACK command with an invalid handle, it transitions into an intermediate state (specified with a star (*)). When there is no input, its slot counter will be checked; if it is nonzero, it will go to Arbitrate and if it is zero it goes to the Reply state.

If a tag is in the Arbitrate or Reply states and receives a Query command, the new session would be what the Query command says and the tag will not change its inventory flag. If the tag is in the Acknowledge, Secured or Open state, and receives a Query command the new session is what the Query says and if it is the same as the current session, the tag changes its inventory flag (from A to B or vice versa):

- A tag in the Reply state with inventory flag set to A and session S0 receives a Query command with session S0; then the new state would be an intermediate state with flag A and session S0
- A tag in the Reply state with inventory flag set to A and session S1 receives a Query command with session S0; then the new state would be an intermediate state with flag A and session S0

- A tag in the Reply state with inventory flag set to A and session S1 receives a Query command with session S1; then the new state would be an intermediate state with flag A and session S1
- A tag in the Acknowledged state with inventory flag set to A and session S0 receives a Query command with session S0; then the new state would be an intermediate state with flag B and session S0
- A tag in the Acknowledged state with inventory flag set to A and session S1 receives a Query command with session S0; then the new state would be an intermediate state with flag A and session S0
- A tag in the Acknowledged state with inventory flag set to A and session S0 receives a Query command with session S1; then the new state would be an intermediate state with flag A and session S1
- A tag in the Acknowledged state with inventory flag set to A and session S1 receives a Query command with session S1; then the new state would be an intermediate state with flag B and session S1

In Appendix B4 of the Standard (EPCglobal Inc., 2008) it is stated that if the tag is in the Acknowledge state and receives a Query Adjust command it should toggle its inventory flag and transition into the Ready state. However, on page 50 of the same standard it is stated that "If a tag receives a Query Adjust whose session number is different from the session number in the Query that initiated the round it shall ignore the command." This is assumed to override the behavior described in Appendix B4. In other words only if the session number of the Query Adjust command matches the current session of the tag will the tag react and transition into the Ready state.

4.0 RESULTS

4.1 TRANSITION TABLE

Below is a snapshot of the filled transition table. As explained before, each row is also called a node and considers a unique situation of the tag's status bits. Each column corresponds to inputs from the state transition machine which is basically playing the role of a reader and exposes the tag to different scenarios that may occur for a tag during a singulation procedure. The numbers in the entries with a pink background are the future states that the tag will transition to in each scenario.

							1	2	3	4	5	6
						type	No_Input	No_Input	'Select'	'Select'	'Select'	'Select'
						session			''	''	''	''
						sl_flag			''	''	''	''
						inv_flag			''	''	''	''
						handle			''	''	''	''
					Commands	memory_access			''	''	''	''
						lock_payload			''	''	''	''
						access_pass			''	''	''	''
						kill_pass			''	''	''	''
						slot_counter	0	1	''	''	''	''
						match			'Match'	'Match'	'Match'	'Match'
						target			'S0'	'S0'	'S0'	'S0'
	+1					action			'000'	'001'	'010'	'011'
449	7	2	2	4	2	2						
	state	sl_flag	inv_flag	session	kill_pass	access_pass						
1	'Ready'	'SL'	'A'	'S0'	'0'	'0'	1	1	1	1	1	17
2	'Ready'	'SL'	'A'	'S0'	'0'	'1'	2	2	2	2	2	18
3	'Ready'	'SL'	'A'	'S0'	'1'	'0'	3	3	3	3	3	19
4	'Ready'	'SL'	'A'	'S0'	'1'	'1'	4	4	4	4	4	20
5	'Ready'	'SL'	'A'	'S1'	'0'	'0'	5	5	5	5	5	5
6	'Ready'	'SL'	'A'	'S1'	'0'	'1'	6	6	6	6	6	6
7	'Ready'	'SL'	'A'	'S1'	'1'	'0'	7	7	7	7	7	7
8	'Ready'	'SL'	'A'	'S1'	'1'	'1'	8	8	8	8	8	8
9	'Ready'	'SL'	'A'	'S2'	'0'	'0'	9	9	9	9	9	9
10	'Ready'	'SL'	'A'	'S2'	'0'	'1'	10	10	10	10	10	10
11	'Ready'	'SL'	'A'	'S2'	'1'	'0'	11	11	11	11	11	11
12	'Ready'	'SL'	'A'	'S2'	'1'	'1'	12	12	12	12	12	12
13	'Ready'	'SL'	'A'	'S3'	'0'	'0'	13	13	13	13	13	13
14	'Ready'	'SL'	'A'	'S3'	'0'	'1'	14	14	14	14	14	14
15	'Ready'	'SL'	'A'	'S3'	'1'	'0'	15	15	15	15	15	15
16	'Ready'	'SL'	'A'	'S3'	'1'	'1'	16	16	16	16	16	16
17	'Ready'	'SL'	'B'	'S0'	'0'	'0'	17	17	1	1	17	1

Figure 6 - Filled state transition table

4.2 AUTOMATIC TABLE GENERATION (USING MATLAB)

Extracting the logic from the standard for all of the possible combinations of states and inputs is a very intricate and time consuming process; as explained in the previous sections there is a good deal of ambiguity in the current standard that might be interpreted differently by different readers. Therefore several reasonable assumptions have been made; some of these are listed in Section 3.5.

There are several reasons for automating the process of filling the table:

1. Scale of the table – time consuming
2. Human errors when dealing with a large number of states
3. Documenting the extracted logic
4. Validation purposes
5. Modification purposes
6. Flexibility with respect to possible changes in the standard
7. Repeatability

Brief description of the simulation procedure:

All of the code is written in MATLAB. There is a main file which accepts the current state and received command as inputs, and returns the future state as output. There is another function that receives a state, locates that state in the transition table and returns the associated row number.

Another file consists mainly of two loops that go through the entire set of rows and also the set of commands, and for each combination, uses the two other functions described above to generate the appropriate future state for that specific combination.

Using these three main functions and a few other functions the table described in the previous section was developed. In case any changes should happen to any command or row, or in case new commands or states need to be added, one only needs to update the list of columns or rows that are subject to change and rerun the program to generate the updated table.

The main function, essentially the brain of the MATLAB simulation, is the transition function. It basically encompasses the rules behind the transition of a tag through different states

based on the EPCglobal RFID Protocols for UHF Class 1 Gen 2 passive tags. Inputting a specific state and a specific command, this function will give the future state the tags must transit into as its output. The logic behind state transitions, which are extracted from EPCglobal standard version 1.0.9, is saved in this function for all of the possible cases. All of the possible combinations of the states and commands are enumerated via other functions of the simulation package, and appropriate changes in the status bits of the tag are implemented and the new state that the tags will transit into are identified.

Given a list of possible states and possible commands, this simulation package generates the transition table, which consists of states as rows and commands as columns and the element (i,j) of the table contains the future state the tags will transit into when it's currently in state i and receives command j.

Since all of the logic is incorporated into the transition function, whenever a modification of the standard happens, or some rules change in the standard, the transition can easily be modified and updated by only modifying the transition function. Modification of the transition function is easy since it basically comprises all of the possible combinations of rows and columns. For example, if the standard were to add some commands, or change some parameters of the commands, extra lines should be added to the transition function to take care of the extra columns that are added. Rules for the transition from all of the states where only these commands are received must be inserted, and the results of these new combinations must be added to the table.

4.3 GRAPHICAL USER INTERFACE (GUI WITH MATLAB)

A graphical user interface is also created for use with this simulation package. This is independent from the state transition machine, and is basically a user-friendly interface for verification purposes. This interface has access to other functions, including the transition function. There are basically three main sections in the interface (see Figure 7). In the top panel to the right, the user enters the type of the command and other parameters specific to that type of command. In the middle panel, the user enters the parameters of each state and each unique combination of these parameters corresponds to one row of the transition table or equivalently, to one state of the state space of our Markovian transition model. In the third section, which is the bottom panel, the user can verify the receiving command and the current state that was entered in the previous sections. If they are different from what was expected, the parameters can be reentered. Once the user verifies the state and the commands the user can click on "Do the transition" and see the future state in front of the "Next State" field in the same section. In some cases where the slot counter values of the tags are chosen randomly by the RN16 generator, the user will be asked to choose either zero or non-zero values (represented by the number 1) for the slot counter. The reasoning behind this is related to the approach taken to resolve the problem associated with the probabilistic nature of how the slot counter value is selected. As stated before, other than the command received from the reader, there are two other inputs (or columns in the context of the transition table) that need to be incorporated into the table in order to maintain the Markovian properties of these transitions and also to avoid the table becoming excessively large. These two columns are basically the slot counter being either zero or non-zero.

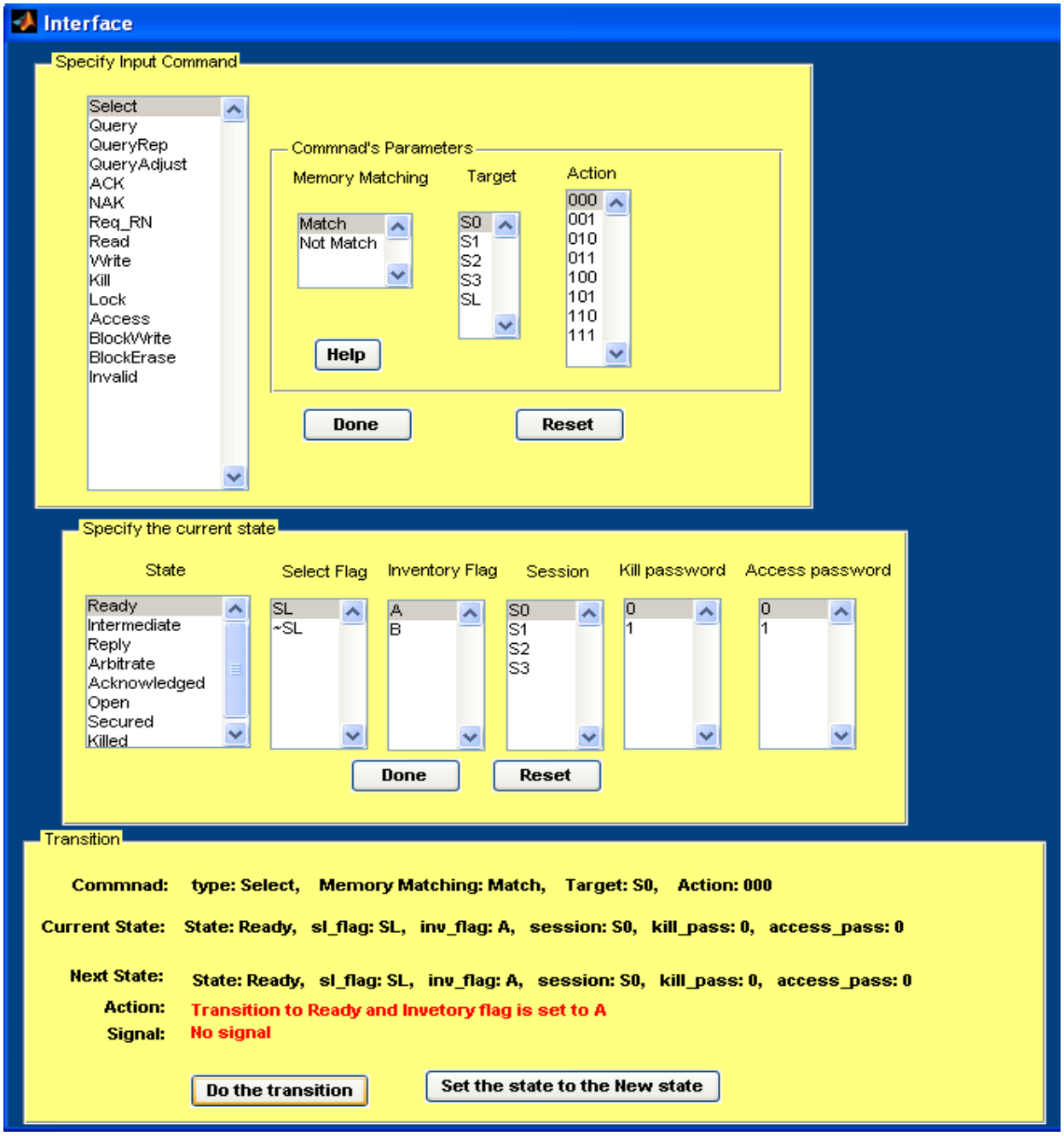

Figure 7 - Snapshot of the GUI

4.4 CONCLUSION AND FUTURE WORK

In summary, a simple simulator for the behavior of a tag has been built and used to obtain a complete transition table representing the tag as a state machine. Each row of this table represents a node[2] of the state transition diagram while each column represents a commands received by a tag which would trigger one of the possible transitions. The entry in each cell of the table is the destination node of the corresponding transition. Using this table one can determine the correct behavior of a tag in response to any sequence of commands, along with the final state.

As stated in Section 2.0 , the overall goal is to create a standard conformity test for tags, and more specifically, to design a sequence of commands which can be used by a reader to test the response of a tag. The first, and perhaps the most important, step in this process is the creation of the transition table herein as a reference for the machines involved (for example, the reader can compare a tag's response to the correct response from the table). A GUI was created as a user-friendly visual aid for the designer of such a system.

The next step would be to convert the table to a machine readable format such as the finite transition table format in Figure 8. Then an optimal algorithm must be designed for the reader that is being used to test tag conformity. This algorithm will transmit to the tag being tested a series of commands designed to start the tag from a specific state and cycle it through a series of states, all the while monitoring its behavior for conformity. It will do so using the fewest number of commands possible. Whether or not this cycle contains all possible states (and if not, what states are omitted) will depend on the particular application. For example, an

[2] An individual node represents a combination of states as defined in the standard, with some other internal tag parameters.

algorithm might be designed to test the most common states, while another might focus on getting from state A to state B in the fewest number of steps.

State	1	2	A	B	C	D	E	F	G	H	I	J	K
R1	**R1**	**R1**	I1	I2									
R1*	R1	R1			**R1***	**R1***	**R1***	**R1***	**R1***				
R2	**R2**	**R2**	I1	I2									
R2*	R2	R2			**R2***	**R2***	**R2***			**R2***			
R3	**R3**	**R3**			I3	I4							
R3*	R3	R3	**R3***	**R3***			**R3***	**R3***	**R3***				
R4	**R4**	**R4**			I3	I4							
R4*	R4	R4	**R4***	**R4***			**R4***			**R4***			
I1	Rep1	A1	**I1**					**I1**	**I1**				**I1**
I2	Rep2	A2		**I2**						**I2**			**I2**
I3	Rep3	A3			**I3**			**I3**	**I3**				**I3**
I4	Rep4	A4				**I4**				**I4**			**I4**
Rep1	**Rep1**		I1	I2	R1*	R1*	R1*	A1*	I1		C1*	A1*	I1
Rep2	**Rep2**		I1	I2	R2*	R2*	R2*			I2	C2*	A2*	I2
Rep3	**Rep3**		R3*	R3*	I3	I4	R3*	A3*	I3		C3*	A3*	I3
Rep4	**Rep4**		R4*	R4*	I3	I4	R4*			I4	C4*	A4*	I4
A1		**A1**	I1	I2	R1*	R1*	R1*	I1	I1				A1*
A1*								**A1***				**A1***	**A1***
A2		**A2**	I1	I2	R2*	R2*	R2*			I2			A2*
A2*												**A2***	**A2***
A3		**A3**	R3*	R3*	I3	I4	R3*	I3	I3				A3*
A3*								**A3***				**A3***	**A3***
A4		**A4**	R4*	R4*	I3	I4	R4*			I4			A4*
A4*												**A4***	**A4***
C1	**C1**	**C1**	R3*	I2	I3	R2*	R3*	R3*	R3*		C1*	A1*	A1*
C1*	C1	C1									**C1***		
C2	**C2**	**C2**	I1	R4*	R1*	I4	R1*			R4*	C2*	A2*	A2*
C2*	C2	C2									**C2***		
C3	**C3**	**C3**	I1	R4*	R1*	I4	R1*	R1*	R1*		C3*	A3*	A3*
C3*	C3	C3									**C3***		
C4	**C4**	**C4**	R3*	I2	I3	R2*	R3*			R2*	C4*	A4*	A4*
C4*	C4	C4									**C4***		

Figure 8 - Finite transition table

Of course, another future task is maintaining the accuracy of this table. The code - and as a result - the structure of the table are directly dependent on many relatively small details from the standard. In order to keep the code and table relevant, any updates, clarifications or modifications of the standard must be reflected in the code and a new table must be generated. Therefore, the code is structured so that detailed, individual modifications are simple.

APPENDIX A

MATLAB CODE OF THE TABLE GENERATION FUNCTION

The following code uses the transition function form Appendix B to generate the table.

```
%load states
global states

for i=1:449
    i
    for j=1:141
        [future_state , signal, action] = transition(states(i) , cmds(j));
        future_states(i,j) =future_state;
        signals(i,j) = cellstr(signal);
        actions(i,j) = cellstr(action) ;
        index_of_future_state(i,j) = locate_state(future_state);
    end
  save table  future_states  signals  actions index_of_future_state
end
```

APPENDIX B

MATLAB CODE OF THE TRANSITION FUNCTION

This code receives a state and a command as input and returns the signal that the tag backscatters and the future state the tag transitions to.

```
function [future_state , signal, action]= transition(state,cmd)

% state is a structure with these fields:
% state=
%      state:
{'Ready','Intermediate','Reply','arbitrate','Acknowledged','Open','Secured','
Killed', 'invalid'}
%      sl_flag: {'SL','~SL'}
%      inv_flag: {'A','B'}
%      session:  {'S0','S1', 'S2', 'S3'}
%      kill_pass:  {'0', '1'}
%      access_pass:  {'0', '1'}
%     ** slot_counter: {0:2^16}
%     ** Q: {1:15

% cmd is a structure with these fields:
% cmd=
%      type:    {'Select' , 'Query' , 'QueryRep' , 'QueryAdjust' , 'Ack' ,
'NAK' , 'Req_RN' , 'Read', 'Write' , 'Kill' , 'Lock', 'Access' , 'BlockWrite'
, 'BlockErase' , 'No_Input', 'invalid'}
%      session:  {'S0','S1', 'S2', 'S3'}       % active only for 'Query',
'QueryRep', 'QueryAdjust'
%      sl_flag:  {'all' , 'SL' , '~SL' }       % active only for 'Query'
%      inv_flag: {'A','B'}                     % active only for 'Query'
%     ** Q: {1:15}                               % active only for 'Query'
%     ** UpDn: {'Up','Down','Same'}              % active only for
'QueryAdjust'
```

```
%      handle: {'valid', 'invalid'}            % active for 'ACK',  'Req_RN'
, 'Read', 'Write' , 'Kill' , 'Lock', 'Access' , 'BlockWrite' , 'BlockErase'
%      memory_access: {'valid', 'invalid'}     % active for 'Read', 'Write'
, 'Lock', 'BlockWrite' , 'BlockErase'
%      slot_counter: {'0' , '~0'}              % active only for 'No_Input'
%      access_pass:  {'valid', 'invalid'}      % active only for 'Access'
%      kill_pass:  {'valid', 'invalid'}        % active only for 'Kill'
%      lock_payload: {'valid', 'invalid'}      % active only for 'Lock'
%    ** these parameters don't play role in state transitions and they'll be
used only if the "tracking tag's slot counter" option has been enabled to
keep track of the tag's slot counter random values for demonstration
purposes.

future_state = state;
signal='No signal';
action='No transition';

Killed_state =
struct('state','Killed','sl_flag','','inv_flag','','session','','kill_pass','
','access_pass','');

switch state.state

    case { 'Ready' , 'ready'}
      switch cmd.type
          case { 'Select' , 'select'}
              state.state= 'Ready';
              switch cmd.action
                  case '000'
                      if strcmpi(cmd.match,'Match')
                          if strcmpi(cmd.target,future_state.session)
                              future_state.inv_flag = 'A';
                              action = 'Transition to Ready and Invetory flag
is set to A';
                          elseif strcmpi(cmd.target,'SL')
                              future_state.sl_flag = 'SL';
                              action = 'Transition to Ready and Select flag
is asserted';
                          end
                      elseif strcmpi(cmd.match,'Not Match')
                          if strcmpi(cmd.target,state.session)
                              future_state.inv_flag = 'B';
                              action = 'Transition to Ready and Invetory flag
is set to B';
                          elseif strcmpi(cmd.target,'SL')
                              future_state.sl_flag = '~SL';
                              action = 'Transition to Ready and Select flag
is deasserted';
                          end
                      end
                  case '001'
                      if strcmpi(cmd.match,'Match')
                          if strcmpi(cmd.target,state.session)
                              future_state.inv_flag = 'A';
                              action = 'Transition to Ready and Invetory flag
is set to A';
```

```
                        elseif strcmpi(cmd.target,'SL')
                            future_state.sl_flag = 'SL';
                            action = 'Transition to Ready and Select flag
is asserted';
                        end
                    elseif strcmpi(cmd.match,'Not Match')
                        action = 'Transition to Ready and do nothing';
                    end
                case '010'
                    if strcmpi(cmd.match,'Match')
                            action = 'Transition to Ready and do nothing';
                    elseif strcmpi(cmd.match,'Not Match')
                        if strcmpi(cmd.target,state.session)
                            future_state.inv_flag = 'B';
                            action = 'Transition to Ready and Invetory flag
is set to B';
                        elseif strcmpi(cmd.target,'SL')
                            future_state.sl_flag = '~SL';
                            action = 'Transition to Ready and Select flag
is deasserted';
                        end
                    end
                case '011'
                    if strcmpi(cmd.match,'Match')
                        if strcmpi(cmd.target,future_state.session)
                            if strcmpi(future_state.inv_flag,'A')  % invert
inventory flag
                                future_state.inv_flag = 'B';
                            else
                                future_state.inv_flag = 'A';
                            end
                            action = 'Transition to Ready and Invert
Invetory flag';
                        elseif strcmpi(cmd.target,'SL')
%negate sl flag
                            if strcmpi(future_state.sl_flag,'SL')
                                future_state.sl_flag = '~SL';
                            else
                                future_state.sl_flag = 'SL';
                            end
                            action = 'Transition to Ready and negate Select
flag';
                        end
                    elseif strcmpi(cmd.match,'Not Match')
                        action = 'Transition to Ready and do nothing';
                    end
                case '100'
                    if strcmpi(cmd.match,'Match')
                        if strcmpi(cmd.target,future_state.session)
                            future_state.inv_flag = 'B';
                            action = 'Transition to Ready and Invetory flag
is set to B';
                        elseif strcmpi(cmd.target,'SL')
                            future_state.sl_flag = '~SL';
                            action = 'Transition to Ready and Select flag
is desserted';
                        end
```

```
                    elseif strcmpi(cmd.match,'Not Match')
                        if strcmpi(cmd.target,future_state.session)
                            future_state.inv_flag = 'A';
                            action = 'Transition to Ready and Invetory flag
is set to A';
                        elseif strcmpi(cmd.target,'SL')
                            future_state.sl_flag = 'SL';
                            action = 'Transition to Ready and Select flag
is asserted';
                        end
                    end
                case '101'
                    if strcmpi(cmd.match,'Match')
                        if strcmpi(cmd.target,future_state.session)
                            future_state.inv_flag = 'B';
                            action = 'Transition to Ready and Invetory flag
is set to B';
                        elseif strcmpi(cmd.target,'SL')
                            future_state.sl_flag = '~SL';
                            action = 'Transition to Ready and Select flag
is desserted';
                        end
                    elseif strcmpi(cmd.match,'Not Match')
                        action = 'Transition to Ready and do nothing';
                    end
                case '110'
                    if strcmpi(cmd.match,'Match')
                            action = 'Transition to Ready and do nothing';
                    elseif strcmpi(cmd.match,'Not Match')
                        if strcmpi(cmd.target,future_state.session)
                            future_state.inv_flag = 'A';
                            action = 'Transition to Ready and Invetory flag
is set to A';
                        elseif strcmpi(cmd.target,'SL')
                            future_state.sl_flag = 'SL';
                            action = 'Transition to Ready and Select flag
is asserted';
                        end
                    end
                case '111'
                    if strcmpi(cmd.match,'Match')
                            action = 'Transition to Ready and do nothing';
                    elseif strcmpi(cmd.match,'Not Match')
                        if strcmpi(cmd.target,future_state.session)
                            if strcmpi(future_state.inv_flag,'A')  % invert
inventory flag
                                future_state.inv_flag = 'B';
                            else
                                future_state.inv_flag = 'A';
                            end
                            action = 'Transition to Ready and Invert
Invetory flag';
                        elseif strcmpi(cmd.target,'SL')
%negate sl flag
                            if strcmpi(future_state.sl_flag,'SL')
                                future_state.sl_flag = '~SL';
                            else
```

```
                                    future_state.sl_flag = 'SL';
                                end
                                action = 'Transition to Ready and negate Select
flag';
                            end
                        end
                end
            case { 'Query' , 'query'}
                if  strcmpi(future_state.inv_flag,cmd.inv_flag) &&  (
strcmpi(future_state.sl_flag,cmd.sl_flag) || strcmpi(cmd.sl_flag ,'all') )
                    future_state.state='Intermediate';
                    future_state.session = cmd.session;
                    action='Transition is done';
                end
            case { 'No_Input' , 'No_input' , 'no_input'}
        end

    case { 'Intermediate' , 'intermediate'}      % When there is No_Input
based on the value of tag's slotcounter it transits either to 'Reply' or
"Arbitrate'
        switch cmd.type
            case { 'No_Input' , 'No_input' , 'no_input'}
                if cmd.slot_counter == '0'
                  future_state.state='Reply';
                  signal = 'Backscatter new RN16'
                  action='Transition is done' ;
                else
                  future_state.state='Arbitrate';
                  action='Transition is done' ;
                end
        end

    case { 'Reply' , 'reply'}
        switch cmd.type
          case { 'Select' , 'select'}
              future_state.state= 'Ready';
              switch cmd.action
                  case '000'
                      if strcmpi(cmd.match,'Match')
                          if strcmpi(cmd.target,future_state.session)
                              future_state.inv_flag = 'A';
                              action = 'Transition to Ready and Invetory flag
is set to A';
                          elseif strcmpi(cmd.target,'SL')
                              future_state.sl_flag = 'SL';
                              action = 'Transition to Ready and Select flag
is asserted';
                          end
                      elseif strcmpi(cmd.match,'Not Match')
                          if strcmpi(cmd.target,future_state.session)
                              future_state.inv_flag = 'B';
                              action = 'Transition to Ready and Invetory flag
is set to B';
                          elseif strcmpi(cmd.target,'SL')
                              future_state.sl_flag = '~SL';
```

```
                            action = 'Transition to Ready and Select flag
is deasserted';
                        end
                    end
                case '001'
                    if strcmpi(cmd.match,'Match')
                        if strcmpi(cmd.target,future_state.session)
                            future_state.inv_flag = 'A';
                            action = 'Transition to Ready and Invetory flag
is set to A';
                        elseif strcmpi(cmd.target,'SL')
                            future_state.sl_flag = 'SL';
                            action = 'Transition to Ready and Select flag
is asserted';
                        end
                    elseif strcmpi(cmd.match,'Not Match')
                        action = 'Transition to Ready and do nothing';
                    end
                case '010'
                    if strcmpi(cmd.match,'Match')
                            action = 'Transition to Ready and do nothing';
                    elseif strcmpi(cmd.match,'Not Match')
                        if strcmpi(cmd.target,future_state.session)
                            future_state.inv_flag = 'B';
                            action = 'Transition to Ready and Invetory flag
is set to B';
                        elseif strcmpi(cmd.target,'SL')
                            future_state.sl_flag = '~SL';
                            action = 'Transition to Ready and Select flag
is deasserted';
                        end
                    end
                case '011'
                    if strcmpi(cmd.match,'Match')
                        if strcmpi(cmd.target,future_state.session)
                            if strcmpi(future_state.inv_flag,'A')  % invert
inventory flag
                                future_state.inv_flag = 'B';
                            else
                                future_state.inv_flag = 'A';
                            end
                            action = 'Transition to Ready and Invert
Invetory flag';
                        elseif strcmpi(cmd.target,'SL')
%negate sl flag
                            if strcmpi(future_state.sl_flag,'SL')
                                future_state.sl_flag = '~SL';
                            else
                                future_state.sl_flag = 'SL';
                            end
                            action = 'Transition to Ready and negate Select
flag';
                        end
                    elseif strcmpi(cmd.match,'Not Match')
                        action = 'Transition to Ready and do nothing';
                    end
                case '100'
```

```
                    if strcmpi(cmd.match,'Match')
                        if strcmpi(cmd.target,future_state.session)
                            future_state.inv_flag = 'B';
                            action = 'Transition to Ready and Invetory flag
is set to B';
                        elseif strcmpi(cmd.target,'SL')
                            future_state.sl_flag = '~SL';
                            action = 'Transition to Ready and Select flag
is desserted';
                        end
                    elseif strcmpi(cmd.match,'Not Match')
                        if strcmpi(cmd.target,future_state.session)
                            future_state.inv_flag = 'A';
                            action = 'Transition to Ready and Invetory flag
is set to A';
                        elseif strcmpi(cmd.target,'SL')
                            future_state.sl_flag = 'SL';
                            action = 'Transition to Ready and Select flag
is asserted';
                        end
                    end
                case '101'
                    if strcmpi(cmd.match,'Match')
                        if strcmpi(cmd.target,future_state.session)
                            future_state.inv_flag = 'B';
                            action = 'Transition to Ready and Invetory flag
is set to B';
                        elseif strcmpi(cmd.target,'SL')
                            future_state.sl_flag = '~SL';
                            action = 'Transition to Ready and Select flag
is desserted';
                        end
                    elseif strcmpi(cmd.match,'Not Match')
                        action = 'Transition to Ready and do nothing';
                    end
                case '110'
                    if strcmpi(cmd.match,'Match')
                            action = 'Transition to Ready and do nothing';
                    elseif strcmpi(cmd.match,'Not Match')
                        if strcmpi(cmd.target,future_state.session)
                            future_state.inv_flag = 'A';
                            action = 'Transition to Ready and Invetory flag
is set to A';
                        elseif strcmpi(cmd.target,'SL')
                            future_state.sl_flag = 'SL';
                            action = 'Transition to Ready and Select flag
is asserted';
                        end
                    end
                case '111'
                    if strcmpi(cmd.match,'Match')
                            action = 'Transition to Ready and do nothing';
                    elseif strcmpi(cmd.match,'Not Match')
                        if strcmpi(cmd.target,future_state.session)
                            if strcmpi(future_state.inv_flag,'A')  % invert
inventory flag
                                future_state.inv_flag = 'B';
```

```
                            else
                                future_state.inv_flag = 'A';
                            end
                            action = 'Transition to Ready and Invert
Invetory flag';
                        elseif strcmpi(cmd.target,'SL')
%negate sl flag
                            if strcmpi(future_state.sl_flag,'SL')
                                future_state.sl_flag = '~SL';
                            else
                                future_state.sl_flag = 'SL';
                            end
                            action = 'Transition to Ready and negate Select
flag';
                        end
                    end
              end
            case { 'Query' , 'query'}
                if  strcmpi(future_state.inv_flag,cmd.inv_flag) &&  (
strcmpi(future_state.sl_flag,cmd.sl_flag) || strcmpi(cmd.sl_flag ,'all') )
                    future_state.state='Intermediate';
                    future_state.session = cmd.session;
                    action='Transition is done' ;
                else
                    future_state.state='Ready';
                    action='Transition is done' ;
                end
            case { 'QueryRep' , 'queryRep', 'queryrep'}   % slot counter
changes from 0000 to 7FFF and tranists into Arbitrate
                if strcmpi(future_state.session,cmd.session)
                    future_state.state='Arbitrate';
                    signal= 'slot counter has been rolled over';
                    action='Transition is done' ;
                end
            case { 'QueryAdjust' , 'queryAdjust', 'queryadjust'}  %if the new
session matches the prior one, tag transits into Intermediate future_state.
                if  strcmpi(future_state.session,cmd.session)
                    future_state.state='Intermediate';
                    action='Transition is done' ;
                end
            case { 'ACK' , 'Ack', 'ack'}   % if ACK command has a valid
handle tag backscatters its EPC and transits into Acknowledged state
otherwise transits into Arbitrate.
                if strcmpi(cmd.handle,'valid')
                    future_state.state='Acknowledged';
                    signal='Backscatter {Pc,EPC, CRC-16} or {00000, truncated
EPC, CRC-16}';
                    action='Transition is done' ;
                elseif strcmpi(cmd.handle,'invalid')
                    future_state.state='Arbitrate';
                    action='Transition is done' ;
                end
            case { 'Invalid' , 'invalid' }
                future_state.state='Reply';
            case { 'No_Input' , 'No_input' , 'no_input'}
            otherwise
                future_state.state='Arbitrate';
```

```
                action='Transition is done' ;
        end

    case { 'Arbitrate' , 'arbitrate'}
        switch cmd.type
          case { 'Select' , 'select'}
              future_state.state= 'Ready';
              switch cmd.action
                  case '000'
                      if strcmpi(cmd.match,'Match')
                          if strcmpi(cmd.target,future_state.session)
                              future_state.inv_flag = 'A';
                              action = 'Transition to Ready and Invetory flag
is set to A';
                          elseif strcmpi(cmd.target,'SL')
                              future_state.sl_flag = 'SL';
                              action = 'Transition to Ready and Select flag
is asserted';
                          end
                      elseif strcmpi(cmd.match,'Not Match')
                          if strcmpi(cmd.target,future_state.session)
                              future_state.inv_flag = 'B';
                              action = 'Transition to Ready and Invetory flag
is set to B';
                          elseif strcmpi(cmd.target,'SL')
                              future_state.sl_flag = '~SL';
                              action = 'Transition to Ready and Select flag
is deasserted';
                          end
                      end
                  case '001'
                      if strcmpi(cmd.match,'Match')
                          if strcmpi(cmd.target,future_state.session)
                              future_state.inv_flag = 'A';
                              action = 'Transition to Ready and Invetory flag
is set to A';
                          elseif strcmpi(cmd.target,'SL')
                              future_state.sl_flag = 'SL';
                              action = 'Transition to Ready and Select flag
is asserted';
                          end
                      elseif strcmpi(cmd.match,'Not Match')
                          action = 'Transition to Ready and do nothing';
                      end
                  case '010'
                      if strcmpi(cmd.match,'Match')
                              action = 'Transition to Ready and do nothing';
                      elseif strcmpi(cmd.match,'Not Match')
                          if strcmpi(cmd.target,future_state.session)
                              future_state.inv_flag = 'B';
                              action = 'Transition to Ready and Invetory flag
is set to B';
                          elseif strcmpi(cmd.target,'SL')
                              future_state.sl_flag = '~SL';
                              action = 'Transition to Ready and Select flag
is deasserted';
                          end
```

```
                    end
                case '011'
                    if strcmpi(cmd.match,'Match')
                        if strcmpi(cmd.target,future_state.session)
                            if strcmpi(future_state.inv_flag,'A')  % invert
inventory flag
                                future_state.inv_flag = 'B';
                            else
                                future_state.inv_flag = 'A';
                            end
                            action = 'Transition to Ready and Invert
Invetory flag';
                        elseif strcmpi(cmd.target,'SL')
%negate sl flag
                            if strcmpi(future_state.sl_flag,'SL')
                                future_state.sl_flag = '~SL';
                            else
                                future_state.sl_flag = 'SL';
                            end
                            action = 'Transition to Ready and negate Select
flag';
                        end
                    elseif strcmpi(cmd.match,'Not Match')
                        action = 'Transition to Ready and do nothing';
                    end
                case '100'
                    if strcmpi(cmd.match,'Match')
                        if strcmpi(cmd.target,future_state.session)
                            future_state.inv_flag = 'B';
                            action = 'Transition to Ready and Invetory flag
is set to B';
                        elseif strcmpi(cmd.target,'SL')
                            future_state.sl_flag = '~SL';
                            action = 'Transition to Ready and Select flag
is desserted';
                        end
                    elseif strcmpi(cmd.match,'Not Match')
                        if strcmpi(cmd.target,future_state.session)
                            future_state.inv_flag = 'A';
                            action = 'Transition to Ready and Invetory flag
is set to A';
                        elseif strcmpi(cmd.target,'SL')
                            future_state.sl_flag = 'SL';
                            action = 'Transition to Ready and Select flag
is asserted';
                        end
                    end
                case '101'
                    if strcmpi(cmd.match,'Match')
                        if strcmpi(cmd.target,future_state.session)
                            future_state.inv_flag = 'B';
                            action = 'Transition to Ready and Invetory flag
is set to B';
                        elseif strcmpi(cmd.target,'SL')
                            future_state.sl_flag = '~SL';
                            action = 'Transition to Ready and Select flag
is desserted';
```

```
                        end
                    elseif strcmpi(cmd.match,'Not Match')
                        action = 'Transition to Ready and do nothing';
                    end
                case '110'
                    if strcmpi(cmd.match,'Match')
                            action = 'Transition to Ready and do nothing';
                    elseif strcmpi(cmd.match,'Not Match')
                        if strcmpi(cmd.target,future_state.session)
                            future_state.inv_flag = 'A';
                            action = 'Transition to Ready and Invetory flag
is set to A';
                        elseif strcmpi(cmd.target,'SL')
                            future_state.sl_flag = 'SL';
                            action = 'Transition to Ready and Select flag
is asserted';
                        end
                    end
                case '111'
                    if strcmpi(cmd.match,'Match')
                            action = 'Transition to Ready and do nothing';
                    elseif strcmpi(cmd.match,'Not Match')
                        if strcmpi(cmd.target,future_state.session)
                            if strcmpi(future_state.inv_flag,'A')  % invert
inventory flag
                                future_state.inv_flag = 'B';
                            else
                                future_state.inv_flag = 'A';
                            end
                            action = 'Transition to Ready and Invert
Invetory flag';
                        elseif strcmpi(cmd.target,'SL')
%negate sl flag
                            if strcmpi(future_state.sl_flag,'SL')
                                future_state.sl_flag = '~SL';
                            else
                                future_state.sl_flag = 'SL';
                            end
                            action = 'Transition to Ready and negate Select
flag';
                        end
                    end
            end
        case { 'Query' , 'query'}     %if inventory and sl_flag matches
transit to Intermediate state and the new session would be whatever command
says otherwise transit into Ready.
            if  strcmpi(future_state.inv_flag,cmd.inv_flag) &&  (
strcmpi(future_state.sl_flag,cmd.sl_flag) || strcmpi(cmd.sl_flag ,'all') )
                future_state.state='Intermediate';
                future_state.session = cmd.session;
                action='Transition is done' ;
            else
                future_state.state='Ready';
                action='Transition is done' ;
            end
        case { 'QueryRep' , 'queryRep', 'queryrep'}   %if sessions matches
transmit into Intermediate state where the value of slot counter will be
```

```
determined and the final transition will happen appropriately (trasit to
Reply if the new slotcounter is 0 and into Arbitrate in case of nonzero
slotcounter)
                if strcmpi(future_state.session,cmd.session)
                    future_state.state='Intermediate';
                    signal = 'slot counter is decremented by one';
                    action='Transition is done' ;
                end
          case { 'QueryAdjust' , 'queryAdjust', 'queryadjust'}
              if  strcmpi(future_state.session,cmd.session)
                  future_state.state='Intermediate';
                  action='Transition is done' ;
              end
            case { 'No_Input' , 'No_input' , 'no_input'}
            otherwise
                future_state.state='Arbitrate';
        end

    case { 'Acknowledged' , 'acknowledged'}
        switch cmd.type
          case { 'Select' , 'select'}
              future_state.state= 'Ready';
              switch cmd.action
                  case '000'
                      if strcmpi(cmd.match,'Match')
                          if strcmpi(cmd.target,future_state.session)
                              future_state.inv_flag = 'A';
                              action = 'Transition to Ready and Invetory flag
is set to A';
                          elseif strcmpi(cmd.target,'SL')
                              future_state.sl_flag = 'SL';
                              action = 'Transition to Ready and Select flag
is asserted';
                          end
                      elseif strcmpi(cmd.match,'Not Match')
                          if strcmpi(cmd.target,future_state.session)
                              future_state.inv_flag = 'B';
                              action = 'Transition to Ready and Invetory flag
is set to B';
                          elseif strcmpi(cmd.target,'SL')
                              future_state.sl_flag = '~SL';
                              action = 'Transition to Ready and Select flag
is deasserted';
                          end
                      end
                  case '001'
                      if strcmpi(cmd.match,'Match')
                          if strcmpi(cmd.target,future_state.session)
                              future_state.inv_flag = 'A';
                              action = 'Transition to Ready and Invetory flag
is set to A';
                          elseif strcmpi(cmd.target,'SL')
                              future_state.sl_flag = 'SL';
                              action = 'Transition to Ready and Select flag
is asserted';
                          end
                      elseif strcmpi(cmd.match,'Not Match')
```

```
                        action = 'Transition to Ready and do nothing';
                    end
                case '010'
                    if strcmpi(cmd.match,'Match')
                            action = 'Transition to Ready and do nothing';
                    elseif strcmpi(cmd.match,'Not Match')
                        if strcmpi(cmd.target,future_state.session)
                            future_state.inv_flag = 'B';
                            action = 'Transition to Ready and Invetory flag
is set to B';
                        elseif strcmpi(cmd.target,'SL')
                            future_state.sl_flag = '~SL';
                            action = 'Transition to Ready and Select flag
is deasserted';
                        end
                    end
                case '011'
                    if strcmpi(cmd.match,'Match')
                        if strcmpi(cmd.target,future_state.session)
                            if strcmpi(future_state.inv_flag,'A')  % invert
inventory flag
                                future_state.inv_flag = 'B';
                            else
                                future_state.inv_flag = 'A';
                            end
                            action = 'Transition to Ready and Invert
Invetory flag';
                        elseif strcmpi(cmd.target,'SL')
%negate sl flag
                            if strcmpi(future_state.sl_flag,'SL')
                                future_state.sl_flag = '~SL';
                            else
                                future_state.sl_flag = 'SL';
                            end
                            action = 'Transition to Ready and negate Select
flag';
                        end
                    elseif strcmpi(cmd.match,'Not Match')
                        action = 'Transition to Ready and do nothing';
                    end
                case '100'
                    if strcmpi(cmd.match,'Match')
                        if strcmpi(cmd.target,future_state.session)
                            future_state.inv_flag = 'B';
                            action = 'Transition to Ready and Invetory flag
is set to B';
                        elseif strcmpi(cmd.target,'SL')
                            future_state.sl_flag = '~SL';
                            action = 'Transition to Ready and Select flag
is desserted';
                        end
                    elseif strcmpi(cmd.match,'Not Match')
                        if strcmpi(cmd.target,future_state.session)
                            future_state.inv_flag = 'A';
                            action = 'Transition to Ready and Invetory flag
is set to A';
                        elseif strcmpi(cmd.target,'SL')
```

```
                            future_state.sl_flag = 'SL';
                            action = 'Transition to Ready and Select flag
is asserted';
                        end
                    end
                case '101'
                    if strcmpi(cmd.match,'Match')
                        if strcmpi(cmd.target,future_state.session)
                            future_state.inv_flag = 'B';
                            action = 'Transition to Ready and Invetory flag
is set to B';
                        elseif strcmpi(cmd.target,'SL')
                            future_state.sl_flag = '~SL';
                            action = 'Transition to Ready and Select flag
is desserted';
                        end
                    elseif strcmpi(cmd.match,'Not Match')
                        action = 'Transition to Ready and do nothing';
                    end
                case '110'
                    if strcmpi(cmd.match,'Match')
                            action = 'Transition to Ready and do nothing';
                    elseif strcmpi(cmd.match,'Not Match')
                        if strcmpi(cmd.target,future_state.session)
                            future_state.inv_flag = 'A';
                            action = 'Transition to Ready and Invetory flag
is set to A';
                        elseif strcmpi(cmd.target,'SL')
                            future_state.sl_flag = 'SL';
                            action = 'Transition to Ready and Select flag
is asserted';
                        end
                    end
                case '111'
                    if strcmpi(cmd.match,'Match')
                            action = 'Transition to Ready and do nothing';
                    elseif strcmpi(cmd.match,'Not Match')
                        if strcmpi(cmd.target,future_state.session)
                            if strcmpi(future_state.inv_flag,'A')  % invert
inventory flag
                                future_state.inv_flag = 'B';
                            else
                                future_state.inv_flag = 'A';
                            end
                            action = 'Transition to Ready and Invert
Invetory flag';
                        elseif strcmpi(cmd.target,'SL')
%negate sl flag
                            if strcmpi(future_state.sl_flag,'SL')
                                future_state.sl_flag = '~SL';
                            else
                                future_state.sl_flag = 'SL';
                            end
                            action = 'Transition to Ready and negate Select
flag';
                        end
                    end
```

```
            end
        case { 'Query' , 'query'}   % if new sessions matches invert
inv_flag then check inventory and sl_flags.
            if strcmpi(future_state.session,cmd.session) % invert Falg if
and only if sessions mathches
                if strcmpi(future_state.inv_flag,'A')
                    future_state.inv_flag = 'B';
                else
                    future_state.inv_flag = 'A';
                end
            end
            if  strcmpi(future_state.inv_flag,cmd.inv_flag) &&  (
strcmpi(future_state.sl_flag,cmd.sl_flag) || strcmpi(cmd.sl_flag ,'all') )
                future_state.state='Intermediate';
                future_state.session = cmd.session;
                action='Transition is done' ;
            else
                future_state.state='Ready';
                action='Transition is done' ;
            end
        case { 'QueryRep' , 'queryRep', 'queryrep', 'QueryAdjust' ,
'queryAdjust', 'queryadjust'}     % if new session matches invert the
inv_flag and transit into Ready
            if strcmpi(future_state.session,cmd.session)
                if strcmpi(future_state.inv_flag,'A')  % invert inventory
flag
                    future_state.inv_flag = 'B';
                else
                    future_state.inv_flag = 'A';
                end
                future_state.state='Ready';   % transition into Ready
                action='Transition is done' ;
            end
        case { 'ACK' , 'Ack', 'ack'}  % if ACK command has a valid handle
send EPC otherwise transit into Arbitrate
            if strcmpi(cmd.handle,'valid')
                future_state.state='Acknowledged';
                signal='Backscatter {Pc,EPC, CRC-16} or {00000, truncated
EPC, CRC-16}';
            elseif strcmpi(cmd.handle,'invalid')
                future_state.state='Arbitrate';
                action='Transition is done' ;
            end
         case { 'Req_RN' , 'Req_rn', 'req_rn'}   % if command has a valid
handle and tag's access_pass is 0, transit into Secured and if access_pass is
not zero transit into Open. In case of invalid handle ignore the command.
            if strcmpi(cmd.handle,'valid')
                if strcmpi(future_state.access_pass,'1')
                    future_state.state='Open';
                    signal='Backscatter handle';
                    action='Transition is done' ;
                else
                    future_state.state='Secured';
                    signal='Backscatter handle' ;
                    action='Transition is done' ;
                end
            end
```

```
            case { 'Invalid' , 'invalid' }
                future_state.state='Acknowledged';
            case { 'No_Input' , 'No_input' , 'no_input'}
            otherwise %  in cases of 'NAK', 'Read', 'Write', 'Kill' , 'Lock',
'Access', 'BlockWrite' , 'BlockErase'
                future_state.state='Arbitrate';
                action='Transition is done' ;
        end

    case { 'Open' , 'open'}
        switch cmd.type
          case { 'Select' , 'select'}
              future_state.state= 'Ready';
              switch cmd.action
                  case '000'
                      if strcmpi(cmd.match,'Match')
                          if strcmpi(cmd.target,future_state.session)
                              future_state.inv_flag = 'A';
                              action = 'Transition to Ready and Invetory flag
is set to A';
                          elseif strcmpi(cmd.target,'SL')
                              future_state.sl_flag = 'SL';
                              action = 'Transition to Ready and Select flag
is asserted';
                          end
                      elseif strcmpi(cmd.match,'Not Match')
                          if strcmpi(cmd.target,future_state.session)
                              future_state.inv_flag = 'B';
                              action = 'Transition to Ready and Invetory flag
is set to B';
                          elseif strcmpi(cmd.target,'SL')
                              future_state.sl_flag = '~SL';
                              action = 'Transition to Ready and Select flag
is deasserted';
                          end
                      end
                  case '001'
                      if strcmpi(cmd.match,'Match')
                          if strcmpi(cmd.target,future_state.session)
                              future_state.inv_flag = 'A';
                              action = 'Transition to Ready and Invetory flag
is set to A';
                          elseif strcmpi(cmd.target,'SL')
                              future_state.sl_flag = 'SL';
                              action = 'Transition to Ready and Select flag
is asserted';
                          end
                      elseif strcmpi(cmd.match,'Not Match')
                          action = 'Transition to Ready and do nothing';
                      end
                  case '010'
                      if strcmpi(cmd.match,'Match')
                              action = 'Transition to Ready and do nothing';
                      elseif strcmpi(cmd.match,'Not Match')
                          if strcmpi(cmd.target,future_state.session)
                              future_state.inv_flag = 'B';
```

```
                            action = 'Transition to Ready and Invetory flag
is set to B';
                        elseif strcmpi(cmd.target,'SL')
                            future_state.sl_flag = '~SL';
                            action = 'Transition to Ready and Select flag
is deasserted';
                        end
                    end
                case '011'
                    if strcmpi(cmd.match,'Match')
                        if strcmpi(cmd.target,future_state.session)
                            if strcmpi(future_state.inv_flag,'A')  % invert
inventory flag
                                future_state.inv_flag = 'B';
                            else
                                future_state.inv_flag = 'A';
                            end
                            action = 'Transition to Ready and Invert
Invetory flag';
                        elseif strcmpi(cmd.target,'SL')
%negate sl flag
                            if strcmpi(future_state.sl_flag,'SL')
                                future_state.sl_flag = '~SL';
                            else
                                future_state.sl_flag = 'SL';
                            end
                            action = 'Transition to Ready and negate Select
flag';
                        end
                    elseif strcmpi(cmd.match,'Not Match')
                        action = 'Transition to Ready and do nothing';
                    end
                case '100'
                    if strcmpi(cmd.match,'Match')
                        if strcmpi(cmd.target,future_state.session)
                            future_state.inv_flag = 'B';
                            action = 'Transition to Ready and Invetory flag
is set to B';
                        elseif strcmpi(cmd.target,'SL')
                            future_state.sl_flag = '~SL';
                            action = 'Transition to Ready and Select flag
is desserted';
                        end
                    elseif strcmpi(cmd.match,'Not Match')
                        if strcmpi(cmd.target,future_state.session)
                            future_state.inv_flag = 'A';
                            action = 'Transition to Ready and Invetory flag
is set to A';
                        elseif strcmpi(cmd.target,'SL')
                            future_state.sl_flag = 'SL';
                            action = 'Transition to Ready and Select flag
is asserted';
                        end
                    end
                case '101'
                    if strcmpi(cmd.match,'Match')
                        if strcmpi(cmd.target,future_state.session)
```

```
                            future_state.inv_flag = 'B';
                            action = 'Transition to Ready and Invetory flag
is set to B';
                        elseif strcmpi(cmd.target,'SL')
                            future_state.sl_flag = '~SL';
                            action = 'Transition to Ready and Select flag
is desserted';
                        end
                    elseif strcmpi(cmd.match,'Not Match')
                        action = 'Transition to Ready and do nothing';
                    end
                case '110'
                    if strcmpi(cmd.match,'Match')
                            action = 'Transition to Ready and do nothing';
                    elseif strcmpi(cmd.match,'Not Match')
                        if strcmpi(cmd.target,future_state.session)
                            future_state.inv_flag = 'A';
                            action = 'Transition to Ready and Invetory flag
is set to A';
                        elseif strcmpi(cmd.target,'SL')
                            future_state.sl_flag = 'SL';
                            action = 'Transition to Ready and Select flag
is asserted';
                        end
                    end
                case '111'
                    if strcmpi(cmd.match,'Match')
                            action = 'Transition to Ready and do nothing';
                    elseif strcmpi(cmd.match,'Not Match')
                        if strcmpi(cmd.target,future_state.session)
                            if strcmpi(future_state.inv_flag,'A')  % invert
inventory flag
                                future_state.inv_flag = 'B';
                            else
                                future_state.inv_flag = 'A';
                            end
                            action = 'Transition to Ready and Invert
Invetory flag';
                        elseif strcmpi(cmd.target,'SL')
%negate sl flag
                            if strcmpi(future_state.sl_flag,'SL')
                                future_state.sl_flag = '~SL';
                            else
                                future_state.sl_flag = 'SL';
                            end
                            action = 'Transition to Ready and negate Select
flag';
                        end
                    end
            end
          case { 'Query' , 'query'}   % if new sessions matches invert
inv_flag then check inventory and sl_flags.
            if strcmpi(future_state.session,cmd.session) % invert Falg if
and only if sessions mathches
                if strcmpi(future_state.inv_flag,'A')
                    future_state.inv_flag = 'B';
                else
```

```
                    future_state.inv_flag = 'A';
                end
            end
            if  strcmpi(future_state.inv_flag,cmd.inv_flag) &&  (
strcmpi(future_state.sl_flag,cmd.sl_flag) || strcmpi(cmd.sl_flag ,'all') )
                future_state.state='Intermediate';
                future_state.session = cmd.session;
                action='Transition is done' ;
            else
                future_state.state='Ready';
                action='Transition is done' ;
            end
        case { 'QueryRep' , 'queryRep', 'queryrep', 'QueryAdjust' ,
'queryAdjust', 'queryadjust'}     % if new session matches invert the
inv_flag and transit into Ready
            if strcmpi(future_state.session,cmd.session)
                if strcmpi(future_state.inv_flag,'A')  % invert inventory
flag
                    future_state.inv_flag = 'B';
                else
                    future_state.inv_flag = 'A';
                end
                future_state.state='Ready';   % transition into Ready
                action='Transition is done' ;
            end
        case { 'ACK' , 'Ack', 'ack'}  % if ACK command has a valid handle
send EPC otherwise transit into Arbitrate
            if strcmpi(cmd.handle,'valid')
                future_state.state='Open';
                signal='Backscatter {PC,EPC, CRC-16} or {00000, truncated
EPC, CRC-16}';
            elseif strcmpi(cmd.handle,'invalid')
                future_state.state='Arbitrate';
                action='Transition is done' ;
            end
        case { 'NAK' , 'Nak', 'nak'}
            future_state.state='Arbitrate';
            action='Transition is done' ;
        case { 'Req_RN' , 'Req_rn', 'req_rn'}
            if strcmpi(cmd.handle,'valid')
                signal='Backscatter new RN16';
            end
        case { 'Read' , 'read', 'Write' , 'write', 'BlockWrite' ,
'Blockwrite' , 'blockwrite', 'BlockErase' , 'Blockerase' , 'blockerase'}
            if strcmpi(cmd.handle,'valid')
                if strcmpi(cmd.memory_access,'valid')
                    signal='Backscatter data and handle';
                else
                    signal='Backscatter error code';
                end
            end
        case { 'Kill' , 'kill' }
            if strcmpi(cmd.handle,'valid')
                if strcmpi(future_state.kill_pass,'0')
                    signal='Backscatter error code';
                elseif strcmpi(cmd.kill_pass,'valid')
                    signal='Backscatter handle';
```

```
                    future_state=Killed_state;
                    action='Transition is done' ;
                else
                   future_state.state='Arbitrate';
                   action='Transition is done' ;
                end
            end

        case { 'Access' , 'access' }
            if strcmpi(cmd.handle,'valid')
                if strcmpi(cmd.access_pass,'valid')
                    signal='Backscatter handle';
                    future_state.state='Secured';
                    action='Transition is done' ;
                else
                    future_state.state='Arbitrate';
                    action='Transition is done' ;
                end
            end
          case { 'No_Input' , 'No_input' , 'no_input'}
          case { 'Lock' , 'lock' }
          case { 'Invalid' , 'invalid' }   % this part should be modified
based on the table B.5 of standard.
        end

    case { 'Secured' , 'secured'}
        switch cmd.type
          case { 'Select' , 'select'}
            future_state.state= 'Ready';
            switch cmd.action
                case '000'
                    if strcmpi(cmd.match,'Match')
                        if strcmpi(cmd.target,future_state.session)
                            future_state.inv_flag = 'A';
                            action = 'Transition to Ready and Invetory flag
is set to A';
                        elseif strcmpi(cmd.target,'SL')
                            future_state.sl_flag = 'SL';
                            action = 'Transition to Ready and Select flag
is asserted';
                        end
                    elseif strcmpi(cmd.match,'Not Match')
                        if strcmpi(cmd.target,future_state.session)
                            future_state.inv_flag = 'B';
                            action = 'Transition to Ready and Invetory flag
is set to B';
                        elseif strcmpi(cmd.target,'SL')
                            future_state.sl_flag = '~SL';
                            action = 'Transition to Ready and Select flag
is deasserted';
                        end
                    end
                case '001'
                    if strcmpi(cmd.match,'Match')
                        if strcmpi(cmd.target,future_state.session)
                            future_state.inv_flag = 'A';
```

```
                            action = 'Transition to Ready and Invetory flag
is set to A';
                        elseif strcmpi(cmd.target,'SL')
                            future_state.sl_flag = 'SL';
                            action = 'Transition to Ready and Select flag
is asserted';
                        end
                    elseif strcmpi(cmd.match,'Not Match')
                        action = 'Transition to Ready and do nothing';
                    end
                case '010'
                    if strcmpi(cmd.match,'Match')
                            action = 'Transition to Ready and do nothing';
                    elseif strcmpi(cmd.match,'Not Match')
                        if strcmpi(cmd.target,future_state.session)
                            future_state.inv_flag = 'B';
                            action = 'Transition to Ready and Invetory flag
is set to B';
                        elseif strcmpi(cmd.target,'SL')
                            future_state.sl_flag = '~SL';
                            action = 'Transition to Ready and Select flag
is deasserted';
                        end
                    end
                case '011'
                    if strcmpi(cmd.match,'Match')
                        if strcmpi(cmd.target,future_state.session)
                            if strcmpi(future_state.inv_flag,'A')  % invert
inventory flag
                                future_state.inv_flag = 'B';
                            else
                                future_state.inv_flag = 'A';
                            end
                            action = 'Transition to Ready and Invert
Invetory flag';
                        elseif strcmpi(cmd.target,'SL')
%negate sl flag
                            if strcmpi(future_state.sl_flag,'SL')
                                future_state.sl_flag = '~SL';
                            else
                                future_state.sl_flag = 'SL';
                            end
                            action = 'Transition to Ready and negate Select
flag';
                        end
                    elseif strcmpi(cmd.match,'Not Match')
                        action = 'Transition to Ready and do nothing';
                    end
                case '100'
                    if strcmpi(cmd.match,'Match')
                        if strcmpi(cmd.target,future_state.session)
                            future_state.inv_flag = 'B';
                            action = 'Transition to Ready and Invetory flag
is set to B';
                        elseif strcmpi(cmd.target,'SL')
                            future_state.sl_flag = '~SL';
```

```
                            action = 'Transition to Ready and Select flag
is desserted';
                        end
                    elseif strcmpi(cmd.match,'Not Match')
                        if strcmpi(cmd.target,future_state.session)
                            future_state.inv_flag = 'A';
                            action = 'Transition to Ready and Invetory flag
is set to A';
                        elseif strcmpi(cmd.target,'SL')
                            future_state.sl_flag = 'SL';
                            action = 'Transition to Ready and Select flag
is asserted';
                        end
                    end
                case '101'
                    if strcmpi(cmd.match,'Match')
                        if strcmpi(cmd.target,future_state.session)
                            future_state.inv_flag = 'B';
                            action = 'Transition to Ready and Invetory flag
is set to B';
                        elseif strcmpi(cmd.target,'SL')
                            future_state.sl_flag = '~SL';
                            action = 'Transition to Ready and Select flag
is desserted';
                        end
                    elseif strcmpi(cmd.match,'Not Match')
                        action = 'Transition to Ready and do nothing';
                    end
                case '110'
                    if strcmpi(cmd.match,'Match')
                            action = 'Transition to Ready and do nothing';
                    elseif strcmpi(cmd.match,'Not Match')
                        if strcmpi(cmd.target,future_state.session)
                            future_state.inv_flag = 'A';
                            action = 'Transition to Ready and Invetory flag
is set to A';
                        elseif strcmpi(cmd.target,'SL')
                            future_state.sl_flag = 'SL';
                            action = 'Transition to Ready and Select flag
is asserted';
                        end
                    end
                case '111'
                    if strcmpi(cmd.match,'Match')
                            action = 'Transition to Ready and do nothing';
                    elseif strcmpi(cmd.match,'Not Match')
                        if strcmpi(cmd.target,future_state.session)
                            if strcmpi(future_state.inv_flag,'A')  % invert
inventory flag
                                future_state.inv_flag = 'B';
                            else
                                future_state.inv_flag = 'A';
                            end
                            action = 'Transition to Ready and Invert
Invetory flag';
                        elseif strcmpi(cmd.target,'SL')
%negate sl flag
```

```
                                if strcmpi(future_state.sl_flag,'SL')
                                    future_state.sl_flag = '~SL';
                                else
                                    future_state.sl_flag = 'SL';
                                end
                                action = 'Transition to Ready and negate Select
flag';
                            end
                        end
                end
            case { 'Query' , 'query'}   % if new sessions matches invert
inv_flag then check inventory and sl_flags.
                if strcmpi(future_state.session,cmd.session) % invert Falg if
and only if sessions mathches
                    if strcmpi(future_state.inv_flag,'A')
                        future_state.inv_flag = 'B';
                    else
                        future_state.inv_flag = 'A';
                    end
                end
                if  strcmpi(future_state.inv_flag,cmd.inv_flag) &&  (
strcmpi(future_state.sl_flag,cmd.sl_flag) || strcmpi(cmd.sl_flag ,'all') )
                    future_state.state='Intermediate';
                    future_state.session = cmd.session;
                    action='Transition is done' ;
                else
                    future_state.state='Ready';
                    action='Transition is done' ;
                end
            case { 'QueryRep' , 'queryRep', 'queryrep', 'QueryAdjust' ,
'queryAdjust', 'queryadjust'}     % if new session matches invert the
inv_flag and transit into Ready
                if strcmpi(future_state.session,cmd.session)
                    if strcmpi(future_state.inv_flag,'A')  % invert inventory
flag
                        future_state.inv_flag = 'B';
                    else
                        future_state.inv_flag = 'A';
                    end
                    future_state.state='Ready';   % transition into Ready
                    action='Transition is done' ;
                end
            case { 'ACK' , 'Ack', 'ack'}  % if ACK command has a valid handle
send EPC otherwise transit into Arbitrate
                if strcmpi(cmd.handle,'valid')
                    future_state.state='Secured';
                    signal='Backscatter {PC,EPC, CRC-16} or {00000, truncated
EPC, CRC-16}';
                elseif strcmpi(cmd.handle,'invalid')
                    future_state.state='Arbitrate';
                    action='Transition is done' ;
                end
            case { 'NAK' , 'Nak', 'nak'}
                future_state.state='Arbitrate';
                action='Transition is done' ;
            case { 'Req_RN' , 'Req_rn', 'req_rn'}
                if strcmpi(cmd.handle,'valid')
```

```
                signal='Backscatter new RN16';
            end
        case { 'Read' , 'read', 'Write' , 'write', 'BlockWrite' ,
'Blockwrite' , 'blockwrite', 'BlockErase' , 'Blockerase' , 'blockerase'}
            if strcmpi(cmd.handle,'valid')
                if strcmpi(cmd.memory_access,'valid')
                    signal='Backscatter data and handle';
                else
                    signal='Backscatter error code';
                end
            end
        case { 'Kill' , 'kill' }
            if strcmpi(cmd.handle,'valid')
                if strcmpi(future_state.kill_pass,'0')
                    signal='Backscatter error code';
                elseif strcmpi(cmd.kill_pass,'valid')
                    signal='Backscatter handle';
                    future_state=Killed_state;
                    action='Transition is done' ;
                else
                   future_state.state='Arbitrate';
                   action='Transition is done' ;
                end
            end
        case { 'Access' , 'access' }
            if strcmpi(cmd.handle,'valid')
                if strcmpi(cmd.access_pass,'valid')
                    signal='Backscatter handle';
                else
                    future_state.state='Arbitrate';
                    action='Transition is done' ;
                end
            end
          case { 'Lock' , 'lock' }
              if strcmpi(cmd.handle,'valid')
                  if strcmpi(cmd.lock_payload,'valid')
                      signal='Backscatter handle';
                  else
                      signal='Backscatter error code';
                  end
              end
          case { 'No_Input' , 'No_input' , 'no_input'}
          case { 'Invalid' , 'invalid' } % this part should be modified
based on the table B.5 of standard.
       end

    case { 'Killed' , 'killed'}
       future_state=Killed_state;

end
```

BIBLIOGRAPHY

Alien Technology. (2005). EPCglobal Class 1 Gen 2 RFID Specification.

EPCglobal Inc. (2008). *EPC Radio-Frequency Identity Protocols Class-1 Generation-2 UHF RFID Protocol for Communications at 860 MHz - 960 MHz, version 1.2.0.*

Puterman, M. L. (2005). *Markov Decision Processes: Descrete Stochastic Dynamic programming.* Hoboken, New Jersey: John Wiley & Sons, Inc.

Tanenbaum, A. S. (2003). *Computer Networks.* Upper Saddle River, NJ: Prentice Hall.