Specification for EPC HF RFID Air Interface

EPC™ Radio-Frequency Identity Protocols
EPC Class-1 HF RFID Air Interface Protocol
for Communications at 13.56 MHz

Version 2.0.3

5 September, 2011

Table of Contents

List of Figures

LIst of Tables

Foreword

This document was prepared by the EPCglobal HF Air Interface Working Group.

This Class-1 specification defines the base (or foundation) of four EPCglobal radio-frequency identification (RFID) protocols. The feature set of all four protocols is located at www.epcglobalinc.org/standards. The feature set for the Class-1 protocol is normative to this document.

The feature set for higher-class protocols is informative to this document.

Regardless of the class definitions, higher class Tags shall not conflict with the operation of, nor degrade the performance of, Class-1 Tags located in the same RF environment.

Disclaimer

Whilst every effort has been made to ensure that this document and the information contained herein are correct, EPCglobal and any other party involved in the creation of the document hereby state that the document is provided on an "as is" basis without warranty, either expressed or implied, including but not limited to any warranty that the use of the information herein will not infringe any rights, of accuracy or fitness for purpose, and hereby disclaim any liability, direct or indirect, for damages or loss relating to the use of the document.

1 Scope

This document specifies:

- Physical interactions (the signalling layer of the communication link) between interrogators and tags,
- Interrogator and tag operating procedures and commands, and
- The collision arbitration scheme used to identify a specific tag in a multiple-tag environment.

2 Conformance

In order to claim conformance with EPC™ Class-1 HF RFID AI, it is necessary to comply with all of the relevant clauses of EPC™ Class-1 HF RFID AI except those marked 'optional' and it is also necessary to operate within the local national radio regulations (which may require further restrictions).

2.1 Claiming conformance

A device shall not claim conformance with this specification unless the device complies with:

- all clauses in this specification (except those marked as optional);
- the conformance document associated with this specification; and
- all local radio regulations.

Conformance may also require a license from the owner of any intellectual property utilized by said device.

2.2 General conformance requirements

2.2.1 Interrogators

To conform to this specification, an Interrogator shall:

- Meet the requirements of this specification,
- Implement the mandatory commands defined in this specification,
- Modulate/transmit and receive/demodulate a sufficient set of the electrical signals defined in the signaling layer of this specification to communicate with conformant Tags, and
- Conform to all local radio regulations.

To conform to this specification, an Interrogator may:

- Implement any subset of the optional commands defined in this specification, and
- Implement any proprietary and/or custom commands in conformance with this specification.

To conform to this specification, an Interrogator shall not:

- Implement any command that conflicts with this specification, or
- Require using an optional, proprietary, or custom command to meet the requirements of this specification.

2.2.2 Tags

To conform to this specification, a Tag shall:

- Meet the requirements of this specification,
- Operate at a frequency of 13,56MHz in the ISM frequency band,
- Implement the mandatory commands defined in this specification,
- Modulate a loadmodulation signal only after receiving the requisite command from an Interrogator, and
- Conform to all local radio regulations.

To conform to this specification, a Tag may:

- Implement any subset of the optional commands defined in this specification, and
- Implement any proprietary and/or custom commands as defined in 2.3.3 and 2.3.4, respectively.

To conform to this specification, a Tag shall not:

- Implement any command that conflicts with this specification,
- Require using an optional, proprietary, or custom command to meet the requirements of this specification, or
- Modulate a loadmodulation signal unless commanded to do so by an Interrogator using the signaling layer defined in this specification.

2.3 Command structure and extensibility

This specification allows four command types: (1) mandatory, (2) optional, (3) proprietary, and (4) custom. Subclause 6.3.4.11 and Table 6.19 define the structure of the command codes used by Interrogators and Tags for each of the four types, as well as the availability of future extensions.

All commands defined by this specification are either mandatory or optional.

Proprietary commands and custom commands are vendor-defined.

2.3.1 Mandatory commands

Conforming Tags shall support all mandatory commands. Conforming Interrogators shall support all mandatory commands.

2.3.2 Optional commands

Conforming Tags may or may not support optional commands. Conforming Interrogators may or may not support optional commands. If a Tag or an Interrogator implements an optional command, the tag shall implement the optional command in the manner specified in this specification.

2.3.3 Proprietary commands

Proprietary commands may be enabled in conformance with this specification, but are not specified herein. All proprietary commands shall be capable of being permanently disabled. Proprietary commands are intended for manufacturing purposes and shall not be used in field-deployed RFID systems.

2.3.4 Custom commands

Custom commands may be enabled in conformance with this specification, but are not specified herein. An Interrogator shall issue a custom command only after (1) singulating a Tag, and (2) reading (or having prior knowledge of the Tag manufacturer's identification in the Tag's TID memory. An Interrogator shall use a custom command only in accordance with the specifications of the Tag manufacturer identified in the Tag ID. A custom command shall not solely duplicate the functionality of any mandatory or optional command defined in this specification by a different method.

2.4 Reserved for Future Use (RFU)

This specification denotes some Tag memory addresses, Interrogator command codes, and bit fields within some Interrogator commands as being RFU. In general, EPCglobal is reserving these RFU values for use in higher-class specifications. Under some circumstances EPCglobal may permit another standards body or related organization to use one or more of these RFU values for standardization purposes. In these circumstances the permitted body shall keep EPCglobal appraised, in a timely manner, of its use or potential use of these RFU values. Third parties, including but not limited to solution providers and end users, shall not use these RFU values for proprietary purposes.

Bit E_h of XPC_W1 (EPC memory location 211_h – see 6.3.4.1.2.5) shall be reserved for use as a protocol functionality indicator. If another standards-setting body authorizes any physical or logical Tag operation that is incompatible with this EPCglobal specification then that standards-setting body shall employ bit E_h as follows:

- If bit E_h of XPC_W1 contains a logical 0 then the application is referred to as an GS1 EPCglobal™ Application and a Tag shall follow the physical and logical requirements specified by this EPC™ Class-1 HF RFID AI Protocol. If bit E_h contains a logical 1 then the application is referred to as a non-EPCglobal™ Application and a Tag may follow the physical and logical requirements specified by a non-EPC™ Protocol. The default value for bit E_h shall be 0.

3 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 3309: *Information technology – Telecommunications and information exchange between systems – High-level data link control (HDLC) procedures – Frame structure*

ISO/IEC 7816-6, *Identification cards — Integrated circuit cards — Part 6: Inter-industry data elements for interchange*

ISO/IEC 15693 (all parts), *Identification cards — Contactless integrated circuit(s) cards — Vicinity cards*

ISO/IEC 15961: *Information technology, Automatic identification and data capture – Radio frequency identification (RFID) for item management – Data protocol: application interface*

ISO/IEC 15962: *Information technology, Automatic identification and data capture techniques – Radio frequency identification (RFID) for item management – Data protocol: data encoding rules and logical memory functions*

ISO/IEC 15963, Information technology — Radio frequency identification for item management — Unique identification for RF tags

ISO/IEC 18000-1, Information technology — Radio frequency identification for item management — Reference architecture and definition of parameters to be standardized

ISO/IEC 18000-3, Information technology — Radio frequency identification for item management — Part 3: Parameters for air interface communications at 13,56MHz

ISO/IEC 18000-6, Information technology — Radio frequency identification for item management — Part 6: Parameters for air interface communications at 860MHz to 960MHz

EPC™ Radio-Frequency Identity Protocols - Class-1 Generation-2 UHF RFID Protocol for Communications at 860MHz – 960MHz

ISO/IEC TR 18047-3, Information technology — Radio frequency identification device conformance test methods — Part 3: Test methods for air interface communications at 13,56MHz

ISO/IEC 19762 (all parts), Information technology — Automatic identification and data capture techniques — Harmonized vocabulary

ETSI EN 300 330, Electromagnetic compatibility and Radio spectrum Matters (ERM); Short Range Devices (SRD); Technical characteristics and test methods for radio equipment in the frequency range 9kHz to 30MHz

U.S. Code of Federal Regulations (CFR), Title 47, Chapter I, Part 15: Radio-frequency devices, U.S. Federal Communications Commission

EPCglobal Tag Data Standards

EPCglobal™ FMCG RFID Physical Operating Requirements Document

EPCglobal™ ILT JRG Protocol Requirements Document

4 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO/IEC 19762 (all parts) and the following apply.

4.1
cover-coding

The method by which an interrogator obscures information that it is transmitting to a tag.

4.2
cover-coded text

Information that is cover-coded.

4.3
full-duplex communications

The communications channel that carries data in both directions at once. Compare with half-duplex communications.

4.4
half-duplex communications

The communications channel that carries data in one direction at a time rather than in both directions at once. Compare with full-duplex communications.

4.5
handle

The 16-bit tag-authentication number.

4.6
loadmodulation

Loadmodulated Tag reply is transmitted to the interrogator by changing the magnetic load of the tag on the RF field (see also 6.3.3.1.3).

4.7
packetCRC

A 16-bit cyclic-redundancy check (CRC) code that a Tag with nonzero-valued XI dynamically calculates over its PC, XPC, and EPC and provides by loadmodulatation during inventory.
(see StoredCRC).

4.8
packetPC

Protocol-control information that a Tag with nonzero-valued XI dynamically calculates and provides by loadmodulation during inventory.
(see StoredPC).

4.9
permalock

A memory location whose lock status is unchangeable (*i.e.* the memory location is permanently locked or permanently unlocked) except by recommissioning the Tag is said to be permalocked.

4.10
Phase Jitter Modulation (PJM)

modulation technique that transmits data as very small phase changes in the powering field

4.11
physical layer

The data coding and modulation waveforms used in interrogator-to-tag and tag-to-interrogator signalling.

4.12
pivot

The average length of an R=>T data symbol: pivot=(data-0 symbol length + data-1 symbol length) / 2.
(See also "R=>T" in section 5.1).

4.13
plaintext

Information that is not cover-coded.

4.14
protocol

The physical layer and tag-identification layer specification.

4.15
recommisioning

A significant altering of a Tag's functionality and/or memory contents, as commanded by an Interrogator, typically in response to a change in the Tag's usage model or purpose.

4.16
slot

The point in an inventory round at which a tag may respond. Slot is the value output by a tag slot counter; tags reply when their slot (*i.e.* the value in their slot counter) is zero. See also *Q* in section 5.1.

4.17
storedCRC

A 16-bit cyclic-redundancy check (CRC) code that a Tag calculates over its StoredPC and EPC and stores in EPC memory at power-up, and may provide by loadmodulation during inventory.
(see PacketCRC).

4.18
storedPC

Protocol-control information stored in EPC memory that a Tag with zero-valued XI provides by loadmodulation during inventory (see PacketPC).

4.19
Tari

The reference time interval for a data-0 in interrogator-to-tag signalling.

4.20
word

16-bits.

5 Symbols (and abbreviated terms)

For the purposes of this document, the symbols and abbreviated terms given in ISO/IEC 18000-1, ISO/IEC 19762 and the following apply.

5.1 Symbols

DR ASK Method: divide ratio

PJM Method: Bit 0 of the reply channel selection

f_c Carrier frequency

M(ASK) Tag reply modulation type

M_h RF signal envelope ripple (overshoot)

M_l RF signal envelope ripple (undershoot)

M(PJM) Bit 1 and bit 2 of the reply channel selection

M_s RF signal level when OFF

Q Slot-count parameter (parameter that an interrogator uses to regulate the probability of tag response)

R **Interrogator (also sometimes called Reader)**

R=>T Interrogator-to-tag

RTcal Interrogator-to-tag calibration symbol

T **Tag**

T_1 Time from interrogator transmission to tag response

T_2 Time from tag response to interrogator transmission

T_3 Time an interrogator waits, after T_1, before it issues another command

T_4 Minimum time between interrogator commands

T_f or $T_{f,10\text{-}90\%}$ RF signal envelope fall time

T_{pri} Link pulse-repetition interval (T_{pri}=1/LF)

T_r or $T_{r,10\text{-}90\%}$ RF signal envelope rise time

TRext ASK Method: chooses whether the T=R preamble is prefixed with a pilot tone

PJM Method: Bit 3 of the reply channel selection

T_s RF signal settling time

T=>R Tag-to-interrogator

TRcal Tag-to-interrogator calibration symbol

X_{fp} Floating-point value

$xxxx_2$ Binary notation

$xxxx_h$ Hexadecimal notation

5.2 Abbreviated terms

AM Amplitude modulation

CRC Cyclic Redundancy Check

NOTE: This specification uses two CRC algorithms: CRC-5 (5-bit CRC) and CRC-16(16-bit CRC) and three different logical CRC-16s: StoredCRC, PacketCRC and CRC-16c.
For the EPC bank word 0 or ACK the following two logical CRC-16s are used:
- StoredCRC = CRC-16 calculated at startup and mapped to EPC word 0
- PacketCRC = CRC-16 calculated over the response data of the tag in case of the ACK command

For all other cases and commands the following logical CRC-16 is used:
- CRC-16c = CRC-16 calculated over the response data of the tag

CW Continuous wave

dBch Decibels referenced to the integrated power in the reference channel

DSB Double sideband

DSB-ASK Double-sideband amplitude-shift keying

DR Divide ratio

EPC Electronic product code

FCC Federal Communications Commission

FT Frequency tolerance

LF Link frequency (LF=1/Tpri)

MFM Modified Frequency Modulation

N/A Not Applicable

NSI Numbering system identifier

PIE Pulse-interval encoding

PJM Phase Jitter Modulation

ppm Parts-per-million

PC Protocol Control

RF Radio frequency

RFU Reserved for future use

RN16 16-bit random or pseudo-random number

RNG Random or pseudo-random number generator

TDM Time-division multiplexing or time-division multiplexed (as appropriate)

TID Tag-identification or tag identifier, depending on context

UMI User-memory indicator

XI XPC_W1 indicator

XPC Extended protocol control

XPC_W1 **XPC word 1**

XPC_W2 **XPC word 2**

XEB **XPC extension bit**

5.3 Notation

This specification uses the following notational conventions:

- States and flags are denoted in bold. Example: **ready**.
- Commands are denoted in italics. Variables are also denoted in italics. Where there might be confusion between commands and variables, this specification shall make an explicit statement. Example: *BeginRound*.
- Procedures are shown as ***italics underline***
- Command parameters are underlined. Example: Pointer.
- For logical negation, labels are preceded by '~'. Example: If flag is true, then ~flag is false.
- The symbol, R=>T, refers to commands or signalling from an interrogator to a tag (reader-to-tag).
- The symbol, T=>R, refers to commands or signalling from a tag to an interrogator (tag-to-reader).

6 Requirements: Physical layer, collision management system and protocol values for 13,56MHz systems

Annex L provides a summary of the optional Tag features

6.1 Protocol overview

6.1.1 Physical layer

An Interrogator sends information to one or more Tags by modulating an RF carrier using double-sideband amplitude shift keying (ASK) using a pulse-interval encoding (PIE) format. Tags receive their operating energy from this same modulated RF carrier.

An Interrogator receives information from a Tag by transmitting an unmodulated RF carrier and listening for a loadmodulated reply. Tags communicate information by loadmodulating the amplitude and/or phase of the RF carrier.
The encoding format, selected in response to Interrogator commands, is either Manchester-, or Miller-modulated subcarrier or FM0 baseband. The communications link between Interrogators and Tags is half-duplex, meaning that Tags shall not be required to demodulate Interrogator commands while loadmodulating. A Tag shall not respond to a mandatory or optional command using full-duplex communications.
Interrogators as well as Tags may optionally provide a physical layer with phase jitter modulation (PJM) using MFM (modified frequency modulation) encoding for Interrogator to Tag transmission and a MFM encoded, binary phase shift keying (BPSK) modulated subcarrier for Tag to Interrogator transmission.

6.1.2 Tag-identification layer

An Interrogator manages Tag populations using three basic operations:

1. **Select.** The operation of choosing a Tag population for inventory and access. A *Select* command may be applied successively to select a particular Tag population based on user-specified criteria. This operation is analogous to selecting records from a database.
2. **Inventory.** The operation of identifying Tags. An Interrogator begins an inventory round by transmitting a *BeginRound* command in one of two sessions. One or more Tags may reply. The Interrogator detects a single Tag reply and requests the PC/XPC word(s), EPC, and if required the packet CRC-16 from the Tag. Inventory comprises multiple commands. An inventory round operates in one and only one session at a time.
3. **Access.** The operation of communicating with (reading from and/or writing to) a Tag. An individual Tag must be uniquely identified prior to access. Access comprises multiple commands, some of which may employ one-time-pad based cover-coding of the R=>T link.

6.2 General

6.2.1 EPC™ Class-1 HF RFID AI interoperability

EPC™ Class-1 HF RFID AI operates at 13,56MHz. EPC™ Class-1 HF RFID AI is not interoperable with ISO/IEC 18000-3 Mode 1 and Mode 2, but all three are expected to operate in the same environment.

6.3 EPC™ Class-1 HF RFID AI: Physical layer, collision management system and protocols

6.3.1 Normative aspects: physical and media access control (MAC) parameters

Table 6.1 and 0 provide an overview of parameters for R=>T and T=>R communications according to this specification; for detailed requirements refer to the referenced Sub-clause. For those parameters that do not apply, or are not used in this specification, the notation “N/A” shall indicate that the parameter is “Not Applicable”. The table is as specified in ISO/IEC 18000-1.

Table 6.1. Interrogator to tag (R=>T) communications

Ref.	Parameter Name	Description	Sub-clause
Int:1	Operating Frequency Range	Fixed single frequency see Int:1a	6.3.3.1
Int:1a	Default Operating Frequency	13,56MHz	6.3.3.1
Int:1b	Operating Channels (spread-spectrum systems)	N/A	N/A
Int:1c	Operating Frequency Accuracy	In accordance with local regulations	6.3.3.1.2.1
Int:1d	Frequency Hop Rate (frequency-hopping [FHSS] systems)	N/A	N/A
Int:1e	Frequency Hop Sequence (frequency-hopping [FHSS] systems)	N/A	N/A
Int:2	Occupied Channel Bandwidth	N/A	N/A
Int:2a	Minimum Receiver Bandwidth	N/A	N/A
Int:3	Interrogator Transmit Maximum Magnetic Field Strength	In accordance with local regulations	N/A
Int:4	Interrogator Transmit Spurious Emissions	In accordance with local regulations	6.3.3.1.2.5
Int:4a	Interrogator Transmit Spurious Emissions, In-Band (spread-spectrum systems)	In accordance with local regulations	6.3.3.1.2.5
Int:4b	Interrogator Transmit Spurious Emissions, Out-of-Band	In accordance with local regulations	6.3.3.1.2.5
Int:5	Interrogator Transmitter Spectrum Mask	In accordance with local regulations	N/A
Int:6	Timing	As specified	6.3.3.1.3
Int:6a	Transmit-to-Receive Turn-Around Time	Less than 73.1 µs maximum ($1024/f_c$-$32/f_c$)	6.3.3.1.6, Figure 6.24, and Table 6.14
Int:6b	Receive-to-Transmit Turn-Around Time	When communicating with a tag, 151 µs minimum; 1208 µs maximum when tag is in **reply** & **acknowledged** states; no maximum limit otherwise	6.3.3.1.2.5, Figure 6.24, and Table 6.14
Int:6c	Dwell Time or Interrogator Transmit Power-On Ramp	1500 µs, maximum settling time	6.3.3.1.2.6, Table 6.7
Int:6d	Decay Time or Interrogator Transmit Power-Down Ramp	500 µs, maximum	6.3.3.1.2.7, Table 6.7
Int:7	Modulation	ASK Method: min. 10%, max. 30% Optional PJM Method: min. deviation +/- 3,0deg. max. deviation +/- 6,0deg.	6.3.3.1.2.2
Int:7a	Spreading Sequence (direct-sequence [DSSS] systems)	N/A	N/A
Int:7b	Chip Rate (spread-spectrum systems)	N/A	N/A
Int:7c	Chip Rate Accuracy (spread-spectrum systems)	N/A	N/A
Int:7d	Modulation Index	ASK Method: (A-B)/(A+B) index 10% minimum to 30% maximum	6.3.3.1.2.5, Figure 6.3, Table 6.5
Int:7e	Duty Cycle	As specified	6.3.3.1.2.5
Int:7f	FM Deviation	N/A	N/A
Int:8	Data Coding	ASK Method: PIE Mode: MFM	6.3.3.1.2.3, Figure 6.1 and optionally Figure 6.2
Int:9	Bit Rate	ASK Method: 26,7 kbit/s to 100 kbit/s (assuming equally probable data) Optional PJM Method: 212 kbit/s	6.3.3.1.2.4 and optionally 6.3.3.1.2
Int:9a	Bit Rate Accuracy	+/– 1%, minimum	6.3.3.1.2.3

Ref.	Parameter Name	Description	Sub-clause
Int:10	Interrogator Transmit Modulation Accuracy	As specified	6.3.3.1.2.3
Int:11	Preamble	Required	6.3.3.1.2.8
Int:11a	Preamble Length	As specified	6.3.3.1.2.8
Int:11b	Preamble Waveform(s)	As specified	Figure 6.6 and optionally Figure 6.7
Int:11c	Bit Sync Sequence	None	N/A
Int:11d	Frame Sync Sequence	Required	6.3.3.1.2.8
Int:12	Scrambling (spread-spectrum systems)	N/A	N/A
Int:13	Bit Transmission Order	MSB is transmitted first	6.3.3.1.4
Int:14	Wake-up process	As specified	6.3.3.1.2.6
Int:15	Polarization	N/A	N/A

Table 6.2. Tag-to-interrogator (T=>R) communications

Ref.	Parameter Name	Description	Sub-clause
Tag:1	Operating Frequency Range	13,56MHz +/- tag sub-carrier frequencies as specified in Tag:7e	6.3.3.1.3, Table 6.9 and Table 6.10
Tag:1a	Default Operating Frequency	Fixed carrier frequency as specified in Int:1a.	6.3.3.1.3
Tag:1b	Operating Channels (spread-spectrum systems)	N/A	N/A
Tag:1c	Operating Frequency Accuracy	As specified	6.3.3.1.2.1
Tag:1d	Frequency Hop Rate (frequency-hopping [FHSS] systems)	N/A	N/A
Tag:1e	Frequency Hop Sequence tags respond to interrogator signals that satisfy (frequency-hopping [FHSS] systems)	N/A	N/A
Tag:2	Occupied Channel Bandwidth	In accordance with local regulations	N/A
Tag:3	Transmit Maximum Magnetic Field Strength	In accordance with local regulations	N/A
Tag:4	Transmit Spurious Emissions	In accordance with local regulations	N/A
Tag:4a	Transmit Spurious Emissions, In-Band (spread spectrum systems)	In accordance with local regulations	N/A
Tag:4b	Transmit Spurious Emissions, Out-of Band	In accordance with local regulations	N/A
Tag:5	Transmit Spectrum Mask	In accordance with local regulations	N/A
Tag:6	Timing	See below	6.3.3.1.2.5,
Tag:6a	Transmit-to-Receive Turn-Around Time	Less than 151 µs;	6.3.3.1.2.5, Figure 6.3, and Table 6.14
Tag:6b	Receive-to-Transmit Turn-Around Time	75,5 µs nominal ($1024/f_c$)	6.3.3.1.2.5, Figure 6.3, and Table 6.14
Tag:6c	Dwell Time or Transmit Power-On Ramp	Ready to receive commands in less than 1500 µs	6.3.3.1.2.6
Tag:6d	Decay Time or Transmit Power-Down Ramp	N/A	N/A
Tag:7	Modulation	By Load Modulation	6.3.3.1.3.1
Tag:7a	Spreading Sequence (direct sequence [DSSS] systems)	N/A	N/A
Tag:7b	Chip Rate (spread spectrum systems)	N/A	N/A
Tag:7c	Chip Rate Accuracy (spread spectrum systems)	N/A	N/A
Tag:7d	On-Off Ratio	Tag dependent; not specified by this document	N/A

Ref.	Parameter Name	Description	Sub-clause
Tag:7e	Sub-carrier Frequency	ASK Method: 424kHz (f_c/32) or 847kHz (f_c/16) (selected by the interrogator) Optional PJM Method: Channel A 969kHz (divide by 14) Channel B 1233kHz (divide by 11) Channel C 1507kHz (divide by 9) Channel D 1808kHz (divide by 7,5) Channel E 2086kHz (divide by 6,5) Channel F 2465kHz (divide by 5,5) Channel G 2712kHz (divide by 5) Channel H 3013kHz (divide by 4,5) for tag replies	6.3.3.1.3.9, Table 6.9 and optionally 6.3.3.1.3.11, and Table 6.10
Tag:7f	Sub-carrier Frequency Accuracy	Carrier frequency synchronized	6.3.3.1.3.9, Table 6.9
Tag:7g	Sub-carrier Modulation	ASK Method: Manchester or Miller, see Tag: 9 and Table 6.8 Optional PJM Method: BPSK at 106 kbit/s	6.3.3.1.3.5 and optionally 6.3.3.1.3.9
Tag:7h	Duty Cycle	FM0: 50%, nominal Sub-carrier: 50%, nominal	6.3.3.1.3.5
Tag:7I	FM Deviation	N/A	N/A
Tag:8	Data Coding	ASK Method: Baseband FM0 or Manchester or Miller-modulated sub-carrier (selected by the interrogator) Optional PJM Mode: MFM for replies	6.3.3.1.3.2
Tag:9	Bit Rate	ASK Method: FM0, 424kbit/s or 848kbit/s; Sub-carrier modulated, 53kbit/s to 212kbit/s Optional PJM Method: 106 kbit/s on each reply channel	6.3.3.1.3.9, Table 6.9 and optionally 6.3.3.1.3.11
Tag:9a	Bit Rate Accuracy	Same as Sub-carrier Frequency Accuracy; see Tag:7f	N/A
Tag:10	Tag Transmit Modulation Accuracy (frequency-hopping [FHSS] systems	N/A	N/A
Tag:11	Preamble	Required	6.3.3.1.3.2
Tag:11a	Preamble Length	As specified	6.3.3.1.3.2
Tag:11b	Preamble Waveform	As specified	6.3.3.1.3.2, Figure 6.11, Figure 6.16 and optionally Figure 6.23
Tag:11c	Bit-Sync Sequence	None	N/A
Tag:11d	Frame-Sync Sequence	ASK Method: None Optional PJM Method: as specified for replies	N/A and optionally 6.3.3.1.3.10
Tag:12	Scrambling (spread-spectrum systems)	N/A	N/A
Tag:13	Bit Transmission Order	MSB is transmitted first	6.3.3.1.4
Tag:14	Reserved	Purposely left blank	N/A
Tag:15	Polarization	N/A	N/A
Tag:16	Minimum tag Receiver Bandwidth	N/A	N/A

6.3.2 Logical – Operating procedure parameters

Table 6.3 and Table 6.4 identify and describe parameters used by an interrogator during the selection, inventory, and access of tags according to this specification. For those parameters that do not apply to or are not used in this specification, the notation "N/A" shall indicate that the parameter is "Not Applicable".

Table 6.3. Tag inventory and access parameters

Ref.	Parameter Name	Description	Sub-clause
P:1	Who talks first	Interrogator	0
P:2	Tag addressing capability	As specified	6.3.4.1
P:3	Tag EPC	Contained in tag memory	6.3.4.1.2
P:3a	EPC Length	As specified	6.3.4.1.2
P:3b	EPC Format	NSI < 100_h: As specified in EPCglobal Tag Data Standards NSI >= 100_h: As specified in ISO/IEC 15961	6.3.4.1.2.2, 6.3.4.1.2.3, and 6.3.4.1.2.4
P:4	Read size	Multiples of 16-bits	6.3.4.11.3.2, Table 6.34
P:5	Write Size	Multiples of 16-bits	6.3.4.11.3.3, Table 6.37, Figure 6.30, Table 6.48
P:6	Read Transaction Time	Varied with R=>T and T=>R link rate and number of bits being read	6.3.4.11.3.2
P:7	Write Transaction Time	20ms (maximum) after end of *Write* command	6.3.4.11.3.3, Figure 6.33, Figure 6.30
P:8	Error detection	Interrogator-to-tag: CRC-5 or CRC-16 as defined in the relative clauses Tag-to-interrogator: CRC-5 or CRC-16 as defined in the relative clauses	6.3.4.10 and its subsections
P:9	Error correction	None	N/A
P:10	Memory size	Tag dependent	N/A
P:11	Command structure and extensibility	As specified	Table 6.19

Table 6.4. Collision management parameters

Ref.	Parameter Name	Description	Sub-clause
A:1	Type (Probabilistic or Deterministic)	Probabilistic	6.3.4.6
A:2	Linearity	Linear up to 2^{15} tags in the interrogator's RF field	6.3.4.8
A:3	Tag inventory capacity	$>2^{15}$ tags	6.3.4.8

6.3.3 Description of operating procedure

The operating procedure defines the physical and logical requirements for an interrogator-talks-first, random slotted anti-collision, RFID system operating at 13,56MHz frequency.

This specification details two methods of operation:

- ASK Method – (Mandatory).
  The interrogator shall communicate with one or more tags using ASK modulated PIE signalling ASK Method.

A tag shall reply to interrogator ASK commands using the tag to interrogator link modulation specified in clause 6.3.3.1.3.11 Table 6.9 and Table 6.10. The tag shall not change the modulation format and data rate.

- PJM Method – (Optional).
  The interrogator may optionally communicate with one or more tags using PJM-modulated, MFM-encoded signalling PJM Method. Tags that support PJM Method shall reply to interrogator PJM Method commands using the tag to interrogator link modulation specified in clause 6.3.3.1.3.11 and Table 6.11. Tags that do not support PJM Method shall not respond. The tag shall not change the modulation format and data rate.

Both methods use a common memory structure and protocol engine with a shared logical command set.

Note: Reference to specific ASK or PJM functions or commands in this specification are referenced "ASK Method:" or "PJM Method:" as appropriate.

6.3.3.1 Signalling

The signalling interface between an interrogator and a tag may be viewed as the physical layer in a layered network communication system. This interface defines frequencies, modulation, data coding, RF envelope, data rates, and other parameters required for RF communications.

6.3.3.1.1 Operational frequencies

Tags shall be capable of receiving power from and communicating with interrogators at the frequency of 13,56 MHz.

6.3.3.1.2 Interrogator-to-tag (R=>T) communications

ASK Method: An interrogator shall communicate with one or more tags by modulating an RF carrier using ASK with PIE encoding. Interrogators shall use a fixed modulation format and data rate for the duration of an inventory round where “inventory round” is defined in 6.3.4.8.

PJM Method: Interrogators shall communicate to one or more tags by modulating the RF carrier using PJM with MFM encoding at a fixed data rate of 212 kbit/s.

6.3.3.1.2.1 Interrogator frequency accuracy

Interrogator frequency accuracy shall comply with local radio regulations.

6.3.3.1.2.2 Modulation

ASK Method: All interrogators shall communicate using ASK, detailed in Annex H.

PJM Method: Interrogators may support PJM as detailed in Annex G.

6.3.3.1.2.3 Data encoding

ASK Method: The R=>T link shall use PIE encoding, shown in Figure 6.1. Tari is the reference time interval for interrogator-to-tag signalling, and is the duration of a data-0. High values represent transmitted CW; low values represent attenuated CW. The tolerance on all parameters shall be +/–1%, except as otherwise specified..

Pulse modulation depth, rise time, fall time, and PW shall be as specified in Table 6.5, and shall be the same for a data-0 and a data-1. Interrogators shall use a fixed modulation depth, rise time, fall time, PW, and Tari for the duration of an inventory round. The RF envelope shall be as specified in Figure 6.3.

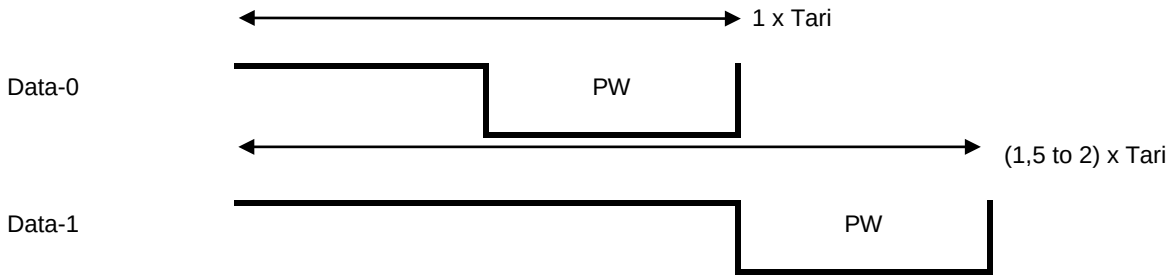


Figure 6.1 ASK Method: PIE symbols

PJM Method: The R=>T link shall use PJM with MFM encoding at 212 kbit/s. The period of a bit interval used for encoding a command is 4,72 us (64 periods of the 13.56 MHz carrier) The rate of phase change is specified in Figure 6.4. Interrogators shall use a fixed phase change value for the duration of an inventory round.

The bit value is defined by a change in state. Bits are encoded using MFM encoding rules. These encoding rules are defined as follows:

- A data-1 is defined by a state change at the middle of a bit interval.
- A data-0 is defined by a state change at the beginning of a bit interval.
- Where a data-0 immediately follows a data-1 there is no state change.

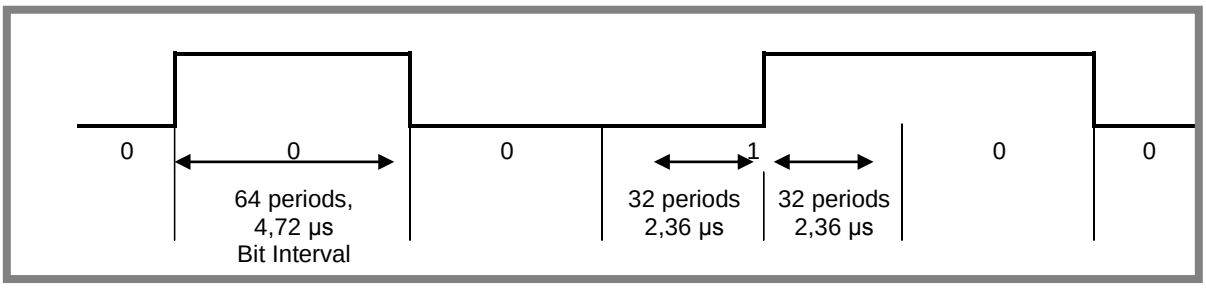


Figure 6.2 PJM Method: Command MFM encoding and timing of binary 000100

An example of command MFM encoding of the command binary string is shown in Figure 6.2. The edges shown represent small (+/- 3deg for example) phase changes. Typically the the edges shown in Figures 6.2 shall be synchronised to the frequency carrier. If not synchronised then these edges will be generated at the Interrogator with a tolerance of +/- one period of the frequency carrier.

6.3.3.1.2.4 ASK Method: Tari values

Interrogators shall communicate using Tari values between 8 µs and 25 µs, inclusive. An interrogator shall use fixed data-0 and data-1 symbol lengths for the duration of an inventory round, where “inventory round” is defined in 6.3.4.8. The choice of Tari value shall be in accordance with local radio regulations.

Note: Interrogator compliance shall be evaluated using at least one Tari value between 8 µs and 25 µs with at least one value of the parameter x=1,5 Tari and x=2,0 Tari.

6.3.3.1.2.5 R=>T RF envelope

ASK Method: The R=>T RF envelope shall comply with Figure 6.3 and Table 6.5. The magnetic field strength A is the maximum amplitude of the RF envelope, measured in A/m, Tari is defined in Figure 6.1. The pulse width is measured at the 50% point on the pulse.

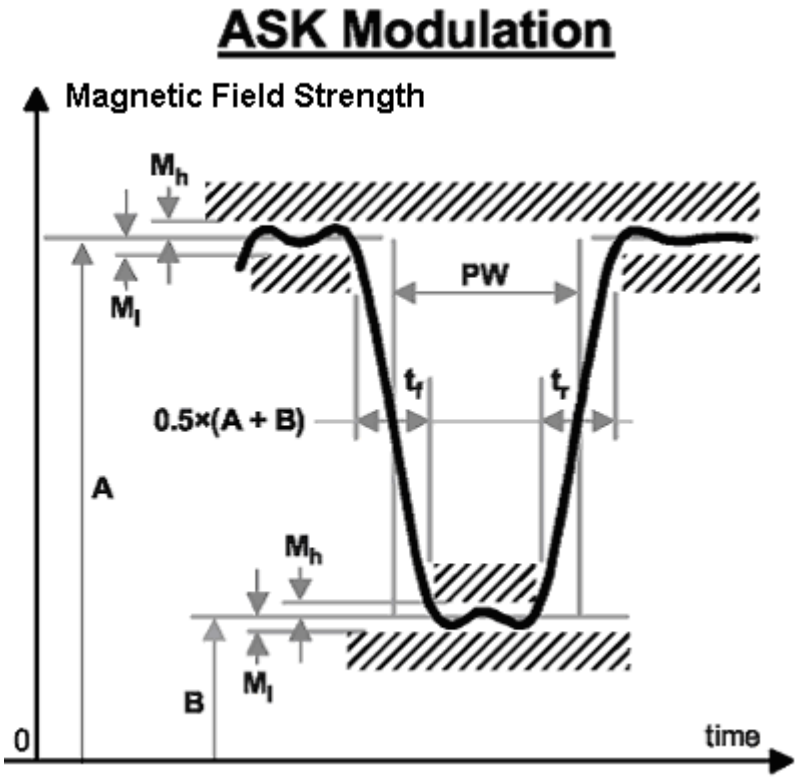


Figure 6.3 ASK Method: Interrogator-to-tag RF envelope

Table 6.5. ASK Method: RF envelope parameters

Tari	Parameter	Symbol	Minimum	Typical	Maximum	Units
8 µs to 25 µs	Modulation Index	(A–B)/(A+B)	10	15	30	%
	RF Envelope Overshoot Ripple	M_h	0		0,1 (A–B)	A/m
	RF Envelope Undershoot Ripple	M_l	0		0,1 (A–B)	A/m
	RF Envelope Rise Time	$t_{r,10–90\%}$	0		MIN(0,33Tari, 4,5)	µs
	RF Envelope Fall Time	$t_{f,10–90\%}$	0		MIN(0,33Tari, 4,5)	µs
	RF Pulsewidth	PW	MAX(0,265Tari, 4)		MIN(0,525Tari, 9,44)	µs

PJM Method: The R=>T link shall use PJM. PJM Method data is transmitted as very small phase changes in the interrogator RF carrier. The PJM Method phase shift waveform of the interrogator magnetic field is described in Figure 6.4 and Table 6.6.

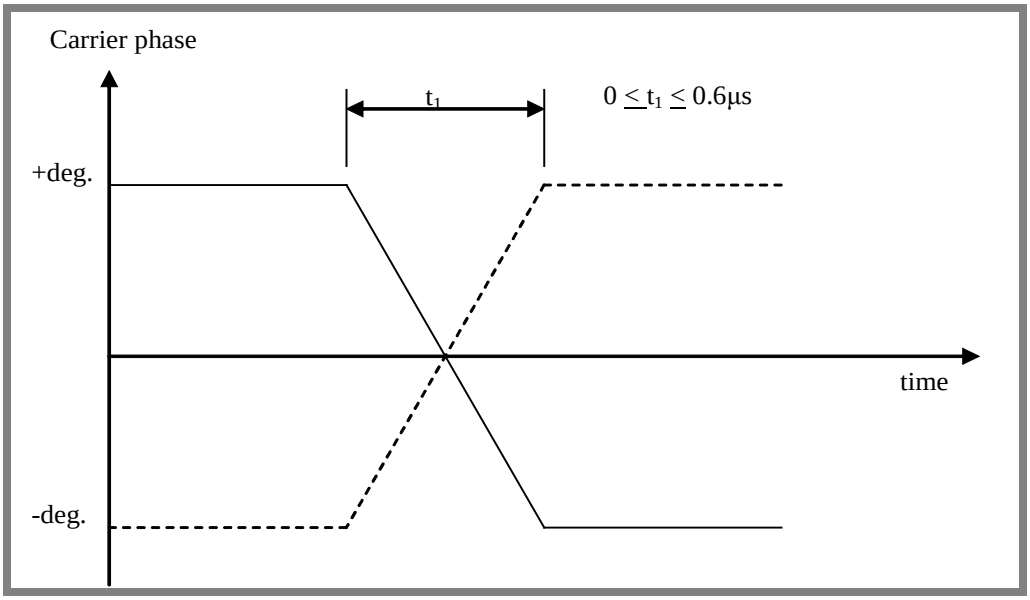


Figure 6.4 PJM Method: Command modulation scheme

Table 6.6. Command modulation parameters

Parameter	Symbol	Nominal	Maximum	Units
Phase Shift	+deg., -deg.	3,0	6,0	deg.
Transition Time	t_1	0,0	0,6	µs

Note: the phase shift cannot exceed the final phase value at any time and the transition time (t_1) is the time allowed to reach 95% of the full phase change.

6.3.3.1.2.6 Interrogator power-up waveform

The interrogator power-up RF envelope shall comply with Figure 6.5 and Table 6.7. Once the carrier level has risen above the 10% level, the power-up envelope shall rise monotonically until at least the ripple limit M_l. The RF envelope shall not fall below the 90% point in Figure 6.5 or rise above the 110% point during interval T_s and after interval Ts shall not fall below 99% or rise above 101%. Interrogators shall not issue commands before the end of the maximum settling-time interval in Table 6.7 (*i.e.* before Ts). Interrogators shall meet the frequency-accuracy requirement specified in 6.3.3.1.2.1 by the end of interval T_s in Figure 6.5.

Note: When specifying compliance test there must be no moving tag in the field.

6.3.3.1.2.7 Interrogator power-down waveform

The interrogator power-down RF envelope shall comply with Figure 6.5 and Table 6.8. Once the carrier level has fallen below the 99% level, the power-down envelope shall fall monotonically until the power-off limit M_s. Once powered off, an interrogator shall remain powered off for at least 1ms before powering up again.

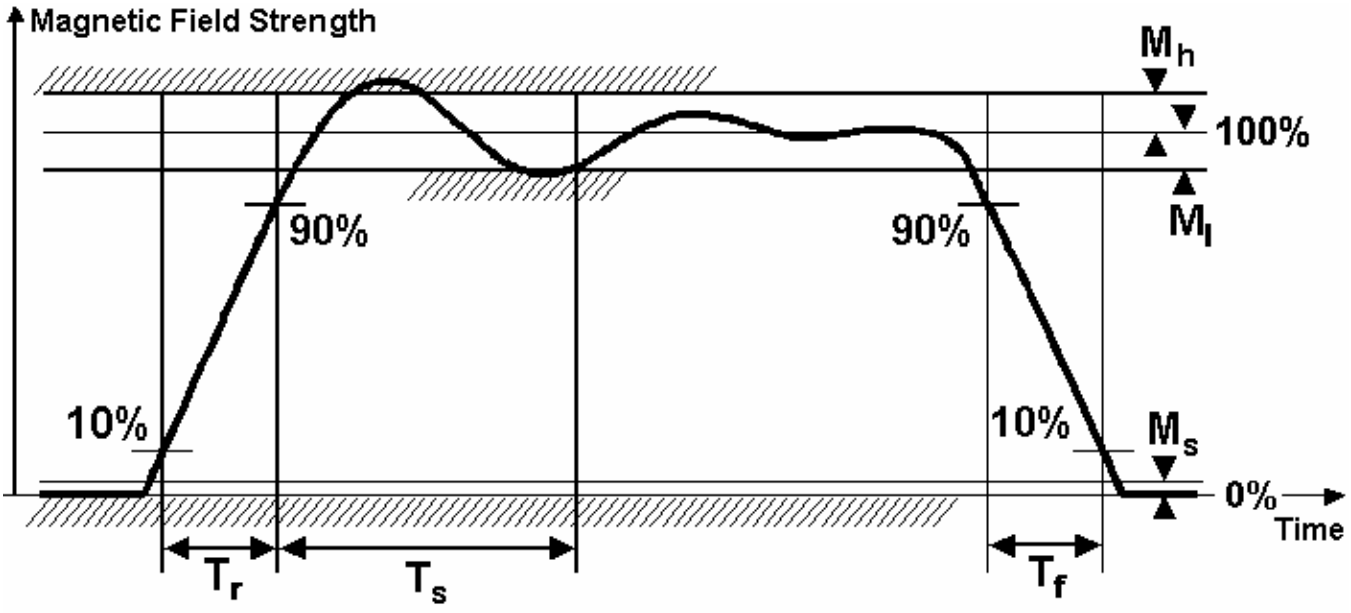


Figure 6.5 Interrogator power-up and power-down RF envelope

Table 6.7. Interrogator power-up waveform parameters

Parameter	Definition	Minimum	Typical	Maximum	Units
T_r	Rise time	1		500	µs
T_s	Settling time			1500	µs
M_s	Signal level when OFF			0.1	% full scale
M_l	Undershoot			10	% full scale
M_h	Overshoot			10	% full scale

Table 6.8. Interrogator power-down waveform parameters

Parameter	Definition	Minimum	Typical	Maximum	Units
T_t	Fall time	1		500	µs
M_s	Signal level when OFF			0.1	% full scale

6.3.3.1.2.8 R=>T preamble and frame-sync

ASK Method: An interrogator shall begin all R=>T signalling with either a preamble or a frame-sync, both of which are shown in Figure 6.6. A preamble shall precede a *BeginRound* command (see 6.3.4.11.2.1) and denotes the start of an inventory round. All other signalling shall begin with a frame-sync. The tolerance on all parameters specified in units of Tari shall be +/–1%. PW shall be as specified in Table 6.5. The RF envelope shall be as specified in Figure 6.3. A tag may compare the length of the data-0 with the length of RTcal to validate the preamble.

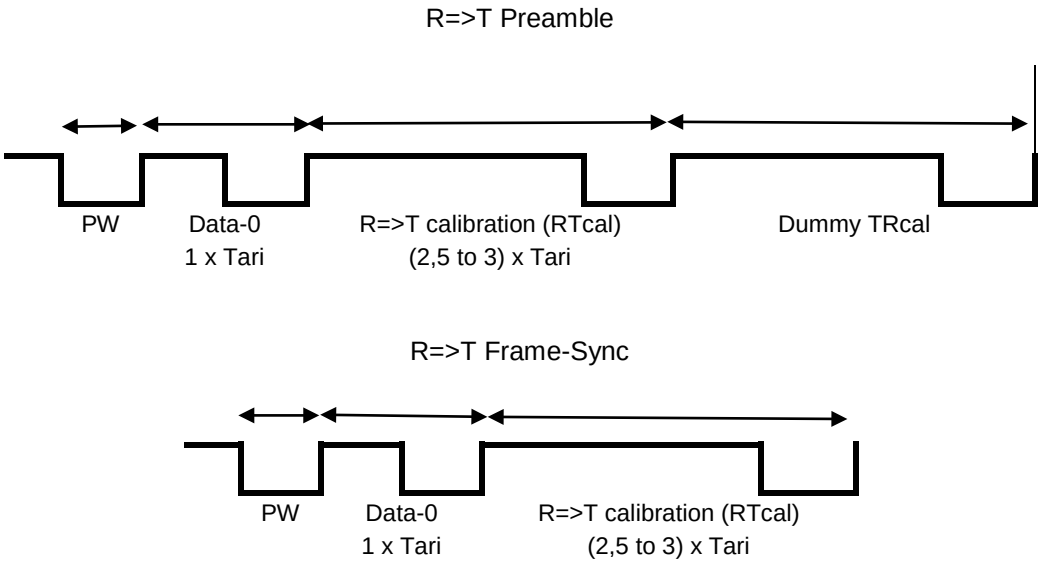


Figure 6.6 ASK Method: R=>T preamble and frame-sync

A preamble shall comprise a modulation with the same length as used in the following data-0 symbol, a data-0 symbol, an R=>T calibration (RTcal) symbol, and a dummy T=>R calibration (TRcal) symbol.

- RTcal: An interrogator shall set RTcal equal to the length of a data-0 symbol plus the length of a data-1 symbol (RTcal=data-0 symbol length + data-1 symbol length). A tag shall measure the length of RTcal and compute pivot=RTcal / 2. The tag shall interpret subsequent interrogator symbols shorter than pivot to be data-0s, and subsequent interrogator symbols longer than pivot to be data-1s. The tag shall interpret symbols longer than 4 RTcal to be invalid. Prior to changing RTcal, an interrogator shall transmit CW for a minimum of 8 RTcal.
- TRcal: For the HF protocol, the TRcal is not used to define the T=>R return link rate. The TRcal value shall be between 1,1*RTcal and 3*RTcal (1.1*RTcal<=TRcal<=3*RTcal).
- A frame-sync is identical to a preamble, minus the TRcal symbol. An interrogator, for the duration of an inventory round, shall use the same length RTcal in a frame-sync as it used in the preamble that initiated the round.

PJM Method: The R=>T link shall use PJM with MFM encoding. An interrogator shall begin all R=>T signalling with a MFM flag (or frame sync). The flag defines the start of a command and the bit interval timings. The flag comprises three parts:

- A synchronising string of 9 bits of valid MFM encoded data. “As an example Figure 6.7 shows 9 data bits with the value "1"
- A MFM encoding violation not present in normal data. The violation consists of a sequence of 4 state changes separated by a 2 bit interval, a 1,5 bit interval and 2 bit interval. The edge of the fourth transition defines the beginning of a bit interval.
- A trailing 0 defining the end of a flag.

The synchronising string, encoding violation and a trailing zero for two possible command flags are illustrated in Figure 6.7.

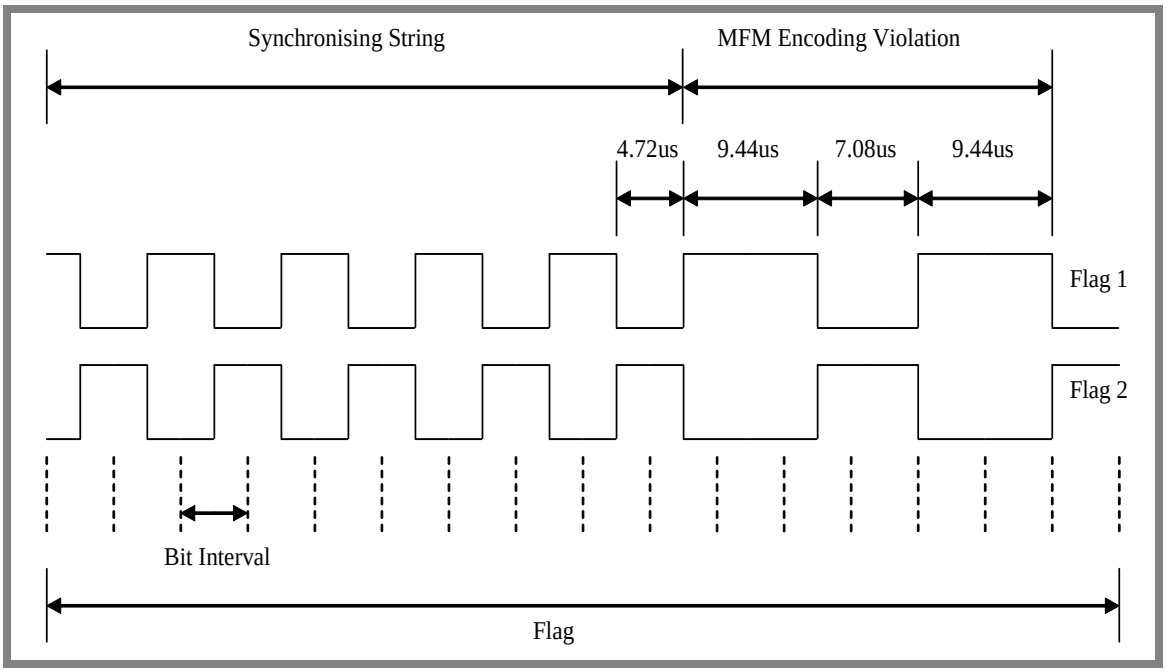


Figure 6.7 PJM Method: MFM Encoding and timing for two possible command flags

6.3.3.1.3 Tag-to-interrogator (T=>R) communications

The tag communicates with an interrogator using loadmodulation. Loadmodulation is a method of impressing a data signal onto the carrier wave by changing the electrical "load" (impedance) of the tag. A tag is able to transmit data by varying its "load", thus effecting voltage changes at the interrogator antenna. The interrogator can then translate the voltage changes in a binary signal.

Load modulation can be achieved in several ways, e.g. by switching:

- load resistors
- shunt diodes.
- the antenna tuning capacitance value or taps on the antenna coil (adjusting the tag tuning). It is possible to consider the switched tuning capacitance as part of the antenna and therefore the load seen by the tag antenna during the reply is fixed. Likewise the tuning capacitor and chip load presented to a tapped coil are constant.

ASK Method: A tag shall loadmodulate using a fixed modulation format, data encoding, and data rate for the duration of an inventory round, where “inventory round” is defined in 6.3.3.4.8. The interrogator sets the encoding and data rate by means of the BeginRound command that initiates the round. From this BeginRound command, the tag selects the T=>R modulation format.

PJM Method: The reply data rate is 106 kbit/s encoded using MFM and modulated onto the sub-carrier as BPSK. Tags can select from one of eight sub-carrier frequencies between 969 kHz and 3013 kHz. The sub-carrier is derived from division of the interrogator operating frequency. During an inventory round a tag shall loadmodulate using a fixed sub-carrier frequency as described in PJM Method sub clause of 6.3.3.1.3.10. The interrogator sets the reply channel by means of the BeginRound command that initiates the round.

6.3.3.1.3.1 Modulation

For subsequent figures, the low values correspond to the tag antenna having a load impedance equal to the load impedance during the CW period immediately before a data transmission, whereas the high values correspond to the tag antenna having a different load impedance to the CW period.

ASK Method: The tag reply waveform may be baseband (for FM0 encoding) or modulated sub-carrier (for Miller or Manchester encoding) as specified in clause 6.3.3.1.3.2 to 6.3.3.1.3.10.

PJM Method: The tag reply waveform is BPSK modulated sub-carrier.

In order to ensure that tags replying on different channels are simultaneously received, all tag replies shall be band limited to reduce data and sub-carrier harmonic levels as shown in Figure 6.8. Intermediate load modulation impedances can be used for band limiting purposes.

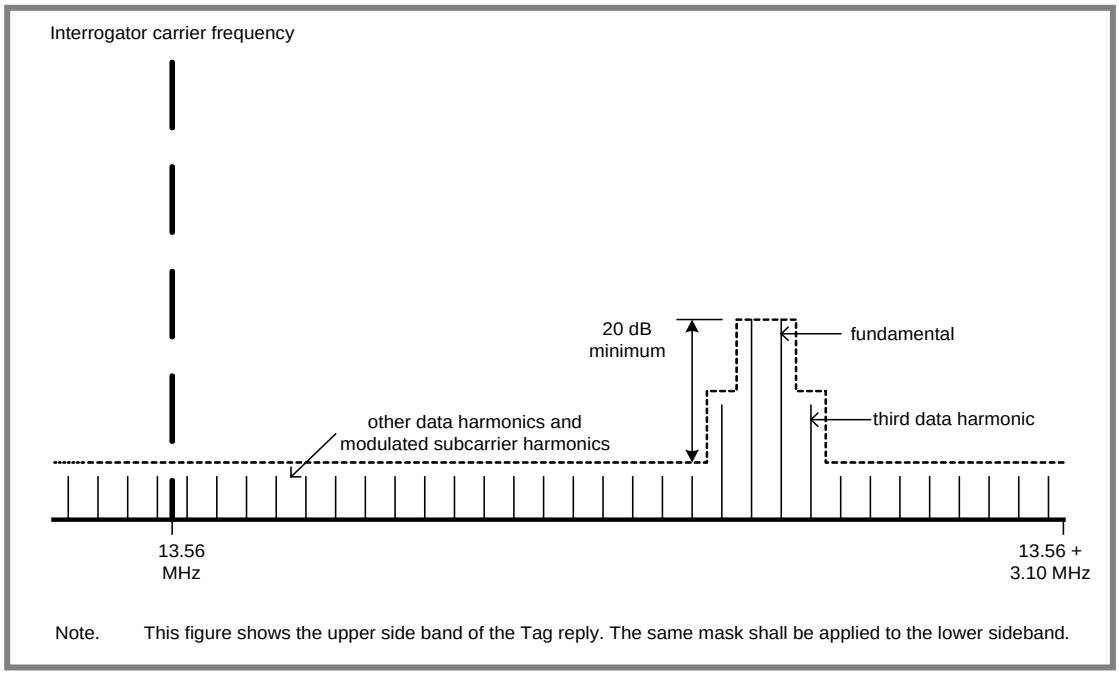


Figure 6.8 PJM Method: Tag reply mask

6.3.3.1.3.2 Data encoding

ASK Method: Tag data shall encode the loadmodulated reply as either FM0 baseband, or as Manchester-encoded sub-carrier, or as Miller-encoded sub-carrier. The interrogator sets the choice of encoding and data rate by parameters in the *BeginRound* command.

PJM Method: Tag data shall encode the loadmodulated reply as MFM of one of eight sub-carriers of different frequencies at a data rate of 106 kbit/s. The selection of sub-carrier frequency is either randomly selected by the tag, or directly selected by parameters in the *BeginRound* command.

6.3.3.1.3.3 ASK Method: FM0 baseband

Figure 6.9 shows basis functions for generating FM0 (bi-phase space) encoding. FM0 inverts the baseband phase at every symbol boundary; a data-0 has an additional mid-symbol phase inversion.

Figure 6.10 shows generated baseband FM0 symbols and sequences. The duty cycle of a 00 or 11 sequence, measured at the modulator output, shall be 50%. FM0 encoding has memory; consequently, the choice of FM0 sequences in Figure 6.10 depends on prior transmissions. FM0 signalling shall always end with a “dedicated EOF at the end of a transmission, as shown in Figure 6.12.

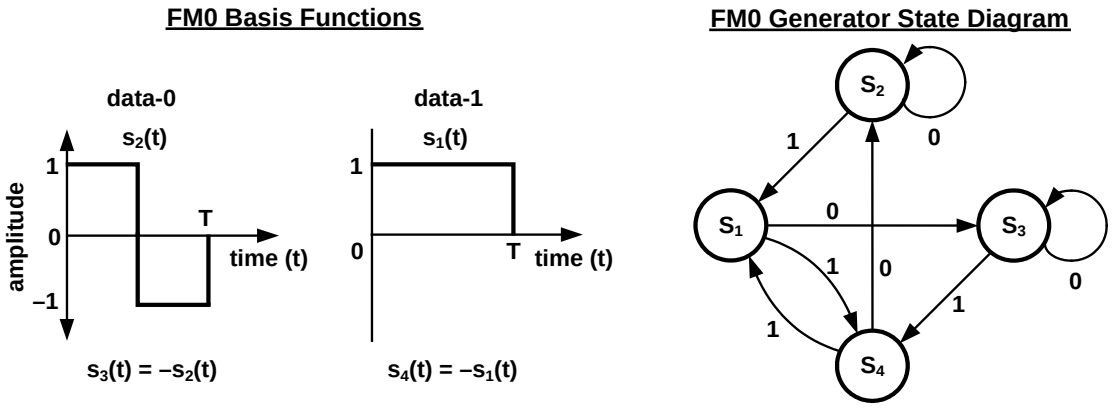


Figure 6.9 ASK Method: FM0 basis functions and generator state diagram

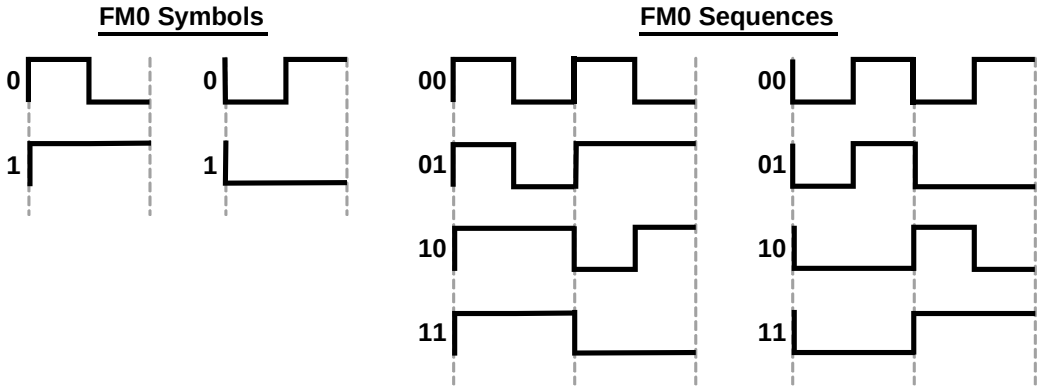


Figure 6.10 ASK Method: FM0 symbols and sequences

6.3.3.1.3.4 ASK Method: FM0 Preamble SOF and EOF

T=>R FM0 signalling shall begin with one of the two preambles shown in Figure 6.11. The preamble comprises four data-0 followed by an FM0-violation of one bit duration, and ending with one data-0. The choice depends on the value of the TRext bit specified in the *BeginRound* command that initiated the inventory round. In Figure 6.11 the pilot tone for TRext=1 consists of additional 12 leading 0s. All tag answers are finalized by an EOF shown in Figure 6.12.

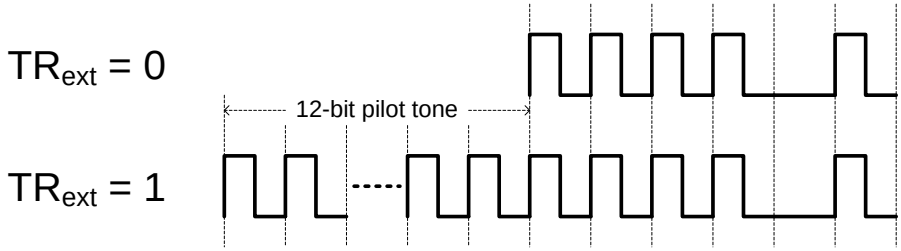


Figure 6.11 ASK Method: FM0 Preamble SOF

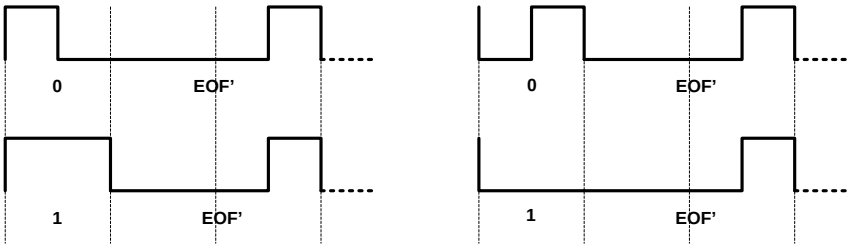


Figure 6.12 ASK Method: Terminating FM0 transmissions EOF

6.3.3.1.3.5 ASK Method: Manchester-modulated sub-carrier

Figure 6.13 shows basis functions for generating Manchester encoding.

Manchester encoding uses a high to low transition for data-0 and a low to high transition for data-1.

Manchester-modulated sub-carrier provides sub-carrier pulses followed by un-modulated time to encode data-0. Data-1 is encoded by un-modulated time followed by sub-carrier pulses.

Figure 6.14 shows Manchester-modulated sub-carrier sequences; the Manchester sequence shall contain exactly four, or eight cycles per bit, depending on the M value specified in the *BeginRound* command that initiated the inventory round (see Table 6.9). The two possible values of the sub-carrier frequency (fc/32 and fc/16) are also defined in the BeginRound command by the DR bit.

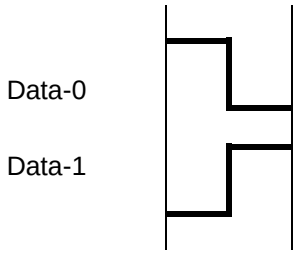


Figure 6.13 ASK Method: Manchester basis functions

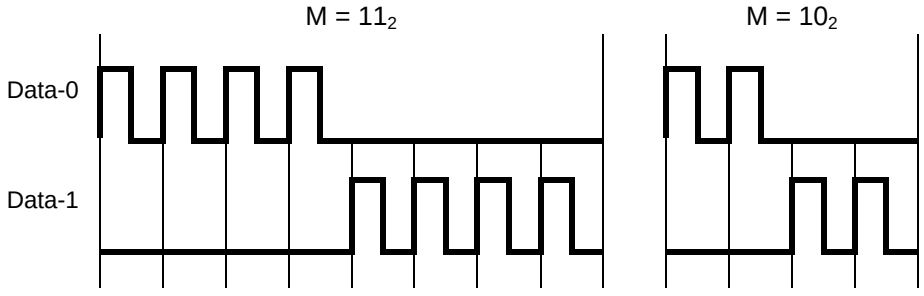


Figure 6.14 ASK Method: Manchester sub-carrier sequences

6.3.3.1.3.6 ASK Method: Manchester sub-carrier preamble SOF and EOF

T=>R sub-carrier signalling shall begin with one of the two preambles shown in Figure 6.15. They contain exactly six (or twelve) sub-carrier pulses followed by an un-modulated time of four (or eight) sub-carrier cycles (that corresponds to a Manchester violation) and ended by a bit "0". The choice depends on the value of the TRext bit specified in the *BeginRound* command that initiated the inventory round. The pilot tone for TRext=1 consists of additional

continuous cycles at the sub-carrier frequency for the duration of 12 bits.. All tag answers are finalized by an EOF shown in Figure 6.16. The EOF comprises a bit "1" followed by a Manchester violation as a group of four (or eight) sub-carrier pulses.

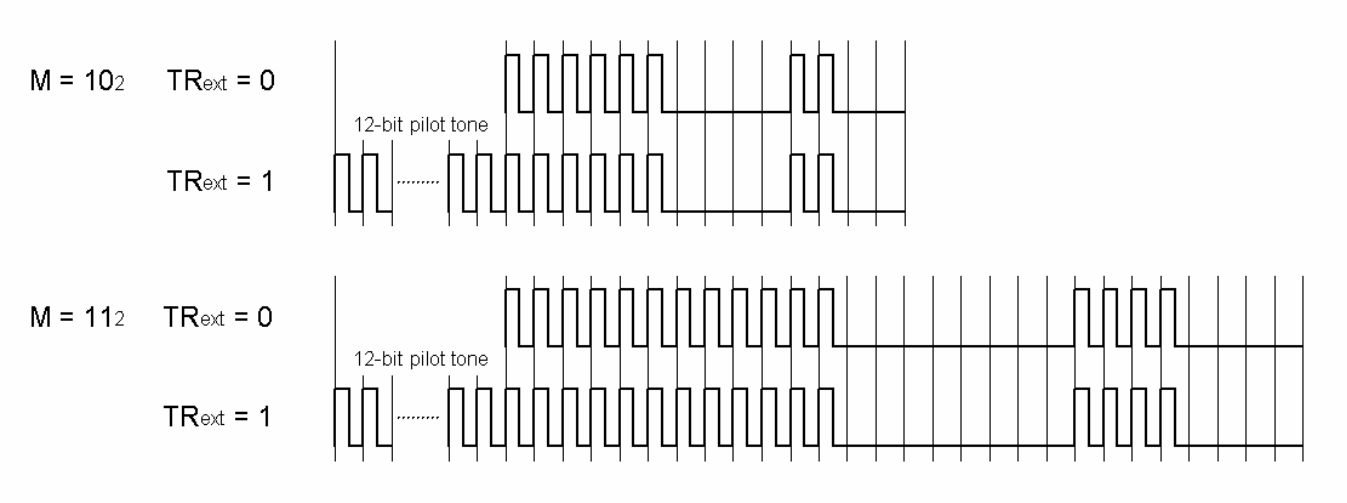


Figure 6.15 ASK Method: Sub-carrier T=>R preamble SOF

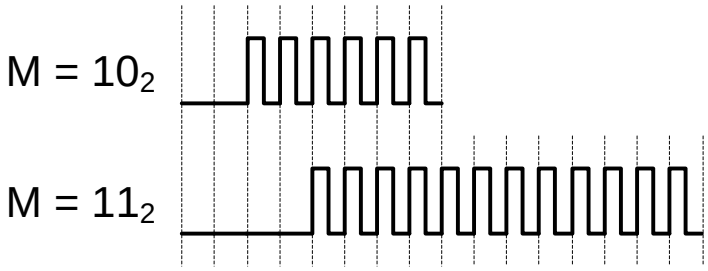


Figure 6.16 ASK Method: Terminating sub-carrier transmissions EOF

6.3.3.1.3.7 ASK Method: Miller-modulated sub-carrier

Figure 6.17 shows basis functions for generating Miller encoding. Baseband Miller inverts its phase between two data-0s in sequence. Baseband Miller also places a phase inversion in the middle of a data-1 symbol. The state diagram in Figure 6.17 maps a logical data sequence to baseband Miller basis functions. The state labels, S_1–S_4, indicate four possible Miller-encoded symbols, represented by the two phases of each of the Miller basis functions. The state labels also represent the baseband Miller waveform that is generated upon entering the state. The transmitted waveform is the baseband waveform multiplied by a square-wave at 8 times the symbol rate. The labels on the state transitions indicate the logical values of the data sequence to be encoded. For example, a transition from state S_1 to S_3 is disallowed because the resulting transmission would have a phase inversion on a symbol boundary between a data-0 and a data-1.

Figure 6.18 shows a Miller-modulated sub-carrier sequence; the Miller sequence shall contain exactly eight sub-carrier cycles per bit (see Table 6.9). Miller encoding has memory; consequently, the choice of Miller sub-carrier sequences as shown in Figure 6.18 depends on prior transmissions.

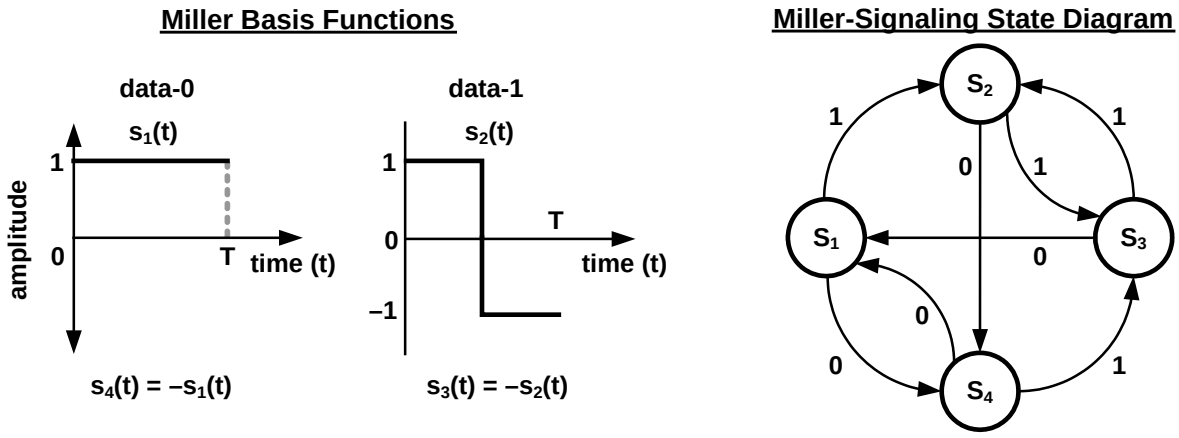


Figure 6.17 ASK Method: Miller basis functions and generator state diagram

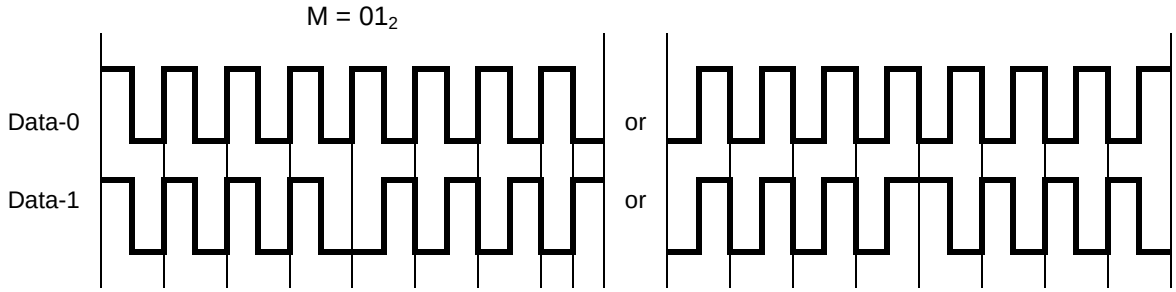


Figure 6.18 ASK Method: Miller sub-carrier sequences

6.3.3.1.3.8 ASK Method: Miller sub-carrier preamble SOF and EOF

T=>R sub-carrier signalling shall begin with one of the two preambles shown in Figure 6.19. The choice depends on the value of the TRext bit specified in the *BeginRound* command that initiated the inventory round. Figure 6.19 shows the preamble for each TRext. This pilot tone consists of continuous cycles at the sub-carrier frequency. Miller encoded communication shall always end with dedicated EOF at the end of a transmission, as shown in Figure 6.20.

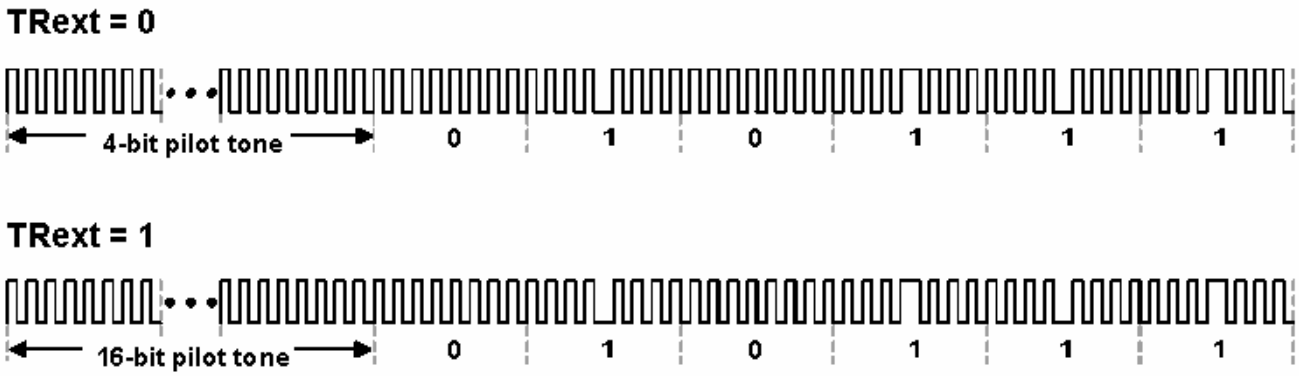


Figure 6.19 ASK Method: Miller Sub-carrier T=>R preamble SOF

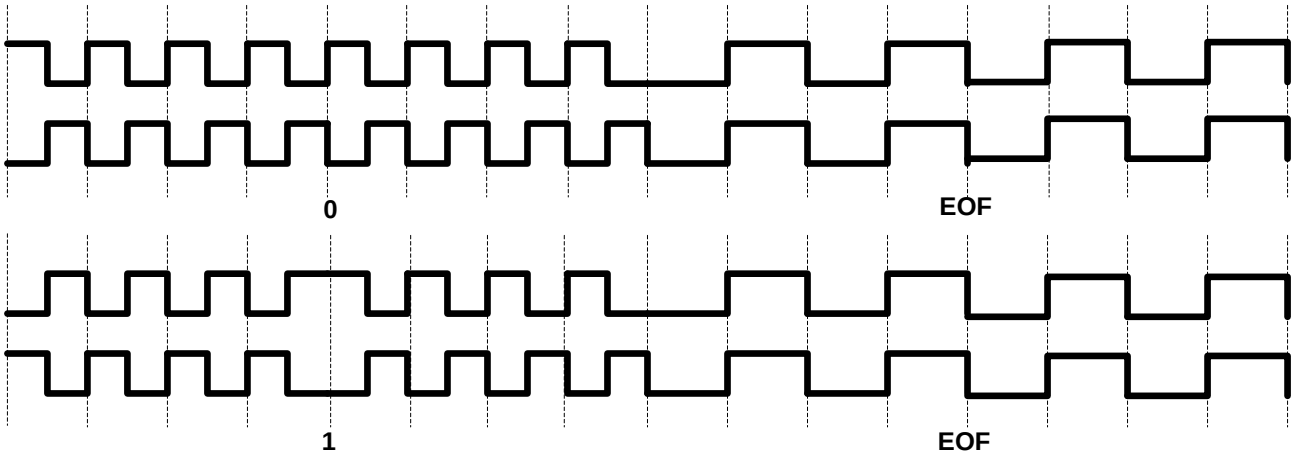


Figure 6.20 ASK Method: Terminating sub-carrier transmissions EOF

6.3.3.1.3.9 PJM Method: MFM modulated sub-carrier

Figure 6.21 shows basis functions for generating MFM encoding. Baseband MFM inverts its phase between two data-0s in sequence. Baseband MFM also places a phase inversion in the middle of a data-1 symbol. The state diagram in Figure 6.21 maps a logical data sequence to baseband MFM basis functions. The state labels, S_1–S_4, indicate four possible MFM-encoded symbols, represented by the two phases of each of the MFM basis functions. The state labels also represent the baseband MFM waveform that is generated upon entering the state. The transmitted waveform is the baseband waveform multiplied by a square wave at f_c divided by n, where n is the division ratio for the 8 sub-carriers defined in Table 6.11. The labels on the state transitions indicate the logical values of the data sequence to be encoded. For example, a transition from state S_1 to S_3 is disallowed because the resulting transmission would have a phase inversion on a symbol boundary between a data-0 and a data-1.

Figure 6.22 shows an example for the BPSK phase changes without sub-carrier and the corresponding MFM-modulated sub-carrier sequence. The phase changes align with MFM state changes shown in Figure 6.21. For clarity the sub-carrier example shows eight cycles per bit period, however the real sub-carrier will have between nine and 28 cycles depending upon the reply channel selected. Also for clarity the phase change shown occurs at the beginning and the middle of a sub-carrier cycle; however the phase change can occur at any point during a sub-carrier cycle because the reply data rate and the sub-carriers are unsynchronized.

The sub-carrier is derived by division of the powering field's frequency and is determined from the DR, M and TRext values specified in the *BeginRound* command that initiated the inventory round (see Table 6.11). The period T of a bit interval used for encoding command data is 9,44µs. MFM encoding has memory; consequently, the choice of MFM sequences in Figure 6.22 depends on prior transmissions.

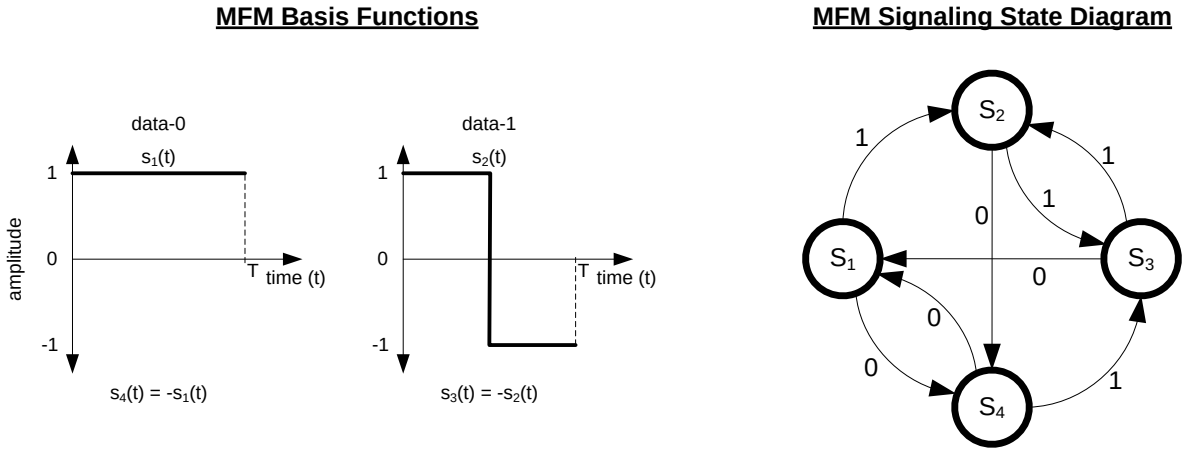


Figure 6.21 PJM Method: MFM basis functions and generator state diagram

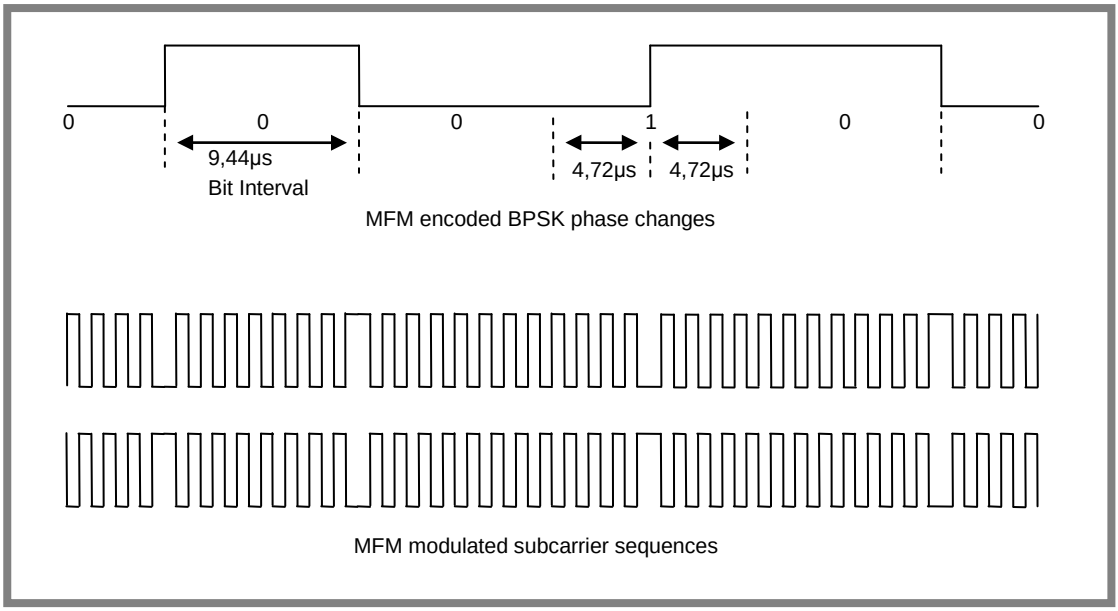


Figure 6.22 PJM Method: Reply MFM encoding and timing of binary 000100

6.3.3.1.3.10 PJM Method: MFM sub-carrier flag (preamble SOF)

PJM Method: The T=>R sub-carrier signalling shall begin with either of the two flags shown in Figure 6.23. This figure shows the flag timing for the BPSK sub-carrier phase changes.

The flag defines the start of a reply and the bit interval timings. The flag is comprised of three parts:

1. A synchronising string of 9 bits of valid MFM data. “As an example Figure 6.23 shows 9 data bits with the value "1".
2. A MFM encoding violation which is not present in normal data. The violation consists of a sequence of 4 state changes separated by a 2 bit interval, a 1,5 bit interval and 2 bit interval. The edge of the fourth transition defines the beginning of a bit interval, and
3. A trailing 0.

A synchronising string, encoding violation and a trailing zero for two possible reply flags are illustrated in Figure 6.23.

Note: an EOF is NOT required

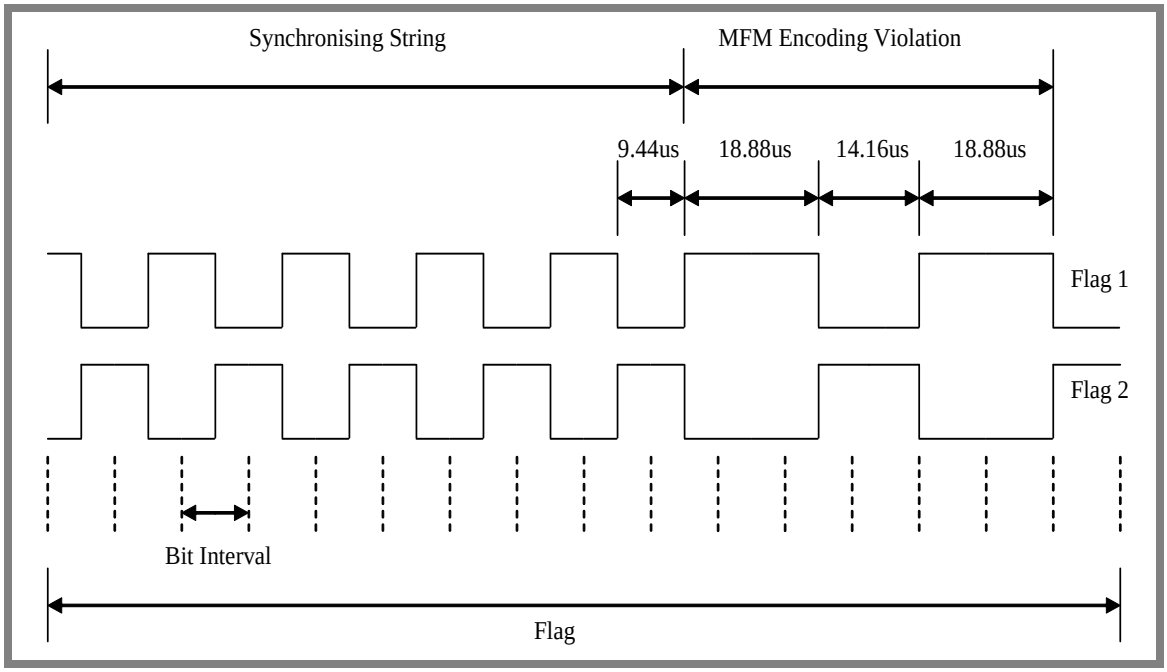


Figure 6.23 PJM Method: Sub-carrier T=>R flags

6.3.3.1.3.11 ASK Method and PJM Method: Selection of modulation type and data rate

- ASK Method: Tags shall support the R=>T Tari values specified in 6.3.3.1.2.4; the T=>R link frequencies specified in Table 6.9; and the T=>R data rates specified in Table 6.10. The *BeginRound* command that initiates an inventory round specifies DR in Table 6.9 and M in Table 6.10.

Table 6.9. ASK Method: Tag-to-interrogator link frequencies

M value	Bit modulation	Subcarrier with DR=0 LF = 424kHz (f_c/32)	Subcarrier with DR=1 LF = 847kHz (f_c/16)
11_2	4 subcarrier pulse Manchester	53 kbit/s (f_c/256)	106 kbit/s (f_c/128)
10_2	2 subcarrier pulse Manchester	106 kbit/s (f_c/128)	212 kbit/s (f_c/64)
01_2	8 subcarrier pulse Miller	53 kbit/s (f_c/256)	106 kbit/s (f_c/128)
00_2	FM0	424kbit/s (f_c/32)	848kbit/s (f_c/16)

Table 6.10. ASK Method: Tag-to-interrogator data rates

M: Number of subcarrier cycles per symbol	Modulation type	Data rate (kbit/s)
M = 00_2: 1 cycles per symbol	FM0 baseband	LF
M = 01_2: 8 cycles per symbol	Miller subcarrier	LF/8
M = 10_2: 4 cycles per symbol	Manchester subcarrier	LF/4
M = 11_2: 8 cycles per symbol	Manchester subcarrier	LF/8

- PJM Method: The T=>R modulation shall be MFM at a data rate of 106 kbit/s on a sub-carrier selected by the *BeginRound* command that initiates an inventory round The *BeginRound* command provides the DR, M, and TRext bits which are (for PJM Method commands) decoded by capable tags as shown in Table 6.11.

Table 6.11. PJM Method: Sub-carrier selection commands

DR	M	TRext	reply channel selected	division ratio	subcarrier frequency kHz
0	00	0	A	14	969
0	00	1	B	11	1233
0	01	0	C	9	1507
0	01	1	D	7,5	1808
0	10	0	E	6,5	2086
0	10	1	F	5,5	2465
0	11	0	G	5	2712
0	11	1	H	4,5	3013
1	00	0	Subcarrier frequency hopping		
all other states			reserved for future use		

When frequency hopping is selected by the *BeginRound* command tags shall select a reply channel. A tag shall use this reply channel for all T=>R communications. The channel selected by a tag shall be determined from the three least significant bits of the StoredCRC. The three least significant bits of the StoredCRC shall be decoded as for the M and TRext bits in Table 6.11 above. For example, if the three least significant bits of the StoredCRC are 110 then channel G would be selected

6.3.3.1.4 Both Methods: Transmission order

The transmission order for all R=>T and T=>R communications shall respect the following conventions:

- within each message, the most-significant word shall be transmitted first; and
- within each word, the most-significant bit (MSB) shall be transmitted first.

6.3.3.1.5 Both Methods: Cyclic-redundancy check (CRC)

A CRC is a cyclic-redundancy check that a Tag uses to ensure the validity of certain R=>T commands, and an Interrogator uses to ensure the validity of certain loadmodulated T=>R replies. This protocol uses two CRC types: (i) a CRC-16, and (ii) a CRC-5. Annex F describes both CRC types.

To generate a CRC-16 a Tag or Interrogator shall first generate the CRC-16 precursor shown in Table 6.12, and then take the ones-complement of the generated precursor to form the CRC-16.

Table 6.12. CRC-16 precursor

CRC Type	Length	Polynomial	Preset	Residue
ISO/IEC 13239	16 bits	$x^{16} + x^{12} + x^5 + 1$	$FFFF_h$	$1D0F_h$

A Tag or Interrogator shall verify the integrity of a received message that uses a CRC-16. The Tag or Interrogator may use one of the methods described in Annex F to verify the CRC-16.

At power-up, a Tag calculates and saves into memory a 16-bit StoredCRC — see 6.3.4.1.2.1.

Tags shall append a CRC-16 to those replies that use a CRC-16 — see 6.3.4.11 for command-specific reply formats.

To generate a CRC-5 an Interrogator shall use the definition in Table 6.13.

Table 6.13. CRC-5 definition. See also Annex F

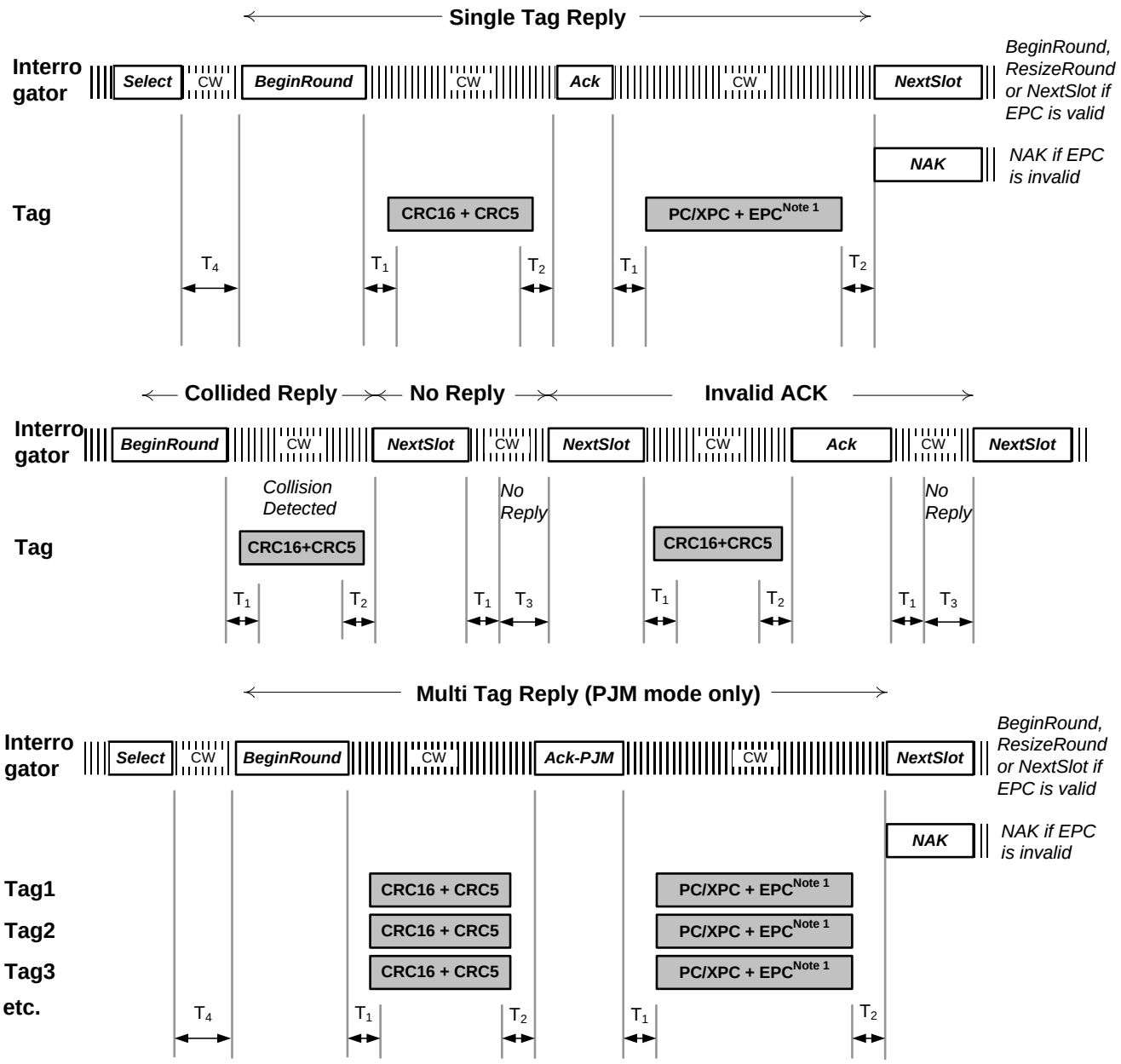
CRC Type	Length	Polynomial	Preset	Residue
—	5 bits	$x^5 + x^3 + 1$	01001_2	00000_2

A Tag shall verify the integrity of a received message that uses a CRC-5. The Tag may use the method described in Annex F to verify a CRC-5.

Interrogators shall append the appropriate CRC to R=>T transmissions as specified in Table 6.19.

6.3.3.1.6 Link timing – Both Methods

Figure 6.24 illustrates R=>T and T=>R link timing. Figure 6.24 (not drawn to scale) defines interrogator interactions with a tag population. Table 6.14 shows the timing requirements for Figure 6.24, while 6.3.4.11 describes the commands. Tags and interrogators shall meet all timing requirements shown in Table 6.14. RTcal is defined in 6.3.3.1.2.8. As described in 6.3.3.1.2.8, an interrogator shall use a fixed R=>T link rate for the duration of an inventory round; prior to changing the R=>T link rate, an interrogator shall transmit CW for a minimum of 8 RTcal.



Note 1: In certain cases a PacketCRC shall be added if required. See Table 6.15 for the definition of the cases.

Figure 6.24 Link timing – Both Methods

Table 6.14. Link timing parameters

Parameter	Minimum	Typical	Maximum	Description
T_1	73,1 µs (1024-32)/f_c	75,5 µs (1024/f_c)	77,9 µs (1024+32)/f_c	Time from interrogator transmission to tag response (specifically, the time from the last rising edge of the last bit of the interrogator transmission to the first rising edge of the tag response), measured at the tag antenna terminals.
T_2	151 µs (2048/f_c)		1208 µs (16384/f_c)	Interrogator response time required if a tag is to demodulate the Interrogator signal, measured from the end of the tag response to the first falling edge of the Interrogator transmission
T_3	Tsof_tag			Time an interrogator waits, after T_1, before it issues another command
T_4	T_{1Typ} + T_{3Min}			Minimum time between interrogator commands

Note 1: A tag may exceed the maximum value of T_1 when responding to commands that write to memory.

Note 2: The maximum value for T_2 shall apply only to tags in the **reply** or **acknowledged** states (see 6.3.4.4.3 and 6.3.4.4.4). For a tag in the **reply** or **acknowledged** states, if T_2 expires (*i.e.* reaches its maximum value):
- Without the tag receiving a valid command, the tag shall transition to the **arbitrate** state (see 6.3.4.4.2);
- During the reception of a valid command, the tag shall execute the command;
- During the reception of an invalid command, the tag shall transition to **arbitrate** upon determining that the command is invalid.

In all other states the maximum value for T_2 shall be unrestricted. "Invalid command" is defined in 6.3.4.11.

Note 3: A Tag shall be allowed a tolerance of $16384/f_c <= T_{2max} <= 16448/f_c$ in determining whether T_2 has expired.

6.3.4 Tag selection, inventory, and access

Tag selection, inventory, and access may be viewed as the lowest level in the data link layer of a layered network communication system.

6.3.4.1 Tag memory

Tag memory shall be logically separated into four distinct banks, each of which may comprise zero or more memory words. A logical memory map is shown in Figure 6.25.

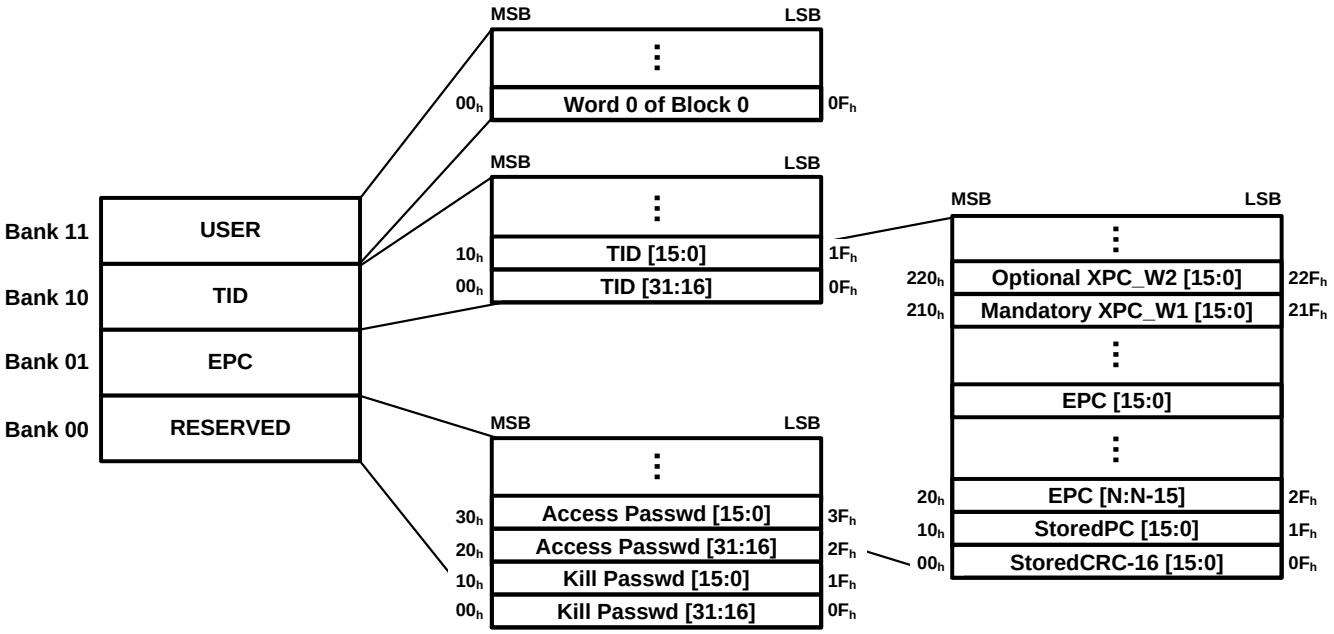


Figure 6.25 Logical memory map

The memory banks are:

Reserved memory shall contain the kill and and/or access passwords, if passwords are implemented on the Tag. The kill password shall be stored at memory addresses 00_h to $1F_h$; the access password shall be stored at memory addresses 20_h to $3F_h$. See 6.3.4.1.1.

EPC memory shall contain a CRC-16 at memory addresses 00_h to $0F_h$; a Protocol-Control (PC) word at addresses 10_h to $1F_h$; a code (such as an EPC, and hereafter referred to as an EPC) that identifies the object to which the Tag is or will be attached beginning at address 20_h; and either one (mandatory) or two (optional) Extended Protocol Control (XPC) word(s) beginning at address 210_h. See 6.3.4.1.2.

TID memory shall contain an 8-bit ISO/IEC 15963 allocation class identifier at memory locations 00_h to 07_h. TID memory shall contain sufficient identifying information above 07_h for an Interrogator to uniquely identify the custom commands and/or optional features that a Tag supports. See 6.3.4.1.3.

User memory is optional. See 6.3.4.1.4.

The logical addressing of all memory banks shall begin at zero (00_h). The physical memory map is vendor-specific. Commands that access memory have a MemBank parameter that selects the bank, an address length parameter and an address parameter to select a particular memory location within that bank. The address parameter has a length of 8, 16, 24 or 32 bits as defined in the address length parameter. Interrogators shall use the shortest address length possible to encode the memory location. When Tags loadmodulate memory contents, this modulation shall fall on word boundaries (except in the case of a truncated reply – see 6.3.4.11.1.1).

MemBank is defined as follows:

- 00_2 Reserved
- 01_2 EPC
- 10_2 TID
- 11_2 User

Operations in one logical memory bank shall not access memory locations in another bank.

Memory writes, detailed in 6.3.4.9, involve the transfer of 16-bit words from Interrogator to Tag. A Write command writes 16 bits (*i.e.* one word) at a time, optionally using link cover-coding to obscure the data during R=>T transmission. The optional BlockWrite command writes one or more 16-bit words at a time, without link cover-coding. The optional BlockErase command erases one or more 16-bit words at a time. A Write, BlockWrite, or BlockErase shall not alter a killed Tag's permanently non-responsive status regardless of the memory address (whether valid or invalid) specified in the command.

Interrogators may lock, permanently lock, unlock, or permanently unlock the kill password, access password, EPC memory, TID memory, and User memory, thereby preventing or allowing subsequent changes (as appropriate). A Tag may optionally have its User memory partitioned into blocks; if it does, then an Interrogator may permanently lock these individual blocks. Recommissioning may alter the memory locking and/or permalocking. See 6.3.4.9 for a description of memory locking and unlocking, and 6.3.4.10 for a description of Tag recommissioning. If the kill and/or access passwords are locked they are usable by only the Kill and Access commands, respectively, and are rendered both unwriteable and unreadable by any other command. Locking or permanently locking the EPC, TID, or User memory banks, or permanently locking blocks within the User memory bank, renders the locked memory location unwriteable but leaves it readable.

6.3.4.1.1 Reserved Memory

Reserved memory contains the kill (see 6.3.4.1.1.1) and/or access (see 6.3.4.1.1.2) passwords, if passwords are implemented on the Tag. If a Tag does not implement the kill and/or access password(s), the Tag shall logically operate as though it has zero-valued password(s) that are permanently read/write locked (see 6.3.4.11.3.5), and the corresponding physical memory locations in Reserved memory need not exist.

6.3.4.1.1.1 Kill password

The kill password is a 32-bit value stored in Reserved memory 00_h to $1F_h$, MSB first. The default (unprogrammed) value shall be zero. An Interrogator may use the kill password to: recommission a Tag, and/or kill a Tag and render it nonresponsive thereafter. A Tag shall not execute a recommissioning or kill operation if its kill password is zero. A Tag that does not implement a kill password operates as if it has a zero-valued kill password that is permanently read/write locked.

6.3.4.1.1.2 Access password

The access password is a 32-bit value stored in Reserved memory 20_h to $3F_h$, MSB first. The default (unprogrammed) value shall be zero. A Tag with a nonzero access password shall require an Interrogator to issue this password before transitioning to the secured state. A Tag that does not implement an access password operates as if it has a zero-valued access password that is permanently read/write locked.

6.3.4.1.2 EPC Memory

EPC memory contains a StoredCRC at memory addresses 00_h to $0F_h$, a StoredPC at 10_h to $1F_h$, an EPC beginning at 20_h, and a first XPC word (XPC_W1) at 210_h to $21F_h$ and an optional second XPC word (XPC_W2) at 220_h to $22F_h$. The StoredCRC, StoredPC, EPC, and XPC word or words shall be stored MSB first (i.e. the EPC's MSB is stored in location 20_h).

The StoredCRC is described in 6.3.4.1.2.1.

The StoredPC, as described in 6.3.4.1.2.2, is subdivided into an EPC length field in memory locations 10_h to 14_h, a User-memory indicator (UMI) in location 15_h, an XPC indicator (XI) in location 16_h, and a Numbering System Identifier (NSI) in locations 17_h to $1F_h$.

The EPC is a code that identifies the object to which a Tag is affixed. The EPC for EPCglobal™ Applications is described in 6.3.4.1.2.3 and the EPC for non-EPCglobal™ Applications is described in 6.3.4.1.2.4. Interrogators may issue a Select command that includes all or part of the EPC in the mask. Interrogators may issue an *ACK* command to cause a Tag to loadmodulate its PC word, XPC word (if XI is asserted), EPC, and a PacketCRC if required. Under certain circumstances the Tag may truncate its reply (see 6.3.2.11.1.1). An Interrogator may issue a Read command to read all or part of the EPC.

The XPC word, as described in 6.3.4.1.2.5, indicates whether and how a Tag has been recommissioned.

6.3.4.1.2.1 CRC-16 (StoredCRC and PacketCRC)

All Tags shall implement a StoredCRC. Tags shall also implement a PacketCRC.

At power-up a Tag shall calculate a CRC-16 over (a) the StoredPC and (b) the EPC specified by the EPC length field in the StoredPC (see 6.3.4.1.2.1) and shall map the calculated CRC-16 into EPC memory 00_h to $0F_h$, MSB first. This CRC is denoted the StoredCRC. Because the StoredPC and EPC comprise an integer number of EPC-memory words, a Tag calculates this StoredCRC on word boundaries. Although Tags include the XI value in the StoredPC in their StoredCRC calculation, regardless of the XI value a Tag shall omit XPC_W1 and XPC_W2 from the calculation. A Tag shall finish its StoredCRC calculation and memory mapping by the end of interval T_s in Figure 6.5. Interrogators may issue a *Select* command that includes all or part of the StoredCRC in Mask. Interrogators may issue a *Read* command to instruct a Tag to loadmodulate its StoredCRC. A Tag shall not recalculate this StoredCRC for a truncated reply (see 6.3.4.11.1.1).

In response to an *ACK* command a Tag loadmodulates a protocol-control (PC) word (either StoredPC or PacketPC – see 6.3.4.1.2.2), a first XPC word (XPC_W1) and an optional second XPC word (XPC_W2), depending on XI (see 6.3.4.1.2.5), EPC (see 6.3.4.1.2.3 and 6.3.4.1.2.4), and a CRC-16 only in case of a PackedCRC is required according to Table 6.15 below.. The CRC-16 in this case shall be a PacketCRC that the Tag shall calculate dynamically over the loadmodulated PC word, XPC word or words (if supported), and EPC. Whether a Tag omits the CRC16 or loadmodulates its PacketCRC, shall be as defined in Table 6.15, depending on the Tag's XI value and whether truncation (see 6.3.4.11.1.1) is asserted or deasserted.

Table 6.15. Tag data and CRC-16 loadmodulated in response to an *ACK* command

XI	XEB	Truncation	Tag Loadmodulation			
			PC	XPC	EPC	CRC-16
0	0	Deasserted	StoredPC	None	Full	no CRC16 in reply
0	0	Asserted	00000_2	None	Truncated	no CRC16 in reply
0	1	Deasserted	Invalid[1]			
0	1	Asserted	Invalid[1]			
1	0	Deasserted	PacketPC	XPC_W1	Full	PacketCRC
1	0	Asserted	00000_2	None	Truncated	no CRC16 in reply
1	1	Deasserted	PacketPC	Both XPC_W1 and XPC_W2	Full	PacketCRC
1	1	Asserted	00000_2	None	Truncated	no CRC16 in reply

Note 1: XI is the bitwise logical OR of the 16 bits of XPC_W1, and XEB is the MSB (bit F_h) of XPC_W1, so if XEB=1 then XI=1

Note 2: The StoredCRC may be different from the Packet CRC if the tag memory has been programmed and the tag has not undergone a power-on-reset.

A Tag shall loadmodulate its CRC MSB first, regardless of the CRC type.

If XI is asserted then a Tag's PacketCRC is different from its StoredCRC.

As required by 6.3.3.1.5 an Interrogator shall verify, using a Tag's loadmodulated CRC-16, the integrity of a received PC word, first XPC word (XPC_W1), optional second XPC word (XPC_W2), and EPC.

6.3.4.1.2.2 Protocol-control (PC) word (StoredPC and PacketPC)

All Tags shall implement a StoredPC whose fields, comprising EPC length, a UMI, an XI, and an NSI, shall be as defined below. Tags shall also implement a PacketPC that differs from the StoredPC in its EPC length field. The type of PC (StoredPC or PacketPC) that a Tag loadmodulates in response to an *ACK* shall be as defined in Table 6.15.

The StoredPC shall be located in EPC memory at addresses 10_h to $1F_h$, with bit values defined as follows:

- Bits 10_h – 14_h: EPC length field. The length of the EPC, in words:
  - 00000_2: Zero words.
  - 00001_2: One word (addresses 20_h to $2F_h$ in EPC memory).
  - 00010_2: Two words (addresses 20_h to $3F_h$ in EPC memory).
    -
    -
    -
  - 11101_2: 29 words (addresses 20_h to $1EF_h$ in EPC memory).

  The maximum value of the EPC length field in the StoredPC shall be 11101_2 (allows a 464-bit EPC). A Tag shall ignore a *Write* or *BlockWrite* command to the StoredPC if the EPC length field exceeds 11101_2, and shall instead loadmodulate an error code (see Annex I).

- Bit 15_h: A User-memory indicator (UMI). If bit 15_h is deasserted then the Tag either does not implement User memory or User memory contains no information. If bit 15_h is asserted then User memory contains information. A Tag may implement the UMI using Method 1 or Method 2 described below, unless the Tag implements block permalocking and/or recommissioning, in which case the Tag shall use Method 1.
  In case that only the mandatory recommisioning option with asserted Recom bit 3SB is supported and no block permalocking is supported also Method 2 may be used instead of Method 1.
  - Method 1: The Tag computes the UMI. At power-up, and prior to calculating its StoredCRC (see 6.3.4.1.2.1), a Tag shall compute the logical OR of bits 03_h to 07_h of User memory and map the

computed value into bit 15_h. A Tag shall use this computed UMI value in its StoredCRC calculation. If an Interrogator modifies any of bits 03_h to 07_h of User memory then the Tag shall recompute and remap its UMI into bit 15_h. If recommissioning renders User memory inaccessible (see 6.3.4.10) then the Tag shall deassert and remap its UMI into bit 15_h. After remapping the UMI the StoredCRC may be incorrect until the Interrogator power cycles the Tag. The UMI shall not be directly writeable by an Interrogator — when an Interrogator writes the StoredPC the Tag shall ignore the data value that the Interrogator provides for bit 15_h.

- Method 2: An Interrogator writes the UMI. If an Interrogator writes a zero value into bits 03_h to 07_h of User memory then the Interrogator shall deassert bit 15_h. If an Interrogator writes a nonzero value into 03_h to 07_h of User memory then the Interrogator shall assert bit 15_h. If an Interrogator locks or permalocks EPC memory then the Interrogator shall also lock or permalock, respectively, the word located at address 00_h of User memory, and vice versa. This latter requirement ensures that a condition in which User memory previously contained data but was subsequently erased does not cause a Tag to wrongly indicate the presence of User memory, and vice versa.

- Bit 16_h: An XPC_W1 indicator (XI). If bit 16_h is deasserted then the XPC_W1 is zero-valued, in which case the Tag shall loadmodulate its StoredPC but not an XPC_W1 during inventory (see 6.3.4.1.2 and Table 6.15). If bit 16_h is asserted then one or more bits of XPC_W1 have nonzero values, indicating that the Tag has been previously re-commissioned (see 6.3.4.1.2.5).
  In this latter case the Tag shall loadmodulate its XPC_W1 immediately after the PacketPC, and before the EPC, during inventory.

  At power-up, and prior to calculating its StoredCRC (see 6.3.4.1.2.1), the Tag shall compute the bitwise logical OR of its XPC_W1 and map the computed value into bit 16_h (i.e. into the XI). A Tag shall use this computed XI value in its StoredCRC calculation. If an Interrogator recommissions the Tag (see 6.3.4.10) then the Tag shall recompute and remap its XI into bit 16_h after recommissioning. After recomputing the XI the StoredCRC may be incorrect until the Interrogator power cycles the Tag. The XI bit shall not be directly writeable by an Interrogator — when an Interrogator writes the StoredPC the Tag shall ignore the data value that the Interrogator provides for bit 16_h.

- Bits 17_h – $1F_h$: A numbering system identifier (NSI). The MSB of the NSI is stored in memory location 17_h. If bit 17_h contains a logical 0, then the application is referred to as an EPCglobal™ Application and bits 18_h – $1F_h$ shall be as defined in the EPCglobal™ Tag Data Standards. If bit 17_h contains a logical 1, then the application is referred to as a non-EPCglobal™ Application and bits 18_h – $1F_h$ shall contain the entire AFI defined in ISO/IEC 15961 parts 2 and 3. The default value for bits 18_h – $1F_h$ is 00000000_2.

The default (unprogrammed) StoredPC value shall be 0000_h.

If an Interrogator changes the EPC length (via a memory write operation), and if it wishes the Tag to subsequently loadmodulate the new EPC length, then it must write a new EPC length field into the Tag's StoredPC. Note: After changing the EPC length field an Interrogator should power-cycle the Tag to ensure a correct StoredCRC.

A Tag shall loadmodulate an error code (see Annex I) if an Interrogator attempts to write an EPC length field that the Tag does not support.

The PacketPC differs from the StoredPC in its EPC length field, which a Tag shall adjust to match the length of the loadmodulated data that follow the PC word. Specifically, if XI is asserted but XEB is not asserted then the Tag loadmodulates an XPC_W1 before the EPC, so the Tag shall add one to (i.e. increment) its EPC length field. If both XI and XEB are asserted then the Tag loadmodulates both an XPC_W1 and an XPC_W2 before the EPC, so the Tag shall add two to (i.e. double increment) its EPC length field. Because Tags that support XPC functionality have a maximum EPC length field of 11101_2, double incrementing will increase the value to 11111_2. A Tag shall not, under any circumstances, allow its EPC length field to roll over to 00000_2. Note that incrementing or double incrementing the EPC length field does not alter the values stored in bits 10_h – 14_h of EPC memory; rather, a Tag increments the EPC length field in the loadmodulated PacketPC but leaves the memory contents unaltered.

If an Interrogator that does not support an XPC_W2 receives a Tag reply with XEB asserted then the Interrogator shall treat the Tag's reply as though its CRC-16 integrity check had failed.

During truncated replies a Tag substitutes 00000_2 for the PC word — see Table 6.15 and 6.3.4.11.1.1.

6.3.4.1.2.3 EPC for an EPCglobal™ Application

The EPC structure for an EPCglobal™ Application shall be as defined in the EPCglobal™ Tag Data Standards.

6.3.4.1.2.4 EPC for a non-EPCglobal™ Application

The EPC structure for a non-EPCglobal™ Application shall be as defined in ISO/IEC 15961.

6.3.4.1.2.5 Extended Protocol Control (XPC) word (mandatory) or words (optional)

A Tag shall implement an XPC_W1 logically located at addresses 210_h to $21F_h$ of EPC memory. A tag may additionally implement an XPC_W2 logically located at address 220_h to $22F_h$ of EPC memory. These XPC words shall be exactly 16 bits in length and are stored MSB first. If a Tag does not support XPC_W2 then the specified memory locations need not exist.

A Tag shall not implement any non-XPC memory element at EPC memory locations 210_h to $22F_h$, inclusive. This requirement shall apply both to Tags that support an XPC_W2 word and to those that do not.

If a Tag implements an XPC_W2 then, at power-up, the Tag shall compute the bitwise logical OR of the XPC_W2 and map the computed value into bit 210_h of EPC memory (i.e. into the most significant bit of XPC_W1). Bit 210_h is denoted the XPC Extension Bit (XEB). If a Tag does not implement an XPC_W2 then the XEB shall be zero.

The remainder of this section 6.3.4.1.2.5 assumes that a Tag implements an XPC_W1 (mandatory) and an XPC_W2 (optional).

When this document refers to the 3 least-significant-bits (LSBs) of XPC_W1 it specifically means locations $21D_h$, $21E_h$, and $21F_h$ of EPC memory. The 3 LSBs of XPC_W1 indicate whether and how a Tag was recommissioned.

For virgin Class-1 Tags the 3 LSBs of XPC_W1 shall be zero-valued. A Tag writes a nonzero value to one or more of these 3 LSBs during Tag recommissioning (see 6.3.4.10). For Class-1 Tags all the other bits in XPC_W1, namely EPC memory locations 210_h to $21C_h$, inclusive, as well as all the bits in XPC_W2, shall be RFU and zero-valued. Tag vendors and end users shall not use these RFU bits for proprietary purposes.

The 3 LSBs of XPC_W1 are not writeable using a *Write* or *BlockWrite*, nor erasable using a *BlockErase*. They can only be asserted by a *Kill* command, meaning that the Tag asserts these bits upon receiving a valid *Kill* command sequence with asserted recommissioning bits (see 6.3.4.11.3.4). A Tag shall not modify the 3 LSBs of XPC_W1 except during Tag recommissioning.

If a Class-1 Tag receives a *Write*, *BlockWrite*, or *BlockErase* command that attempts to write to XPC_W1 or to XPC_W2 it responds with an error code (see Annex I).

An Interrogator may issue a *Select* command (see 6.3.4.11.1) with a Mask that covers all or part of XPC_W1 and/or XPC_W2. For example, Mask may have the value 000_2 for the 3 LSBs of XPC_W1, in which case recommissioned Tags shall be non-matching.

An Interrogator may read a Tag's XPC_W1 and XPC_W2 using a *Read* command (see 6.3.4.11.3.2).

Bit E_h of XPC_W1 (EPC memory location 211_h) is reserved for use as a protocol functionality indicator. See 2.4.

The following encoding provides a general mapping betwen the 3 XPC_W1 LSBs and a Tag's recommissioned status. Table 6.16 provides a detailed mapping between these 3 LSBs and a Tag's recommissioned status:

- **An asserted LSB** (EPC memory location $21F_h$) indicates that block permalocking has been disabled, and any blocks of User memory that were previously block permalocked are no longer block permalocked. An asserted LSB also indicates that the BlockPermalock command has been disabled. If a Tag did not implement block permalocking prior to recommissioning then block permalocking shall remain disabled.
- **An asserted 2SB** (EPC memory location $21E_h$) indicates that User memory has been rendered inaccessible. The 2SB has precedence over the LSB — if both are asserted then User memory is inaccessible.

- **An asserted 3SB** (EPC memory location $21D_h$) indicates that the Tag has unlocked its EPC, TID, and User memory banks. The Tag has also write-unlocked its kill and access passwords, and left the read lock status of the kill and access passwords in the state prior to the recommissioning. If an Interrogator subsequently attempts to read the Tag's kill or access passwords, the Tag loadmodulates an error code (see Annex I) if the kill or access password was unreadable prior to the recommisioning. Note that portions or banks of Tag memory, if factory set and locked, may not be unlockable regardless of recommissioning. Note also that an Interrogator may subsequently re-lock any memory banks that have been unlocked by recommissioning.

Table 6.16. XPC_W1 LSBs and a Tag's recommissioned status

LSBs of XPC Word	Tag Status	Notes
000_2	1. The tag has not been recommissioned	1. The tag's XI bit is deasserted
001_2	1. Block permalocking has been disabled, and any blocks of User memory that were previously block permalocked are no longer block permalocked 2. The *BlockPermalock* command has been disabled	1. The lock bits are the sole determinant of the User memory bank's lock status (6.3.4.11.3.5)
010_2	1. The User memory bank has been rendered inaccessible 2. A tag whose XPC_W1 LSBs are 010 acts identically to one whose XPC_W1 LSBs are 011	1. The tag deasserts its UMI bit 2. User memory is inaccessible, so block permalocking and the *BlockPermalock* command are disabled
011_2	1. The User memory bank has been rendered inaccessible 2. A tag whose XPC_W1 LSBs are 011 acts identically to one whose XPC_W1 LSBs are 010	1. The tag deasserts its UMI bit 2. User memory is inaccessible, so block permalocking and the *BlockPermalock* command are disabled
100_2	1. The EPC, TID, and User memory banks have been unlocked 2. The kill and access passwords have been write-unlocked 3. The Read/Lock status of the kill and access passwords shall be the same state as prior to the recommissioning	1. If the tag supports block permalocking then the *BlockPermalock* command remains enabled 2. Any blocks of User memory that were previously block permalocked remain block permalocked, and vice versa 3. Portions or banks of tag memory, if factory set and locked, may not be unlockable regardless of recommissioning 4. If an Interrogator attempts to read the tag's kill or access passwords the tag responds by loadmodulating an error code (see Annex I) if the kill or access password was unreadable prior to the recommisioning 5. The kill and/or access passwords, as well as one or more memory blocks and/or banks, may be changed and/or relocked after recommissioning
101_2	1. Block permalocking has been disabled, and any blocks of User memory that were previously block permalocked are no longer block permalocked 2. The *BlockPermalock* command has been disabled 3. The EPC, TID, and User memory banks have been unlocked 4. The kill and access passwords have been write-unlocked 5. The Read/Lock status of the kill and access passwords shall be the same state as prior to the recommissioning	1. The lock bits are the sole determinant of the User memory bank's lock status (see 6.3.4.11.3.5) 2. Portions or banks of tag memory, if factory set and locked, may not be unlockable regardless of recommissioning 3. If an Interrogator attempts to read the tag's kill or access passwords the tag responds by loadmodulating an error code (see Annex I) if the kill or access password was unreadable prior to the recommisioning 4. The kill and/or access passwords, as well as one or more memory blocks and/or banks, may be changed and/or relocked after recommissioning

LSBs of XPC Word	Tag Status	Notes
110_2	1. The EPC and TID memory banks have been unlocked 2. The kill and access passwords have been write-unlocked 3. The Read/Lock status of the kill and access passwords shall be the same state as prior to the recommissioning 4. The User memory bank has been rendered inaccessible 5. A tag whose XPC_W1 LSBs are 110_2 acts identically to one whose XPC_W1 LSBs are 111_2	1. Portions or banks of tag memory, if factory set and locked, may not be unlockable regardless of recommissioning 2. If an Interrogator attempts to read the tag's kill or access passwords the tag responds by loadmodulating an error code (see Annex I) if the kill or access password was unreadable prior to the recommisioning 3. The kill and/or access passwords, as well as one or more memory blocks and/or banks, may be changed and/or relocked after recommissioning 4. The tag deasserts its UMI bit 5. User memory is inaccessible, so block permalocking and the *BlockPermalock* command are disabled
111_2	1. The EPC and TID memory banks have been unlocked 2. The kill and access passwords have been write-unlocked 3. The Read/Lock status of the kill and access passwords shall be the same state as prior to the recommissioning 4. The User memory bank has been rendered inaccessible 5. A tag whose XPC_W1 LSBs are 111_2 acts identically to one whose XPC_W1 LSBs are 110_2	1. Portions or banks of tag memory, if factory set and locked, may not be unlockable regardless of recommissioning 2. If an Interrogator attempts to read the tag's kill or access passwords the tag responds by loadmodulating an error code (see Annex I) if the kill or access password was unreadable prior to the recommisioning 3. The kill and/or access passwords, as well as one or more memory blocks and/or banks, may be changed and/or relocked after recommissioning 4. The tag deasserts its UMI bit 5. User memory is inaccessible, so block permalocking and the *BlockPermalock* command are disabled

6.3.4.1.3 TID Memory

TID memory locations 00_h to 07_h shall contain one of two ISO/IEC 15963 class-identifier values — either $E0_h$ or $E2_h$. TID memory locations above 07_h shall be defined according to the registration authority defined by this class-identifier value and shall contain, at a minimum, sufficient identifying information for an Interrogator to uniquely identify the custom commands and/or optional features that a Tag supports. TID memory may also contain Tag- and vendor-specific data (for example, a Tag serial number).

Note: The Tag manufacturer assigns the class-identifier value (i.e. $E0_h$ or $E2_h$), for which ISO/IEC 15963 defines the registration authorities. The class-identifier does not specify the Application. If the class identifier is $E0_h$, TID memory locations 08_h to $0F_h$ contain an 8-bit manufacturer identifier, TID memory locations 10_h to $3F_h$ contain a 48-bit Tag serial number (assigned by the Tag manufacturer), the composite 64-bit Tag ID (i.e. TID memory 00_h to $3F_h$) is unique among all classes of Tags defined in ISO/IEC 15693, and TID memory is permalocked at the time of manufacture. If the class identifier is $E2_h$, TID memory location 08_h to 13_h contain a 12-bit Tag mask-designer identifier (obtainable from the registration authority), TID memory locations 14_h to $1F_h$ contain a vendor-defined 12-bit Tag model number, and the usage of TID memory locations above $1F_h$ is defined in the EPCglobal™ Tag Data Standards.

6.3.4.1.4 User Memory

A Tag may contain User memory. User memory allows user-specific data storage.

If a Tag's User memory has not yet been programmed, then the 5 LSBs of the first byte of User memory (*i.e.* memory addresses 03_h to 07_h) shall have the default value 00000_2.

During recommissioning an Interrogator may instruct a Tag to render its User memory inaccessible, causing the entire memory bank to become unreadable, unwriteable, and unselectable. A Tag with inaccessible User memory shall function as though its User memory bank no longer exists.

6.3.4.1.4.1 User memory for an EPCglobal™ Application

If User memory is included on a Tag, then its encoding shall be as defined in the EPCglobal™ Tag Data Standards.

6.3.4.1.4.2 User memory for a non-EPCglobal™ Application

If User memory is included on a Tag, then its encoding shall be as defined in ISO/IEC 15961 and ISO/IEC 15962.

6.3.4.2 ASK and PJM Method: Sessions and inventoried flags

Note: An interrogator chooses one of two 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 (defined above). For each session, tags maintain a corresponding inventoried flag. Sessions allow tags to keep track of their inventoried status separately for each of two possible time-interleaved inventory processes, using an independent inventoried flag for each process.

Interrogators shall support and tags shall provide two mandatory sessions (denoted S0 and S2) and two optional sessions (denoted S1 and S3). Tags shall participate in one and only one session during an inventory round. Two or more interrogators can use sessions to independently inventory a common tag population. The session's concept is illustrated in Figure 6.26.

Tags shall maintain an independent **inventoried** flag for each session. Each of the two mandatory **inventoried** flags has two values, denoted *A* and *B.* Tags participating in an inventory round in one session shall neither use nor modify the **inventoried** flag for a different session. The **inventoried** flags are the only resource a tag provides separately and independently to a given session; all other tag resources are shared among sessions.

Sessions allow tags to associate a separate and independent **inventoried** flag to each of several Interrogators.

After singulating a tag an interrogator may issue a command that causes the tag to set its **inventoried** flag for that session (*A*→*B).*

Only tags with an inventoried Flag of A respond in an inventory round.

Note: A change from B to A can be achieved with a select command. (Reset)

The following example illustrates how two interrogators can use sessions and **inventoried** flags to independently and completely inventory a common tag population, on a time-interleaved basis:

- Interrogator #1 powers-on, then
  - It initiates an inventory round during which it singulates *A* tags in session S0 to *B*,
  - It powers off.
- Interrogator #2 powers-on, then
  - It initiates an inventory round during which it singulates *A* tags in session S2 to *B*,
  - It powers off.

This process repeats until interrogator #1 has placed all tags in session S0 into *B*. Similarly, interrogator #2 places all tags in session S2 into *B*.

Note: Tags maintain a separate inventoried flag for each of two mandatory sessions.

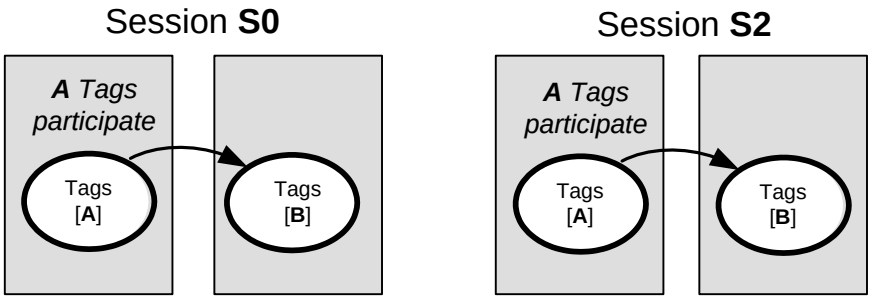


Figure 6.26 Session diagram

A tag **inventoried** flags shall have the persistence times shown in Table 6.17. A tag shall power-up with its **inventoried** flags set as follows:

- The S0 **inventoried** flag shall be set to *A*.
- The S1 **inventoried** flag (optional) shall be set to either *A* or *B*, depending on its stored value, unless the flag was set longer in the past than its persistence time, in which case the tag shall power-up with its S1 inventoried flag set to *A*. Because the S1 **inventoried** flag (optional) is not automatically refreshed, it may revert from *B* to *A* even when the tag is powered.
- The S2 **inventoried** flag shall be set to either *A* or *B*, depending on its stored value, unless the tag has lost power for a time greater than its persistence time, in which case the tag shall power-up with the S2 inventoried flag set to *A*.
- The S3 **inventoried** flag (optional) shall be set to either *A* or *B*, depending on its stored value, unless the tag has lost power for a time greater than its persistence time, in which case the tag shall power-up with its S3 inventoried flag (optional) set to *A*.

A tag shall be capable of setting any of its inventoried flags to either *A* or *B* in 2ms or less, regardless of the initial flag value. A tag shall refresh its S2 and S3 flags while powered, meaning that every time a tag loses power its S2 and S3 inventoried flags shall have the persistence times shown in Table 6.17.

Table 6.17. Tag flags and persistence values

Flag	Required persistence
S0 **inventoried** flag (mandatory)	Tag energized: Indefinite Tag not energized: None
S1 **inventoried** flag (optional)	Tag energized: Nominal temperature range: 500ms < persistence < 5s Extended temperature range: Not specified Tag not energized: Nominal temperature range: 500ms < persistence < 5s Extended temperature range: Not specified
S2 **inventoried** flag (mandatory)	Tag energized: Indefinite Tag not energized: Nominal temperature range: 2s < persistence Extended temperature range: Not specified
S3 **inventoried** flag (optional)	Tag energized: Indefinite Tag not energized: Nominal temperature range: 2s < persistence Extended temperature range: Not specified
Selected (**SL**) flag	Tag energized: Indefinite Tag not energized: Nominal temperature range: 2s < persistence Extended temperature range: Not specified

Note 1: Nominal temperature range is –25 °C to +40 °C (see extended temperature range)

Note 2: Extended temperature range is –40 °C to +65 °C (see nominal temperature range)

6.3.4.3 ASK and PJM Method: Selected flag

Tags shall implement a selected flag, **SL,** which an interrogator may assert or deassert using a *Select* command. The *Select* parameter in the *BeginRound* command allows an interrogator to inventory tags that have **SL** either asserted or deasserted (*i.e.* **SL** or ~**SL),** or to ignore the flag and inventory tags regardless of their **SL** value. **SL** is not associated with any particular session; **SL** applies to all tags regardless of session.

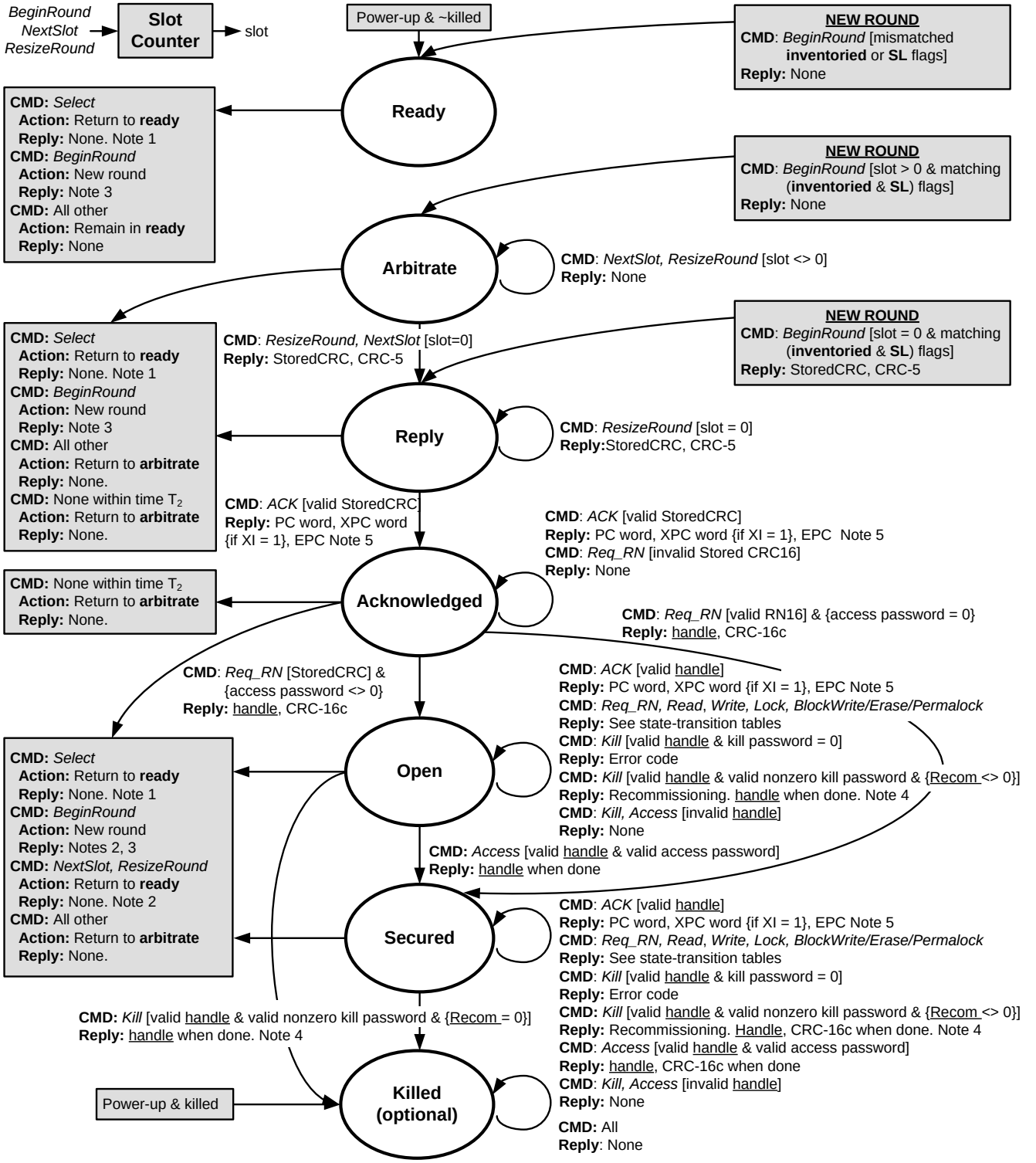
A tag's **SL** flag shall have the persistence times shown in Table 6.17. A tag shall power-up with its **SL** flag either asserted or deasserted, depending on the stored value, unless the tag has lost power for a time greater than the **SL** persistence time, in which case the tag shall power-up with its **SL** flag deasserted (set to **~SL**). A tag shall be capable of asserting or deasserting its **SL** flag in 2ms or less, regardless of the initial flag value. A tag shall refresh its **SL** flag when powered, meaning that every time a tag loses power its **SL** flag shall have the persistence times shown in Table 6.17.

6.3.4.4 ASK and PJM Method: Tag states and slot counter

Tags shall implement the states and the slot counter shown in Figure 6.27. Annex B shows the associated state transition tables; Annex C shows the associated command-response tables.

Note: The tag collision arbitration protocol uses the popular and industry proven slotted Aloha algorithm. Slotted ALOHA is a refinement over the pure Aloha. It requires that time be segmented into slots that are separated by slot markers. The probability that a tag chooses a timeslot is equal to 1/n (where "n" is the number of timeslots and equals the maximum value of the described Q-bit random number). This algorithm ensures that each tag, if selected, replies once and only once during an inventory round.

Example: for Q=3 the maximum number of slots is 8, Q is a 3 bit random number, ranging from 0-7, in each slot the probability of a specific tag reply is 1/8.



NOTES 1. *Select*: Assert/deassert **SL** or set **inventoried** to *A* or *B*.
2. *BeginRound*: *A*→ *B* if the new session matches the prior session; otherwise no change to the **inventoried** flag.
NextSlot/ResizeRound: *A*→ *B* if the session matches the prior *BeginRound*; otherwise, the command is invalid and ignored by the Tag.
3. *BeginRound* starts a new round and may change the session. Tags may go to **ready**, **arbitrate**, or **reply**.
4. If a Tag does not implement Recom Bits LSB and 2SB then the Tag treats nonzero Recom bits as though Recom = 0. Two part reply see *Kill*
5. In certain cases a PacketCRC should be added if required. See Table 6.15 for the definition of the cases.

Figure 6.27 Tag state diagram

6.3.4.4.1 Ready state

Tags shall implement a **ready** state. **Ready** can be viewed as a "holding state" for energized tags that are neither killed nor currently participating in an inventory round. Upon entering an energizing RF field a tag that is not killed shall enter **ready.** The tag shall remain in **ready** until it receives a *BeginRound* command (see 6.3.4.11.2.1) whose inventoried parameter (for the session specified in the *BeginRound)* and *Select* parameter match its current flag values. Matching tags shall draw a *Q*-bit number from their RNG (see 6.3.4.5), load this number into their slot counter, and transition to the **arbitrate** state if the number is non-zero, or to the **reply** state if the number is zero. If a tag in any state except **killed** loses power, it shall return to **ready** upon regaining power.

6.3.4.4.2 Arbitrate state

Tags shall implement an **arbitrate** state. **Arbitrate** can be viewed as a "holding state" for tags that are participating in the current inventory round but whose slot counters (see 6.3.4.4.8) hold non-zero values. A tag in **arbitrate** shall decrement its slot counter every time it receives a *NextSlot* command (see 6.3.4.11.2.3) whose session parameter matches the session for the inventory round currently in progress, and it shall transition to the **reply** state when its slot counter reaches 0000_h. Tags that return to **arbitrate** (for example, from the **reply** state) with a slot value of 0000_h shall decrement their slot counter from 0000_h to $7FFF_h$ at the next *NextSlot* (with matching session) and, because their slot value is now non-zero, shall remain in **arbitrate.**

6.3.4.4.3 Reply state

Tags shall implement a **reply** state. Upon entering **reply** a tag shall loadmodulate the StoredCRC. If the tag receives a valid acknowledgement (*ACK*), it shall transition to the **acknowledged** state, loadmodulates its PC word, XPC word (if XI is asserted), EPC and PacketCRC if required (see Table 6.15). If the tag fails to receive an *ACK*, or receives an invalid *ACK*, it shall return to **arbitrate**. Tag and interrogator shall meet all timing requirements specified in Table 6.14.

6.3.4.4.4 Acknowledged state

Tags shall implement an **acknowledged** state. A Tag in the **acknowledged** state may transition to any state except **killed**, depending on the received command (see Figure 6.27). If a Tag in the **acknowledged** state receives a valid *ACK* containing the correct StoredCRC it shall loadmodulate the reply shown in Table 6.15. If a Tag in the **acknowledged** state fails to receive a valid command within time $T_{2(max)}$, it shall return to **arbitrate**. Tag and interrogator shall meet all timing requirements specified in Table 6.14.

6.3.4.4.5 Open state

Tags shall implement an **open** state. A tag in the **acknowledged** state whose access password is non-zero shall transition to **open** upon receiving a *Req_RN* command, loadmodulating an RN16 (denoted handle) that the interrogator shall use in subsequent commands and that the tag shall use in subsequent replies. Tags in the **open** state can execute all access commands except *Lock* and *BlockPermalock.* A tag in **open** may transition to any state except **acknowledged,** depending on the received command (see Figure 6.27). If a Tag in the **open** state receives a valid *ACK* containing the correct handle, it shall loadmodulate the reply shown in Table 6.15. Tag and interrogator shall meet all timing requirements specified in Table 6.14 except $T_{2(max)}$; in the **open** state the maximum delay between tag response and interrogator transmission is unrestricted.

6.3.4.4.6 Secured state

Tags shall implement a **secured** state. A tag in the **acknowledged** state whose access password is zero shall transition to **secured** upon receiving a *Req_RN* command, loadmodulate an RN16 (denoted handle) that the interrogator shall use in subsequent commands and the tag shall use in subsequent replies. A tag in the **open** state whose access password is non-zero shall transition to **secured** upon receiving a valid *Access* command, maintaining the same handle that it previously transmitted when it transitioned from the **acknowledged state** to the **open** state. Tags in the **secured** state can execute all access commands. A tag in **secured state** may transition to any state except **open** or **acknowledged,** depending on the received command (see Figure 6.27). If a Tag in the **secured** state receives a valid *ACK* containing the correct handle, it shall loadmodulate the reply

shown in Table 6.15. Tag and interrogator shall meet all timing requirements specified in Table 6.14 except $T_{2(max)}$; in the **secured** state the maximum delay between tag response and interrogator transmission is unrestricted.

6.3.4.4.7 Killed state (optional)

Tags may implement a **killed** state. If the killed state is supported, a tag in either the **open** or **secured** states shall enter the **killed** state upon receiving a valid *Kill* command (see 6.3.4.11.3.4) with a valid non-zero kill password, zero-valued Recom bits (see 6.3.4.10), and valid handle. If a Tag does not implement Recom bits LSB and 2SB , it then treats nonzero Recom bits as though Recom=0. *Kill* permanently disables a tag. Upon entering the **killed** state a tag shall notify the interrogator that the kill operation was successful, and shall not respond to an interrogator thereafter. Killed tags shall remain in the **killed** state under all circumstances, and shall immediately enter the killed state upon subsequent power-ups. Killing a tag is not reversible.

6.3.4.4.8 Slot counter

Tags shall implement a 15-bit slot counter. Upon receiving a *BeginRound* or *ResizeRound* command, a tag shall preload a value between 0 and 2^Q–1, drawn from the tag RNG (see 6.3.4.5), into its slot counter. Q is an integer in the range (0 - 15). A *BeginRound* specifies Q; a *ResizeRound* may modify Q from the prior *BeginRound* or prior *ResizeRound*. Upon receiving a *NextSlot* command a tag shall decrement its slot counter. The slot counter shall be capable of continuous counting: meaning that, after the slot counter decrements to 0000_h, it shall roll over and begin counting down from $7FFF_h$. See also Annex J.

Tags shall generate 16-bit random or pseudo-random numbers (RN16) using the RNG, and shall have the ability to extract *Q*-bit subsets from an RN16 to preload the tag slot counter.

Tags in the **arbitrate** state decrement their slot counter every time they receive a *NextSlot* with matching session, transitioning to the **reply** state and loadmodulating the StoredCRCwhen their slot counter reaches 0000_h. Tags whose slot counter reached 0000_h, who replied, and who were not acknowledged (including Tags that responded to an original *BeginRound* and were not acknowledged) shall return to **arbitrate** with a slot value of 0000_h and shall decrement this slot value from 0000_h to $7FFF_h$ at the next *NextSlot*. The slot counter shall be capable of continuous counting, meaning that, after the slot counter rolls over to $7FFF_h$ it begins counting down again, thereby effectively preventing subsequent replies until the Tag loads a new random value into its slot counter. See also Annex J.

Note: An interrogator commands tags in an inventory round to load a Q-bit random (or pseudo-random) number into their slot counter; the interrogator may also command tags to decrement their slot counter. Tags reply when the value in their slot counter (i.e. their slot – see below) is zero. Q is an integer in the range (0 - 15); the corresponding tag response probabilities range from $2^0 = 1$ to $2^{-15} = 0{,}000031$.

6.3.4.5 ASK and PJM Method: Tag random or pseudo-random number generator

Tags shall implement a random or pseudo-random number generator (RNG). The RNG shall meet the following randomness criteria independent of the strength of the energizing field, the R=>T link rate, and the data stored in the tag (including but not limited to the StoredPC, XPC word or words, EPC, and StoredCRC). The RNG is used to produce pseudo-random numbers for RN16, Handle, and Slot Counter. Tags that support cover-coding shall have the ability to temporarily store at least two RN16s while powered, to use, for example, as a handle and a 16-bit cover-code during password transactions (see Figure 6.31 or Figure 6.33).

- **Probability of a single RN16:** The probability that any RN16 drawn from the RNG has value RN16 = *j*, for any *j*, shall be bounded by $0.8/2^{16} < P(RN16 = j) < 1.25/2^{16}$.
- **Probability of simultaneously identical sequences:** For a tag population of up to ten thousand tags, the probability that any two or more tags simultaneously generate the same sequence of RN16s shall be less than 0.1%, regardless of when the tags are energized.
- **Probability of predicting an RN16:** An RN16 drawn from a tag RNG 10ms after the end of T_r in Figure 6.3 shall not be predictable with a probability greater than 0,025% if the outcomes of prior draws from the RNG, performed under identical conditions, are known.

6.3.4.6 ASK and PJM Method: Managing tag populations

Interrogators manage tag populations using the three basic operations shown in Figure 6.28. Each of these operations comprises one or more commands. The operations are defined as follows:

a. **Select (ASK and PJM Method):** The process by which an interrogator selects a tag population for inventory and access. Interrogators may use one or more Select commands to select a particular tag population prior to inventory.

b. **Inventory ASK Method:** The process by which an interrogator identifies tags. An interrogator begins an inventory round by transmitting a *BeginRound* command in one of two sessions. One or more tags may reply. The interrogator detects a single tag reply and requests the PC word XPC_W1 if XI is set and optionally XPC_W2, EPC, and PacketCRC is required (see Table 6.15) from the tag. An inventory round operates in one and only one session at a time. Annex E shows an example of an interrogator inventorying and accessing a single tag.

   Inventory PJM Method: The process by which an interrogator identifies tags. An interrogator begins an inventory round by transmitting a *BeginRound* command in one of two sessions. One or more tags may reply. The interrogator detects one or more tag replies and requests the PC, EPC, and CRC-16 from the tag or tags. An inventory round operates in one and only one session at a time. Annex E shows an example of an interrogator inventorying and accessing a single tag or multiple tags.

c. **Access ASK Method:** The process by which an interrogator transacts with (reads from or writes to) individual tags. An individual tag must be uniquely identified prior to access. Access comprises multiple commands, some of which may employ one-time-pad based cover-coding of the R=>T link.

   Access PJM Method: The process by which an interrogator transacts with (reads from or writes to) individual or multiple tags. Tags must be uniquely identified prior to access. Access comprises multiple commands, some of which may employ one-time, pad based cover-coding of the R=>T link.

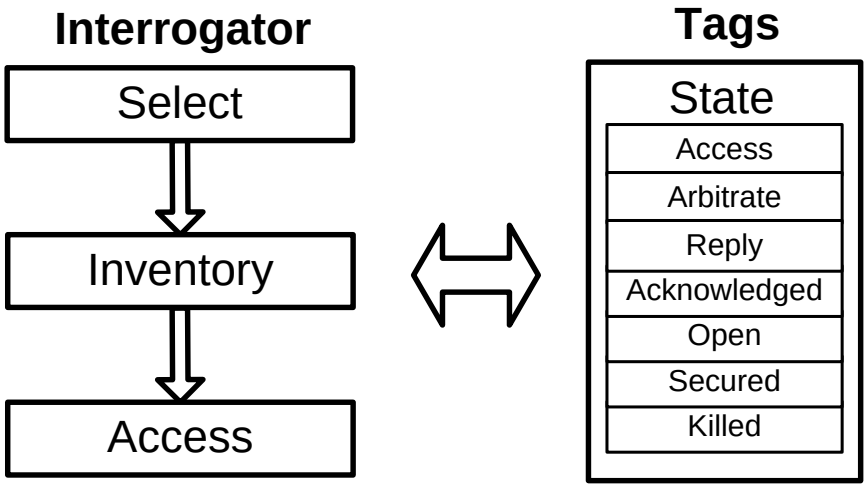


Figure 6.28 Interrogator/tag operations and tag state

6.3.4.7 ASK and PJM Method: Selecting tag populations

The selection process employs a single command, *Select,* which an interrogator may apply successively to select a particular tag population based on user-defined criteria, enabling union (U), intersection (∩), and negation (~) based tag partitioning. Interrogators perform U and ∩ operations by issuing successive *Select* commands. *Select* can assert or deassert a tag **SL** flag, or it can set a tag **inventoried** flag to either *A* or *B* in any one of the four sessions. *Select* contains the parameters Target, Action, MemBank, Pointer, Length, Mask, and Truncate.

- Target and Action indicate whether and how a *Select* modifies a tag **SL** or **inventoried** flag, and in the case of the **inventoried** flag, for which session. A *Select* that modifies **SL** shall not modify **inventoried,** and vice versa.

- MemBank specifies if the mask applies to EPC, TID, or User memory. *Select* commands apply to a single memory bank. Successive *Selects* may apply to different memory banks.
- Pointerlength, Pointer, Length, and Mask: Pointer and Length describe a memory range. Pointer references a memory bit address (Pointer is not restricted to word boundaries) and has a length of 8, 16, 24 or 32 bits as defined in Pointerlength. Length is 8-bits, allowing Masks from 0 to 255 bits in length. Mask, which is Length bits long, contains a bit string that a tag compares against the memory location that begins at Pointer and ends Length bits later.
- Truncate specifies whether a tag loadmodulates its entire EPC, or only that portion of the EPC immediately following Mask. Truncated EPCs are always followed by the Tag's StoredCRC (see Table 6.15). A tag does not recompute its StoredCRC for a truncated reply.

By issuing multiple identical *Select* commands an interrogator can asymptotically single out all tags matching the selection criteria even though tags may undergo short-term RF fades.

A *BeginRound* command uses **inventoried** and **SL** to decide which tags participate in an inventory. Interrogators may inventory and access **SL** or ~**SL** tags, or they may choose to ignore the **SL** flag entirely.

6.3.4.8 ASK and PJM Method: Inventorying tag populations

The inventory command set includes *BeginRound, ResizeRound, NextSlot, ACK,* and *NAK. BeginRound* initiates an inventory round and decides which tags participate in the round (where "inventory round" is defined as the period between successive *BeginRound* commands).

BeginRound contains a slot-count parameter *Q.* Upon receiving a *BeginRound* participating tags shall pick a random value in the range (0 to 2^Q–1), inclusive, and shall load this value into their slot counter. Tags that pick a zero shall transition to the **reply** state and reply immediately. Tags that pick a non-zero value shall transition to the **arbitrate** state and await a *ResizeRound* or a *NextSlot* command. Assuming that a single tag replies, the query-response algorithm proceeds as follows:

a. The tag loadmodulates the StoredCRC as it enters **reply,**

b. The interrogator acknowledges the tag with an *ACK* containing this same StoredCRC,

c. The acknowledged tag transitions to the **acknowledged** state, loadmodulating the reply shown in Table 6.15,

d. The interrogator issues a *ResizeRound* or *NextSlot,* causing the identified tag to set its **inventoried** flag (*A*→*B*) and transition to **ready,** and potentially causing another tag to initiate a query/response dialog with the interrogator, starting in step (a), above.

If the tag fails to receive the *ACK* in step (b) within time T_2 (see Figure 6.24 and Figure 6.27), or receives the *ACK* with an erroneous StoredCRC, it shall return to **arbitrate.**

If multiple tags reply in step (a) but the interrogator, by detecting and resolving collisions at the waveform level, can resolve a StoredCRC from one of the tags, the interrogator can *ACK* the resolved tag. Unresolved tags receive erroneous StoredCRC s and return to **arbitrate** without loadmodulating the reply shown in Table 6.15.

If the interrogator sends a valid *ACK* (*i.e.* an *ACK* containing the correct StoredCRC) to the tag in the **acknowledged** state, the tag shall loadmodulate the reply shown in Table 6.15.

At any point the interrogator may issue a *NAK,* in response to which all tags in the inventory round shall return to **arbitrate** without changing their **inventoried** flag.

After issuing a *BeginRound* to initiate an inventory round, the interrogator typically issues one or more *ResizeRound* or *NextSlot* commands. *ResizeRound* repeats a previous *BeginRound* and may increment or decrement *Q,* but does not introduce new tags into the round. *NextSlot* repeats a previous *BeginRound* without changing any parameters and without introducing new tags into the round. An inventory round can contain multiple *ResizeRound* or *NextSlot* commands. At some point the interrogator shall issue a new *BeginRound,* thereby starting a new inventory round.

Tags in the **arbitrate** or **reply** states that receive a *ResizeRound* first adjust *Q* (increment, decrement, or leave unchanged), then pick a random value in the range (0 to 2^Q–1), inclusive, and load this value into their slot counter. Tags that pick zero shall transition to the **reply** state and reply immediately. Tags that pick a non-zero value shall transition to the **arbitrate** state and await a *ResizeRound* or a *NextSlot* command.

Tags in the **arbitrate** state decrement their slot counter every time they receive a *NextSlot,* transitioning to the **reply** state and loadmodulate the StoredCRC when their slot counter reaches 0000_h. Tags whose slot counter reached 0000_h, who replied, and who were not acknowledged (including tags that responded to the original *BeginRound* and were not acknowledged), shall return to **arbitrate** with a slot value of 0000_h and shall decrement this slot value from 0000_h to $7FFF_h$ at the next *NextSlot*, thereby effectively preventing subsequent replies until the tag loads a new random value into its slot counter. Tags shall reply at least once in 2^Q–1 *NextSlot* commands.

Annex D describes an exemplary interrogator algorithm for choosing *Q.*

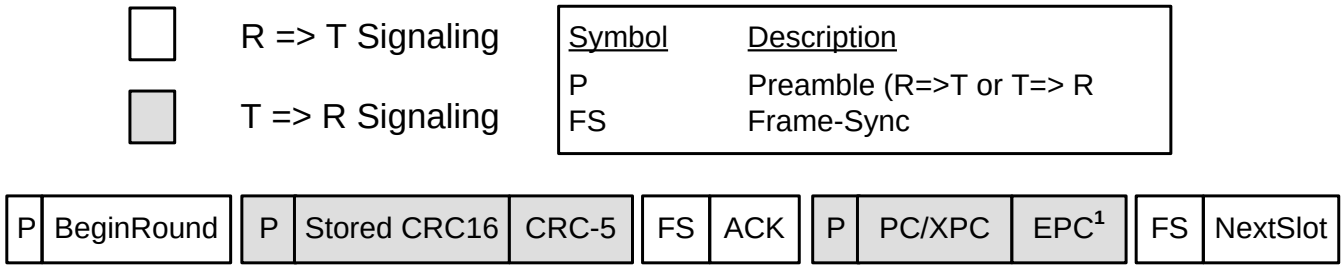
The scenario outlined above assumed a single interrogator operating in a single session. However, as described in 6.3.4.2, an interrogator can inventory a tag population in one of two sessions. Furthermore, as described in 6.3.4.11.2, the *BeginRound, ResizeRound,* and *NextSlot* commands, each contain a session parameter. How a tag responds to these commands varies with the command, session parameter, and tag state, as follows:

- *BeginRound*: A *BeginRound* command starts an inventory round and chooses the session for the round. Tags in any state except **killed** shall execute a *BeginRound*, starting a new round in the specified session and transitioning to **ready**, **arbitrate**, or **reply**, as appropriate (see Figure 6.27).
  - If a tag in the **acknowledged**, **open**, or **secured** states receives a *BeginRound* whose session parameter matches the prior session it shall set its **inventoried** flag (*A*→*B*) for the session before it evaluates whether to transition to **ready**, **arbitrate**, or **reply**.
  - If a tag in the **acknowledged**, **open**, or **secured** states receives a *BeginRound* whose session parameter does not match the prior session it shall leave its **inventoried** flag for the prior session unchanged as it evaluates whether to transition to **ready**, **arbitrate**, or **reply**.
- *ResizeRound*, *NextSlot*: Tags in any state except **ready** or **killed** shall execute a *ResizeRound* or *NextSlot* command if, and only if, the session parameter in the command matches the session parameter in the *BeginRound* that started the round. Tags shall ignore a *ResizeRound* or *NextSlot* with mismatched session.
  - If a tag in the **acknowledged**, **open**, or **secured** states receives a *ResizeRound* or *NextSlot* whose session parameter matches the session parameter in the prior *BeginRound*, it shall set its **inventoried** flag (*A*→*B*) for the current session then transition to **ready**.

To illustrate an inventory operation, consider a specific example: Assume a population of 64 powered tags in the **ready** state. An interrogator first issues a *Select* to select a subpopulation of tags. Assume that 16 tags match the selection criteria. Further assume that 12 of the 16 selected tags have their **inventoried** flag set to *A* in session S0. The interrogator issues a *BeginRound* specifying (**SL,** *Q*=4, S0). Each of the 12 tags picks a random number in the range (0 - 15) and loads the value into its slot counter. Tags that pick a zero respond immediately. The *BeginRound* has 3 possible outcomes:

1) **No tags reply:** The interrogator may issue another *BeginRound,* or it may issue a *ResizeRound* or *NextSlot.*

2) **One tag replies** (see Figure 6.29): The tag transitions to the **reply** state and loadmodulates the StoredCRC. The interrogator acknowledges the tag by sending an *ACK.* If the tag receives the *ACK* with a correct StoredCRC it loadmodulates the reply shown in Table 6.15 and transitions to the **acknowledged** state. The diagram in Figure 6.29 assumes that XI is deasserted. If the tag receives the *ACK* with an incorrect StoredCRC, it transitions to **arbitrate.** Assuming a successful *ACK,* the interrogator may either access the acknowledged tag or issue a *ResizeRound* or *NextSlot* to set the tag **inventoried** flag from *A*→*B* and send the tag to **ready** (a *BeginRound* with matching prior-round session parameter shall also set the **inventoried** flag from *A*→*B*).



Note 1: In certain cases a PacketCRC shall be added if required. See Table 6.15 for the definition of the cases.

Figure 6.29 One tag reply

3) **Multiple tags reply:** The interrogator observes a loadmodulated waveform comprising multiple StoredCRC s. It may try to resolve the collision and issue an *ACK;* not resolve the collision and issue a *ResizeRound, NextSlot,* or *NAK;* or quickly identify the collision and issue a *ResizeRound* or *NextSlot* before the collided tags have finished loadmodulation. In the latter case the collided tags, not observing a valid reply within T_2 (see Figure 6.24), shall return to **arbitrate** and await the next *BeginRound* or *ResizeRound* command.

PJM Method: Interrogators may receive multiple tag replies on multiple channels in response to parameters in the *BeginRound, NextSlot or ResizeRound* command. It is possible that such an interrogator may receive (without any clashing error) up to eight tags per *BeginRound, NextSlot or ResizeRound* command. However, with a correctly set Q, compared with the unknown tag population, the average reception rate for an interrogator equipped with eight reply channels would be as follows, per *BeginRound, NextSlot or ResizeRound* command:

- three channels with one tag in each (therefore three tags correctly received, for example, tag 1 on channel A, tag 2 on channel B and tag 3 on channel E),
- one channel with two tags clashing,
- one channel with three tags clashing, and
- three empty channels.

Channel selection is determined by the StoredCRC (see clause 6.3.3.1.3.11) and the RN16 is produced by the RNG (see clause 6.3.4.5).

If three tags are correctly received the interrogator can acknowledge all three tags by sending a single *ACK-PJM*, see 6.3.4.11.2.4. In the current example, the *ACK-PJM* includes the three tag StoredCRC s that have just been received. At the completion of the *ACK-PJM,* if a tag has received a correct StoredCRC it loadmodulates its PC, EPC and PacketCRC if required (see Table 6.15) using the same reply channel used to respond to the previous *BeginRound, NextSlot* or *ResizeRound* command. The tag then transitions to the **acknowledged** state. In this case the three tags shall simultaneously reply their PC, EPC and PacketCRC if required (see Table 6.15) each on a different channel (tag 1 on channel A, tag 2 on channel B and tag 3 on channel E). If required, the interrogator can correlate the tag StoredCRCs to the EPC codes, because each tag uses the same channel for replies to the *BeginRound, NextSlot* or *ResizeRound* commands and the subsequent *ACK-PJM*.

If a tag receives the *ACK-PJM* where all StoredCRCs are incorrect, it transitions to **arbitrate**.

Assuming a successful *ACK-PJM* the interrogator may either access the acknowledged tags or issue a *ResizeRound* or *NextSlot* to set the tag's **inventoried** flags from *A→B* and send the tags to **ready** (a *BeginRound* with prior-round session parameter shall also set the **inventoried** flags from *A→B*).

The tags referred to above are the types that are equipped to receive PJM commands.

6.3.4.9 ASK and PJM Method: Accessing individual tags

After acknowledging a tag, an interrogator may choose to access it. The access command set comprises *Req_RN, Read, Write, Kill, Lock, Access, BlockWrite, BlockErase* and *BlockPermalock.* Tags execute *Req_RN* from the **acknowledged, open,** or **secured** states. Tags execute *Read, Write, Kill, Access, BlockWrite,* and *BlockErase* from the **open** or **secured** states. Tags execute *Lock* only from the **secured** state.

An interrogator accesses a tag in the **acknowledged** state as follows:

Step 1. The interrogator issues a Req_RN to the acknowledged tag.

Step 2. The tag generates and stores an RN16 (denoted handle), loadmodulates the handle, and transitions to the open state if its access password is non-zero, or transitions to the secured state if its access password is zero. The interrogator may now issue further access commands.

All access commands issued to a tag in the **open** or **secured** states include the tag handle as a parameter in the command. When in either of these two states, tags shall verify that the handle is correct prior to executing an access command, and shall ignore access commands with an incorrect handle. The handle value is fixed for the entire duration of an access sequence.

Tags in the **open** state can execute all access commands except *Lock* and *BlockPermalock.* Tags in the **secured** state can execute all access commands. A tag response to an access command includes, at a minimum, the tag handle; the response may include other information as well (for example, the result of a *Read* operation).

An interrogator may issue an *ACK* to a tag in the **open** or **secured** states, causing the tag to loadmodulate the reply shown in Table 6.15

Interrogator and tag can communicate indefinitely in the **open** or **secured** states. The interrogator may terminate the communications at any time by issuing a *BeginRound, ResizeRound, NextSlot,* or a *NAK.* The tag response to a *BeginRound, ResizeRound,* or *NextSlot* is described in 6.3.4.8. A *NAK* causes all tags in the inventory round to return to **arbitrate** without changing their **inventoried** flag.

The *Write, Kill,* and *Access* commands send 16-bit words (either data or half-passwords) from interrogator to tag. These commands could optionally use one-time, pad-based link cover-coding to obscure the word being transmitted as follows:

Step 1. The interrogator issues a Req_RN, to which the tag responds by loadmodulating a new RN16. The interrogator then generates a 16-bit cover-coded text string comprising a bit-wise EXOR of the 16-bit word to be transmitted with this new RN16, both MSB first, and issues the command with this cover-coded text string as a parameter.

Step 2. The tag decrypts the received cover-coded text string by performing a bit-wise EXOR of the received 16-bit cover-coded text string with the original RN16.

An interrogator shall not use handle for cover-coding purposes.

An interrogator shall not re-use an RN16 for cover-coding, if it is used. If an interrogator reissues a command that contained cover-coded data, then the interrogator shall reissue the command unchanged. If the interrogator changes the data and uses cover-coding, then it shall first issue a Req_RN to obtain a new RN16 and shall use this new RN16 for cover-coding.

If no RN16 is requested the following 16 bit words (either data or half-passwords) shall be transmitted in plain text without cover-coding. The interrogator shall decide at the beginning of an access sequence whether to use cover-coding or not. If a tag does not support cover-coding it shall respond with 0000_h to each Req_RN command which requests a RN16 for cover-coding.

Table 6.18. Access commands and tag states in which they are permitted

Command	State			Remark
	Acknowledged	Open	Secured	
Req_RN	Permitted	Permitted	Permitted	-
Read	-	Permitted	Permitted	-
Write	-	Permitted	Permitted	Optional prior *Req_RN*
Kill	-	Permitted	Permitted	Optional prior *Req_RN*
Lock	-	-	Permitted	
Access	-	Permitted	Permitted	Optional command: Optional prior *Req_RN*
BlockWrite	-	Permitted	Permitted	Optional command
BlockErase	-	Permitted	Permitted	Optional command
BlockPermalock	-	-	Permitted	Optional command

The *BlockWrite* command (see Table 6.48) communicates multiple 16-bit words from interrogator to tag. Unlike *Write, BlockWrite* does not use link cover-coding.

A Tag responds to a command that writes to or erases memory (*i.e. Write*, *Kill*, *Lock*, *BlockWrite*, *BlockErase*, and *BlockPermalock* with Read/Lock=1 (see 6.3.4.11.3.9)) by loadmodulating its handle, indicating that the operation was successful, or by loadmodulating an error code (see Annex I), indicating that the operation was unsuccessful. The Tag reply uses the preamble shown in Figure 6.11, Figure 6.15, and Figure 6.19, as appropriate. See 6.3.4.11.3 for detailed descriptions of a Tag's reply to each particular access command.

Issuing an access password to a Tag is a multi-step procedure described in 6.3.4.11.3.6 and outlined in Figure 6.33.

Tag memory may be unlocked or locked. The lock status may be changeable or permalocked (*i.e.* permanently unlocked or permanently locked). Recommissioning the Tag may change the lock status, even if the memory was previously permalocked. An Interrogator may write to unlocked memory from either the **open** or **secured** states. An Interrogator may write to locked memory that is not permalocked from the **secured** state only. See 6.3.4.10, 6.3.4.11.3.5, 6.3.4.11.3.9, Table 6.43, and Table 6.53 for a detailed description of memory lock, permalock, recommissioning, and the Tag state required to modify memory.

Killing a tag is a multi-step procedure, described in 6.3.4.11.3.4 and outlined in Figure 6.31.

Issuing an access password to a tag is a multi-step procedure, described in 6.3.4.11.3.6 and outlined in Figure 6.33.

Interrogator and tag shall transmit all strings MSB first.

Interrogators shall not power-off while a tag is in the **reply, acknowledged, open** or **secured** states. Interrogators should end their dialog with a tag before powering off, leaving the tag in either the **ready** or **arbitrate** state.

PJM Method: Interrogators can communicate to and receive replies from multiple tags simultaneously.

Note: This is specific to PJM multi-channel method. For *Kill* and *Access* commands the interrogator shall communicate with only one tag at a time. However for *Req_RN, Read, Write, Lock, BlockWrite, BlockErase* and *BlockPermalock* commands the interrogator can extend the RN field to include multiple StoredCRCs or handles, see 6.3.4.11, such that the interrogator can communicate with multiple tags at a time. A tag shall reply to such commands using the same channel that was used for its reply to the previous *BeginRound, NextSlot* or *ResizeRound* command.

6.3.4.10 Killing (optional) or recommissioning a Tag

Killing or recommissioning a Tag is a multi-step procedure, described in 6.3.4.11.3.4 and shown in Figure 6.31, in which an Interrogator sends two successive *Kill* commands to a Tag. The first *Kill* command contains the first half of the kill password, and the second *Kill* command contains the second half. Each *Kill* command also contains 3 RFU/Recom bits. In the first *Kill* command these bits are RFU and zero valued; in the second *Kill* command they are called recommissioning (or Recom) bits and may be nonzero valued. The procedures for killing or recommissioning a Tag are identical, except that the recommissioning bits in the second *Kill* command are zero when killing a Tag "dead", and are nonzero when recommissioning it. Regardless of the intended operation, a Tag shall not kill or re-commission itself without first receiving the correct kill password by the procedure shown in Figure 6.31.

If a Tag does not implement Recom bits LSB and 2SB, then it shall ignore these recommissioning bits and treat them as though they were zero; If the Tag receives a properly formatted *Kill* command sequence with the correct kill password and all three Recom bits set to zero and it does not support this command as it is optional it shall interpret it in the same way as with 3SB of the Recom bits is set to one. The remainder of this section 6.3.4.10 assumes that a Tag implements Recom bits LSB and 2SB.

Upon receiving a properly formatted *Kill* command sequence with the correct kill password and one or more non-zero recommissioning bits, a Tag shall assert those LSBs of its XPC_W1 word that are asserted in the recommissioning bits (for example, if a Tag receives 100_2 for the recommissioning bits, then it asserts the 3SB of its XPC_W1 word). The XPC LSBs shall be one-time-writeable, meaning that they cannot be deasserted after they are asserted. By storing the Tag's recommissioned status in the XPC_W1 word, a subsequent Interrogator can know how the Tag was re-commissioned (see Table 6.16). A Tag shall perform the following operations based on the recommissioning bit values it receives (see also 6.3.4.1.2.5):

- **Asserted LSB**: The Tag shall disable block permalocking and unlock any blocks of User memory that were previously permalocked. The Tag shall disable support for the the BlockPermalock command. If the Tag did not implement block permalocking prior to recommissioning, then block permalocking shall remain disabled. The lock status of User memory shall be determined solely by the lock bits (see 6.3.4.11.3.5).
- **Asserted 2SB**: The Tag shall render its User memory inaccessible; causing the entire memory bank to become unreadable, unwriteable, and unselectable (*i.e.* the Tag functions as though its User memory bank no longer exists). The 2SB has precedence over the LSB — if both are asserted then User memory is inaccessible.
- **Asserted 3SB**: The Tag shall unlock its EPC, TID, and User memory banks, regardless of whether these banks were locked or permalocked. Portions of User memory that were block permalocked shall remain block permalocked, and vice versa; unless the LSB is also asserted, the Tag shall unlock its permalocked blocks. The Tag shall write-unlock its kill and access passwords The Read/Lock state of the kill and access passwords shall be the same as prior to the recommissioning.. If an Interrogator subsequently attempts to read the Tag's kill or access passwords the Tag shall loadmodulate an error code (see Annex I) if the kill or access password was unreadable prior to the recommisioning. A Tag that receives a subsequent Lock command with pwd-read/write=0 shall lock or permalock the indicated password(s) in the writeable state.

A Tag shall not execute any of the above recommissioning operations more than once. As one example, a Tag does not allow any of its memory banks to be unlocked more than once by recommissioning.

A Tag may execute multiple *Kill* command sequences, depending on the nature and ordering of the operations specified in these command sequences. Specifically:

- A Tag that is killed shall not allow a subsequent recommissioning.
- A previously recommissioned Tag that receives a properly formatted *Kill* command sequence with the correct kill password and Recom=000_2 shall be killed.
- A Tag that receives a properly formatted *Kill* command sequence with the correct kill password, but with redundant recommissioning bits (for example, the recommissioning bits are 100_2 but the Tag's XPC_W1 word already contains 100_2), shall not perform the requested recommissioning operation again. Instead, the Tag shall merely verify that its XPC_W1 word contains the asserted values and respond affirmatively to the Interrogator. An Interrogator may choose to send a Kill sequence with redundant recommissioning bits if, for example, it had sent a prior Kill sequence but did not observe an affirmative response from the Tag.
- A Tag that receives a properly formatted *Kill* command sequence with the correct kill password and with newly asserted recommissioning bits shall perform the recommissioning operation indicated by the newly asserted

bits, responding affirmatively to the Interrogator when done. A Tag shall compute the logical OR of the LSBs of its current XPC_W1 word and the recommissioning bits, and shall store the resulting value into its XPC_W1 word. For example, if a Tag whose XPC_W1 LSBs are 100_2 receives a *Kill* command sequence whose recommissioning bits are 010_2, the Tag renders its User memory bank inaccessible and stores 110_2 into its XPC_W1 LSBs.

An Interrogator may subsequently re-lock any of the memory banks or passwords that have been unlocked by recommissioning.

Portions or entire banks of Tag memory, if factory locked, may not be unlocked by recommissioning. An Interro-gator may determine whether a Tag supports Recom bits LSB and 2SB, and if so which (if any) memory portions are not recommissionable, by reading the Tag's TID memory prior to recommissioning.

A Tag that does not implement a kill password, or a Tag whose kill password is zero, shall not execute a kill or recommissioning operation. Such a Tag shall respond with an error code (as shown in Figure 6.23) to a Kill com-mand sequence regardless of the RFU or recommissioning bit settings.

A Tag shall accept all eight possible combinations of the three recommissioning bits if Recom bits LSB and 2SB are supported, executing those portions that it is capable of executing, ignoring those it cannot, and responding affirmatively to the Interrogator when done. Several examples of operations that a Tag may be incapable or partially capable of executing are as follows:

- a Tag that does not have User memory cannot unlock it,
- a Tag that does not implement an Access password cannot unlock it for writing, and
- a Tag in which a portion of TID memory is factory locked cannot unlock this portion.

6.3.4.11 ASK and PJM Method: Interrogator commands and tag replies

Interrogator-to-tag commands shall have the format shown in Table 6.19.

- *ACK* has a 2-bit command code 01_2.
- *BeginRound*, *NextSlot*, *ResizeRound*, and *Select* have 4-bit command codes (see Table 6.19).
- All other base commands have 8-bit command codes beginning with 110_2.
- All extended commands have 16-bit command codes beginning with 1110_2.
- *NextSlot*, *ACK*, *BeginRound*, *ResizeRound*, and *NAK* have the unique command lengths shown in Table 6.19. No other commands shall have these lengths. If a tag receives one of these commands with an incorrect length it shall ignore the command.
- BeginRound, ResizeRound, and NextSlot contain a session parameter.
- *BeginRound* is protected by a CRC-5, shown in Table 6.22 and detailed in Annex F.
- *Select*, *Req_RN*, *Read, Write, Kill, Lock, Access, BlockWrite, BlockErase* and *BlockPermalock* are protected by a CRC-16c, defined in 6.3.4.1.4 and detailed in Annex F.
- R=>T commands begin with either a preamble or a frame-sync, as described in 6.3.3.1.3.4, 6.3.3.1.3.6 and 6.3.3.1.3.8. The command code lengths specified in Table 6.19 do not include the preamble or frame-sync.
- Tags shall ignore invalid commands. In general, "invalid" means a command that (1) is incorrect given the current tag state, (2) is unsupported by the tag, (3) has incorrect parameters, (4) has a CRC error, (5) specifies an incorrect session, or (6) is in any other way not recognized or not executable by the tag. The actual definition of "invalid" is state-specific and defined, for each tag state, in Annex B and Annex C.

Table 6.19. Commands

Command	Code	Length (bits)	Mandatory?	Protection
ACK (ASK Method)	01	18	Yes	Unique command length
ACK (PJM Method)	01	>/= 23	Yes	CRC-5
NextSlot	0000	6	Yes	Unique command length
BeginRound	1000	22	Yes	Unique command length and a CRC-5
ResizeRound	1001	9	Yes	Unique command length
Select	1010	> 46	Yes	CRC-16c
Reserved for future use	1011	–	–	–
NAK	11000000	8	Yes	Unique command length
Req_RN	11000001	40	Yes	CRC-16c
Read	11000010	> 59	Yes	CRC-16c
Write	11000011	> 60	Yes	CRC-16c
Kill [1]	11000100	59	Yes	CRC-16c
Lock	11000101	60	Yes	CRC-16c
Access	11000110	56	No	CRC-16c
BlockWrite	11000111	> 59	No	CRC-16c
BlockErase	11001000	> 59	No	CRC-16c
BlockPermalock	11001001	> 68	No	CRC-16c
Reserved for future use	11001010 … 11011111	–	–	–
Reserved for custom commands	11100000 00000000 … 11100000 11111111	–	–	Manufacturer specified
Reserved for proprietary commands	11100001 00000000 … 11100001 11111111	–	–	Manufacturer specified
Reserved for future use	11100010 00000000 … 11101111 11111111	–	–	–

Note 1: For the *Kill* command only the support of asserted Recom bit 3SB is mandatory. See section 6.3.4.10

PJM Method: The *ACK-PJM*, *Req_RN, Read, Write, Lock, BlockWrite, BlockErase* and *BlockPermalock* commands can include multiple StoredCRCs or handles (within the RN field). PJM Method *Kill* and *Access* commands only include a single RN16 or handle.

Therefore for PJM Method the *ACK-PJM, Req_RN, Read, Write, Lock, BlockWrite, BlockErase* and *BlockPermalock* commands may have different lengths than those shown in Table 6.19. The length of these commands shown in the Table only include a single StoredCRC or handle. This length (or minimum length) shall be increased by 16 bits for each additional StoredCRC or handle.

R=>T commands begin with either a flag (preamble or a frame-sync), as described in 6.3.3.1.3.10. The command code lengths specified in Table 6.19 do not include the flag.

6.3.4.11.1 Select commands

The Select command set comprises a single command: *Select.*

6.3.4.11.1.1 *Select* (mandatory)

The use of the multiple *Select* commands is optional.

Select selects a particular tag population based on user-defined criteria, enabling union (U), intersection (∩), and negation (~) based tag partitioning. Interrogators perform U and ∩ operations by issuing successive *Select* commands. *Select* can assert or deassert a tag **SL** flag, which applies across all four sessions, or it can set a tag **inventoried** flag to either *A* or *B* in any one of the four sessions.

Interrogators and tags shall implement the *Select* command shown in Table 6.20. Target shall indicate whether the *Select* modifies a tag **SL** or **inventoried** flag, and in the case of the **inventoried** flag, for which session. Action shall elicit the tag response shown in 0. The criteria for determining whether a tag is matching or non-matching are specified in the MemBank, Pointer, Length and Mask fields. Truncate indicates whether a tag transmitted reply shall be truncated to include only those EPC and StoredCRC bits that follow Mask. *Select* passes the following parameters from interrogator to tags:

- Target indicates whether the *Select* command modifies a tag **SL** flag or its **inventoried** flag, and in the case of **inventoried** it further specifies one of four sessions. A *Select* command that modifies **SL** does not modify **inventoried,** and vice versa. Class-1 tags shall ignore *Select* commands whose Target is 101_2, 110_2, or 111_2.
- Action indicates whether matching tags assert or deassert **SL,** or set their **inventoried** flag to *A* or to *B.* Tags conforming to the contents of the MemBank, Pointer, Length, and Mask fields are considered matching. Tags not conforming to the contents of these fields are considered non-matching.
- MemBank specifies whether Mask applies to EPC, TID, or User memory. *Select* commands shall apply to a single memory bank. Successive *Selects* may apply to different banks. MemBank shall not specify Reserved memory; if a tag receives a *Select* specifying MemBank=00_2 it shall ignore the *Select.* MemBank parameter value 00_2 is reserved for future use (RFU).
- PointerLength, Pointer, Length, and Mask: Pointer and Length describe a memory range. Pointer references a memory bit address (Pointer is not restricted to word boundaries) and has a length of 8,16,24 or 32 bits as defined in Pointerlength. Length is 8-bits, allowing Masks from 0 to 255 bits in length. Mask, which is Length bits long, contains a bit string that a tag compares against the memory location that begins at Pointer and ends Length bits later. If Pointer and Length reference a memory location that does not exist on the tag, then the tag shall consider the *Select* to be non-matching. If Length is zero then all tags shall be considered matching, unless Pointer references a memory location that does not exist on the tag, or Truncate=1 and Pointer is outside the EPC specified in the StoredPC, in which case the tag shall consider the *Select* to be non-matching.
- Truncate: If an interrogator asserts Truncate, and if a subsequent *BeginRound* specifies Sel=10 or Sel=11, then a matching tag shall truncate its reply to an *ACK* to that portion of the EPC immediately following Mask, if remainder of EPC length > 0.

  Interrogators shall assert Truncate:

  - in the last (and only in the last) *Select* that the interrogator issues prior to sending a *BeginRound*,
  - only if the *Select* has *Target*=100_2, and
  - only if Mask ends in the EPC.

  These constraints *do not* preclude an interrogator from issuing multiple *Select* commands that target the **SL** and/or **inventoried** flags. They *do* require that an interrogator assert Truncate only in the last *Select,* and only if this last *Select* targets the **SL** flag. Tags shall power-up with Truncate deasserted.

  Tags shall decide whether to truncate their loadmodulated EPC on the basis of the most recently received *Select.* If a tag receives a *Select* with Truncate=1 but Target<>100_2 the tag shall ignore the *Select.* If a tag receives a *Select* in which Truncate=1 but MemBank<>01, the tag shall consider the *Select* to be invalid. If a tag receives a *Select* in which Truncate=1, MemBank=01, but Mask ends outside the EPC specified in the StoredPC, the tag shall consider the *Select* to be not matching.

Mask may end at the last bit of the EPC, in which case a selected tag shall loadmodulate SOF followed by 00000_2.and EOF. Truncated replies never include XPC_W1 or an XPC_W2, because Mask must end in the EPC. A recommissioned tag shall not truncate its replies.

A recommissioned tag that receives a *Select* with Truncate=1 shall evaluate the *Select* normally, but when replying to a subsequent *ACK* it shall loadmodulate its PacketPC, XPC_W1, optionally its XPC_W2 (if XEB is asserted), an EPC whose length is as specified in the EPC length field in the StoredPC, and its PacketCRC. Interrogators can use a *Select* command to reset all tags in a session to **inventoried** state *A,* by issuing a *Select* with Action = 000_2 and a Length value of zero.

Because a Tag stores its StoredPC and StoredCRC in EPC memory, a *Select* command may select on them.

Because a Tag computes its PacketPC and PacketCRC dynamically and does not store them in memory, a *Select* command is unable to select on them.

Because a Tag may compute its PC and/or CRC dynamically, its response to a *Select* command whose Pointer, Length, and Mask include the StoredPC or StoredCRC may produce unexpected behavior. Specifically, a Tag's loadmodulated reply may appear to not match Mask even though the Tag's behavior indicates matching, and vice versa. For example, suppose an Interrogator sends a *Select* to match a 00100_2 EPC length field in the StoredPC. Further assume that a Tag matches, but has an asserted XI. The Tag will dynamically increment its EPC length field to 00101_2 when responding to an *ACK*, and will loadmodulate this incremented value in the PacketPC. The Tag was matching, but the loadmodulated EPC length field appears to be non-matching.

Interrogators shall prefix a *Select* with a frame-sync (see 6.3.4.3). The CRC-16c that protects a Select is calculated over the first command-code bit to the Truncate bit.

Tags shall not reply to a *Select.*

Note: The interrogator shall not transmit Target 001 and 011.

Table 6.20. *Select* command

	Command	Target	Action	Mem Bank	Pointer length	Pointer	Length	Mask	Truncate	CRC-16c
# of bits	4	3	3	2	2	8, 16, 24 or 32	8	Variable	1	16
Description	1010	000: **Inventoried** (S0) LSB always 0 001: Not Permitted 010: **Inventoried** (S2) LSB always 0 011: Not Permitted 100: **SL** 101: RFU 110: RFU 111: RFU	See Table 6.21	00: RFU 01: EPC 10: TID 11: User	Length of pointer 00: 8 bit 01: 16 bit 10: 24 bit 11: 32 bit	Starting Mask address	Mask length (bits)	Mask value	0: Disable truncation 1: Enable truncation	

Table 6.21. Tag response to Action parameter

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

6.3.4.11.2 Inventory commands

The inventory command set comprises *BeginRound, ResizeRound, NextSlot, ACK,* and *NAK.*

6.3.4.11.2.1 *BeginRound* (mandatory)

Interrogators and tags shall implement the *BeginRound* command shown in Table 6.22. *BeginRound* initiates and specifies an inventory round. *BeginRound* includes the following fields:

- DR (divide ratio) sets the T=>R link frequency as described in 6.3.3.1.3.9 and Table 6.9,
- M (cycles per symbol) sets the T=>R data rate and modulation format as shown in Table 6.10,
- TRext chooses whether the T=>R preamble is prefixed with a pilot tone as described in 6.3.3.1.3.4, 6.3.3.1.3.6 and 6.3.3.1.3.8,
- Sel chooses which tags respond to the *BeginRound* (see 6.3.4.11.1.1 and 6.3.4.8),
- Session chooses a session for the inventory round (see 6.3.4.8),
- Q sets the number of slots in the round (see 6.3.4.8).

Interrogators shall prefix a *BeginRound* with a preamble.

The CRC-5 that protects a *BeginRound* is calculated over the first command-code bit to the last Q bit. If a tag receives a *BeginRound* with a CRC-5 error, it shall ignore the command.

Upon receiving a *BeginRound,* tags with matching *Select* and Target pick a random value in the range (0 to 2^{Q}–1), inclusive, and load this value into their slot counter. If a tag, in response to the *BeginRound,* loads its slot counter with zero, then its reply to a *BeginRound* shall be as shown in Table 6.23; otherwise the tag shall remain silent.

A *BeginRound* may initiate an inventory round in a new session, or in the prior session. If a tag in the **acknowledged, open,** or **secured** states receives a *BeginRound* whose session parameter matches the prior session, it shall set its **inventoried** flag (*A*→*B*) for the session. If a tag in the **acknowledged, open,** or **secured** states receives a *BeginRound* whose session parameter does not match the prior session it shall leave its **inventoried** flag for the prior session unchanged when beginning the new round.

ASK Method: The tags shall support all DR, M, and TRext parameters specified in Table 6.9 and Table 6.10, respectively.

PJM Method: The tags shall interpret DR, M, and TRext parameters as specified in Table 6.11.

Tags in any state other than **killed** shall execute a *BeginRound* command; tags in the **killed** state shall ignore a *BeginRound.*

Table 6.22. *BeginRound* command

	Command	DR	M	TRext	Sel	Session	RFU	Q	CRC-5
# of bits	4	1	2	1	2	2	1	4	5
description	1000	0: FL=424KHz (fc/32) 1: FL=847KHz (fc/16)	00: FM0 01: Miller 8 10: Manchester 2 11: Manchester 4	0: No pilot tone 1: Use pilot tone	00: All 01: All 10: ~**SL** 11: **SL**	00: S0 01: not permitted 10: S2 11: not permitted	0: A 1: *RFU*	0–15	

Table 6.23. Tag reply to a *BeginRound* command

	Reply	CRC-5
# of bits	16	5
description	StoredCRC	

PJM Method: Tags equipped to receive PJM Method shall decode the DR, M, and TRext bits as per Table 6.11.

6.3.4.11.2.2 *ResizeRound* (mandatory)

Interrogators and tags shall implement the *ResizeRound* command shown in Table 6.24. *ResizeRound* adjusts *Q* (*i.e.* the number of slots in an inventory round – see 6.3.4.8) without changing any other round parameters.

ResizeRound includes the following fields:

- Session corroborates the session number for the inventory round (see 6.3.4.8). If a tag receives a *ResizeRound* whose session number is different from the session number in the *BeginRound* that initiated the round, it shall ignore the command.
- UpDn determines whether and how the tag adjusts *Q,* as follows:
  110: Increment *Q* (*i.e. Q*=*Q* + 1)
  000: No change to *Q*
  011: Decrement *Q* (*i.e. Q*=*Q* – 1)

  If a tag receives a *ResizeRound* with an UpDn value different from those specified above it shall ignore the command. If a tag whose *Q* value is 15 receives a *ResizeRound* with UpDn =110 it shall change UpDn to 000 prior to executing the command; likewise, if a tag whose *Q* value is 0 receives a *ResizeRound* with UpDn=011 it shall change UpDn to 000 prior to executing the command.

Tags shall maintain a running count of the current Q value. The initial Q value is specified in the *BeginRound* command that started the inventory round; one or more subsequent *ResizeRound* commands may modify Q.

A *ResizeRound* shall be prefixed with a frame-sync (see 6.3.4.8).

Upon receiving a *ResizeRound* tags first update *Q,* then pick a random value in the range (0 to 2^{Q}–1), inclusive, and load this value into their slot counter. If a tag, in response to the *ResizeRound,* loads its slot counter with zero, then its reply to a *ResizeRound* shall be shown in Table 6.25; otherwise, the tag shall remain silent. Tags shall respond to a *ResizeRound* only if they received a prior *BeginRound.*

Tags in the **acknowledged, open,** or **secured** states that receive a *ResizeRound* set their **inventoried** flag (*A*→*B*) for the current session and transition to ready.

Table 6.24. *ResizeRound* command

	Command	Session	UpDn
# of bits	4	2	3
description	1001	00: S0 01: Not permitted 10: S2 11: Not permitted	110: Q = Q + 1 000: No change to Q 011: Q = Q – 1

Table 6.25. Tag reply to a *ResizeRound* command

	Reply	CRC-5
# of bits	16	5
description	StoredCRC	

6.3.4.11.2.3 *NextSlot* (mandatory)

Interrogators and tags shall implement the *NextSlot* command shown in Table 6.26. *NextSlot* instructs tags to decrement their slot counters and, if slot=0 after decrementing, to loadmodulate the StoredCRC+CRC-5 to the interrogator.

NextSlot includes the following field.

- Session corroborates the session number for the inventory round (see 6.3.4.8 and Table 6.26). If a tag receives a *NextSlot* whose session number is different from the session number in the *BeginRound* that initiated the round, it shall ignore the command.

A *NextSlot* shall be prefixed with a frame-sync (see 6.3.3.1.3.4, 6.3.3.1.3.6 and 6.3.3.1.3.8).

If a tag, in response to the *NextSlot,* decrements its slot counter and the decremented slot value is zero, then its reply to a *NextSlot* shall be as shown in Table 6.27; otherwise the tag shall remain silent. Tags shall respond to a *NextSlot* only if they received a prior *BeginRound.*

Tags in the **acknowledged, open,** or **secured** states that receive a *NextSlot* set their **inventoried** flag ($A \rightarrow B$) for the current session and transition to **ready.**

Table 6.26. *NextSlot* command

	Command	**Session**
# of bits	4	2
description	0000	00: S0 01: Not permitted 10: S2 11: Not permitted

Table 6.27. Tag reply to a *NextSlot* command

	Reply	**CRC-5**
# of bits	16	5
description	StoredCRC	

6.3.4.11.2.4 *ACK* (mandatory)

Interrogators and tags shall implement the *ACK* command shown in Table 6.28. An interrogator sends an *ACK* to acknowledge a single tag. *ACK* echoes the StoredCRC if the Tag is in the reply or in the acknowledged state or the tag handle if the tag is in the **open** or **secured** state.

An *ACK* shall be prefixed with a frame-sync (see 6.3.3.1.3.4, 6.3.3.1.3.6 and 6.3.3.1.3.8).

The tag reply to a successful *ACK* shall be as shown in Table 6.29. As described in 6.3.4.11.1.1, the reply may be truncated. A tag that receives an *ACK* with an incorrect StoredCRC or an incorrect handle (as appropriate) shall return to **arbitrate** without responding, unless the tag is in **ready** or **killed,** in which case it shall ignore the *ACK* and remain in its present state.

Table 6.28. *ACK* command

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

Table 6.29. Tag reply to a successful *ACK* command

	Reply
# of bits	5 to 528
description	See Table 6.15

PJM Method: Refer to the PJM Method part of clauses 6.3.4.8 and 6.3.4.11.

Interrogators can simultaneously receive replies from up to eight tags at a time. This Method allows for these multiply received tags to all be addressed in certain commands by extending the RN field to include multiple StoredCRCs or handles. To acknowledge a tag or tags, this Method uses a *ACK-PJM* shown in Figure 6.24. This command includes one or more echoed StoredCRCs or handles. The selection of StoredCRC or handle shall be as described above for ASK Method.

One or more tags shall reply to a successful *ACK-PJM* as shown in Table 6.29. The tag or tags response to the *ACK-PJM* shall be as described above for ASK Method.

Table 6.30. *ACK-PJM* command

	Command	RN	CRC-5
# of bits	2	Number of tags to be acknowledged multiplied by 16	5
description	01	Echoed StoredCRCs or handles	

6.3.4.11.2.5 *NAK* (mandatory)

Interrogators and tags shall implement the *NAK* command shown in Table 6.31. *NAK* shall return all tags to the **arbitrate** state unless they are in **ready** or **killed,** in which case they shall ignore the *NAK* and remain in their current state.

A *NAK* shall be prefixed with a frame-sync (see 6.3.3.1.3.4, 6.3.3.1.3.6 and 6.3.3.1.3.8).

Tags shall not reply to a *NAK.*

Table 6.31. *NAK* command

	Command
# of bits	8
description	11000000

6.3.4.11.3 Access commands

The set of access commands comprises *Req_RN, Read, Write, Kill, Lock, Access, BlockWrite,* and *BlockErase*, and *BlockPermalock*. As described in 6.3.4.9, tags execute *Req_RN* from the **acknowledged, open,** or **secured** states. Tags execute *Read, Write, BlockWrite,* and *BlockErase* from the **secured** state; if allowed by the lock status of the memory location being accessed, they may also execute these commands from the **open** state. Tags execute *Access* and *Kill* from the **open** or **secured** states. Tags execute *Lock* and *BlockPermalock* only from the **secured** state.

All access commands issued to a tag in the **open** or **secured** states include the tag handle as a parameter in the command. When in either of these two states, tags verify that the handle is correct prior to executing an access command, and ignore access commands with an incorrect handle. The handle value is fixed for the duration of an access sequence.

A tag reply to all access commands that read or write memory (*i.e. Read, Write, Kill, Lock, BlockWrite,* and *BlockErase*, and *BlockPermalock)* includes a 1-bit header. Header=0 indicates that the operation was successful and the reply is valid; header=1 indicates that the operation was unsuccessful and the reply is an error code.

After an interrogator writes to a Tag's StoredPC or EPC, or changes bits 03_h to 07_h of User memory from zero to a nonzero value (or vice versa), or recommissions the Tag, the StoredCRC may be incorrect until the interrogator first powers-down and then re-powers-up its energizing RF field, because the Tag computes its StoredCRC at power-up.

If an Interrogator attempts to write to EPC memory locations 00_h to $0F_h$ (*i.e.* to the StoredCRC) using a *Write*, *BlockWrite*, or *BlockErase* command the Tag shall ignore the command and respond with an error code (see Annex I for error-code definitions and for the reply format).

If an Interrogator attempts to write to EPC memory locations 210_h to $22F_h$ (*i.e.* to the XPC_W1 or XPC_W2) of a Class-1 Tag using a *Write*, *BlockWrite*, or *BlockErase* command the Tag shall ignore the command and respond with an error code (see Annex I for error-code definitions and for the reply format).

If an Interrogator attempts to write to a memory bank or password that is permalocked unwriteable, to a memory bank or password that is locked unwriteable and the Tag is not in the **secured** state, or to a memory block that is permalocked, using a *Write*, *BlockWrite*, or *BlockErase* command; or if a portion of the Data in a *Write* command overlaps a permalocked memory block, the Tag shall ignore the command and respond with an error code (see *Annex I* for error-code definitions and for the reply format).

Req_RN, Read, Write, *Kill* (only with asserted Recom bit 3SB), and *Lock* are required commands; *Access*, *BlockWrite*, *BlockErase*, and *BlockPermalock* are optional. Tags shall ignore optional access commands that they do not support.

See Annex K for an example of a data-flow exchange during which an interrogator accesses a tag and reads its kill password.

6.3.4.11.3.1 *Req_RN* (mandatory)

Interrogators and tags shall implement the *Req_RN* command shown in Table 6.32. *Req_RN* instructs a tag to loadmodulate a new RN16. Both the interrogator's command, and the tag response, depends on the tag state:

- **Acknowledged** state: When issuing a Req_RN command to a tag in the acknowledged state, an interrogator shall include the tag's StoredCRC as a parameter in the *Req_RN.* The *Req_RN* command is protected by a CRC-16c calculated over the command bits and the StoredCRC. If the tag receives the *Req_RN* with a valid CRC-16c and a valid StoredCRC it shall generate and store an RN16 (denoted handle), loadmodulate this handle and transition to the open or secured state. The choice of ending state depends on the tag access password, as follows:

  - Access password<>0: tag transitions to **open** state.
  - Access password=0: tag transitions to **secured** state.

  If the tag receives the *Req_RN* command with a valid CRC-16c but an invalid StoredCRC it shall ignore the *Req_RN* and remain in the **acknowledged** state.

- **Open** or **secured** states: When issuing a Req_RN command to a tag in the open or secured states, an interrogator shall include the tag handle as a parameter in the *Req_RN.* The *Req_RN* command is protected by a CRC-16c calculated over the command bits and the handle. If the tag receives the *Req_RN* with a valid CRC-16c and a valid handle, it shall generate and loadmodulate a new RN16. If the tag receives the *Req_RN* with a valid CRC-16c but an invalid handle, it shall ignore the *Req_RN.* In either case the tag shall remain in its current state (open or secured, as appropriate).

If an interrogator wishes to ensure that only a single tag is in the acknowledged state, it may issue a Req_RN, causing the tag or tags to each loadmodulate a handle and transition to the open or secured state (as appropriate). The interrogator may then issue an *ACK* with handle as a parameter. Tags that receive an *ACK* with an invalid handle shall return to arbitrate (Note: If a tag receives an *ACK* with an invalid handle, it returns to arbitrate, whereas if it receives an access command with an invalid handle it ignores the command).

The first bit of the transmitted RN16 shall be denoted the MSB; the last bit shall be denoted the LSB.

A Req_RN shall be prefixed with a frame-sync (see 6.3.3.1.3.4, 6.3.3.1.3.6 and 6.3.3.1.3.8).

The tag reply to a Req_RN shall be as shown in Table 6.33. The StoredCRC and handle are protected by a CRC-16c.

Table 6.32. *Req_RN* command

	Command	RN	CRC-16c
# of bits	8	16	16
description	11000001	StoredCRC or handle	

Table 6.33. Tag reply to a *Req_RN* command

	RN	CRC-16c
# of bits	16	16
description	New RN16 or handle	

PJM Method: Refer to the PJM Method part of clauses 6.3.4.9 and 6.3.4.11. As for the *ACK-PJM*, see 6.3.4.11.2.4 and Table 6.30, this command can include multiple StoredCRCs or handles in the RN field in order to address multiple tags. Interrogators and tags shall operate (when using this command) as described above for ASK

Method except that multiple tags can be addressed by this command and multiple tags can reply to this command.

6.3.4.11.3.2 *Read* (mandatory)

Interrogators and tags shall implement the *Read* command shown in Table 6.35. *Read* allows an interrogator to read part or all of a tag Reserved, EPC, TID, or User memory. *Read* has the following fields:

- MemBank specifies whether the *Read* accesses Reserved, EPC, TID, or User memory. *Read* commands shall apply to a single memory bank. Successive *Read*s may apply to different banks.
- WordPtrlength, WordPtr specifies the starting word address for the memory read, where words are 16-bits in length. For example, WordPtr=00_h specifies the first 16-bit memory word, WordPtr=01_h specifies the second 16-bit memory word, etc. WordPtr has a length of 8,16,24 or 32 bits as defined in WordPtrlength.
- WordCount specifies the number of 16-bit words to be read. If WordCount=00_h the tag shall loadmodulate the contents of the chosen memory bank starting at WordPtr and ending at the end of the bank, unless MemBank=01_2, in which case the tag shall loadmodulate the EPC memory contents specified in Table 6.34.

Table 6.34. Tag loadmodulation when WordCount=00_h and MemBank=01_2

WordPtr Memory Address	Tag Implements XPC_W2?	What the Tag Loadmodulates
Within the StoredCRC, StoredPC, or the EPC specified by bits 10_h–14_h of the StoredPC	Do not care	EPC memory starting at WordPtr and ending at the EPC length specified by bits 10_h–14_h of the StoredPC
Within physical EPC memory but above the EPC specified by bits 10_h–14_h of the StoredPC	No	EPC memory starting at WordPtr and ending at the end of physical EPC memory, including the XPC_W1 if WordPtr is less than or equal to 210_h and physical EPC memory extends to or above address 210_h
Within physical EPC memory but above the EPC specified by bits 10_h–14_h of the StoredPC	Yes	EPC memory starting at WordPtr and ending at the end of physical EPC memory, including the XPC_W1 and XPC_W2 if WordPtr is less than or equal to 210_h and physical EPC memory extends to or above address 210_h includes the XPC_W2 if WordPtr is equal to 220_h and physical EPC memory extends to or above address 220_h
210_h. Above physical EPC memory.	No	XPC_W1
210_h. Above physical EPC memory.	Yes	XPC_W1 and XPC_W2
220_h. Above physical EPC memory.	No	Error code
220_h. Above physical EPC memory.	Yes	XPC_W2
Not 210_h or 220_h. Above physical EPC memory.	Do not care	Error code

The *Read* command also includes the tag handle and a CRC-16c. The CRC-16c is calculated over the first command-code bit to the last handle bit.

If a tag receives a *Read* with a valid CRC-16c but an invalid handle, it shall ignore the *Read* and remain in its current state **(open** or **secured,** as appropriate).

A *Read* shall be prefixed with a frame-sync (see 6.3.3.1.3.4, 6.3.3.1.3.6 and 6.3.3.1.3.8).

If all of the memory words specified in a *Read* exist and none are read-locked, the tag reply to the *Read* shall be as shown in 0. The tag responds by loadmodulating a header (a 0-bit), the requested memory words, and its handle. The reply includes a CRC-16c calculated over the 0-bit, memory words, and handle.

If a one or more of the memory words specified in the *Read* command either do not exist or are read-locked, the tag shall loadmodulate an error code, within time T_1 in Table 6.14, rather than the reply shown in Table 6.36 (see Annex I for error-code definitions and for the reply format).

Table 6.35. *Read* command

	Command	MemBank	WordPtrLength	WordPtr	WordCount	RN	CRC-16c
# of bits	8	2	2	8, 16, 24 or 32	8	16	16
description	11000010	00: Reserved 01: EPC 10: TID 11: User	Length of Wordpointer 00: 8 bit 01: 16 bit 10: 24 bit 11: 32 bit	Starting address pointer	Number of words to read	handle	

Table 6.36. Tag reply to a successful *Read* command

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

PJM Method: Refer to the PJM Method part of clauses 6.3.4.9 and 6.3.4.11. As for the *ACK-PJM*, see 6.3.4.11.2.4 and Table 6.30, this command can include multiple handles in the RN field in order to address multiple tags. Interrogators and tags shall operate (when using this command) as described above for ASK Method except that multiple tags can be addressed by this command and multiple tags can reply to this command.

6.3.4.11.3.3 *Write* (mandatory)

Interrogators and tags shall implement the *Write* command shown in Table 6.37. *Write* allows an interrogator to write a word in a tag Reserved, EPC, TID, or User memory. *Write* has the following fields:

- MemBank specifies whether the *Write* occurs in Reserved, EPC, TID, or User memory.
- WordPtrlength, WordPtr specifies the word address for the memory write, where words are 16-bits in length. For example, WordPtr=00_h specifies the first 16-bit memory word, WordPtr=01_h specifies the second 16-bit memory word, etc. WordPtr has a length of 8, 16, 24 or 32 bits as defined in the WordPtrlength.
- Data contains a 16-bit word to be written. Before each and every *Write* the interrogator may optionally first issue a *Req_RN* command; the tag responds by loadmodulating a new RN16. The interrogator shall cover-code the data by EXORing it with this new RN16 prior to transmission if a *Req_RN* command was issued.

The *Write* command also includes the tag handle and a CRC-16c. The CRC-16c is calculated over the first command-code bit to the last handle bit. If a tag in the **open** or **secured** states receives a *Write* with a valid CRC-16c but an invalid handle, it shall ignore the *Write* and remain in its current state. If it receives a *Write* where the immediately preceding command was not a *Req_RN*, it shall interpret the data field as plain data

A *Write* shall be prefixed with a frame-sync (see 6.3.3.1.3.4, 6.3.3.1.3.6 and 6.3.3.1.3.8). After issuing a *Write* an interrogator shall transmit CW for the lesser of *TREPLY* or 20ms, where *TREPLY* is the time between the interrogator's *Write* command and the tag transmitted reply. The time *TREPLY* shall be a multiple of T_1 typical (see Table 6.14) with a tolerance of +/- 2.4us so that *TREPLY*=n*1024/f_c +/- 32/f_c. An interrogator may observe several possible outcomes from a *Write,* depending on the success or failure of the tag memory-write operation:

- **The *Write* succeeds:** After completing the *Write,* a tag shall loadmodulate the reply shown in Table 6.38 and Figure 6.30 comprising a header (a 0-bit), the tag handle, and a CRC-16c calculated over the 0-bit and handle. If the interrogator observes this reply within 20ms then the *Write* completed successfully.
- **The tag encounters an error:** The tag shall loadmodulate an error code during the CW period rather than the reply shown in Figure 6.30 (see Annex I for error-code definitions and for the reply format).
- **The *Write* does not succeed:** If the interrogator does not observe a reply within 20ms then the *Write* did not complete successfully. The interrogator may issue a *Req_RN* command (containing the tag handle) to verify that the tag is still in the interrogator's field if cover-coding is used, and may reissue the *Write* command regardless the use of cover-coding.

Upon receiving a valid *Write* command, a tag shall write the commanded Data into memory. The tag reply to a successful Write shall use the preamble as specified in the *BeginRound* command that initiated the round.

Table 6.37. *Write* command

	Command	MemBank	WordPtrLength	WordPtr	Data	RN	CRC-16c
# of bits	8	2	2	8, 16, 24 or 32	16	16	16
description	11000011	00: Reserved 01: EPC 10: TID 11: User	Length of Wordpointer 00: 8 bit 01: 16 bit 10: 24 bit 11: 32 bit	Address pointer	RN16 ⊗ word to be written or word to be written	handle	

Table 6.38. Tag reply to a successful *Write* command

	Header	RN	CRC-16c
# of bits	1	16	16
description	0	handle	

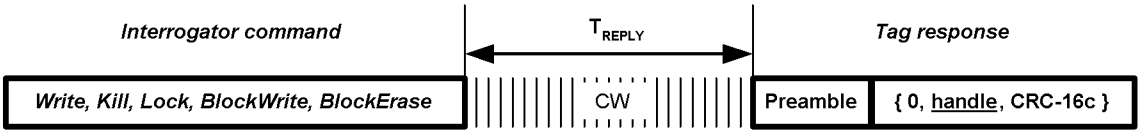


Figure 6.30 Successful *Write* sequence

PJM Method: Refer to the PJM Method part of clauses 6.3.4.9 and 6.3.4.11. As for the *ACK-PJM*, see 6.3.4.11.2.4 and Table 6.30, this command can include multiple handles in the RN field in order to address multiple tags. Interrogators and tags shall operate (when using this command) as described above for ASK Method except that multiple tags can be addressed by this command and multiple tags can reply to this command.

6.3.4.11.3.4 *Kill* (only support of asserted Recom bit 3SB is **mandatory)**

ASK Method and PJM Method: Interrogators and tags shall implement the *Kill* command (only support of asserted Recom bit 3SB is mandatory) shown in Table 6.39 and Figure 6.31. *Kill* allows an interrogator to permanently disable a tag. *Kill* also allows an Interrogator to recommission Tags.

To kill or recommission a tag, an interrogator shall follow the multi-step procedure outlined in Figure 6.31. Briefly, an interrogator issues two *Kill* commands, the first containing the 16 MSBs of the tag kill password optionally EXORed with an RN16, and the second containing the 16 LSBs of the tag kill password optionally EXORed with a different RN16. Each EXOR operation shall be performed MSB first (*i.e.* the MSB of each half-password shall be EXORed with the MSB of its respective RN16). Just prior to issuing each *Kill* command the interrogator first issues a *Req_RN* to obtain a new RN16 if cover-coding is used, otherwise no *Req_RN* command shall be issued.

Tags shall incorporate the necessary logic to successively accept two 16-bit sub-portions of a 32-bit kill password. Interrogators shall not intersperse commands other than *Req_RN* between the two successive *Kill* commands. If a tag, after receiving a first *Kill,* receives any command other than *Req_RN* before the second *Kill,* it shall return to **arbitrate,** unless the intervening command is a *BeginRound,* in which case the tag shall execute the *BeginRound* (setting its **inventoried** flag if the session parameter in the *BeginRound* matches the prior session).

Kill contains 3 RFU/Recom bits. In the first *Kill* command these bits are RFU. Interrogators shall set the 3 RFU bits to 000_2 when communicating with Class-1 Tags, and Class-1 Tags shall ignore those bits. Higher-functionality Tags may use the 3 RFU bits to expand the functionality of the *Kill* command. As described in 6.3.4.10 in the second *Kill* command, the 3 RFU bits are called *recommissioning* (or Recom) bits and may be nonzero. The procedures for killing or recommissioning a Tag are identical, except that the recommissioning bits in the second *Kill* command are zero when killing a Tag (optional) and are nonzero when recommissioning (support of Recom bit 3SB is mandatory) a tag. Regardless of the intended operation, a Tag does not kill or recommission itself without first receiving the correct kill password by the procedure shown in 6.3.4.10.

If a Tag does not implement Recom bits LSB and 2SB (see 6.3.4.10), then the Tag ignores the LSB and 2SB recommissioning bits and treats them as though they were zero. If the Tag receives a properly formatted *Kill* command sequence with the correct kill password and all 3 Recom bits set to zero, and the Tag does not support the second *Kill* command with all 3 Recom bits set to zero - because it is optional, the Tag shall interpret this *Kill* command sequence in the same way as if the 3SB of the Recom bits is set to one. If the Tag does not support killing, then the Tag recommissions itself regardless of the values of the recommissioning bits if recommissioning is permitted.

A tag whose kill password is zero, does not execute a kill or a recommissioning operation; if such a tag receives a *Kill* command, it ignores the command and loadmodulates an error code (see Figure 6.31).

The tag reply to the first *Kill* command shall be as shown in Table 6.41. The reply shall use the TRext value specified in the *BeginRound* command that initiated the round.

After issuing the second *Kill* command, an interrogator shall transmit CW for the lesser of T_{REPLY} or 20ms, where T_{REPLY} is the time between the interrogator's second *Kill* command and the tag's transmitted reply. The time T_{REPLY} shall be a multiple of T_1 typical (see Table 6.14) with a tolerance of +/- 2.4μs so that $T_{REPLY} = n*1024/f_c$ +/- $32/f_c$. An interrogator may observe several possible outcomes from a *Kill* command sequence, depending on the success or failure of the tag kill or recommissioning operation:

- **The *Kill* or recommissioning succeeds:** After completing the operation the tag shall loadmodulate the reply shown in Table 6.42 and Figure 6.31. comprising a header (a 0-bit), the tag handle, and a CRC-16c calculated over the 0-bit and handle. If the Interrogator observes this reply within 20ms then the operation completed successfully. If the Tag is killed, then immediately after this reply the Tag shall render itself silent and shall not respond to an Interrogator thereafter.
- **The tag encounters an error:** The tag shall loadmodulate an error code during the CW period rather than the reply shown in Table 6.42 (see Annex I for error-code definitions and for the reply format).
- **The *Kill* or recommissioning does not succeed:** If the interrogator does not observe a reply within 20ms then the operation did not complete successfully. The interrogator may issue a *Req_RN* command (containing the tag handle) to verify that the tag is still in the interrogator's field if cover-coding is used, and may reinitiate the multi-step kill procedure outlined in Figure 6.31 regardless the use of cover-coding.

A *Kill* shall be prefixed with a frame-sync (see 6.3.3.1.3.4, 6.3.3.1.3.6 and 6.3.3.1.3.8).

Upon receiving a valid *Kill* command sequence a tag shall render itself killed or recommissioned as appropriate. The tag reply to the second *Kill* command shall use the preamble as specified in the *BeginRound* command that initiated the round.

Table 6.39. First *Kill* command

	Command	Password	RFU/Recom	RN	CRC-16c
# of bits	8	16	3	16	16
description	11000100	(½ kill password) ⊗ RN16 or plain	000	handle	

Table 6.40. Second *Kill* command

	Command	Password	RFU/Recom	RN	CRC-16c
# of bits	8	16	3	16	16
description	11000100	(½ kill password) ⊗ RN16 or plain	Recommissioning bits (see 6.3.4.10)	handle	

Table 6.41. Tag reply to the first *Kill* command

	RN	CRC-16c
# of bits	16	16
description	handle	

Table 6.42. Tag reply to a successful *Kill* procedure

	Header	RN	CRC-16c
# of bits	1	16	16
description	0	handle	

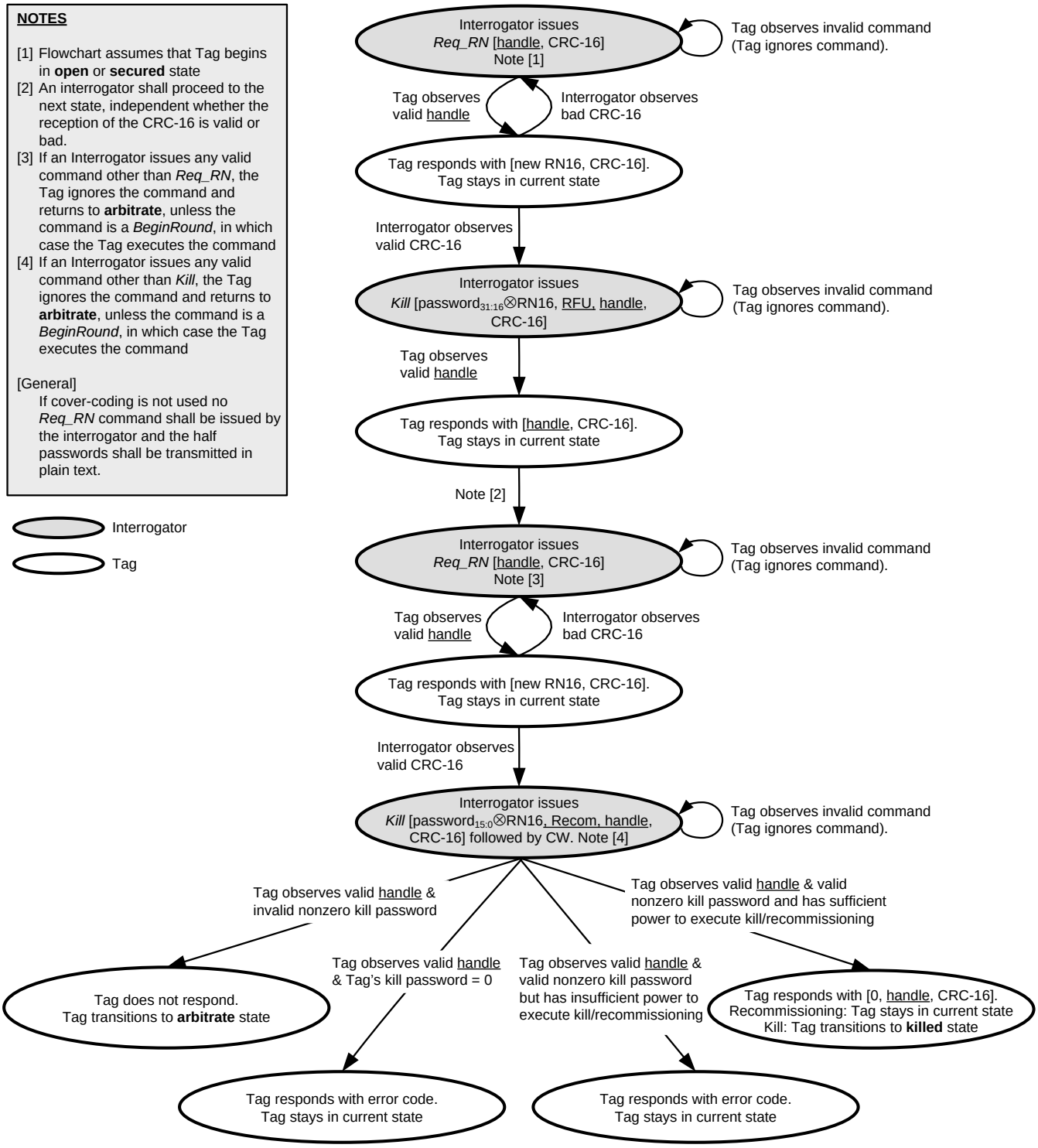


Figure 6.31 *Kill* procedure

6.3.4.11.3.5 *Lock* (mandatory)

Interrogators and tags shall implement the *Lock* command shown in Table 6.43 and Figure 6.32. Only tags in the **secured** state shall execute a *Lock* command. *Lock* allows an interrogator to:

- Lock individual passwords, thereby preventing or allowing subsequent reads and writes of that password,
- Lock individual memory banks, thereby preventing or allowing subsequent writes to that bank, and
- Permalock (make permanently unchangeable) the lock status for a password or memory bank.

Lock contains a 20-bit payload defined as follows:

- The first 10 payload bits are Mask bits. A tag shall interpret these bit values as follows:
  - Mask=0: Ignore the associated Action field and retain the current lock setting
  - Mask=1: Implement the associated Action field and overwrite the current lock setting
- The last 10 payload bits are Action bits. A tag shall interpret these bit values as follows:
  - Action=0: Deassert lock for the associated memory location
  - Action=1: Assert lock or permalock for the associated memory location

The functionality of the various Action fields is described in Table 6.45.

The payload of a *Lock* command shall always be 20 bits in length. If an interrogator issues a *Lock* command whose Mask and Action fields attempt to change the lock status of a nonexistent memory bank or nonexistent password, the tag shall ignore the entire *Lock* command and instead loadmodulate an error code (see Annex I).

The *Lock* command differs from the optional *BlockPermalock* command in that *Lock* reversibly or permanently locks a password or an entire EPC, TID, or User memory bank in a writeable or unwriteable state, whereas *BlockPermalock* permanently locks blocks of User memory in an unwriteable state. Table 6.52 specifies how a Tag shall react to a *Lock* command that follows a prior *BlockPermalock* command, or vice versa.

Permalock bits, once asserted, cannot be deasserted except by recommissioning the Tag (see 6.3.4.10). If a tag receives a *Lock* whose payload attempts to deassert a previously asserted permalock bit, the tag shall ignore the *Lock* and loadmodulate an error code (see Annex I). If a tag receives a *Lock* whose payload attempts to reassert a previously asserted permalock bit, the tag shall simply ignore this particular Action field and implement the remainder of the *Lock* payload, unless the Tag has been recommissioned and the corresponding memory location is no longer permalocked, in which case the Tag shall reassert the permalock bit.

A tag's lock bits cannot be read directly; they can be inferred by attempting to perform other memory operations.

All tags shall implement memory locking, and all tags shall implement the *Lock* command. However, tags need not support all the Action fields shown in Figure 6.32, depending on whether the password location or memory bank associated with an Action field exists and is lockable and/or unlockable. Specifically, if a tag receives a *Lock* it cannot execute because one or more of the passwords or memory banks do not exist, or one or more of the Action fields attempt to change a permalocked value, or one or more of the passwords or memory banks are either not lockable or not unlockable; the tag shall ignore the entire *Lock* and instead loadmodulate an error code (see Annex I). The only exception to this general rule relates to tags whose only lock functionality is to permanently lock **all** memory (*i.e.* all memory banks and all passwords) at once; these tags shall execute a *Lock* whose payload is $FFFFF_h$, and shall loadmodulate an error code for any payload other than $FFFFF_h$.

A *Lock* shall be prefixed with a frame-sync (see 6.3.3.1.3.4, 6.3.3.1.3.6 and 6.3.3.1.3.8).

After issuing a *Lock* an interrogator shall transmit CW for the lesser of *TREPLY* or 20ms, where *TREPLY* is the time between the interrogator's *Lock* command and the tag transmitted reply. The time *TREPLY* shall be a multiple of T_1 typical (see Table 6.14) with a tolerance of +/- 2.4μs so that *TREPLY*=n*1024/f_c +/- 32/f_c. An interrogator may observe several possible outcomes from a *Lock,* depending on the success or failure of the tag memory-write operation:

- **The *Lock* succeeds:** After completing the *Lock* the tag shall loadmodulate the reply shown in Table 6.44 and Figure 6.30 comprising a header (a 0-bit), the tag handle and a CRC-16c calculated over the 0-bit and handle. If the interrogator observes this reply within 20ms then the Lock completed successfully.

- **The tag encounters an error:** The tag shall loadmodulate an error code during the CW period rather than the reply shown in Table 6.44 (see Annex I for error-code definitions and for the reply format).
- **The *Lock* does not succeed:** If the interrogator does not observe a reply within 20ms, then the Lock did not complete successfully. The interrogator may issue a Req RN command (containing the tag handle) to verify that the tag is still in the interrogator's field, and may reissue the Lock.

Upon receiving a valid *Lock* command, a tag shall perform the commanded lock operation The tag reply to a Lock shall use the preamble as specified in the *BeginRound* command that initiated the round.

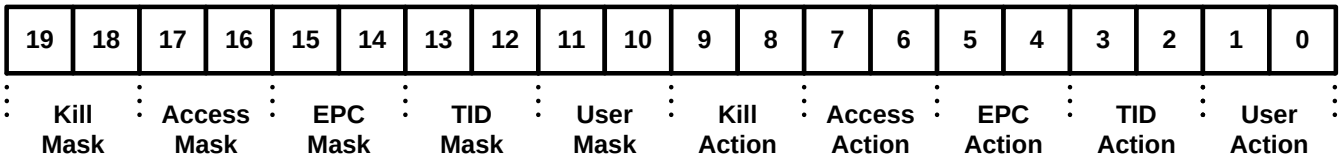
Table 6.43. *Lock* command

	Command	Payload	RN	CRC-16c
# of bits	8	20	16	16
description	11000101	Mask and Action Fields	Handle	

Table 6.44. Tag reply to a *Lock* command

	Header	RN	CRC-16c
# of bits	1	16	16
description	0	handle	

Lock-Command Payload



Masks and Associated Action Fields

	Kill pwd		Access pwd		EPC memory		TID memory		User memory	
	19	18	17	16	15	14	13	12	11	10
Mask	skip/ write	skip/ write	skip/ write	skip/ write	skip/ write	skip/ write	skip/ write	skip/ write	skip/ write	skip/ write
	9	8	7	6	5	4	3	2	1	0
Action	pwd read/ write	perma lock	pwd read/ write	perma lock	pwd write	perma lock	pwd write	perma lock	pwd write	perma lock

Figure 6.32 *Lock* payload and usage

Table 6.45. *Lock* Action-field functionality

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

PJM Method: Refer to the PJM Method part of clauses 6.3.4.9 and 6.3.4.11. As for the *ACK-PJM*, see 6.3.4.11.2.4 and Table 6.30, this command can include multiple handles in the RN field in order to address multiple tags. Interrogators and tags shall operate (when using this command) as described above for ASK Method except that multiple tags can be addressed by this command and multiple tags can reply to this command.

6.3.4.11.3.6 *Access* (optional)

ASK Method and PJM Method: Interrogators and tags may implement an Access command; if they do, the command shall be as shown in Table 6.46. *Access* causes a tag with a non-zero-valued access password to transition from the **open** to the **secured** state (a tag with a zero-valued access password is never in the **open** state — see Figure 6.27) or, if the tag is already in the **secured** state, to remain **secured.**

To access a tag, an interrogator shall follow the multi-step procedure outlined in Figure 6.33. Briefly, an interrogator issues two Access commands, the first containing the 16 MSBs of the tag access password optionally EXORed with an RN16, and the second containing the 16 LSBs of the tag access password optionally EXORed with a different RN16. Each EXOR operation shall be performed MSB first (i.e. the MSB of each half-password shall be EXORed with the MSB of its respective RN16). Just prior to issuing each Access command, the interrogator first issues a *Req_RN* to obtain a new RN16 if cover-coding is used otherwise no Req_RN command shall be issued.

Tags shall incorporate the necessary logic to successively accept two 16-bit sub-portions of a 32-bit access password. Interrogators shall not intersperse commands other than *Req_RN* if cover-coding is used between the two successive *Access* commands. If a tag, after receiving a first *Access,* receives any command other than *Req_RN* before the second *Access,* it shall return to **arbitrate,** unless the intervening command is a *BeginRound,* in which case the tag shall execute the *BeginRound* (setting its **inventoried** flag if the session parameter in the *BeginRound* matches the prior session).

An *Access* shall be prefixed with a frame-sync (see 6.3.3.1.3.4, 6.3.3.1.3.6 and 6.3.3.1.3.8).

The tag reply to an *Access* command shall be as shown in Table 6.47. If the *Access* is the first in the sequence, then the tag loadmodulates its handle to acknowledge that it received the command. If the *Access* is the second in the sequence and the entire received 32-bit access password is correct, then the tag loadmodulates its handle to acknowledge that it has executed the command successfully and has transitioned to the **secured** state; otherwise the tag does not reply. The reply includes a CRC-16c calculated over the handle.

Table 6.46. *Access* command

	Command	Password	RN	CRC-16c
# of bits	8	16	16	16
description	11000110	(½ access password) ⊗ RN16 or plain	handle	

Table 6.47. Tag reply to an *Access* command

	RN	CRC-16c
# of bits	16	16
description	handle	

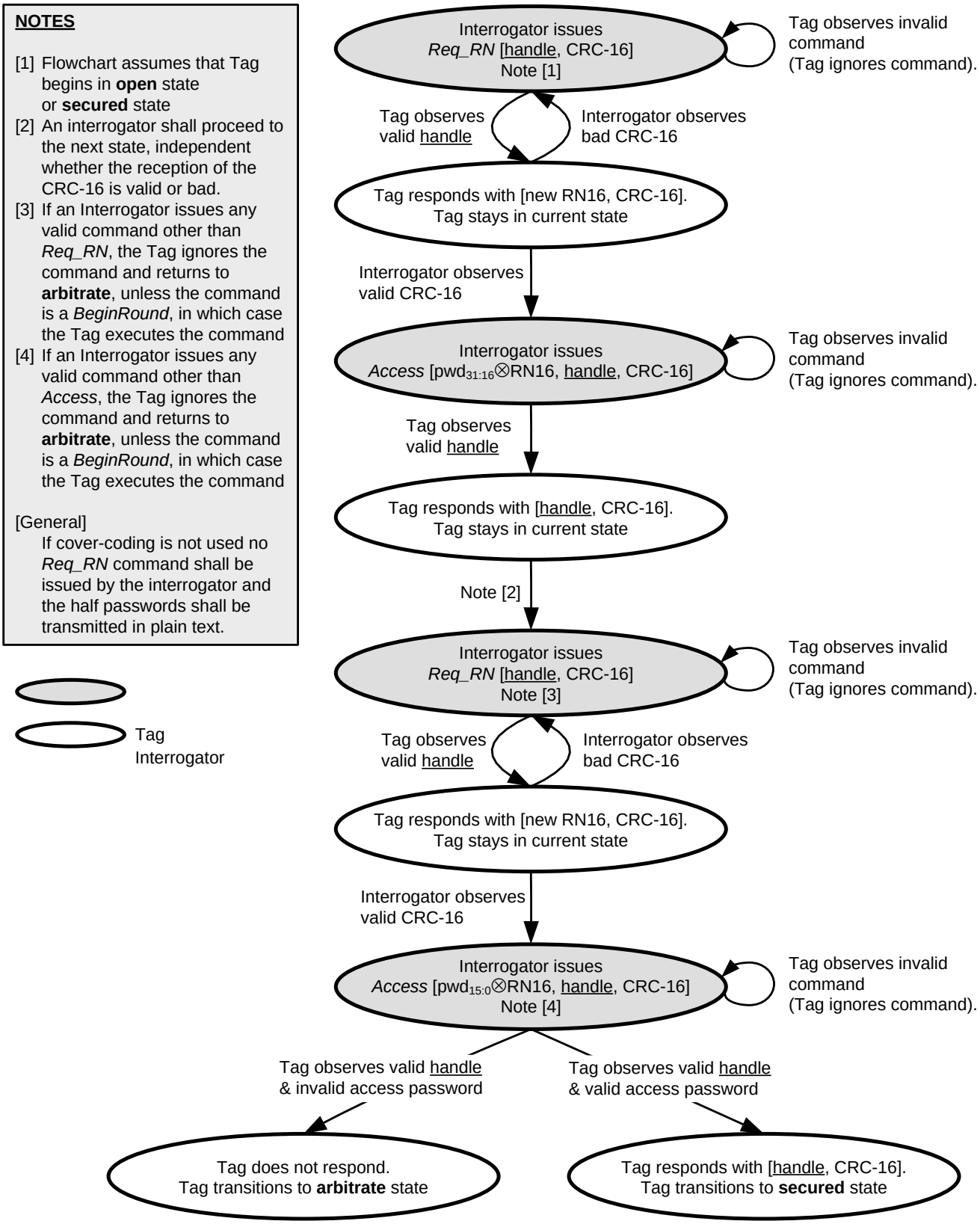


Figure 6.33 *Access* procedure

6.3.4.11.3.7 *BlockWrite* (optional)

Interrogators and tags may implement a *BlockWrite* command; if they do, they shall implement it as shown in Table 6.48. *BlockWrite* allows an interrogator to write multiple words in a tag Reserved, EPC, TID, or User memory using a single command. *BlockWrite* has the following fields:

- MemBank specifies whether the BlockWrite occurs in Reserved, EPC, TID, or User memory. BlockWrite commands shall apply to a single memory bank. Successive BlockWrites may apply to different banks.
- WordPtrlength, WordPtr specifies the starting word address for the memory write, where words are 16-bits in length. For example, WordPtr=00_h specifies the first 16-bit memory word, WordPtr=01_h specifies the second 16-bit memory word, etc. WordPtr has a length of 8, 16, 24 or 32 bits as defined in WordPtrlength.
- WordCount specifies the number of 16-bit words to be written. If WordCount=00_h, the tag shall ignore the *BlockWrite.* If WordCount=01_h, the tag shall write a single data word.
- Data contains the 16-bit words to be written, and shall be 16×WordCount bits in length. Unlike a *Write,* the data in a *BlockWrite* are not cover-coded, and an interrogator need not issue a *Req_RN* before issuing a *BlockWrite.*

The *BlockWrite* command also includes the tag handle and a CRC-16c. The CRC-16c is calculated over the first command-code bit to the last handle bit.

If a tag receives a *BlockWrite* with a valid CRC-16c but an invalid handle, it shall ignore the *BlockWrite* and remain in its current state **(open** or **secured,** as appropriate). A *BlockWrite* shall be prefixed with a frame-sync (see 6.3.3.1.3.4, 6.3.3.1.3.6 and 6.3.3.1.3.8).

After issuing a *BlockWrite*, an interrogator shall transmit CW for the lesser of *TREPLY* or 20ms, where *TREPLY* is the time between the interrogator's *BlockWrite* command and the tag transmitted reply. The time *TREPLY* shall be a multiple of T_1 typical (see Table 6.14) with a tolerance of +/- 2.4μs so that *TREPLY*=n*1024/f_c +/- 32/f_c. An interrogator may observe several possible outcomes from a *BlockWrite,* depending on the success or failure of the tag memory write operation:

- **The *BlockWrite* succeeds:** After completing the *BlockWrite* a tag shall loadmodulate the reply shown in Table 6.49 and Figure 6.30 comprising a header (a 0-bit), the tag handle and a CRC-16c calculated over the 0-bit and handle. If the interrogator observes this reply within 20ms then the *BlockWrite* completed successfully.
- **The tag encounters an error:** The tag shall loadmodulate an error code during the CW period rather than the reply shown in Table 6.49 (see Annex I for error-code definitions and for the reply format).
- **The *BlockWrite* does not succeed:** If the interrogator does not observe a reply within 20ms then the BlockWrite did not complete successfully. The interrogator may issue a Req_RN command (containing the tag handle) to verify that the tag is still in the interrogator's field, and may reissue the BlockWrite.

Upon receiving a valid *BlockWrite* command a tag shall write the commanded Data into memory. The tag reply to a Block Write shall use the preamble as specified in the *BeginRound* command that initiated the round.

Table 6.48. *BlockWrite* command

	Command	MemBank	WordPtrLength	WordPtr	WordCount	Data	RN	CRC-16c
# of bits	8	2	2	8, 16, 24 or 32	8	Variable	16	16
Description	11000111	00: Reserved 01: EPC 10: TID 11: User	Length of Wordpointer 00: 8 bit 01: 16 bit 10: 24 bit 11: 32 bit	Starting address pointer	Number of words to write	Data to be written	handle	

Table 6.49. Tag reply to a successful *BlockWrite* command

	Header	RN	CRC-16c
# of bits	1	16	16
description	0	handle	

PJM Method: Refer to the PJM Method part of clauses 6.3.4.9 and 6.3.4.11. As for the *ACK-PJM*, see 6.3.4.11.2.4 and Table 6.30, this command can include multiple handles in the RN field in order to address multiple tags. Interrogators and tags shall operate (when using this command) as described above for ASK Method except that multiple tags can be addressed by this command and multiple tags can reply to this command.

6.3.4.11.3.8 *BlockErase* (optional)

Interrogators and tags may implement a *BlockErase* command; if they do, they shall implement it as shown in Table 6.50. *BlockErase* allows an interrogator to erase multiple words in a tag Reserved, EPC, TID, or User memory using a single command. *BlockErase* has the following fields:

- MemBank specifies whether the *BlockErase* occurs in Reserved, EPC, TID, or User memory. *BlockErase* commands shall apply to a single memory bank. Successive *BlockErases* may apply to different banks.
- WordPtrlength, WordPtr specifies the starting word address for the memory erase, where words are 16-bits in length. For example, WordPtr=00_h specifies the first 16-bit memory word, WordPtr=01_h specifies the second 16-bit memory word, etc. WordPtr has a length of 8, 16, 24 or 32 bits as defined in WordPtrlength.
- WordCount specifies the number of 16-bit words to be erased. If WordCount=00_h, the tag shall ignore the *BlockErase.* If WordCount=01_h the tag shall erase a single data word.

The *BlockErase* command also includes the tag handle and a CRC-16c. The CRC-16c is calculated over the first command-code bit to the last handle bit.

If a tag receives a *BlockErase* with a valid CRC-16c but an invalid handle it shall ignore the *BlockErase* and remain in its current state **(open** or **secured,** as appropriate).

A *BlockErase* shall be prefixed with a frame-sync (see 6.3.3.1.3.4, 6.3.3.1.3.6 and 6.3.3.1.3.8).

After issuing a *BlockErase* an interrogator shall transmit CW for the lesser of *TREPLY* or 20ms, where *TREPLY* is the time between the interrogator's *BlockErase* command and the tag transmitted reply. The time *TREPLY* shall be a multiple of T_1 typical (see Table 6.14) with a tolerance of +/- 2.4μs so that *TREPLY*=n*1024/f_c +/- 32/f_c. An interrogator may observe several possible outcomes from a *BlockErase,* depending on the success or failure of the tag memory erase operation:

- **The *BlockErase* succeeds:** After completing the *BlockErase* a tag shall loadmodulate the reply shown in Table 6.51 and Figure 6.30 comprising a header (a 0-bit), the tag handle and a CRC-16c calculated over the 0-bit and handle. If the interrogator observes this reply within 20ms then the *BlockErase* completed successfully.
- **The tag encounters an error:** The tag shall loadmodulate an error code during the CW period rather than the reply shown in Figure 6.30 (see Annex I for error-code definitions and for the reply format).
- **The *BlockErase* does not succeed:** If the interrogator does not observe a reply within 20ms then the *BlockErase* did not complete successfully. The interrogator may issue a *Req_RN* command (containing the tag handle) to verify that the tag is still in the interrogator's field, and may reissue the BlockErase.

Upon receiving a valid *BlockErase* command a tag shall erase the commanded memory words. The tag reply to a *BlockErase* shall use the preamble as specified in the BeginRound command that initiated the round.

Table 6.50. *BlockErase* command

	Command	MemBank	WordPtrLength	WordPtr	WordCount	RN	CRC-16c
# of bits	8	2	2	8, 16, 24 or 32	8	16	16
description	11001000	00: Reserved 01: EPC 10: TID 11: User	Length of Wordpointer 00: 8 bit 01: 16 bit 10: 24 bit 11: 32 bit	Starting address Pointer	Number of words to write	handle	

Table 6.51. Tag reply to a successful *BlockErase* command

	Header	RN	CRC-16c
# of bits	1	16	16
description	0	Handle	

Optionally the interrogators shall communicate using PJM.

PJM Method: See the PJM Method part of clauses 6.3.4.9 and 6.3.4.11. As for the *ACK-PJM*, see 6.3.4.11.2.4 and Table 6.30, this command can include multiple handles in the RN field in order to address multiple tags. Interrogators and tags shall operate (when using this command) as described above for ASK Method except that multiple tags can be addressed by this command and multiple tags can reply to this command.

6.3.4.11.3.9 *BlockPermalock* (optional)

Interrogators and Tags may implement a *BlockPermalock* command; if they do, they shall implement it as shown in Table 6.53. *BlockPermalock* allows an Interrogator to:

- Permalock one or more blocks (individual sub-portions) in a Tag's User memory, or
- Read the permalock status of the memory blocks in a Tag's User memory.

A single *BlockPermalock* command can permalock between 0 and 4080 blocks of User memory. The block size is vendor-defined. The memory blocks need not be contiguous.

Only Tags in the **secured** state shall execute a *BlockPermalock* command.

The *BlockPermalock* command differs from the *Lock* command in that *BlockPermalock* permanently locks blocks of User memory in an unwriteable state, whereas *Lock* reversibly or permanently locks a password or an entire memory bank in a writeable or unwriteable state. 6.3.4.11.3.9 specifies how a Tag shall react to a *BlockPermalock* command (with Read/Lock=1) that follows a prior *Lock* command, or vice versa.

Table 6.52. Precedence for *Lock* and *BlockPermalock* commands

<table>
<tr><th colspan="3">First Command</th><th colspan="3">Second Command</th><th>Tag Action and Response to 2nd Command</th></tr>
<tr><td rowspan="5">Lock</td><td>pwd-write</td><td>permalock</td><td colspan="3" rowspan="5">BlockPermalock
(Read/Lock = 1)</td><td></td></tr>
<tr><td>0</td><td>0</td><td>Permalock the blocks indicated by Mask; respond as described in this section 6.3.4.11.3.9</td></tr>
<tr><td>0</td><td>1</td><td>Reject the BlockPermalock; respond with an error code</td></tr>
<tr><td>1</td><td>0</td><td>Permalock the blocks indicated by Mask; respond as described in this section 6.3.4.11.3.9</td></tr>
<tr><td>1</td><td>1</td><td>Permalock the blocks indicated by Mask; respond as described in this section 6.3.4.11.3.9</td></tr>
<tr><td colspan="3" rowspan="5">BlockPermalock
(Read/Lock = 1)</td><td rowspan="5">Lock</td><td>pwd-write</td><td>permalock</td><td></td></tr>
<tr><td>0</td><td>0</td><td>Implement the Lock, but do not un-permalock any blocks that were previously permalocked; respond as described in 6.3.4.11.3.5</td></tr>
<tr><td>0</td><td>1</td><td>Reject the Lock; respond with an error code</td></tr>
<tr><td>1</td><td>0</td><td>Implement the Lock, but do not un-permalock any blocks that were previously permalocked; respond as described in 6.3.4.11.3.5</td></tr>
<tr><td>1</td><td>1</td><td>Implement the Lock; respond as described in 6.3.4.11.3.56.3.4.11.3.5</td></tr>
</table>

The *BlockPermalock* command has the following fields:

- MemBank specifies whether the *BlockPermalock* applies to EPC, TID, or User memory. *BlockPermalock* commands shall apply to a single memory bank. Successive *BlockPermalocks* may apply to different memory banks. Class-1 Tags shall only execute a *BlockPermalock* command if MemBank=11 (User memory); if a Class-1 Tag receives a *BlockPermalock* with MemBank<>11 it shall ignore the command and instead loadmodulation an error code (see Annex I), remaining in the **secured** state. Higher-functionality Tags may use the other MemBank values to expand the functionality of the *BlockPermalock* command.
- Read/Lock specifies whether a Tag loadmodulations the permalock status of, or permalocks, one or more blocks within the memory bank specified by MemBank. A Tag shall interpret the Read/Lock bit as follows:
  - Read/Lock=0: A Tag shall loadmodulation the permalock status of blocks in the specified memory bank, starting from the memory block located at BlockPtr and ending at the memory block located at

BlockPtr+(16×BlockRange)–1. A Tag shall loadmodulation a “0” if the memory block corresponding to that bit is not permalocked and a “1” if the block is permalocked. An Interrogator omits Mask from the *BlockPermalock* command when Read/Lock=0.

  - Read/Lock=1: A Tag shall permalock those blocks in the specified memory bank that are specified by Mask, starting at BlockPtr and ending at BlockPtr+(16×BlockRange)–1.
- BlockPtrlength, BlockPtr specifies the starting address for Mask, in units of 16 blocks. For example, BlockPtr=00_h indicates block 0, BlockPtr=01_h indicates block 16, and BlockPtr=02_h indicates block 32. BlockPtr has a length of 8, 16, 24 or 32 bits as defined in BlockPtrlength.
- BlockRange specifies the range of Mask, starting at BlockPtr and ending (16×BlockRange)–1 blocks later. If BlockRange=00_h, then the Tag shall ignore the *BlockPermalock* command and instead loadmodulate an error code (see Annex I), remaining in the **secured** state.
- Mask specifies which memory blocks a Tag permalocks. Mask depends on the Read/Lock bit as follows:
  - Read/Lock=0: The Interrogator shall omit Mask from the *BlockPermalock* command.
  - Read/Lock=1: The Interrogator shall include a Mask of length 16×BlockRange bits in the *BlockPermalock* command. The Mask bits shall be ordered from lower-order block to higher (i.e. if BlockPtr=00_h then the leading Mask bit refers to Block 0). The Tag shall interpret each bit of Mask as follows:
    - Mask bit=0: Retain the current permalock setting for the corresponding memory block.
    - Mask bit=1: Permalock the corresponding memory block. If a block is already permalocked then the Tag shall retain the current permalock setting. A memory block, once permalocked, cannot be un-permalocked except by recommissioning the Tag (see 6.3.4.10).

The following examples illustrate the usage of Read/Lock, BlockPtr, BlockRange, and Mask:

- If Read/Lock=1, BlockPtr=01_h, and BlockRange=01_h, the Tag operates on sixteen blocks starting at block 16 and ending at block 31, permalocking those blocks whose corresponding bits are asserted in Mask.

The *BlockPermalock* command contains 8 RFU bits. Interrogators shall set these bits to 00_h when communicating with Class-1 Tags. If a Class-1 Tag receives a *BlockPermalock* command containing nonzero RFU bits it shall ignore the command and instead loadmodulates an error code (see Annex I), remaining in the **secured** state. Higher-functionality Tags may use these bits to expand the functionality of the *BlockPermalock* command.

The *BlockPermalock* command also includes the Tag’s handle and a CRC-16c. The CRC-16c is calculated over the first command-code bit to the last handle bit. If a Tag receives a *BlockPermalock* with a valid CRC-16c but an invalid handle, it shall ignore the *BlockPermalock* and remain in the **secured** state.

If a Tag receives a *BlockPermalock* command that it cannot execute because User memory does not exist, or in which the LSB and/or the 2SB of a Tag’s XPC_W1 is/are asserted (see Table 6.16), or in which one of the asserted Mask bits references a non-existent block, then the Tag shall ignore the *BlockPermalock* command and instead loadmodulate an error code (see Annex I), remaining in the **secured** state. A Tag shall treat as invalid a *BlockPermalock* command in which Read/Lock=1 but Mask has a length that is not equal to 16xBlockRange bits (see 6.3.4.11 for the definition of an “invalid” command).

Certain Tags, depending on the Tag manufacturer’s implementation, may be unable to execute a *BlockPermalock* command with certain BlockPtr and BlockRange values, in which case the Tag shall ignore the *BlockPermalock* command and instead loadmodulate an error code (see Annex I), remaining in the **secured** state. Because a Tag contains information in its TID memory that an Interrogator can use to uniquely identify the optional features that the Tag supports (see 6.3.4.1.3), Interrogators shall read a Tag’s TID memory prior to issuing a *BlockPermalock* command.

If an Interrogator issues a *BlockPermalock* command in which BlockPtr and BlockRange specify one or more nonexistent blocks, but Mask only asserts permalocking on existent blocks, then the Tag shall execute the command.

A *BlockPermalock* shall be prepended with a frame-sync (see 6.3.3.1.2.8).

After issuing a *BlockPermalock* command an Interrogator shall transmit CW for the lesser of T_{REPLY} or 20ms, where T_{REPLY} is the time between the Interrogator’s *BlockPermalock* and the Tag’s loadmodulated reply. The time T_{REPLY} shall be a multiple of T_1 typical (see Table 6.14) with a tolerance of +/- 2.4μs so that T_{REPLY} =n*1024/f_c +/- 32/f_c. An Interrogator may observe several possible outcomes from a *BlockPermalock* command, depending on

the value of the Read/Lock bit in the command and, if Read/Lock=1, the success or failure of the Tag's memory-lock operation:

- **Read/Lock=0 and the Tag is able to execute the command:** The Tag shall loadmodulate the reply shown in Table 6.54, within time T_1 in Table 6.14, comprising a header (a 0-bit), the requested permalock bits, the Tag's handle, and a CRC-16c calculated over the 0-bit, permalock bits, and handle. The Tag's reply shall use the preamble specified by the TRext value in the *BeginRound* that initiated the round.
- **Read/Lock=0 and the Tag is unable to execute the command:** the Tag shall loadmodulate an error code, within time T_1 in Table 6.14; rather than the reply shown in Table 6.54 (see Annex I for error-code definitions and for the reply format). The Tag's reply shall use the preamble specified by the TRext value in the *BeginRound* that initiated the round.
- **Read/Lock=1 and The *BlockPermalock* succeeds:** After completing the *BlockPermalock* the Tag shall loadmodulate the reply shown in Table 6.55 comprising a header (a 0-bit), the Tag's handle, and a CRC-16c calculated over the 0-bit and handle. If the Interrogator observes this reply within 20ms then the *BlockPermalock* completed successfully. The tag reply shall use the preamble as specified in the *BeginRound* command that initiated the round.
- **Read/Lock=1 and the *BlockPermalock* does not succeed:** If the Interrogator does not observe a reply within 20ms then the *BlockPermalock* did not complete successfully. The Interrogator may issue a *Req_RN* command (containing the Tag's handle) to verify that the Tag is still in the Interrogator's field, and may reissue the *BlockPermalock*.
- **Read/Lock=1 and the Tag encounters an error:** The Tag shall loadmodulate an error code during the CW period rather than the reply shown in Table 6.52 (see Annex I for error-code definitions and for the reply format). The tag reply shall use the preamble as specified in the *BeginRound* command that initiated the round. The time T_{REPLY} shall be a multiple of T1 typical (see Table 6.12) with a tolerance of +/- 2.4μs so that $T_{REPLY} = n*1024/f_c$ +/- $32/f_c$.

Table 6.53. *BlockPermalock* command

	Command	RFU	Read/ Lock	Mem Bank	BlockPtr Length	BlockPtr	BlockRange	Mask	RN	CRC-16c
# of bits	8	8	1	2	2	8, 16, 24 or 32	8	Variable	16	16
Description	11001001	00_h	0: Read 1: Permalock	00: RFU 01: EPC 10: TID 11: User	Length of Wordpointer 00: 8 bit 01: 16 bit 10: 24 bit 11: 32 bit	Mask starting address, specified in units of 16 blocks	Mask range, specified in units of 16 blocks	0: Retain current permalock setting 1: Assert perma-lock	handle	

Table 6.54. Tag reply to a successful *BlockPermalock* command with Read/Lock=0

	Header	Data	RN	CRC-16c
# of bits	1	Variable	16	16
description	0	Permalock bits	Handle	

Table 6.55. Tag reply to a successful *BlockPermalock* command with Read/Lock=1

	Header	RN	CRC-16c
# of bits	1	16	16
description	0	handle	

Upon receiving a valid *BlockPermalock* command a Tag shall perform the commanded operation, unless the Tag does not support block permalocking, in which case it shall ignore the command.

PJM Method: Refer to the PJM Method part of clauses 6.3.4.9 and 6.3.4.11. As for the *ACK-PJM*, see 6.3.4.11.2.4 and Table 6.30, this command can include multiple handles in the RN field in order to address multiple tags. Interrogators and tags shall operate (when using this command) as described above for ASK Method except that multiple tags can be addressed by this command and multiple tags can reply to this command.

7 Marking of equipment

All interrogators/readers (or the associated user manuals) shall be clearly and permanently marked stating with which National Regulations they comply.

All interrogators/readers (or the associated user manuals) shall be clearly permanently marked to show which functionalities of the EPC™ Class-1 HF RFID AI that they support.

Annex A
(informative)

Revision History

Table A.1. Revision History

Date & Version	Section(s)	Change description
Jun 23, 2008 Version 1.0.3	n.a.	Initial version
April 21, 2010 Version 2.0.1	All	Modifications to remove potential Intellectual Property
September 5, 2011 Version 2.0.3	All	Addition of statement on use of multiple *Select* commands

Annex B
(normative)

State-transition tables

State-transition Table B.1. Ready state-transition table to Table B.7.Killed state-transition table shall define a tag response to interrogator commands. The term handle used in the state-transition tables is defined in 6.3.4.4; error codes are defined in Table I.2; "slot" is the slot-counter output shown in Figure 6.27 and detailed in Annex J; "–" in the "Action" column means that a tag neither modifies its **SL** or **inventoried** flags nor loadmodulates a reply.

B.1 Present state: Ready

Table B.1. Ready state-transition table

Command	Condition	Action	Next State
BeginRound [1]	slot=0; matching **inventoried** & **SL** flags	loadmodulate StoredCRC	**reply**
	slot<>0; matching **inventoried** & **SL** flags	–	**arbitrate**
	Otherwise	–	**ready**
NextSlot	All	–	**ready**
ResizeRound	All	–	**ready**
ACK	All	–	**ready**
NAK	All	–	**ready**
Req_RN	All	–	**ready**
Select	All	assert or deassert **SL**, or set **inventoried** to *A* or *B*	**ready**
Read	All	–	**ready**
Write	All	–	**ready**
Kill	All	–	**ready**
Lock	All	–	**ready**
Access	All	–	**ready**
BlockWrite	All	–	**ready**
BlockErase	All	–	**ready**
BlockPermalock	All	–	**ready**
Invalid [2]	All	–	**ready**

Note 1: *BeginRound* starts a new round and may change the session. *BeginRound* also instructs a tag to load a new random value into its slot counter.

Note 2: "Invalid" shall mean an erroneous command, an unsupported command, a command with invalid parameters, a command with a CRC error, or any other command either not recognized or not executable by the tag.

B.2 Present state: Arbitrate

Table B.2. Arbitrate state-transition table

Command	Condition	Action	Next State
BeginRound [1,2]	slot=0; matching **inventoried** & **SL** flags	loadmodulate StoredCRC	**Reply**
	slot<>0; matching **inventoried** & **SL** flags	–	**arbitrate**
	Otherwise	–	**Ready**
NextSlot	slot=0 after decrementing slot counter	loadmodulate StoredCRC	**Reply**
	slot<>0after decrementing slot counter	–	**arbitrate**
ResizeRound [2]	slot=0	loadmodulate StoredCRC	**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**
BlockWrite	All	–	**arbitrate**
BlockErase	all	–	**arbitrate**
BlockPermalock	all	–	**arbitrate**
Invalid [3]	all	–	**arbitrate**

Note 1: *BeginRound* starts a new round and may change the session.
Note 2: *BeginRound* and *ResizeRound* instruct a tag to load a new random value into its slot counter.
Note 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 *BeginRound*) 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.

B.3 Present state: Reply

Table B.3. Reply state-transition table

Command	Condition	Action	Next State
BeginRound [1,2]	slot=0; matching **inventoried** & **SL** flags	loadmodulate StoredCRC	**Reply**
	slot<>0; matching **inventoried** & **SL** flags	–	**Arbitrate**
	otherwise	–	**Ready**
NextSlot	all	–	**Arbitrate**
ResizeRound [2]	slot=0	loadmodulate StoredCRC	**Reply**
	slot<>0	–	**Arbitrate**
ACK	valid StoredCRC	see Table 6.15	**Acknowledged**
	invalid StoredCRC	–	**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**
BlockWrite	all	–	**Arbitrate**
BlockErase	all	–	**Arbitrate**
BlockPermalock	all	–	**Arbitrate**
T_2 timeout	See Figure 6.24 and Table 6.14	–	**Arbitrate**
Invalid [3]	all	–	**Reply**

Note 1: *BeginRound* starts a new round and may change the session.
Note 2: *BeginRound* and *ResizeRound* instruct a tag to load a new random value into its slot counter.
Note 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 *BeginRound*) 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.

B.4 Present state: Acknowledged

Table B.4. Acknowledged state-transition table

Command	Condition	Action	Next State
BeginRound[1]	slot=0; matching **inventoried**[2] & **SL** flags	loadmodulate StoredCRC; transition **inventoried**[2] from *A*→*B* if and only if new session matches prior session	**reply**
	slot<>0; matching **inventoried**[2] & **SL** flags	transition **inventoried**[2] from *A*→*B* if and only if new session matches prior session	**arbitrate**
	otherwise	transition **inventoried** from *A*→*B* if and only if new session matches prior session	**ready**
NextSlot	all	transition **inventoried** from *A*→*B*	**ready**
ResizeRound	all	transition **inventoried** from *A*→*B*	**ready**
ACK	valid StoredCRC	see Table 6.15	**acknowledged**
	invalid StoredCRC	–	**arbitrate**
NAK	all	–	**arbitrate**
Req_RN	valid StoredCRC & access password<>0	loadmodulate handle	**open**
	valid StoredCRC & access password=0	loadmodulate handle	**secured**
	invalid StoredCRC	–	**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**
BlockPermalock	all	–	**arbitrate**
T_2 timeout	See Figure 6.24 and Table 6.14	–	**arbitrate**
Invalid[3]	all	–	**acknowledged**

Note 1: *BeginRound* starts a new round and may change the session. *BeginRound* also instructs a tag to load a new random value into its slot counter.

Note 2: As described 6.3.4.8, a tag transitions its **inventoried** flag prior to evaluating the condition.

Note 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 *BeginRound*) 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.

B.5 Present state: Open

Table B.5. Open state-transition table

Command	Condition	Action	Next State
BeginRound [1]	slot=0; matching **inventoried**[2] & **SL** flags	loadmodulate StoredCRC; transition **inventoried**[2] from *A*→*B* if and only if new session matches prior session	**Reply**
	slot<>0; matching **inventoried**[2] & **SL** flags	transition **inventoried**[2] from *A*→*B* if and only if new session matches prior session	**Arbitrate**
	otherwise	transition **inventoried** from *A*→*B* if and only if new session matches prior session	**Ready**
NextSlot	all	transition **inventoried** from *A*→*B*	**Ready**
ResizeRound	all	transition **inventoried** from *A*→*B*	**Ready**
ACK	valid handle	see Table 6.15	**Open**
	invalid handle	–	**Arbitrate**
NAK	all	–	**Arbitrate**
Req_RN	valid handle	loadmodulate new RN16	**Open**
	invalid handle	–	**Open**
Select	all	assert or deassert **SL**, or set **inventoried** to *A* or *B*	**Ready**
Read	valid handle & valid memory access	loadmodulate data and handle	**Open**
	valid handle & invalid memory access	loadmodulate error code	**Open**
	invalid handle	–	**Open**
Write	valid handle & valid memory access	loadmodulate handle when done	**Open**
	valid handle & invalid memory access	loadmodulate error code	**Open**
	invalid handle	–	**Open**
Kill [3] (see also 6.3.4.11.3.4)	valid handle & valid nonzero kill password & Recom = 0	loadmodulate handle when done	**Killed**
	valid handle & valid nonzero kill password & Recom <> 0	loadmodulate handle when done	**Open**
	valid handle & invalid nonzero kill password	–	**Arbitrate**
	valid handle & kill password=0	loadmodulate error code	**Open**
	invalid handle	–	**Open**
Lock	all	–	**Open**
Access (see also Figure 6.33)	valid handle & valid access password	loadmodulate handle	**Secured**
	valid handle & invalid access password	–	**Arbitrate**
	invalid handle	–	**Open**
BlockWrite	valid handle & valid memory access	loadmodulate handle when done	**Open**
	valid handle & invalid memory access	loadmodulate error code	**Open**
	invalid handle	–	**Open**
BlockErase	valid handle & valid memory access	loadmodulate handle when done	**Open**
	valid handle & invalid memory access	loadmodulate error code	**Open**
	invalid handle	–	**Open**
BlockPermalock	all	–	**Open**
Invalid [4]	all, excluding valid commands interspersed between successive *Kill* or *Access* commands in a kill or access sequence, respectively (see 6.3.4.11.3.4 and 6.3.4.11.3.6).	–	**Open**

Command	Condition	Action	Next State
	otherwise valid commands, except Req_RN or BeginRound, interspersed between successive *Kill* or *Access* commands in a kill or access sequence, respectively (see Figure 6.31 and Figure 6.33).	–	**Arbitrate**

Note 1: *BeginRound* starts a new round and may change the session. *BeginRound* also instructs a tag to load a new random value into its slot counter.
Note 2: As described in 6.3.4.8, a tag transitions its **inventoried** flag prior to evaluating the condition.
Note 3: As described in 6.3.4.11.3.4, if a tag does not implement Recom bits LSB and 2SB it has to ignore their value and treat them as if their value were zero.
Note 4: "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 *BeginRound*) 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.

B.6 Present state: Secured

Table B.6. Secured state-transition table

Command	Condition	Action	Next State
BeginRound[1]	slot=0; matching **inventoried**[2] & **SL** flags	loadmodulate StoredCRC; transition **inventoried**[2] from *A*→*B* if and only if new session matches prior session	**Reply**
	slot<>0; matching **inventoried**[2] & **SL** flags	transition **inventoried**[2] from *A*→*B* if and only if new session matches prior session	**Arbitrate**
	otherwise	transition **inventoried** from *A*→*B* if and only if new session matches prior session	**Ready**
NextSlot	all	transition **inventoried** from *A*→*B*	**Ready**
ResizeRound	all	transition **inventoried** from *A*→*B*	**Ready**
ACK	valid handle	see Table 6.15	**Secured**
	invalid handle	–	**Arbitrate**
NAK	all	–	**Arbitrate**
Req_RN	valid handle	loadmodulate new RN16	**Secured**
	invalid handle	–	**Secured**
Select	all	assert or deassert **SL**, or set **inventoried** to *A* or *B*	**Ready**
Read	valid handle & valid memory access	loadmodulate data and handle	**Secured**
	valid handle & invalid memory access	loadmodulate error code	**Secured**
	invalid handle	–	**Secured**
Write	valid handle & valid memory access	loadmodulate handle when done	**Secured**
	valid handle & invalid memory access	loadmodulate error code	**Secured**
	invalid handle	–	**Secured**
Kill[3] (see also 6.3.4.11.3.4)	valid handle & valid nonzero kill password & Recom = 0	loadmodulate handle when done	**Killed**
	valid handle & valid nonzero kill password & Recom <> 0	loadmodulate handle when done	**Secured**
	valid handle & invalid nonzero kill password	–	**Arbitrate**
	valid handle & kill password=0	loadmodulate error code	**Secured**

Command	Condition	Action	Next State
	invalid handle	–	**Secured**
Lock	valid handle & valid lock payload	loadmodulate handle when done	**Secured**
	valid handle & invalid lock payload	loadmodulate error code	**Secured**
	invalid handle	–	**Secured**
*Access (see **also** **Figure 6.33**)*	valid handle & valid access password	loadmodulate handle	**Secured**
	valid handle & invalid access password	–	**Arbitrate**
	invalid handle	–	**Secured**
BlockWrite	valid handle & valid memory access	loadmodulate handle when done	**Secured**
	valid handle & invalid memory access	loadmodulate error code	**Secured**
	invalid handle	–	**Secured**
BlockErase	valid handle & valid memory access	loadmodulate handle when done	**Secured**
	valid handle & invalid memory access	loadmodulate error code	**Secured**
	invalid handle	–	**Secured**
BlockPermalock	valid handle, valid payload, & Read/Lock = 0	loadmodulate permalock bits and handle	**Secured**
	valid handle, invalid payload, & Read/Lock = 0	loadmodulate error code	**Secured**
	valid handle, valid payload, & Read/Lock = 1	loadmodulate handle when done	**Secured**
	valid handle, invalid payload, & Read/Lock = 1	loadmodulate error code	**Secured**
	invalid handle	–	**Secured**
Invalid[4]	all, excluding valid commands interspersed between successive Kill or Access commands in a kill or access sequence, respectively (see 6.3.4.11.3.4 and 6.3.4.11.3.6).	–	**Secured**
	otherwise valid commands, except Req_RN or BeginRound, interspersed between successive Kill or Access commands in a kill or access sequence, respectively (see 6.3.4.11.3.4 and 6.3.4.11.3.6).	–	**Arbitrate**

Note 1: *BeginRound* starts a new round and may change the session. *BeginRound* also instructs a tag to load a new random value into its slot counter.

Note 2: As described in 6.3.4.8, a tag transitions its **inventoried** flag prior to evaluating the condition.

Note 3: As described in 6.3.4.11.3.4, if a tag does not implement Recom bits LSB and 2SB it has to ignore their value and treat them as if their value were zero.

Note 4: "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 *BeginRound*) 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.

B.7 Present state: Killed (optional)

Table B.7. Killed state-transition table

Command	Condition	Action	Next State
BeginRound	all	–	**killed**
NextSlot	all	–	**killed**
ResizeRound	all	–	**killed**
ACK	all	–	**killed**
NAK	all	–	**killed**
Req_RN	all	–	**killed**
Select	all	–	**killed**
Read	all	–	**killed**
Write	all	–	**killed**
Kill	all	–	**killed**
Lock	all	–	**killed**
Access	all	–	**killed**
BlockWrite	all	–	**killed**
BlockErase	all	–	**killed**
BlockPermalock	all	–	**killed**
Invalid[1]	all	–	**killed**

Note 1: “Invalid” shall mean an erroneous command, an unsupported command, a command with invalid parameters, a command with a CRC error, or any other command either not recognized or not executable by the tag.

Annex C
(normative)

Command-Response tables

Command-response Table C.1. Power-up command-response table to Table C.18. Invalid command-response table shall define a tag response to interrogator commands. The term "handle" used in the state-transition tables is defined in 6.3.4.4; error codes are defined in Table I.2; "slot" is the slot counter output shown in Figure 6.27 and detailed in Annex J; "–" in the "Response" column means that a tag neither modifies its **SL** or **inventoried** flags nor loadmodulates a reply.

C.1 Command response: Power-up

Table C.1. Power-up command-response table

Starting State	Condition	Response	Next State
ready, arbitrate, reply, acknowledged, open, secured	power-up	–	**ready**
killed	all	–	**killed**

C.2 Command response: *BeginRound*

Table C.2. *BeginRound*[1] command-response table

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

Note 1: *BeginRound* (in any other state than **killed**) starts a new round and may change the session. *BeginRound* also instructs a tag to load a new random value into its slot counter.
Note 2: As described in 6.3.4.8, a tag transitions its **inventoried** flag prior to evaluating the condition.

C.3 Command response: *NextSlot*

Table C.3. *NextSlot* command-response table[1]

Starting State	Condition	Response	Next State
ready	All	–	**ready**
arbitrate	slot<>0 after decrementing slot counter	–	**arbitrate**
	slot=0 after decrementing slot counter	loadmodulate StoredCRC	**reply**
reply	All	–	**arbitrate**
acknowledged, open, secured	All	transition **inventoried** from *A*→*B*	**ready**
killed	all	–	**killed**

Note 1: See Table C.17 for the tag response to a *NextSlot* whose session parameter does not match that of the current inventory round.

C.4 Command response: *ResizeRound*

Table C.4. *ResizeRound*[1] command-response table[2]

Starting State	Condition	Response	Next State
ready	All	–	**ready**
arbitrate, reply	slot<>0	–	**arbitrate**
	slot=0	loadmodulate StoredCRC	**reply**
acknowledged, open, secured	All	transition **inventoried** from *A*→*B*	**ready**
killed	All	–	**killed**

Note 1: *ResizeRound,* in the **arbitrate** or **reply** states, instructs a tag to load a new random value into its slot counter.
Note 2: See Table C.17 for the tag response to a *NextSlot* whose session parameter does not match that of the current inventory round.

C.5 Command response: *Ack*

Table C.5. *Ack* command-response table

Starting State	Condition	Response	Next State
ready	all	–	**ready**
arbitrate	all	–	**arbitrate**
reply	valid StoredCRC	see Table 6.15	**acknowledged**
	invalid StoredCRC	–	**arbitrate**
acknowledged	valid StoredCRC	see Table 6.15	**acknowledged**
	invalid StoredCRC	–	**arbitrate**
open	valid handle	see Table 6.15	**open**
	invalid handle	–	**arbitrate**
secured	valid handle	see Table 6.15	**secured**
	invalid handle	–	**arbitrate**
killed	all	–	**killed**

C.6 Command response: *NAK*

Table C.6. *Nak* command-response table

Starting State	Condition	Response	Next State
ready	All	–	**ready**
arbitrate, reply, acknowledged, open, secured	All	–	**arbitrate**
killed	All	–	**killed**

C.7 Command response: *Req_RN*

Table C.7. *Req_RN* command-response table

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

C.8 Command response: *Select*

Table C.8. *Select* command-response table

Starting State	Condition	Response	Next State
ready, arbitrate, reply, acknowledged, open, secured	All	assert or deassert **SL**, or set **inventoried** to *A* or *B*	**ready**
killed	All	–	**killed**

C.9 Command response: *Read*

Table C.9. *Read* command-response table

Starting State	Condition	Response	Next State
ready	All	–	**ready**
arbitrate, reply, acknowledged	All	–	**arbitrate**
open	valid handle & invalid memory access	loadmodulate error code	**open**
	valid handle & valid memory access	loadmodulate data and handle	**open**
	invalid handle	–	**open**
secured	valid handle & invalid memory access	loadmodulate error code	**secured**
	valid handle & valid memory access	loadmodulate data and handle	**secured**
	invalid handle	–	**secured**
killed	All	–	**killed**

C.10 Command response: *Write*

Table C.10. *Write* command-response table

Starting State	Condition	Response	Next State
ready	All	–	**ready**
arbitrate, reply, acknowledged	All	–	**arbitrate**
open	valid handle & invalid memory access	loadmodulate error code	**open**
	valid handle & valid memory access	loadmodulate handle when done	**open**
	invalid handle	–	**open**
secured	valid handle & invalid memory access	loadmodulate error code	**secured**
	valid handle & valid memory access	loadmodulate handle when done	**secured**
	invalid handle	–	**secured**
killed	All	–	**killed**

C.11 Command response: *Kill*

Table C.11. *Kill*[1] command-response table

Starting State	Condition	Response	Next State
ready	all	–	**ready**
arbitrate, reply, acknowledged	all	–	**arbitrate**
open[2]	valid handle & kill password=0	loadmodulate error code	**open**
	valid handle & invalid nonzero kill password	–	**arbitrate**
	valid handle & valid nonzero kill password & Recom=0	loadmodulate handle when done	**killed**
	valid handle & valid nonzero kill password & Recom<>0	loadmodulate handle when done	**open**
	invalid handle	–	**open**
secured[2]	valid handle & kill password=0	loadmodulate error code	**secured**
	valid handle & invalid nonzero kill password	–	**arbitrate**
	valid handle & valid nonzero kill password & Recom=0	loadmodulate handle when done	**killed**
	valid handle & valid nonzero kill password & Recom<>0	loadmodulate handle when done	**secured**
	invalid handle	–	**secured**
Killed	all	–	**killed**

Note 1: See also Figure 6.31.
Note 2: As described in 6.3.4.11.3.4, if a Tag does not implement Recom bits LSB and 2SB it has to ignore their value and treat them as if their value were zero

C.12 Command response: *Lock*

Table C.12. *Lock* command-response table

Starting State	Condition	Response	Next State
ready	all	–	**ready**
arbitrate, reply, acknowledged	all	–	**arbitrate**
open	all	–	**open**
secured	valid handle & invalid lock payload	loadmodulate error code	**secured**
	valid handle & valid lock payload	loadmodulate handle when done	**secured**
	invalid handle	–	**secured**
killed	all	–	**killed**

C.13 Command response: *Access*

Table C.13. *Access*[1] command-response table

Starting State	Condition	Response	Next State
ready	all	–	**ready**
arbitrate, reply, acknowledged	all	–	**arbitrate**
open	valid handle & invalid access password	–	**arbitrate**
	valid handle & valid access password	loadmodulate handle	**secured**
	invalid handle	–	**open**
secured	valid handle & invalid access password	–	**arbitrate**
	valid handle & valid access password	loadmodulate handle	**secured**
	invalid handle	–	**secured**
killed	all	–	**killed**

Note 1: See also Figure 6.33.

C.14 Command response: *BlockWrite*

Table C.14. *BlockWrite* command-response table

Starting State	Condition	Response	Next State
ready	All	–	**ready**
arbitrate, reply, acknowledged	All	–	**arbitrate**
open	valid handle & invalid memory access	loadmodulate error code	**open**
	valid handle & valid memory access	loadmodulate handle when done	**open**
	invalid handle	–	**open**
secured	valid handle & invalid memory access	loadmodulate error code	**secured**
	valid handle & valid memory access	loadmodulate handle when done	**secured**
	invalid handle	–	**secured**
killed	All	–	**killed**

C.15 Command response: *BlockErase*

Table C.15. *BlockErase* command-response table

Starting State	Condition	Response	Next State
ready	All	–	**ready**
arbitrate, reply, acknowledged	All	–	**arbitrate**
open	valid handle & invalid memory access	loadmodulate error code	**open**
	valid handle & valid memory access	loadmodulate handle when done	**open**
	invalid handle	–	**open**
secured	valid handle & invalid memory access	loadmodulate error code	**secured**
	valid handle & valid memory access	loadmodulate handle when done	**secured**
	invalid handle	–	**secured**
killed	All	–	**killed**

C.16 Command response: *BlockPermalock*

Table C.16. *BlockPermalock* command-response table

Starting State	Condition	Response	Next State
ready	all	–	**ready**
arbitrate, reply, acknowledged	all	–	**arbitrate**
open	all	–	**open**
secured	valid handle, valid payload, & **Read/Lock** = 0	loadmodulate permalock bits and handle	**secured**
	valid handle, invalid payload, & **Read/Lock** = 0	loadmodulate error code	**secured**
	valid handle, valid payload, & **Read/Lock** = 1	loadmodulate handle when done	**secured**
	valid handle, invalid payload, & **Read/Lock** = 1	loadmodulate error code	**secured**
	invalid handle	–	**secured**
killed	all	–	**killed**

C.17 Command response: T_2 timeout

Table C.17. T_2 timeout command-response table

Starting State	Condition	Response	Next State
ready	all	–	**ready**
arbitrate	all	–	**arbitrate**
reply, acknowledged	See Figure 6.24 and Table 6.14	–	**arbitrate**
open	all	–	**open**
secured	all	–	**secured**
killed	all	–	**killed**

C.18 Command response: Invalid command

Table C.18. Invalid command-response table

Starting State	Condition	Response	Next State
ready[1]	All	–	**Ready**
arbitrate[2]	All	–	**Arbitrate**
reply[2]	All	–	**Reply**
acknowledged[2]	All	–	**Acknowledged**
open[2]	all, excluding valid commands interspersed between successive *Kill* or *Access* commands in a kill or access sequence, respectively (see 6.3.4.11.3.4 and 6.3.4.11.3.6).	–	**Open**
	otherwise valid commands, except *Req_RN* or *BeginRound*, interspersed between successive Kill or Access commands in a kill or access sequence, respectively (see 6.3.4.11.3.4 and 6.3.4.11.3.6).	–	**Arbitrate**
secured[2]	all, excluding valid commands interspersed between successive *Kill* or *Access* commands in a kill or access sequence, respectively (see 6.3.4.11.3.4 and 6.3.4.11.3.6).	–	**Secured**
	otherwise valid commands, except *Req_RN* or *BeginRound*, interspersed between successive *Kill* or *Access* commands in a kill or access sequence, respectively (see 6.3.4.11.3.4 and 6.3.4.11.3.6).	–	**Arbitrate**
killed[1]	All	–	**Killed**

Note 1: "Invalid" shall mean an erroneous command, an unsupported command, a command with invalid parameters, a command with a CRC error, or any other command either not recognized or not executable by the tag.

Note 2: "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 *BeginRound*) 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.

Annex D
(informative)

Example slot-count (Q) selection algorithm

D.1 Example algorithm an interrogator might use to choose *Q*

Figure D.1 shows an algorithm an interrogator might use for setting the slot-count parameter *Q* in a *BeginRound* command. Q_{fp} is a floating-point representation of *Q;* an interrogator rounds Q_{fp} to an integer value and substitutes this integer value for *Q* in the *BeginRound.* Typical values for *C* are $0{,}1 < C < 0{,}5$. An interrogator typically uses small values of *C* when *Q* is large, and larger values of *C* when *Q* is small.

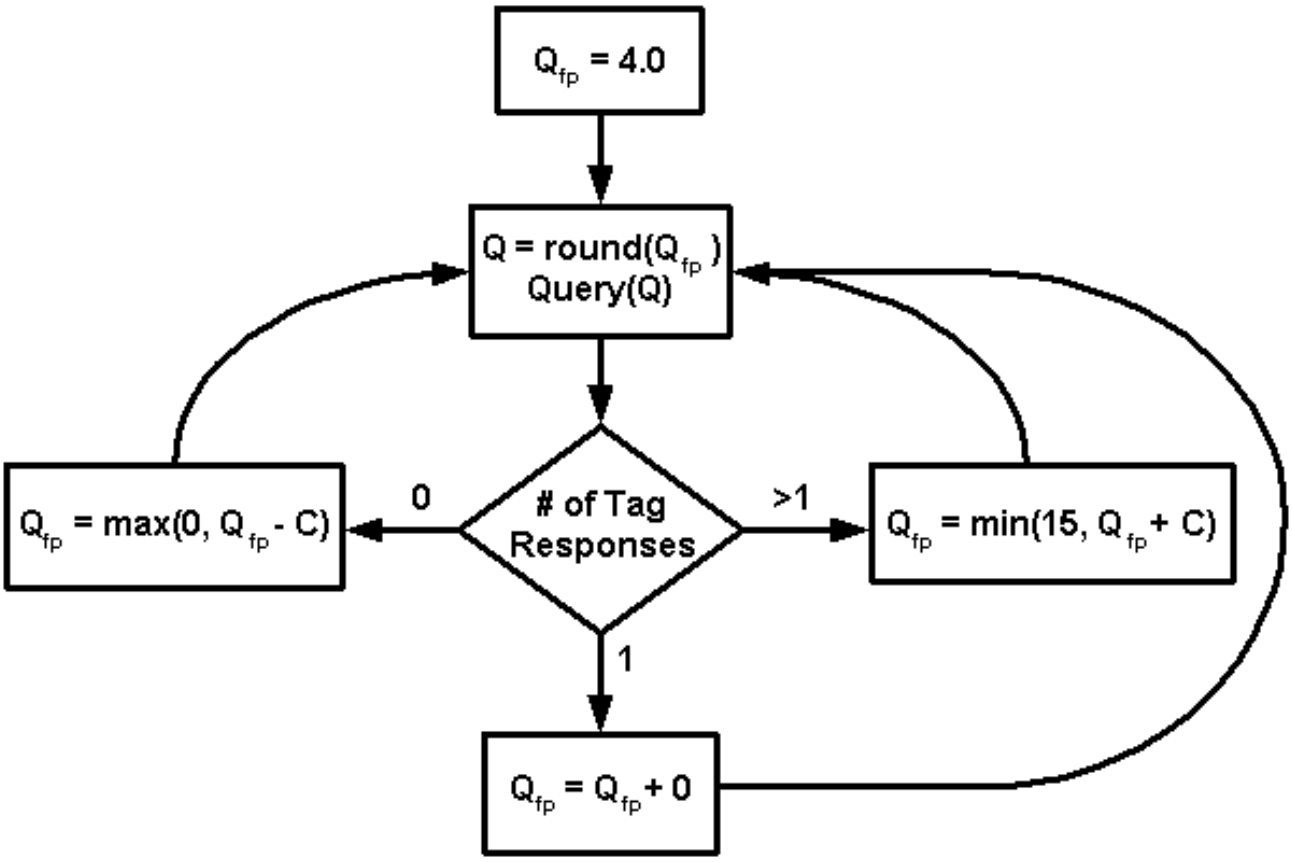


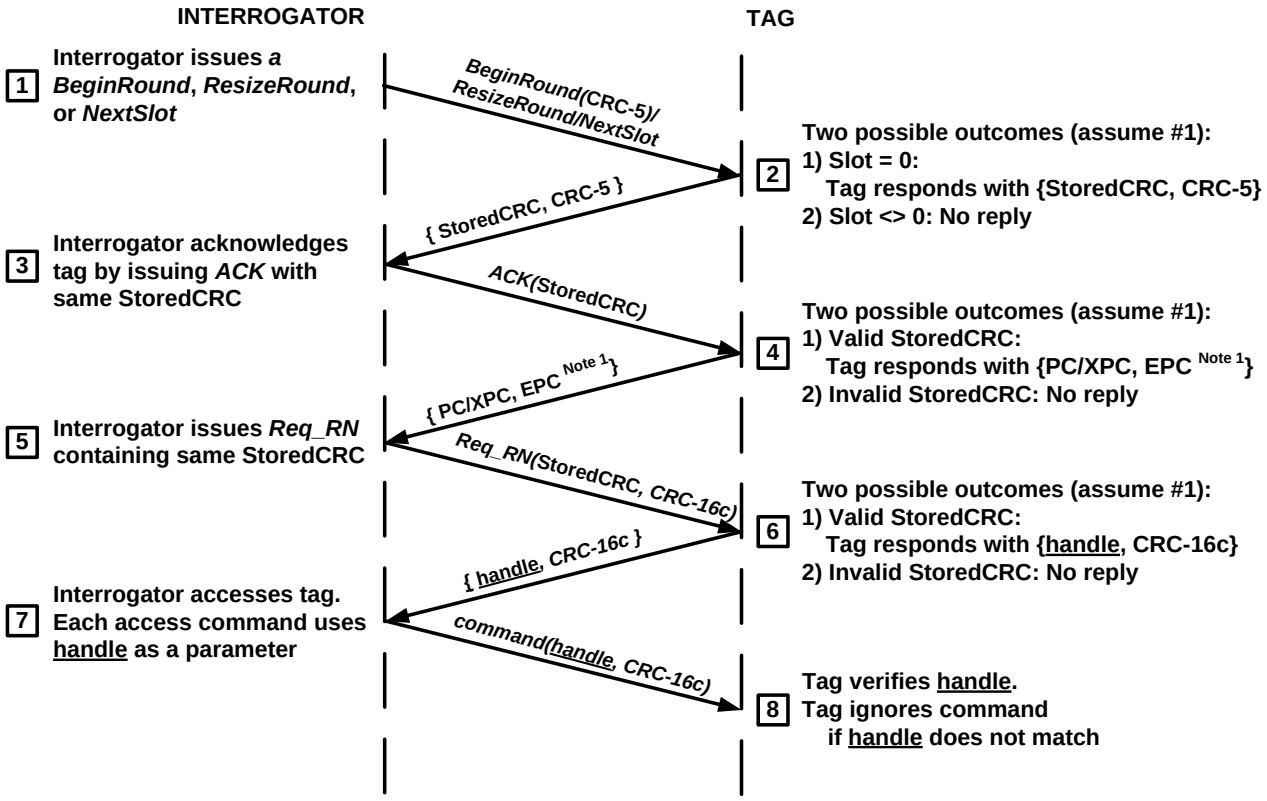
Figure D.1 Example algorithm for choosing the slot count parameter Q

Annex E
(informative)

Example of tag inventory and access

E.1 ASK Method: Example inventory and access of a single tag.

Figure E.1 shows the steps by which an interrogator inventories and accesses a single tag for ASK Method.

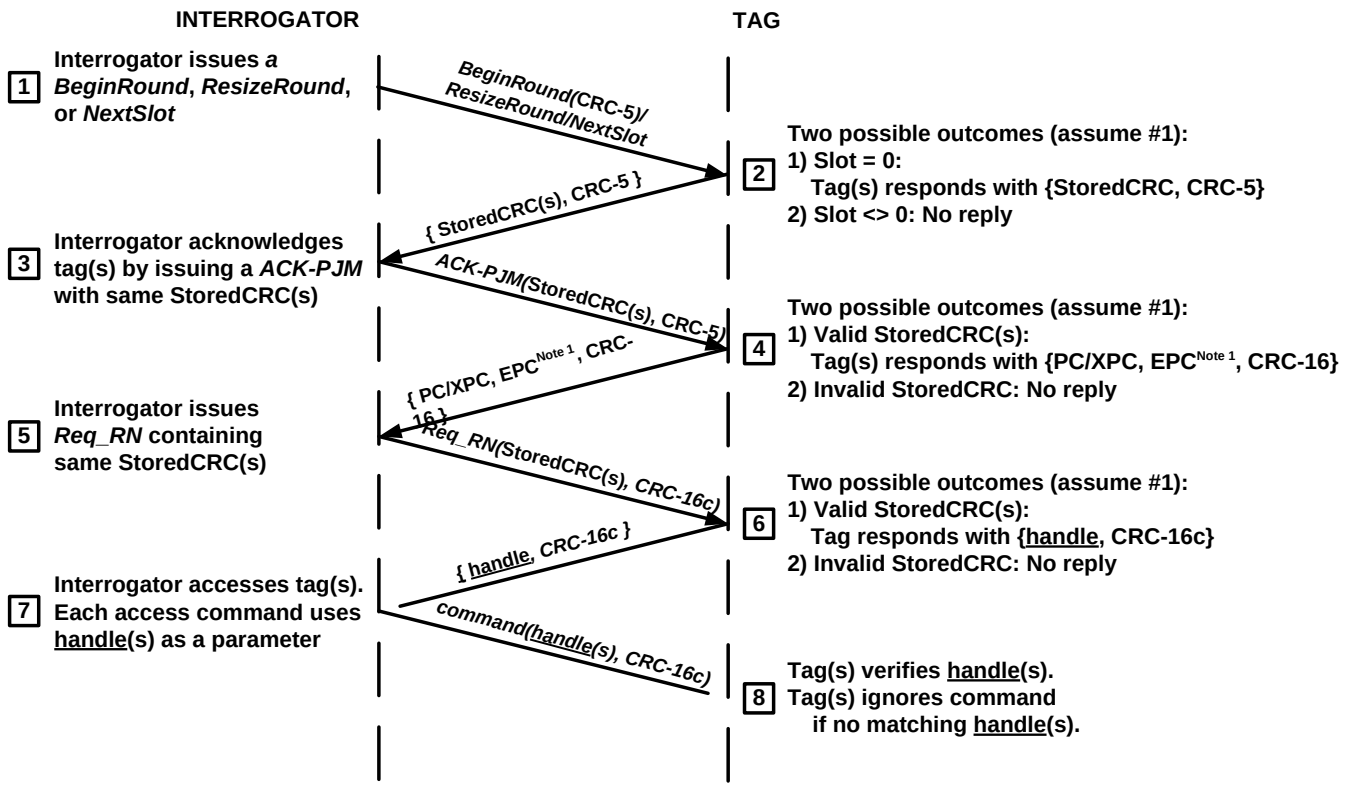


Note 1: In certain cases a PacketCRC should be added if required. See Table 6.15 for the definition of the cases.

Figure E.1 ASK Method: Example of tag inventory and access

E.2 PJM Method: Example inventory and access of a single or multiple tags

Figure E.2 shows the steps by which an Interrogator inventories and accesses a single or multiple tags for PJM Method.



Note 1: In certain cases a PacketCRC should be added if required. See Table 6.15 for the definition of the cases.

Figure E.2 PJM Method: Example of tag inventory and access

Annex F
(informative)

Calculation of 5-bit and 16-bit cyclic redundancy checks

F.1Example CRC-5 encoder/decoder

An exemplary schematic diagram for a CRC-5 encoder/decoder is shown in Figure F.1, using the polynomial and preset defined in Figure 6.24.

To calculate a CRC-5, first preload the entire CRC register (*i.e.* C[4:0]) with 01001_2, then clock the data bits to be encoded into the input labeled DATA, MSB first. After clocking in all the data bits, C[4:0] holds the CRC-5 value.

To decode a CRC-5, first preload the entire CRC register (C[4:0]) with 01001_2, then clock the received data and CRC-5 {data, CRC-5} bits into the input labeled DATA, MSB first. The CRC-5 check passes if C[4:0]=00000_2.

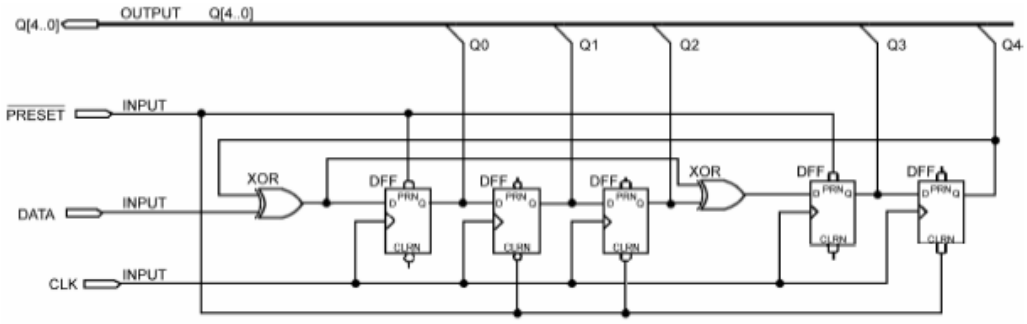


Figure F.1 Example CRC-5 circuit

Table F.1. CRC-5 register preload values

Register	Preload value
Q0	1
Q1	0
Q2	0
Q3	1
Q4	0

F.2Example CRC-16 calculations

This example shows the StoredCRC (a CRC-16) that a Tag would calculate at power-up. As shown in Figure 6.25, EPC memory contains a StoredCRC starting at address 00_h, a StoredPC starting at address 10_h, zero or more EPC words starting at address 20_h, and an XPC_W1 starting at address 210_h, and an optional XPC_W2 starting at address 220_h. As described in 6.3.4.1.2.1, a Tag calculates its StoredCRC over its StoredPC and EPC, but omits the XPC_W1 and XPC_W2 from the calculation. Figure 6.24 shows the StoredCRC that a Tag would

calculate and logically map into EPC memory at power-up, for the indicated example StoredPC and EPC word values. In each successive column, one more word of EPC memory is written, with the entire EPC memory written in the rightmost column. The indicated StoredPC values correspond to the number of EPC words written, with StoredPC bits 15_h–$1F_h$ set to zero. Entries marked N/A mean that that word of EPC memory is not included as part of the CRC calculation.

Table F.2. EPC memory contents for an example **tag**

EPC word starting address	EPC word contents	EPC wordvalues						
00_h	CRC-16	$E2F0_h$	$CCAE_h$	$968F_h$	$78F6_h$	$C241_h$	$2A91_h$	1835_h
10_h	PC	0000_h	0800_h	1000_h	1800_h	2000_h	2800_h	3000_h
20_h	EPC word 1	N/A	1111_h	1111	1111_h	1111_h	1111_h	1111_h
30_h	EPC word 2	N/A	N/A	2222_h	2222_h	2222_h	2222_h	2222_h
40_h	EPC word 3	N/A	N/A	N/A	3333_h	3333_h	3333_h	3333_h
50_h	EPC word 4	N/A	N/A	N/A	N/A	4444_h	4444_h	4444_h
60_h	EPC word 5	N/A	N/A	N/A	N/A	N/A	5555_h	5555_h
70_h	EPC word 6	N/A	N/A	N/A	N/A	N/A	N/A	6666_h

F.3 Example CRC-16c encoder/decoder

An exemplary schematic diagram for a CRC-16c encoder/decoder is shown in Figure F.2, using the polynomial and preset defined in Figure 6.24 (the polynomial used to calculate the CRC-16c, $x^{16} + x^{12} + x^5 + 1$, is the CRC-CCITT International Standard, ITU Recommendation X.25). This is applicable for both CRC-16 and CRC-16c calculations.

To encode a CRC-16c, first preload the entire CRC register (*i.e.* C[15:0]) with $FFFF_h$, then clock the data bits to be encoded into the input labeled DATA, MSB first. After clocking in all the data bits, C[15:0] holds the ones complement of the CRC-16c value.

To decode a CRC-16c, first preload the entire CRC register (C[15:0]) with $FFFF_h$, then clock the received data and CRC-16c {data, CRC-16c} bits into the input labeled DATA, MSB first. The CRC-16c check passes if C[15:0] $=1D0F_h$

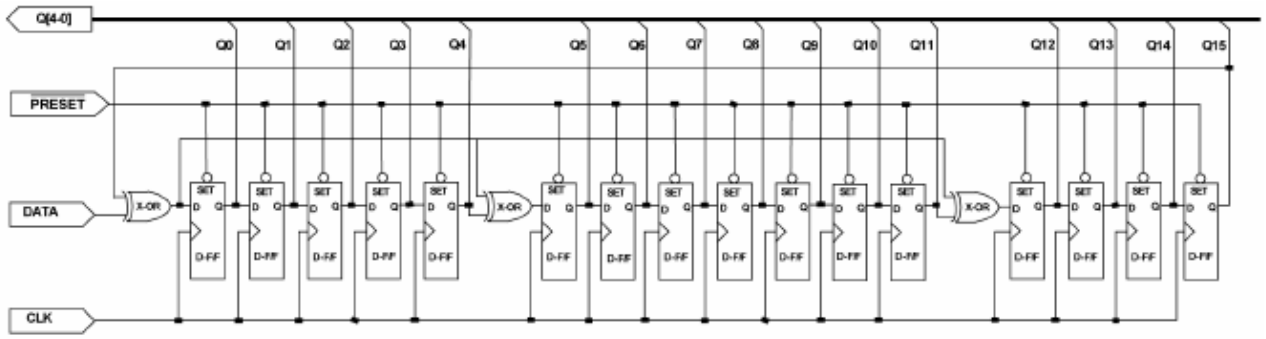


Figure F.2 Example CRC-16c circuit

Annex G
(informative)

Phase Jitter Modulation (PJM)

G.1 PJM concept

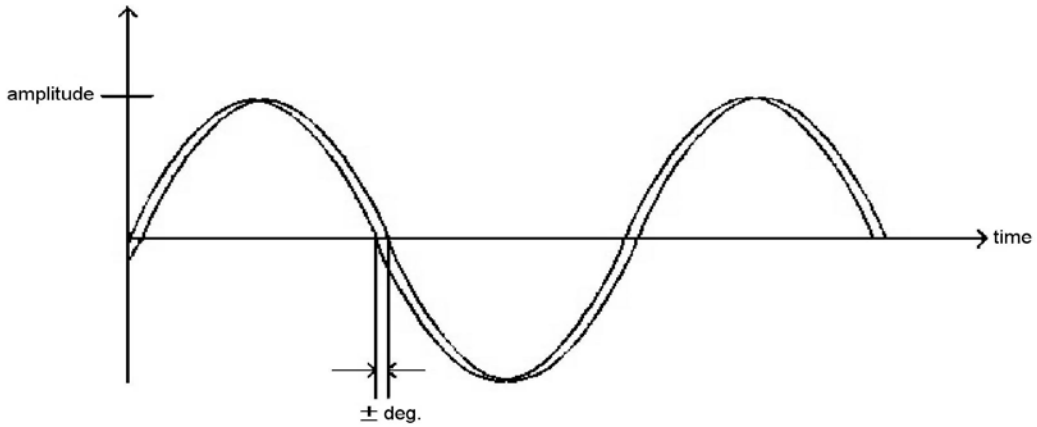


Figure G.1 Phase jitter modulation

PJM consists of two components:

1. An in-phase (0°) powering signal I.
2. A low level quadrature (90°) data signal ±Q.

The PJM waveform is the sum of these two signals. In phasor notation, these can be represented as shown in Figure G.2.

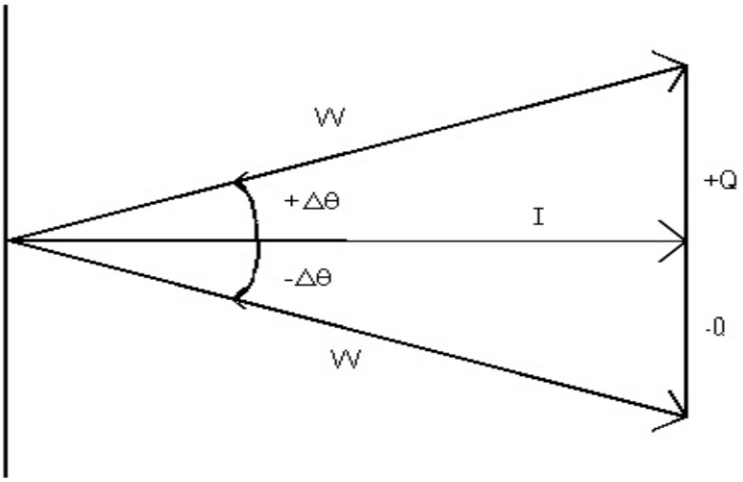


Figure G.2 Frequency spectrum

The frequency spectrum of the phasor components are shown in Figure G.3:

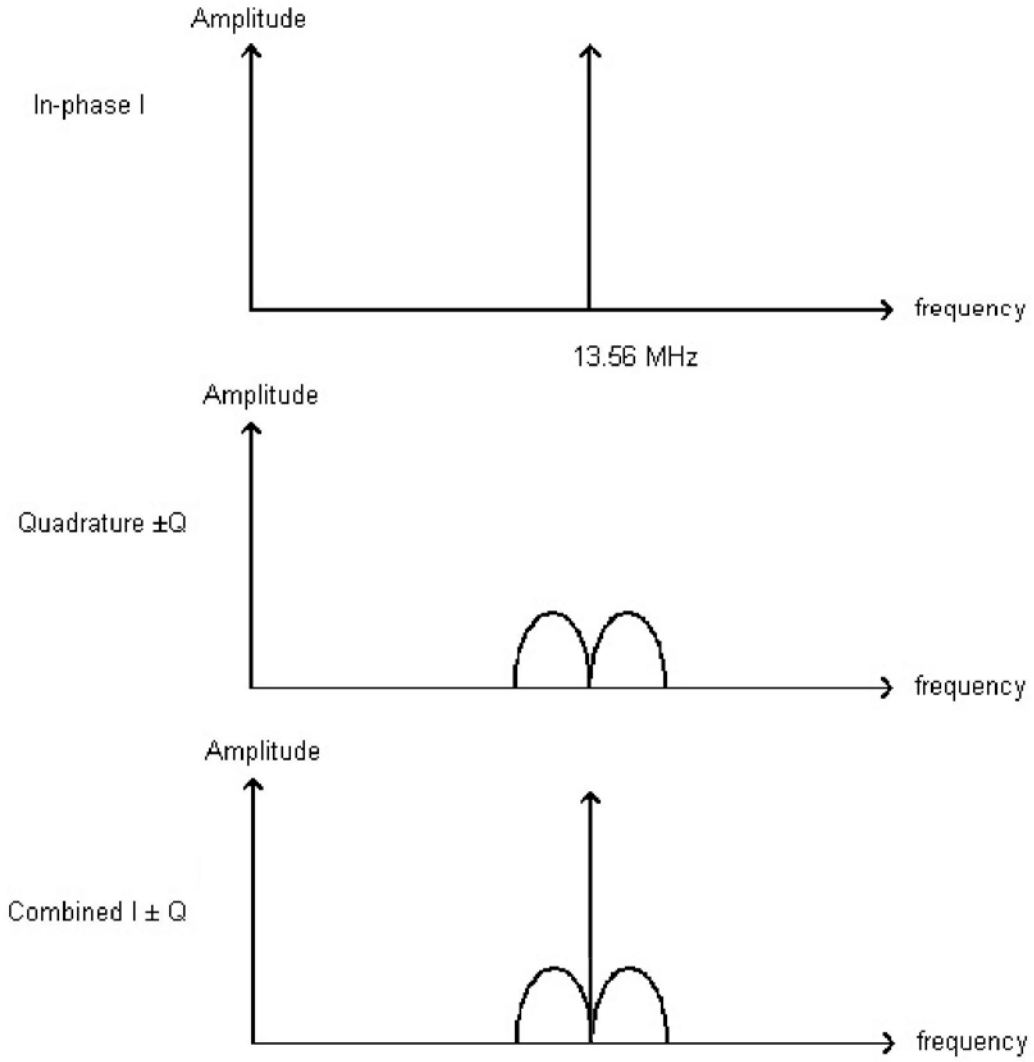


Figure G.3 Generation of PJM

- The phase change details are defined in Int:7 and 6.3.3.1.2.5.
- The interrogator command bit rate is 212 kbit/s, see Int:9 in Table 6.1.

Features of PJM are:

- Constant amplitude signal with constant power transfer.
- Sideband levels independent of data rate and can be adjusted to suit regulations.
- Very high speed data can be transmitted because PJM bandwidth is no wider than the original double-sided data bandwidth.
- Narrow bandwidth antennas do not limit high speed PJM signals. PJM can be pre-compensated to cancel for the effect of antenna bandwidth.

G.2 PJM Example implementations

Example implementations of PJM are given in figures G.4 and G.5:

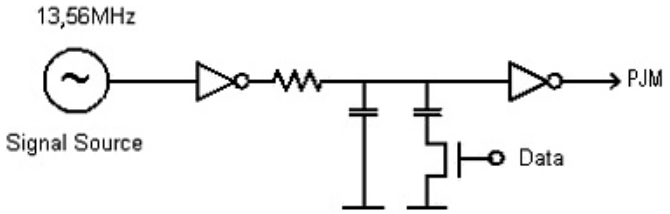


Figure G.4 Circuit for providing a data controlled variable phase delay for generating PJM

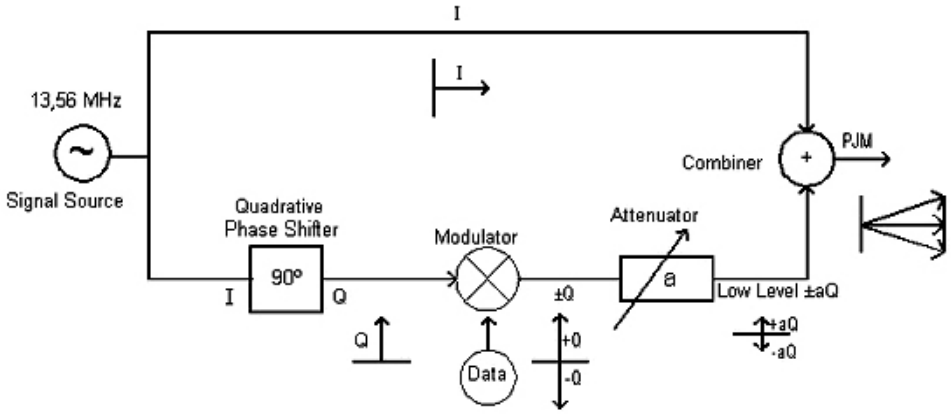


Figure G.5 Circuit for the generation of PJM showing the various elements of a PJM signal

Annex H
(informative)

ASK Method: Interrogator-to-tag link modulation

H.1 Baseband waveforms, modulated RF, and detected waveforms

Figure H.1 shows R=>T baseband and modulated waveforms as generated by an interrogator, and the corresponding waveforms envelope-detected by a tag, for DSB-ASK modulation.

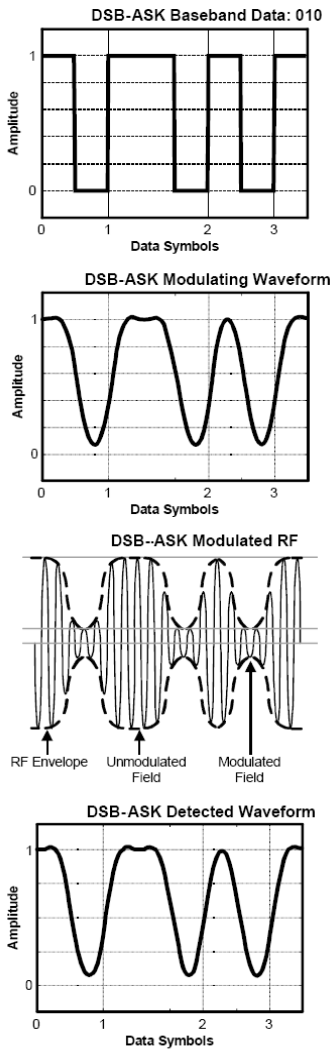


Figure H.1 Interrogator-to-tag modulation

Note: Drawing not to scale. The DSB-ASK Modulated RF waveform has to respect the modulation index specified in Table 6.5.

Annex I
(normative)

Error codes

I.1 Tag error codes and their usage

If a tag encounters an error when executing an access command that reads from or writes to memory, and if the command is a handle-based command (*i.e. Read, Write, Kill, Lock, BlockWrite,* or *BlockErase*, or *BlockPermalock),* then the tag shall loadmodulate an error code as shown in Table I.1 instead of its normal reply.

- If the tag supports error-specific codes, it shall use the error-specific codes shown in Table I.2. Tag error codes.
- If the tag does not support error-specific codes, it shall loadmodulate error code 00001111_2 (indicating a non-specific error) as shown in Table I.2. Tag error codes.
- Tags shall loadmodulate error codes only from the **open** or **secured** states.
- A tag shall not loadmodulate an error code if it receives an invalid access command; instead, it shall ignore the command.
- If an error is described by more than one error code, the more specific error code shall take precedence and shall be the code that the tag loadmodulates.
- The header for an error code is a 1-bit, unlike the header for a normal tag response, which is a 0-bit.

Table I.1. Tag-error reply format

	Header	Error Code	RN	CRC-16c
# of bits	1	8	16	16
description	1	Error code	handle	

Table I.2. Tag error codes

Error-Code Support	Error Code	Error-Code Name	Error Description
Error-specific	00000000_2	Other error	"Catch-all" for errors not covered by other codes
	00000011_2	Memory overrun	The specified memory location does not exist or the EPC length field is not supported by the tag
	00000100_2	Memory locked	The specified memory location is locked and/or permalocked and is either not writeable or not readable
	00001011_2	Insufficient power	The tag has insufficient power to perform the memory-write operation
Non-specific	00001111_2	Non-specific error	The tag does not support error-specific codes

Annex J
(normative)

Slot counter

J.1 Slot-counter operation

As described in 6.3.4.4.8, tags implement a 15-bit slot counter. As described in 6.3.4.8, interrogators use the slot counter to regulate the probability of a tag responding to a *BeginRound, ResizeRound,* or *NextSlot* command. Upon receiving a *BeginRound* or *ResizeRound* a tag preloads a *Q*-bit value, drawn from the tag RNG (see 6.3.4.4.8), into its slot counter. *Q* is an integer in the range (0 - 15). A *BeginRound* specifies *Q;* a *ResizeRound* may modify *Q* from the prior *BeginRound.* Upon receiving a *NextSlot,* a tag decrements its slot value. Tags transition to the **reply** state when their slot value reaches 0000_h. The slot counter implements continuous counting, meaning that, after the slot value decrements to 0000_h, the next *NextSlot* causes it to roll over and begin counting down from $7FFF_h$. Tags that return to **arbitrate** (for example, from the **reply** state) with a slot value of 0000_h decrement their slot value from 0000_h to $7FFF_h$ at the next *NextSlot* (with matching session) and, because their slot value is now non-zero, remain in **arbitrate.**

Annex B and Annex C contain tables describing a tag response to interrogator commands; “slot” is a parameter in these tables.

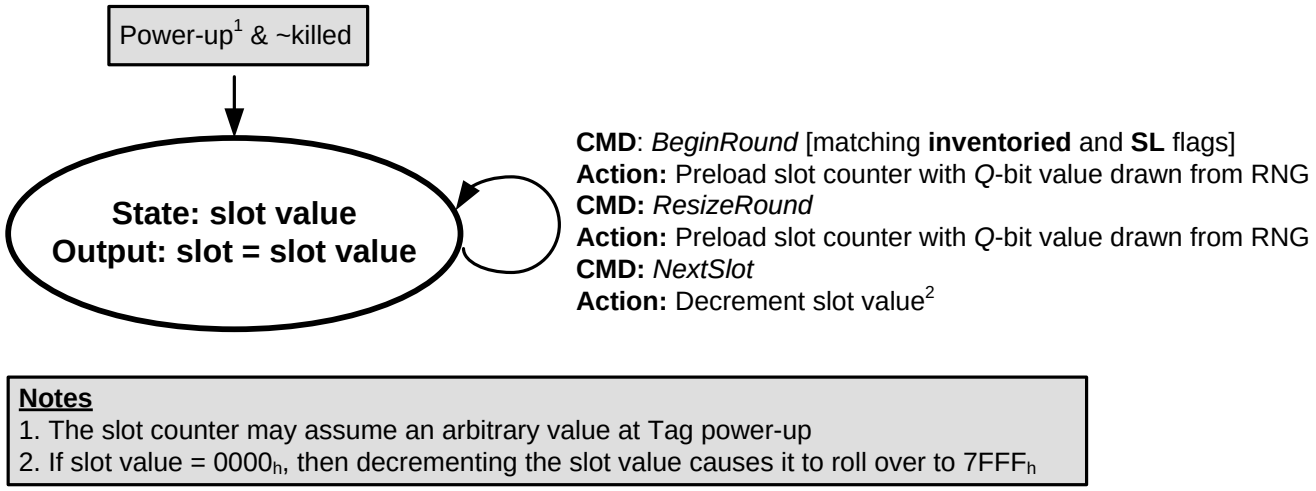


Figure J.1 Slot-counter state diagram

Annex K
(informative)

Example data flow exchange

K.1 Overview of the data-flow exchange

The following example describes a data exchange, between an interrogator and a single tag, during which the interrogator reads the kill password stored in the tag Reserved memory. This example assumes that:

- The tag has been singulated and is in the **acknowledged** state.
- The tag Reserved memory is locked but not permalocked, meaning that the interrogator must issue the access password and transition the tag to the **secured** state before performing the read operation.
- The random numbers the tag generates (listed in sequence, and not random for reasons of clarity) are:
  - CRC16 $XXXX_h$ (the StoredCRC the tag transmitted prior to entering **acknowledged**)
  - RN16_1 1601_h (shall become the handle for the entire access sequence)
  - RN16_2 1602_h
  - RN16_3 1603_h
- The tag EPC is 64 bits in length.
- The tag access password is $ACCEC0DE_h$.
- The tag kill password is $DEADC0DE_h$.
- The 1st half of the access password EXORed with RN16_2=$ACCE_h \otimes 1602_h=BACC_h$.
- The 2nd half of the access password EXORed with RN16_3=$C0DE_h \otimes 1603_h=D6DD_h$.

K.2 Tag memory contents and lock-field values

Table K.1. Tag memory contents and Table K.2. Lock-field values show the example tag memory contents and lock-field values, respectively.

Table K.1. Tag memory contents

Memory Bank	Memory Contents	Memory Addresses	Memory Values
TID	TID[15:0]	10_h–$1F_h$	$54E2_h$
	TID[31:16]	00_h–$0F_h$	$A986_h$
EPC	EPC[15:0]	50_h–$5F_h$	3210_h
	EPC[31:16]	40_h–$4F_h$	7654_h
	EPC[47:32]	30_h–$3F_h$	$BA98_h$
	EPC[63:48]	20_h–$2F_h$	$FEDC_h$
	StoredPC[15:0]	10_h–$1F_h$	2000_h
	StoredCRC-16[15:0]	00_h–$0F_h$	$XXXX_h$ as calculated (see Annex F)
Reserved	access password[15:0]	30_h–$3F_h$	$C0DE_h$
	access password[31:16]	20_h–$2F_h$	$ACCE_h$
	kill password[15:0]	10_h–$1F_h$	$C0DE_h$
	kill password[31:16]	00_h–$0F_h$	$DEAD_h$

Table K.2. Lock-field values

Kill Password		Access Password		EPC Memory		TID Memory		User Memory	
1	0	1	0	0	0	0	0	N/A	N/A

K.3 Data-flow exchange and command sequence

The data-flow exchange follows the ***Access*** procedure outlined in Figure 6.33 with a *Read* command added at the end. The sequence of interrogator commands and tag replies is:

Step 1: *Req_RN*[CRC16, CRC-16c]
Tag loadmodulates RN 16_1, which becomes the handle for the entire access sequence

Step 2: *Req_RN*[handle CRC-16c]
Tag loadmodulates RN 16_2

Step 3: *Access*[access password[31:16] EXORed with RN16_2, handle CRC-16c]
Tag loadmodulates handle

Step 4: *Req_RN*[handle CRC-16c]|
Tag loadmodulates RN16_3

Step 5: *Access*[access password[15:0] EXORed with RN16_3, handle CRC-16c]
Tag loadmodulates handle

Step 6: *Read*[MemBank=Reserved, WordPtr=00_h, WordCount=2, handle CRC-16c]
Tag loadmodulates kill password

Table K.3. Interrogator commands and tag replies shows the detailed interrogator commands and tag replies. For reasons of clarity, the CRC-16c has been omitted from all commands and replies.

Table K.3. Interrogator commands and tag replies

Step	Data Flow	Command	Parameter and/or Data	Tag State
1a: *Req_RN* command	R => T	11000001	0001 0110 0000 0000 (CRC16=$XXXX_h$)	**acknowledged**
1b: tag response	T => R		0001 0110 0000 0001 (handle=1601_h)	**→ open**
2a: *Req_RN* command	R => T	11000001	0001 0110 0000 0001 (handle=1601_h)	**open**
2b: tag response	T => R		0001 0110 0000 0010 (RN16_2=1602_h)	**→ open**
3a: *Access* command	R => T	11000110	1011 1010 1100 1100 ($BACC_h$) 0001 0110 0000 0001 (handle=1601_h)	**open**
3b: tag response	T => R		0001 0110 0000 0001 (handle=1601_h)	**→ open**
4a: *Req_RN* command	R => T	11000001	0001 0110 0000 0001 (handle=1601_h)	**open**
4b: tag response	T => R		0001 0110 0000 0011 (RN16_2=1603_h)	**→ open**
5a: *Access* command	R => T	11000110	1101 0110 1101 1101 ($D6DD_h$) 0001 0110 0000 0001 (handle=1601_h)	**open**
5b: tag response	T => R		0001 0110 0000 0001 (handle=1601_h)	**→ secured**
6a: *Read* command	R => T	11000010	00 (MemBank=Reserved) 00000000(WordPtr=kill password) 00000010(WordCount=2) 0001 0110 0000 0001 (handle=1601_h)	**secured**
6b: tag response	T => R		0 (header) 1101 1110 1010 1101 ($DEAD_h$) 1100 0000 1101 1110 ($C0DE_h$)	**→ secured**

Note: The example above shows the optional usage of the cover-coding of the access password.

Annex L
(informative)

Optional Tag Features

The following options are available to Tags certified to this protocol.

L.1Optional Tag passwords

Kill password: A Tag may optionally implement a kill password. A Tag that does not implement a kill password operates as if it has a zero-valued kill password that is permanently read/write locked. See 6.3.4.1.1.1.

Access password: A Tag may optionally implement an access password. A Tag that does not implement an access password operates as if it has a zero-valued access password that is permanently read/write locked. See 6.3.4.1.1.2.

L.2Optional Tag memory banks and memory-bank sizes

Reserved memory: Reserved memory is optional. If a Tag does not implement either a kill password or an access password, then the Tag need not physically implement Reserved memory. Because a Tag with non-implemented passwords operates as if it has zero-valued password(s) that are permanently read/write locked, these passwords must still be logically addressable in Reserved memory at the memory locations specified in 6.3.4.1.1.1 and 6.3.4.1.1.2.

EPC memory: EPC memory is required, but its size is vendor-defined. The minimum size is 32 bits, to contain a 16-bit StoredCRC and 16-bit StoredPC. EPC memory may be larger than 32 bits, to contain an EPC whose vendor-specified length may be 16 to 464 bits in 16-bit increments, as well as a mandatory XPC_W1 and an optional XPC_W2. See 6.3.4.1.2.

TID memory: TID memory is required, but its size is vendor-defined. The minimum-size TID memory contains an 8-bit ISO/IEC 15963 allocation class identifier, as well as sufficient identifying information for an Interrogator to uniquely identify the custom commands and/or optional features that a Tag supports. TID memory may optionally contain vendor-specific data. See 6.3.4.1.4.

User memory: User memory is optional. See 6.3.4.1.4, 6.3.4.1.4.1, and 6.3.4.1.4.2.

L.3Optional Tag commands

Proprietary: A Tag may support proprietary commands. See 2.3.3.

Custom: A Tag may support custom commands. See 2.3.4.

Access: A Tag may support the Access command. See 6.3.4.11.3.6.

BlockWrite: A Tag may support the BlockWrite command. See 6.3.4.11.3.7.

BlockErase: A Tag may support the BlockErase command. See 6.3.4.11.3.8.

BlockPermalock: A Tag may support the BlockPermalock command. See 6.3.4.11.3.9.

Kill: A Tag may support the *Kill* command. Only support of Recom bit 3SB is mandatory. See 6.3.4.11.3.4

L.4 Optional Tag error-code reporting format

A Tag may support error-specific or non-error-specific error-code reporting. See Annex I.

L.5 Optional Tag functionality

A Tag may implement the UMI by one of two methods. See 6.3.4.1.2.2.

A Tag may implement a second XPC word (XPC_W2). See 6.3.4.1.2.2 and 6.3.4.1.2.5.

A Tag may implement Recom bits LSB and 2SB. See 6.3.4.4.7, 6.3.4.10, and 6.3.4.11.3.4.

A tag may implement cover-coding. See 6.3.4.11.3.3

L.6 Optional Tag Feature

A Tag may optionally implement a PJM Method. (See Annex G)