**ISO/IEC JTC 1/SC 17 1692**

Date: 2000-03-10

**ISO/IEC FCD 15693-3**

ISO/IEC JTC 1/SC 17/WG 8

Secretariat: DIN

# Identification cards — Contactless integrated circuit(s) cards - Vicinity cards — Part 3: Anti-collision and transmission protocol

*Cartes d'identification — Cartes à circuit(s) intégré(s) sans contact - Cartes de voisinage— Partie 3 : Anti-collision et protocole de transmission*



Document type: International Standard
Document subtype:
Document stage: (40) Enquiry
Document language: E
D:\1Standards\ISO-TF3\15693-3\ISO-IEC FCD15693-3 V4(E).doc STD Version 1.0

## Copyright notice

# Contents

**Foreword**

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 3.

In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this part of ISO/IEC 15693 may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

International Standard ISO/IEC 15693-3 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 17, *Cards and personal identification*.

ISO/IEC 15693 consists of the following parts, under the general title *Identification cards — Contactless integrated circuit(s) cards - Vicinity cards*:

— *Part 1: Physical characteristics*

— *Part 2: Radio frequency power and signal interface*

— *Part 3: Anticollision and transmission protocol*

# Introduction

ISO/IEC 15693 is one of a series of International Standards describing the parameters for identification cards as defined in ISO/IEC 7810 and the use of such cards for international interchange.

This part of ISO/IEC 15693 describes the anticollision and transmission protocols

This International Standard does not preclude the incorporation of other standard technologies on the card.

Contactless card standards cover a variety of types as embodied in ISO/IEC 10536 (Close-coupled cards), ISO/IEC 14443 (Proximity cards), ISO/IEC 15693 (Vicinity cards). These are intended for operation when very near, nearby and at a longer distance from associated coupling devices respectively.

# Identification cards — Contactless integrated circuit(s) cards - Vicinity cards — Part 3: Anti-collision and transmission protocol

## 1 Scope

This part of ISO/IEC 15693 describes:

- protocol and commands,
- other parameters required to initialize communications between a VICC and a VCD,
- methods to detect and communicate with one card among several cards ("anticollision"),
- optional means to ease and speed up the selection of one among several cards based on application criteria.

## 2 Normative references

The following standards contain provisions that, through reference in this text, constitute provisions of this part of ISO/IEC 15693. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this part of ISO/IEC 15693 are encouraged to investigate the possibility of applying the most recent editions of the standards listed below. Members of ISO and IEC maintain registers of currently valid International Standards.

ISO/IEC 15693-1, *Identification cards - Contactless integrated circuit(s) cards – Vicinity cards - Part 1: Physical characteristics*

ISO/IEC 15693-2, *Identification cards - Contactless integrated circuit(s) cards – Vicinity cards - Part 2: Radio frequency power and signal interface*

ISO/IEC 10373-7, *Identification cards - Test methods – Part 7: Vicinity cards*

ISO/IEC 13239, *Information technology – Telecommunications and information exchange between systems – High level data link control (HDLC) procedures*

ISO/IEC 7816-5, *Identification cards – Integrated circuit(s) card with contacts – Part 5: Numbering system and registration procedure for application identifiers*

## 3 Definitions, abbreviations and symbols

### 3.1 Definitions

#### 3.1.1 Anticollision loop

Algorithm used to prepare for and handle a dialogue between a VCD and one or more VICCs from several in its energizing field.

#### 3.1.2 Byte

A byte consists of 8 bits of data designated b1 to b8, from the most significant bit (MSB, b8) to the least significant bit (LSB, b1).

### 3.2 Abbreviations

| | |
|---|---|
| AFI | Application family identifier |
| CRC | Cyclic redundancy check |
| DSFID | Data storage format identifier |
| EOF | End of frame |
| LSB | Least significant bit |
| MSB | Most significant bit |
| RFU | Reserved for future use |
| SOF | Start of frame |
| UID | Unique identifier |
| VCD | Vicinity coupling device |
| VICC | Vicinity integrated circuit card |

### 3.3 Symbols

$f_c$ Frequency of operating field (carrier frequency)

## 4 Definition of data elements

### 4.1 Unique identifier (UID)

The VICCs are uniquely identified by a 64 bits unique identifier (UID). This is used for addressing each VICC uniquely and individually, during the anticollision loop and for one-to-one exchange between a VCD and a VICC.

The UID shall be set permanently by the IC manufacturer in accordance with figure 1.

MSB LSB

| 64 57 | 56 49 | 48 1 |
|---|---|---|
| 'E0' | IC Mfg code | IC manufacturer serial number |

**Figure 1 — UID format**

The UID comprises

- The 8 MSB bits shall be 'E0',

- The IC manufacturer code, on 8 bits according to ISO/IEC 7816-6/AM1,
- A unique serial number on 48 bits assigned by the IC manufacturer.

## 4.2 Application family identifier (AFI)

AFI (Application family identifier) represents the type of application targeted by the VCD and is used to extract from all the VICCs present only the VICCs meeting the required application criteria.

It may be programmed and locked by the respective commands.

AFI is coded on one byte, which constitutes 2 nibbles of 4 bits each.

The most significant nibble of AFI is used to code one specific or all application families, as defined in table 1.

The least significant nibble of AFI is used to code one specific or all application sub-families. Sub-family codes different from 0 are proprietary.

**Table 1 — AFI coding**

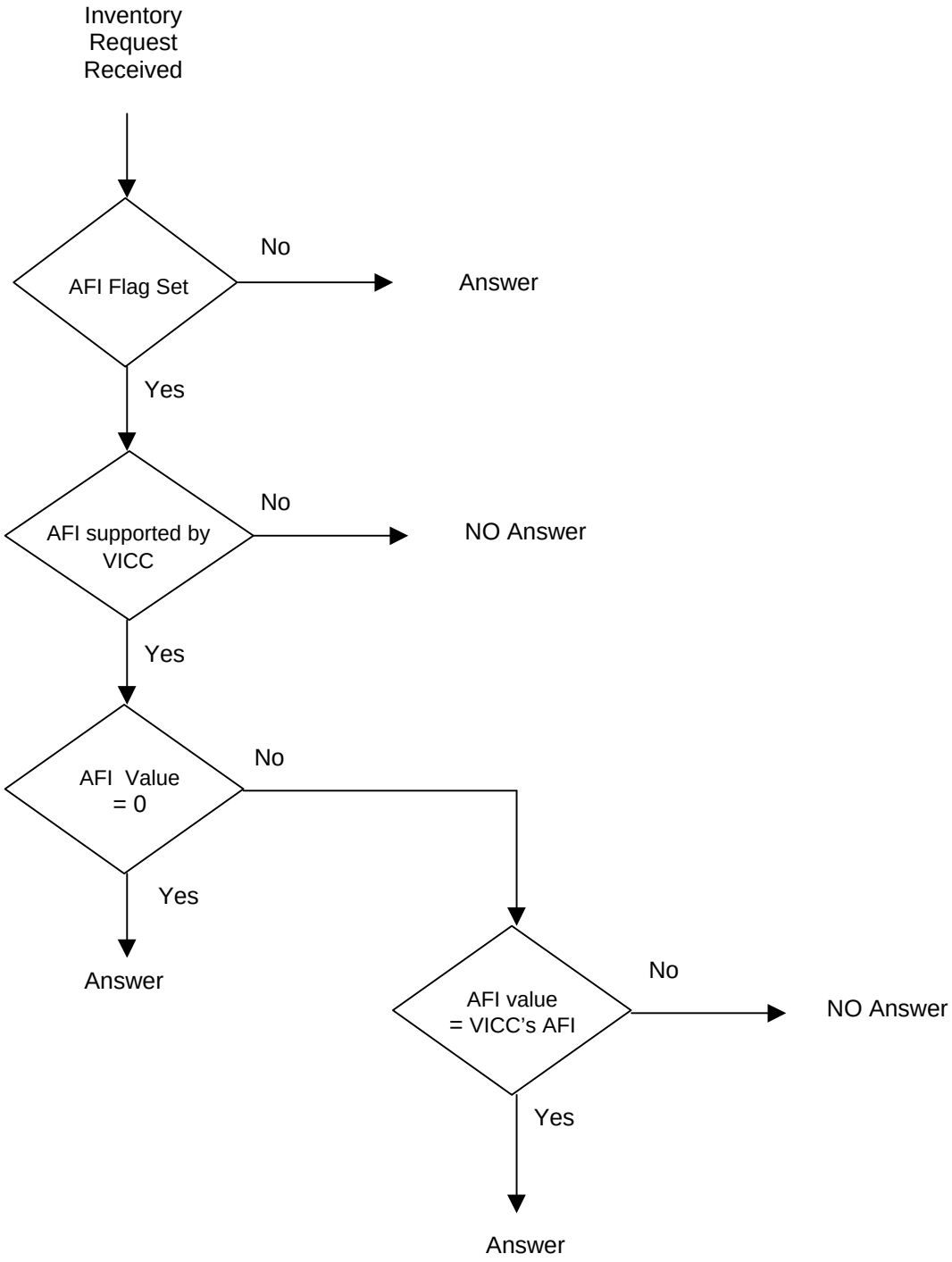
| AFI most significant nibble | AFI least significant nibble | Meaning<br>VICCs respond from | Examples / note |
|---|---|---|---|
| ‘0’ | ‘0’ | All families and sub-families | No applicative preselection |
| X | '0' | All sub-families of family X | Wide applicative preselection |
| X | Y | Only the Yth sub-family of family X | |
| ‘0’ | Y | Proprietary sub-family Y only | |
| ‘1' | ‘0’, Y | Transport | Mass transit, Bus, Airline |
| '2' | ‘0’, Y | Financial | IEP, Banking, Retail |
| '3' | ‘0’, Y | Identification | Access control |
| '4' | ‘0’, Y | Telecommunication | Public telephony, GSM |
| ‘5’ | ‘0’, Y | Medical | |
| '6' | ‘0’, Y | Multimedia | Internet services |
| '7' | ‘0’, Y | Gaming | |
| '8' | ‘0’, Y | Data storage | Portable files |
| '9' | ‘0’, Y | Item management | |
| 'A' | ‘0’, Y | Express parcels | |
| 'B' | ‘0’, Y | Postal services | |
| 'C' | ‘0’, Y | Airline bags | |
| 'D' | ‘0’, Y | | |
| 'E' | ‘0’, Y | | |
| ‘F’ | ‘0’, Y | | |

NOTE X = ‘1’ to ‘F’, Y = ‘1’ to ‘F’

The support of AFI by the VICC is optional.

If AFI is not supported by the VICC and if the AFI flag is set, the VICC shall not answer whatever the AFI value is in the request.

If AFI is supported by the VICC, it shall answer according to the matching rules described in table 1.




NOTE "Answer" means that the VICC shall answer to the Inventory request.

**Figure 2 — VICC decision tree for AFI**

### 4.3 Data storage format identifier (DSFID)

The Data storage format identifier indicates how the data is structured in the VICC memory.

It may be programmed and locked by the respective commands. It is coded on one byte. It allows for instant knowledge on the logical organisation of the data.

If its programming is not supported by the VICC, the VICC shall respond with the value zero ('00').

### 4.4 CRC

The CRC shall be calculated as per the definition in ISO/IEC 13239.

The initial register content shall be all ones: 'FFFF'.

The two bytes CRC are appended to each request and each response, within each frame, before the EOF. The CRC is calculated on all the bytes after the SOF up to but not including the CRC field.

Upon reception of a request from the VCD, the VICC shall verify that the CRC value is valid. If it is invalid, it shall discard the frame and shall not answer (modulate).

Upon reception of a response from the VICC, it is recommended that the VCD verify that the CRC value is valid. If it is invalid, actions to be performed are left to the responsibility of the VCD designer.

The CRC is transmitted least significant byte first.

Each byte is transmitted least significant bit first.

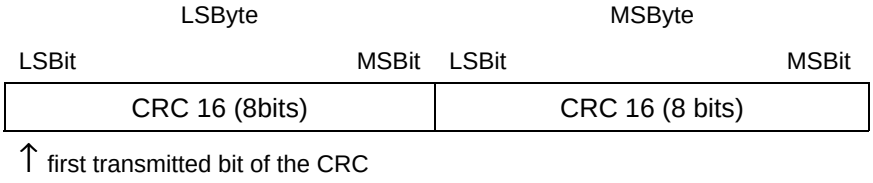



**Figure 3 — CRC bits and bytes transmission rules**

## 5 VICC memory organization

The commands specified in this standard assume that the physical memory is organized in blocks (or pages) of fixed size.

- Up to 256 blocks can be addressed.
- Block size can be of up to 256 bits.
- This leads to a maximum memory capacity of up to 8 kBytes (64 kBits).

Note: The structure allows for future extension of the maximum memory capacity.

The commands described in this standard allow the access (read and write) by block(s). There is no implicit or explicit restriction regarding other access method (e.g. by byte or by logical object in future revision(s) of the standard or in custom commands.

# 6 Block security status

The block security status is coded on one byte. It is an element of the protocol. There is no implicit or explicit assumption that the 8 bits are actually implemented in the physical memory structure of the VICC.

When it is present in a request or a response, the VICC shall interpret (request) or set (response) the supported bits according to the description below:

**Table 2 — Block security status**

| Bit Nb | Flag name | State | Description |
|---|---|---|---|
| Bit 1 | Locked | 0 | Not locked |
| | | 1 | Locked |
| Bit 2 to 8 | RFU | | Shall be set to 0 |

# 7 Overall protocol description

## 7.1 Protocol concept

The transmission protocol (or protocol) defines the mechanism to exchange instructions and data between the VCD and the VICC, in both directions.

It is based on the concept of "VCD talks first".

This means that any VICC does not start transmitting (i.e. modulating according to 15693-2) unless it has received and properly decoded an instruction sent by the VCD.

- The protocol is based on an exchange of
  - a request from the VCD to the VICC
  - a response from the VICC(s) to the VCD
- Each request and each response are contained in a frame. The frame delimiters (SOF, EOF) are specified in ISO/IEC 15693-2.
- Each request consists of the following fields:
  - Flags
  - Command code
  - Mandatory and optional parameters fields, depending on the command
  - Application data fields
  - CRC
- Each response consists of the following fields:
  - Flags

  - Mandatory and optional parameters fields, depending on the command
  - Application data fields
  - CRC
- The protocol is bit-oriented. The number of bits transmitted in a frame is a multiple of eight (8), i.e. an integer number of bytes.
- A single-byte field is transmitted least significant bit (LSBit) first.
- A multiple-byte field is transmitted least significant byte (LSByte) first, each byte is transmitted least significant bit (LSBit) first.
- The setting of the flags indicates the presence of the optional fields. When the flag is set (to one), the field is present. When the flag is reset (to zero), the field is absent.
- RFU flags shall be set to zero (0).

## 7.2 Modes

The term mode refers to the mechanism to specify in a request the set of VICC's that shall answer to the request.

### 7.2.1 Addressed mode

When the Address_flag is set to 1 (addressed mode), the request shall contain the unique ID (UID) of the addressed VICC.

Any VICC receiving a request with the Address_flag set to 1 shall compare the received unique ID (address) to its own ID.

If it matches, it shall execute it (if possible) and return a response to the VCD as specified by the command description.

If it does not match, it shall remain silent.

### 7.2.2 Non-addressed mode

When the Address_flag is set to 0 (non-addressed mode), the request shall not contain a unique ID.

Any VICC receiving a request with the Address_flag set to 0 shall execute it (if possible) and shall return a response to the VCD as specified by the command description.

### 7.2.3 Select mode

When the Select_flag is set to 1 (select mode), the request shall not contain a VICC unique ID.

The VICC in the selected state receiving a request with the Select_flag set to 1 shall execute it (if possible) and shall return a response to the VCD as specified by the command description.

Only the VICC in the selected state shall answer to a request having the select flag set to 1.

## 7.3 Request format

The request consists of the following fields:

- Flags
- Command code (see clause 9)
- Parameters and data fields
- CRC (see clause 4.4)

| SOF | Flags | Command code | Parameters | Data | CRC | EOF |
|---|---|---|---|---|---|---|

**Figure 4 — General request format**

### 7.3.1 Request flags

In a request, the field "flags" specifies the actions to be performed by the VICC and whether corresponding fields are present or not.

It consists of eight bits.

**Table 3 — Request flags 1 to 4 definition**

| Bit Nb | Flag name | State | Description |
|---|---|---|---|
| Bit 1 | Sub-carrier_flag | 0 | A single sub-carrier frequency shall be used by the VICC |
| | | 1 | Two sub-carriers shall be used by the VICC |
| Bit 2 | Data_rate_flag | 0 | Low data rate shall be used |
| | | 1 | High data rate shall be used |
| Bit 3 | Inventory_flag | 0 | Flags 5 to 8 meaning is according to table 4 |
| | | 1 | Flags 5 to 8 meaning is according to table 5 |
| Bit 4 | Protocol Extension_flag | 0 | No protocol format extension |
| | | 1 | Protocol format is extended. Reserved for future use |

Note: 1. Sub-carrier_flag refers to the VICC-to-VCD communication as specified in ISO/IEC 15693-2.

2. Data_rate_flag refers to the VICC-to-VCD communication as specified in ISO/IEC 15693-2.

**Table 4 — Request flags 5 to 8 definition when inventory flag is NOT set**

| Bit Nb | Flag name | State | Description |
|---|---|---|---|
| Bit 5 | Select_flag | 0 | Request shall be executed by any VICC according to the setting of Address_flag |
| | | 1 | Request shall be executed only by VICC in selected state |
| Bit 6 | Address_flag | 0 | Request is not addressed. UID field is not present. It shall be executed by any VICC. |
| | | 1 | Request is addressed. UID field is present. It shall be executed only by the VICC whose UID matches the UID specified in the request. |
| Bit 7 | Option_flag | 0 | Meaning is defined by the command description. It shall be set to 0 if not otherwise defined by the command. |
| | | 1 | Meaning is defined by the command description. |
| Bit 8 | RFU | 0 | Shall be set to 0. |

Note: if the Select_flag is set to 1, the Address_flag shall be set to 0 and the UID field shall not be present in the request.

**Table 5 — Request flags 5 to 8 definition when inventory flag is set**

| Bit Nb | Flag name | State | Description |
|---|---|---|---|
| Bit 5 | AFI_flag | 0 | AFI field is not present |
| | | 1 | AFI field is present |
| Bit 6 | Nb_slots_flag | 0 | 16 slots |
| | | 1 | 1 slot |
| Bit 7 | Option_flag | 0 | Meaning is defined by the command description. It shall be set to 0 if not otherwise defined by the command. |
| | | 1 | Meaning is defined by the command description. |
| Bit 8 | RFU | 0 | Shall be set to 0. |

## 7.4 Response format

The response consists of the following fields:

- Flags
- one or more parameter fields
- Data
- CRC (see clause 4.4)

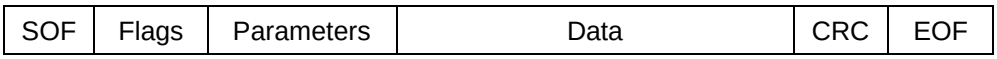



**Figure 5 — General response format**

### 7.4.1 Response flags

In a response, it indicates how actions have been performed by the VICC and whether corresponding fields are present or not.

It consists of eight bits.

**Table 6 — Response flags 1 to 8 definition**

| Bit Nb | Flag name | State | Description |
|---|---|---|---|
| Bit 1 | Error_flag | 0 | No error |
| | | 1 | Error detected. Error code is in the "Error" field. |
| Bit 2 | RFU | | Shall be set to 0. |
| Bit 3 | RFU | | Shall be set to 0. |
| Bit 4 | Extension_flag | 0 | No protocol format extension. |
| | | 1 | Protocol format is extended. Reserved for future use. |
| Bit 5 | RFU | | Shall be set to 0. |
| Bit 6 | RFU | | Shall be set to 0. |
| Bit 7 | RFU | | Shall be set to 0. |
| Bit 8 | RFU | | Shall be set to 0. |

### 7.4.2 Response error code

If the Error_flag is set by the VICC in the response, the error code field is present and provides information about the error that occurred.

The following error codes are specified. Other codes are reserved for future use.

**Table 7 — Response error code definition**

| Error code | Meaning |
|---|---|
| '01' | The command is not supported, i.e. the request code is not recognised. |
| '02' | The command is not recognised, for example: a format error occurred. |
| '03' | The option is not supported. |
| '0F' | Unknown error. |
| '10' | The specified block is not available (doesn't exist). |
| '11' | The specified block is already -locked and thus cannot be locked again |
| '12' | The specified block is locked and its content cannot be changed. |
| '13' | The specified block was not successfully programmed. |
| '14' | The specified block was not successfully locked. |
| 'A0' – 'DF' | Custom command error codes |
| all others | RFU |

Note: If the VICC does not support error codes listed in table 7, it shall answer with the error code '0F' (unknown error).

## 7.5 VICC states

A VICC can be in 4 states:

- Power-off
- Ready
- Quiet
- Selected

The transition between these states is specified in figure 6.

The support of power-off, ready and quiet states is mandatory.

The support of selected state is optional.

### 7.5.1 Power-off state

The VICC is in the power-off state when it cannot be activated by the VCD.

### 7.5.2 Ready state

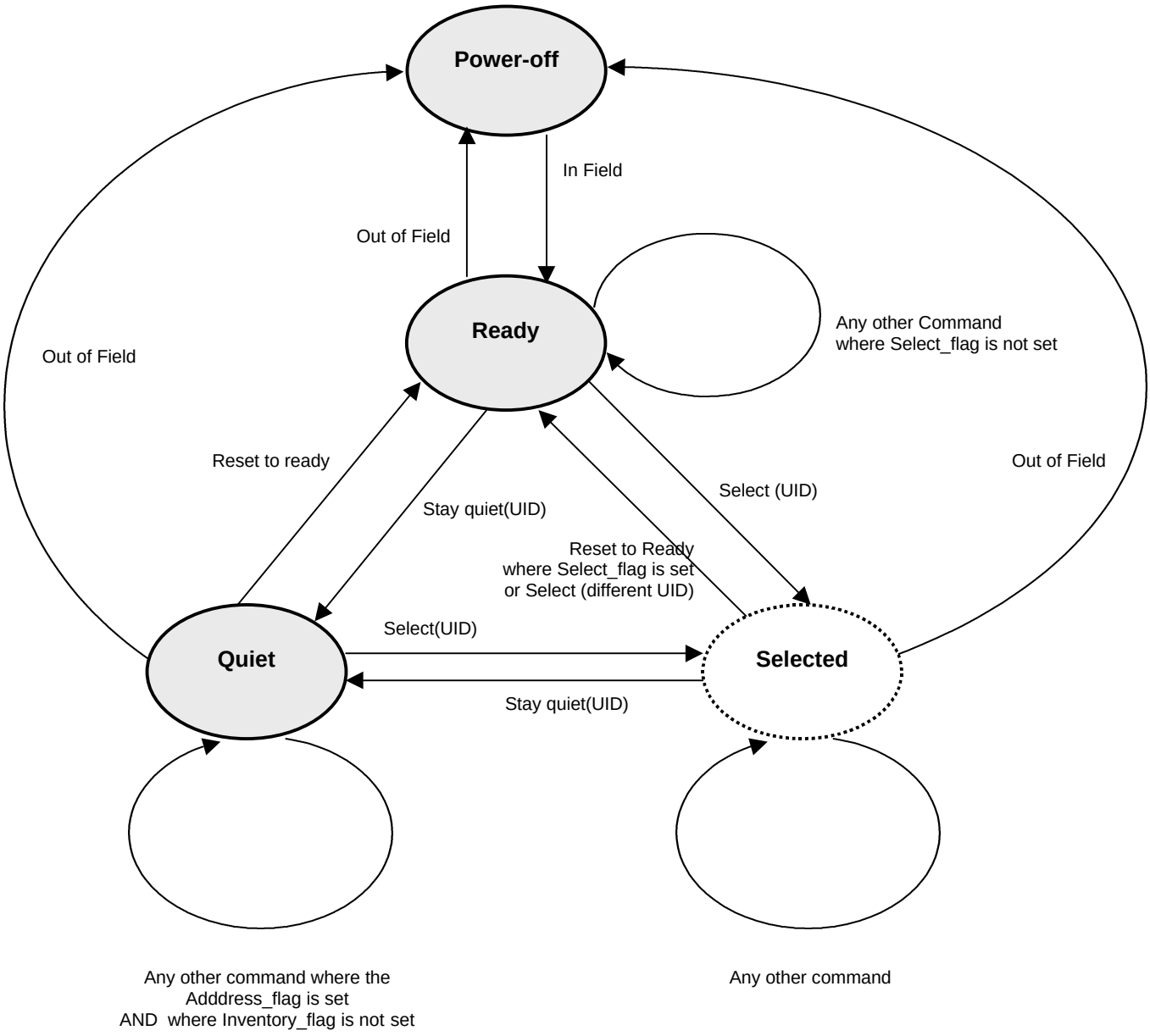
The VICC is in the Ready state when it is activated by the VCD. It shall process any request where the select flag is not set.

### 7.5.3 Quiet state

When in the quiet state, the VICC shall process any request where the Inventory_flag is not set and where the Adddress_flag_flag is set.

### 7.5.4 Selected state

Only a VICC in the selected state shall process requests having the Select_flag set.




NOTE 1 The intention of the state transition method is that only one VICC should be in the selected state at a time.

NOTE 2 The VICC state transition diagram shows only valid transitions. In all other cases the current VICC state remains unchanged. When the VICC cannot process a VCD request (e.g. CRC error etc... ), it shall stay in its current state.

NOTE 3 The Selected state is represented with a dotted line to show its support by the VICC is optional.

**Figure 6 — VICC state transition diagram**

# 8 Anticollision

The purpose of the anticollision sequence is to make an inventory of the VICCs present in the VCD field by their unique ID (UID).

The VCD is the master of the communication with one or multiple VICCs. It initiates card communication by issuing the inventory request.

The VICC shall send its response in the slot determined or shall not respond, according to the algorithm described in clause 8.2.

## 8.1 Request parameters

When issuing the inventory command, the VCD shall:

- set the Nb_slots_flag to the desired setting,
- add after the command field the mask length and the mask value.
- The mask length is the number of significant bits of the mask value.
- The mask value is contained in an integer number of bytes. The mask length indicates the number of significant bits. LSB shall be transmitted first.
- If the mask length is not a multiple of 8 (bits), the mask value MSB shall be padded with the required number of null (set to 0) bits so that the mask value is contained in an integer number of bytes.
- The next field starts on the next byte boundary.

| SOF | Flags | Command | Mask length | Mask value | EOF |
|---|---|---|---|---|---|
| | 8 bits | 8 bits | 8 bits | 0 to 8 bytes | |

**Figure 7 — Inventory request format**

| MSB | LSB |
|---|---|
| 0000 | 0100 1100 1111 |
| Pad | Value |

**Figure 8 — Example of the padding of the mask**

In the example of the figure 8, the mask length is 12 bits. The mask value MSB is padded with four bits set to 0

The AFI field shall be present if the AFI_flag is set.

The pulse shall be generated according to the definition of the EOF in ISO/IEC 15693-2.

The first slot starts immediately after the reception of the request EOF.

To switch to the next slot, the VCD sends an EOF.

The following rules and restrictions apply:

If no VICC answer is detected, the VCD may switch to the next slot by sending an EOF. If one or more VICC answers are received, the VCD shall wait until the VICC frames are completely received before sending an EOF for switching to the next slot.

This is illustrated in figures 8, 9 10 and 11. The timing is described in clause 8.4 and table 8.

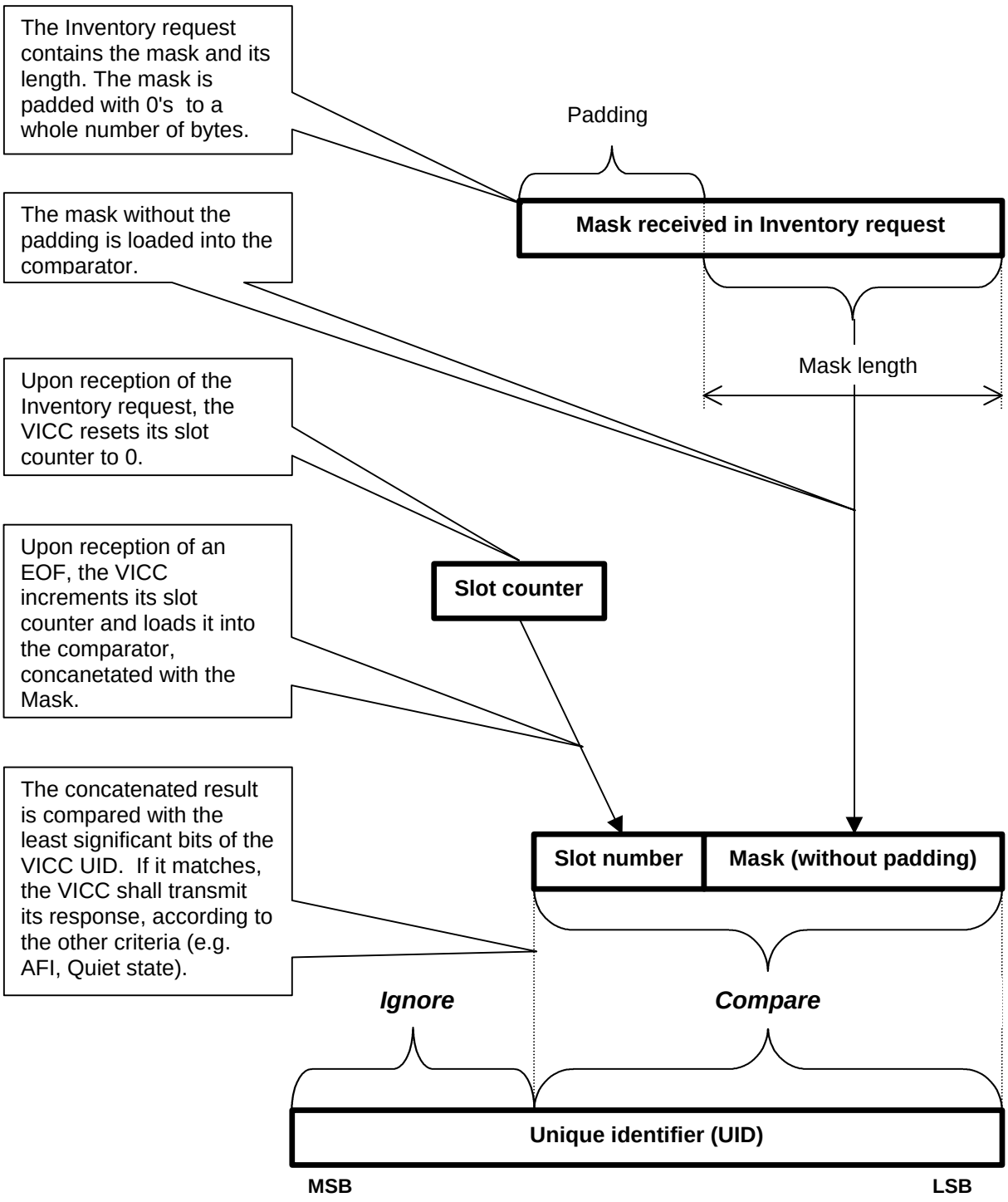
## 8.2 Request processing by the VICC

Upon reception of a valid request, the VICC shall perform the algorithm described in figure 9 or its functional equivalent (implementation dependent).

```
NbS is the total number of slots (1 or 16)

SN is the current slot number (0 to 15)

LSB (value, n) function returns the n less significant bits of value

"&" is the concatenation operator

Slot_Frame is either a SOF or an EOF

    SN= 0

    if Nb_slots_flag    then NbS =1  SN_length=0

                                 else      NbS = 16      SN_length=4

    endif

label1:  if LSB(UID, SN_length + Mask_length) = LSB(SN, SN_length)&LSB(Mask, Mask_length) then

          transmit response to inventory request
```

```
endif
wait (Slot_Frame)
if Slot_Frame= SOF then
    Stop anticollision and decode/process request
    exit
endif
if SN<NbS-1 then
    SN = SN +1
    goto label1
    exit
endif
exit
```

**Figure 9 — Example of the inventory algorithm to be executed by the VICC**




NOTE When the slot number is 1 (Nb_slots_flag is set to 1), the comparison is made only on the mask (without padding).

**Figure 10 — Principle of comparison between the mask, slot number and UID**

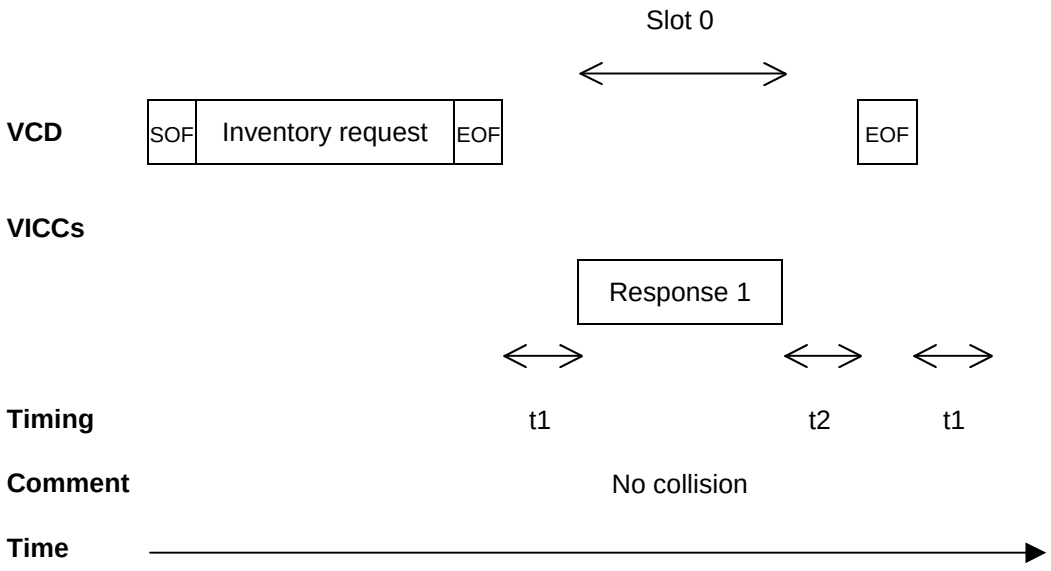
### 8.3 Explanation of an anticollision sequence

Figure 11 summarises the main cases that can occur during a typical anticollision sequence where the number of slots is 16.

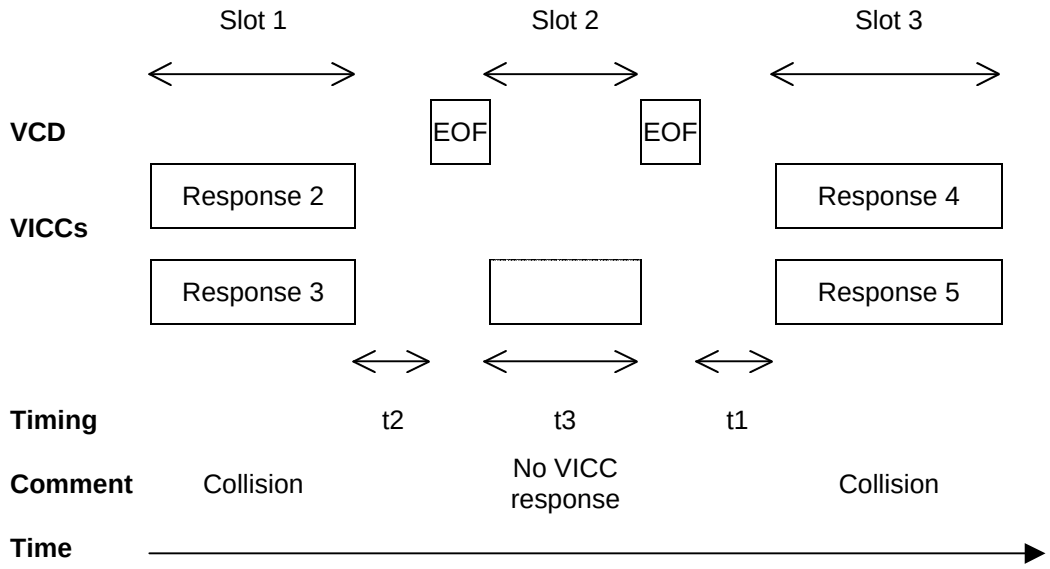
The different steps are:

- The VCD sends an inventory request, in a frame, terminated by a EOF. The number of slots is 16.
- VICC 1 transmits its response in slot 0. It is the only one to do so, therefore no collision occurs and its UID is received and registered by the VCD;
- The VCD sends an EOF, meaning to switch to the next slot.
- In slot 1, two VICCs 2 and 3 transmits their response, this generates a collision. The VCD detects it and remembers that a collision was detected in slot 1.
- The VCD sends an EOF, meaning to switch to the next slot.
- In slot 2, no VICC transmits a response. Therefore the VCD does not detect a VICC SOF and decides to switch to the next slot by sending a EOF.
- In slot 3, there is another collision caused by responses from VICC 4 and 5
- The VCD then decides to send a request (for instance a Read Block) to VICC 1, which UID was already correctly received.
- All VICCs detect a SOF and exit the anticollision sequence. They process this request and since the request is addressed to VICC 1, only VICC1 transmit its response.
- All VICCs are ready to receive another request. If it is an inventory command, the slot numbering sequence restarts from 0.

NOTE The decision to interrupt the anticollision sequence is up to the VCD. It could have continued to send EOF's till slot 15 and then send the request to VICC 1.



Continued…




Continued…

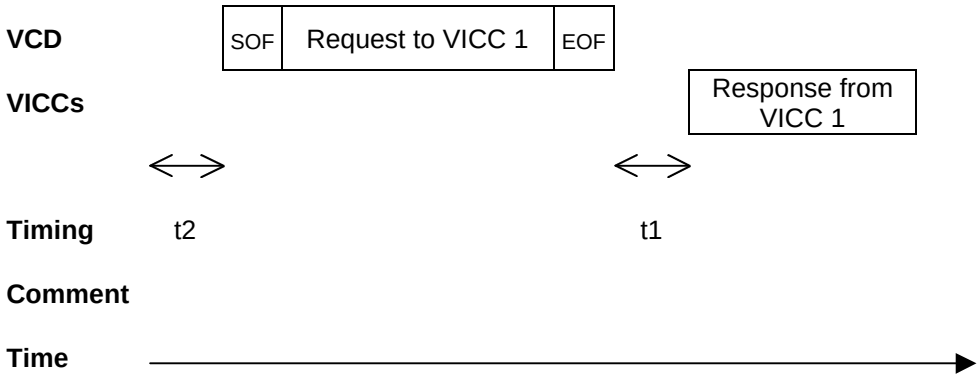



**Figure 11 — Description of a possible anticollision sequence**

## 8.4 Timing definition

- t1 is as defined in table 8.

  Upon detection of the rising edge of the EOF, the VICC shall wait for a time equal to

  $128/f_c$ (9,44µs) + t1

  before starting to transmit its response to a VCD request or switch to the next slot when in an inventory process.

  The EOF is defined in Part 2, clause 7.3.3.

If the VICC detects a carrier modulation during this time, it shall reset its t1 timer and wait for a further time equal to the sum of

128/ $f_c$ (9,44µs) + t1

before starting to transmit its response to a VCD request or to switch to the next slot when in an inventory process.

NOTE the synchronisation on the rising edge of the EOF pulse is needed for ensuring the required synchronisation of the VICC responses.

- t2 is the time after which the VCD may send an EOF to switch to the next slot when one or more VICC responses have been received. It starts from the reception of the EOF received from the VICCs.

  NOTE Such EOF sent by the VCD may be either 10% or 100% modulated independent of the modulation index used for transmitting the VCD request to the VICC.

- t3 is the time after which the VCD may send an EOF to switch to the next slot when no VICC response has been received.

  NOTE Such EOF sent by the VCD may be either 10% or 100% modulated whatever the modulation index used for transmitting the VCD request to the VICC was.

From the time the VCD has generated the rising edge of an EOF,

If this EOF is 100% modulated, the VCD shall wait a time at least equal to the sum of

128/$f_c$ (9,44µs) + the minimum value of t3

before sending a subsequent EOF.

If this EOF is 10% modulated, the VCD shall wait a time at least equal to the sum of

128/$f_c$ (9,44µs) + the minimum value of t3 + the nominal response time of a VICC, which is depending from the VICC datarate and subcarrier modulation mode.

before sending a subsequent EOF.

**Table 8 — Timing values for Read alike requests**

| | minimum | nominal | maximum |
|---|---|---|---|
| t1 | 4192/$f_c$ (309,2us) | 4224/$f_c$ (311,5us) | 4256/$f_c$ (313,9us) |
| t2 | 4192/$f_c$ (309,2us) | - | - |
| t3 | t1maximum + tsof (see notes 1 and 2) | - | - |

NOTE 1 tsof is the duration for a VICC to transmit an SOF to the VCD. tsof is dependant on the current data rate, as per ISO/IEC 15693-2.

NOTE 2 t1max does not apply for Write alike requests. Timing conditions for Write alike requests are defined in the command descriptions.

# 9 Commands

Four sets of commands are defined:

- **Mandatory** '01' to '1F' all VICCs shall support them
- **Optional** '20' to '9F' VICCs may support them, at their option. If supported, request and response formats shall comply with the definition given in this standard.

  If the VICC does not support an optional command and if the Adddress_flag or the Select_flag is set, it may return an error code ("Not supported") or remain silent. If neither the Adddress_flag nor the Select_flag is set, the VICC shall remain silent.

  If a command has different options they may be supported by the VICC otherwise an error code shall be returned.

- **Custom** 'A0' to 'DF' VICCs support them, at their option, to implement manufacturer specific functions. The function of flags (including reserved bits) shall not be modified except the custom flag. The only fields that can be customized are the parameters and the data fields.

  Any custom command contains as its first parameter the IC manufacturer code. This allows IC manufacturers to implement custom commands without risking duplication of command codes and thus misinterpretation.

  If the VICC does not support a custom command, it may return an error code ("Not supported") or remain silent.

  If a command has different options they may be supported by the VICC otherwise an error code shall be returned.

- **Proprietary** 'E0' to 'FF' these commands are used by IC and VICC manufacturers for various purposes such as tests, programming of system information, etc... They are not specified in this standard. The IC manufacturer may at its option document them or not. It is allowed that these commands are disabled after IC and/or VICC manufacturing.

  All VICCs with the same IC manufacturer code and same IC version number shall behave the same.

## 9.1 Command codes

**Table 9 — Command codes**

| Command code | Type | Function |
|---|---|---|
| '01' | Mandatory | Inventory |
| '02' | Mandatory | Stay quiet |
| '03' – '1F' | Mandatory | RFU |
| '20' | Optional | Read single block |
| '21' | Optional | Write single block |
| '22' | Optional | Lock block |
| '23' | Optional | Read multiple blocks |
| '24' | Optional | Write multiple blocks |
| '25' | Optional | Select |
| '26' | Optional | Reset to ready |

| | | |
|---|---|---|
| '27' | Optional | Write AFI |
| '28' | Optional | Lock AFI |
| '29' | Optional | Write DSFID |
| '2A' | Optional | Lock DSFID |
| '2B' | Optional | Get system information |
| '2C' | Optional | Get multiple block security status |
| '2D' – '9F' | Optional | RFU |
| 'A0' – 'DF' | Custom | IC Mfg dependent |
| 'E0' – 'FF' | Proprietary | IC Mfg dependent |

## 9.2 Mandatory commands

### 9.2.1 Inventory

**Command code = '01'**

When receiving the Inventory request, the VICC shall perform the anticollision sequence.

The request contains:

- The flags,
- The Inventory command code
- The AFI if the AFI flag is set
- The mask length
- The mask value
- The CRC

The Inventory_flag shall be set to 1.

The meaning of flags 5 to 8 is according to table 5.

| SOF | Flags | Inventory | Optional AFI | Mask length | Mask value | CRC16 | EOF |
|---|---|---|---|---|---|---|---|
| | 8 bits | 8 bits | 8 bits | 8 bits | 0 – 64 bits | 16 bits | |

**Figure 12 — Inventory request format**

The response shall contain:

- the DSFID
- the unique ID

| SOF | Flags | DSFID | UID | CRC16 | EOF |
|---|---|---|---|---|---|
| | 8 bits | 8 bits | 64 bits | 16 bits | |

**Figure 13 — Inventory response format**

### 9.2.2 Stay quiet

**Command code = '02'**

When receiving the Stay quiet command, the VICC shall enter the quiet state and shall NOT send back a response. There is NO response to the Stay quiet command.

When in quiet state:

- the VICC shall not process any request where Inventory_flag is set,
- the VICC shall process any addressed request

The VICC shall exit the quiet state when:

- reset (power off),
- receiving a Select request. It shall then go to the selected state if supported or return an error if not supported,
- receiving a Reset to ready request. It shall then go to the Ready state.

| SOF | Flags | Stay quiet | UID | CRC16 | EOF |
|---|---|---|---|---|---|
| | 8 bits | 8 bits | 64 bits | 16 bits | |

**Figure 14 — Stay quiet request format**

**Request parameter:**

UID (mandatory)

The Stay quiet command shall always be executed in Addressed mode (Select_flag is set to 0 and Adddress_flag is set to 1).

## 9.3 Optional commands

### 9.3.1 Read single block

**Command code = '20'**

When receiving the Read single block command, the VICC shall read the requested block and send back its value in the response.

If the Option_flag is set in the request, the VICC shall return the block security status, followed by the block value.

If it is not set, the VICC shall return only the block value.

| SOF | Flags | Read single block | UID | Block number | CRC16 | EOF |
|---|---|---|---|---|---|---|
| | 8 bits | 8 bits | 64 bits | 8 bits | 16 bits | |

**Figure 15 — Read single block request format**

**Request parameter:**

(Optional) UID

Block number

| SOF | Flags | Error code | CRC16 | EOF |
|---|---|---|---|---|
| | 8 bits | 8 bits | 16 bits | |

**Figure 16 — Read single block response format when Error_flag is set**

| SOF | Flags | Block security status | Data | CRC16 | EOF |
|---|---|---|---|---|---|
| | 8 bits | 8 bits | Block length | 16 bits | |

**Figure 17 — Read single block response format when Error_flag is NOT set**

**Response parameter:**

Error_flag (and code if Error_flag is set)if Error_flag is not set

Block security status (if Option_flag is set in the request)

Block data

### 9.3.2 Write single block

**Command code = '21'**

When receiving the Write single block command, the VICC shall write the requested block with the data contained in the request and report the success of the operation in the response.

If the Option_flag is not set, the VICC shall return its response when it has completed the write operation starting after a multiple of t1 nominal (see table 8) with a total tolerance of +-32/$f_c$ and latest after 20ms.

If it is set, the VICC shall wait for the reception of an 100% modulated EOF from the VCD and upon such reception shall return its response.

| SOF | Flags | Write single block | UID | Block number | Data | CRC16 | EOF |
|---|---|---|---|---|---|---|---|
| | 8 bits | 8 bits | 64 bits | 8 bits | Block length | 16bits | |

**Figure 18 — Write single block request format**

**Request parameter:**

(Optional) UID

Block number

Data

| SOF | Flags | Error code | CRC16 | EOF |
|---|---|---|---|---|
| | 8 bits | 8 bits | 16 bits | |

**Figure 19 — Write single block response format when Error_flag is set**

| SOF | Flags | CRC16 | EOF |
|---|---|---|---|
| | 8 bits | 16 bits | |

**Figure 20 — Write single block response format when Error_flag is NOT set**

**Response parameter:**

Error_flag (and code if Error_flag is set)

### 9.3.3 Lock block

**Command code = '22'**

When receiving the Lock block command, the VICC shall lock permanently the requested block.

If the Option_flag is not set, the VICC shall return its response when it has completed the lock operation starting after a multiple of t1 nominal ( see table 8 ) with a total tolerance of +-32/$f_c$ and latest after 20ms.

If it is set, the VICC shall wait for the reception of an EOF from the VCD and upon such reception shall return its response.

| SOF | Flags | Lock block | UID | Block number | CRC16 | EOF |
|---|---|---|---|---|---|---|
| | 8 bits | 8 bits | 64 bits | 8 bits | 16 bits | |

**Figure 21 — Lock single block request format**

**Request parameter:**

(Optional) UID

Block number

| SOF | Flags | Error code | CRC16 | EOF |
|---|---|---|---|---|
| | 8 bits | 8 bits | 16 bits | |

**Figure 22 — Lock block response format when Error_flag is set**

| SOF | Flags | CRC16 | EOF |
|---|---|---|---|
| | 8 bits | 16 bits | |

**Figure 23 — Lock block response format when Error_flag is NOT set**

**Response parameter:**

Error_flag (and code if Error_flag is set)

### 9.3.4 Read multiple blocks

**Command code = '23'**

When receiving the Read multiple block command, the VICC shall read the requested block(s) and send back their value in the response.

If the Option_flag is set in the request, the VICC shall return the block security status, followed by the block value sequentially block by block.

If the Option_flag is not set in the request, the VICC shall return only the block value.

The blocks are numbered from '00' to 'FF' (0 to 255).

The number of blocks in the request is one less than the number of blocks that the VICC shall return in its response.

E.g. a value of '06' in the "Number of blocks" field requests to read 7 blocks. A value of '00' requests to read a single block.

| SOF | Flags | Read multiple block | UID | First block number | Number of blocks | CRC16 | EOF |
|---|---|---|---|---|---|---|---|
| | 8 bits | 8 bits | 64 bits | 8 bits | 8 bits | 16 bits | |

**Figure 24 — Read multiple blocks request format**

**Request parameter:**

(Optional) UID

First block number

Number of blocks

| SOF | Flags | Error code | CRC16 | EOF |
|---|---|---|---|---|
| | 8 bits | 8 bits | 16 bits | |

**Figure 25 — Read multiple blocks response format when Error_flag is set**

| SOF | Flags | Block security status | Data | CRC16 | EOF |
|---|---|---|---|---|---|
| | 8 bits | 8 bits | Block length | 16 bits | |
| | | Repeated as needed | | | |

**Figure 26 — Read multiple block response format when Error_flag is NOT set**

**Response parameter:**

Error_flag (and code if Error_flag is set)

if Error_flag is not set (the following order shall be respected in the VICC response)

| |
|---|
| Block security status N (if Option_flag is set in the request) |
| Block value N |
| Second Block security status N+1 (if Option_flag is set in the request) |
| Block value N+1 |
| etc… |
| where N is the first requested (and returned) block. |

### 9.3.5 Write multiple blocks

**Command code = '24'**

When receiving the Write multiple single block command, the VICC shall write the requested block(s) with the data contained in the request and report the success of the operation in the response.

If the Option_flag is not set, the VICC shall return its response when it has completed the write operation starting after a multiple of t1 nominal ( see table 8 ) with a total tolerance of +-32/$f_c$ and latest after 20ms.

If it is set, the VICC shall wait for the reception of an EOF from the VCD and upon such reception shall return its response.

The blocks are numbered from '00' to 'FF' (0 to 255).

The number of blocks in the request is one less than the number of blocks that the VICC shall write.

E.g. a value of '06' in the "Number of blocks" field requests to write 7 blocks. The "Data" field shall contain 7 blocks. A value of '00' in the "Number of blocks" field requests to write 1 block. The "Data" field shall contain 1 block.

| SOF | Flags | Write multiple block | UID | First block number | Number of blocks | Data | CRC16 | EOF |
|---|---|---|---|---|---|---|---|---|
| | 8 bits | 8 bits | 64 bits | 8 bits | 8 bits | Block length | 16 bits | |
| | | | | | | Repeated as needed | | |

**Figure 27 — Write multiple blocks request format**

**Request parameter:**

(Optional) UID

First block number

Number of blocks

Block data (repeated as defined in figure 27)

| SOF | Flags | Error code | CRC16 | EOF |
|---|---|---|---|---|
| | 8 bits | 8 bits | 16 bits | |

**Figure 28 — Write multiple blocks response format when Error_flag is set**

| SOF | Flags | CRC16 | EOF |
|---|---|---|---|
| | 8 bits | 16 bits | |

**Figure 29 — Write multiple block response format when Error_flag is NOT set**

**Response parameter:**

Error_flag (and code if Error_flag is set)

### 9.3.6 Select

**Command code = '25'**

When receiving the Select command:

- if the UID is equal to its own UID, the VICC shall enter the selected state and shall send a response.
- if it is different, the VICC shall return to the Ready state and shall not send a response.The Select command shall always be executed in Addressed mode. (The Select_flag is set to 0. The Address_flag is set to 1.)

| SOF | Flags | Select | UID | CRC16 | EOF |
|---|---|---|---|---|---|
| | 8 bits | 8 bits | 64 bits | 16 bits | |

**Figure 30 — Select request format**

**Request parameter:**

UID (mandatory)

| SOF | Flags | Error Code | CRC16 | EOF |
|---|---|---|---|---|
| | 8 bits | 8 bits | 16 bits | |

**Figure 31 — Select response format when Error_flag is set**

| SOF | Flags | CRC16 | EOF |
|---|---|---|---|
| | 8 bits | 16 bits | |

**Figure 32 — Select block response format when Error_flag is NOT set**

**Response parameter:**

Error_flag (and Error code if the Error_flag is set)

### 9.3.7 Reset to ready

**Command code = '26'**

When receiving a Reset to ready command, the VICC shall return to the Ready state.

| SOF | Flags | Reset to ready | UID | CRC16 | EOF |
|---|---|---|---|---|---|
| | 8 bits | 8 bits | 64 bits | 16 bits | |

**Figure 33 — Reset to ready request format**

**Request parameter:**

(Optional) UID

| SOF | Flags | Error code | CRC16 | EOF |
|---|---|---|---|---|
| | 8 bits | 8 bits | 16 bits | |

**Figure 34 — Reset to ready response format when Error_flag is set**

| SOF | Flags | CRC16 | EOF |
|---|---|---|---|
| | 8 bits | 16 bits | |

**Figure 35 — Reset to ready response format when Error_flag is NOT set**

**Response parameter:**

Error_flag (and Error code if the Error_flag is set)

### 9.3.8 Write AFI

**Command code = '27'**

When receiving the Write AFI request, the VICC shall write the AFI value into its memory.

If the Option_flag is not set, the VICC shall return its response when it has completed the write operation starting after a multiple of t1 nominal (see table 8) with a total tolerance of +-32/$f_c$ and latest after 20ms.

If it is set, the VICC shall wait for the reception of an EOF from the VCD and upon such reception shall return its response.

| SOF | Flags | Write AFI | UID | AFI | CRC16 | EOF |
|---|---|---|---|---|---|---|
| | 8 bits | 8 bits | 64 bits | 8 bits | 16 bits | |

**Figure 36: Write AFI request format**

**Request parameter:**

(Optional) UID

AFI

| SOF | Flags | Error code | CRC16 | EOF |
|---|---|---|---|---|
| | 8 bits | 8 bits | 16 bits | |

**Figure 37 — Write AFI response format when Error_flag is set**

| SOF | Flags | CRC16 | EOF |
|---|---|---|---|
| | 8 bits | 16 bits | |

**Figure 38 — Write AFI response format when Error_flag is NOT set**

### 9.3.9 Lock AFI

**Command code = '28'**

When receiving the Lock AFI request, the VICC shall lock the AFI value permanently into its memory.

If the Option_flag is not set, the VICC shall return its response when it has completed the lock operation starting after a multiple of t1 nominal (see table 8) with a total tolerance of +-32/$f_c$ and latest after 20ms.

If it is set, the VICC shall wait for the reception of an EOF from the VCD and upon such reception shall return its response.

| SOF | Flags | Lock AFI | UID | CRC16 | EOF |
|---|---|---|---|---|---|
| | 8 bits | 8 bits | 64 bits | 16 bits | |

**Figure 39 — Lock AFI request format**

| SOF | Flags | Error Code | CRC16 | EOF |
|---|---|---|---|---|
| | 8 bits | 8 bits | 16 bits | |

**Figure 40 — Lock AFI response format when Error_flag is set**

| SOF | Flags | CRC16 | EOF |
|---|---|---|---|
| | 8 bits | 16 bits | |

**Figure 41 — Lock AFI response format when Error_flag is NOT set**

### 9.3.10 Write DSFID command

**Command code = '29'**

When receiving the Write DSFID request, the VICC shall write the DSFID value into its memory.

If the Option_flag is not set, the VICC shall return its response when it has completed the write operation starting after a multiple of t1 nominal ( see table 8 ) with a total tolerance of +-32/$f_c$ and latest after 20ms.

If it is set, the VICC shall wait for the reception of an EOF from the VCD and upon such reception shall return its response.

| SOF | Flags | Write DSFID | UID | DSFID | CRC16 | EOF |
|---|---|---|---|---|---|---|
| | 8 bits | 8 bits | 64 bits | 8 bits | 16 bits | |

**Figure 42 — Write DSFID request format**

**Request parameter:**

(Optional) UID

DSFID

| SOF | Flags | Error code | CRC16 | EOF |
|---|---|---|---|---|
| | 8 bits | 8 bits | 16 bits | |

**Figure 43 — Write DSFID response format when Error_flag is set**

| SOF | Flags | CRC16 | EOF |
|---|---|---|---|
| | 8 bits | 16 bits | |

**Figure 44 — Write DSFID response format when Error_flag is NOT set**

### 9.3.11 Lock DSFID

**Command code = '2A'**

When receiving the Lock DSFID request, the VICC shall lock the DSFID value permanently into its memory.

If the Option_flag is not set, the VICC shall return its response when it has completed the lock operation starting after a multiple of t1 nominal (see table 8) with a total tolerance of +-32/$f_c$ and latest after 20ms.

If it is set, the VICC shall wait for the reception of an EOF from the VCD and upon such reception shall return its response.

| SOF | Flags | Lock DSFID | UID | CRC16 | EOF |
|---|---|---|---|---|---|
| | 8 bits | 8 bits | 64 bits | 16 bits | |

**Figure 45 — Lock DSFID request format**

| SOF | Flags | Error code | CRC16 | EOF |
|---|---|---|---|---|
| | 8 bits | 8 bits | 16 bits | |

**Figure 46 — Lock DSFID response format when Error_flag is set**

| SOF | Flags | CRC16 | EOF |
|---|---|---|---|
| | 8 bits | 16 bits | |

**Figure 47 — Lock DSFID response format when Error_flag is NOT set**

### 9.3.12 Get system information

**Command code = '2B'**

This command allows for retrieving the system information value from the VICC.

| SOF | Flags | Get system info | UID | CRC16 | EOF |
|---|---|---|---|---|---|
| | 8 bits | 8 bits | 64 bits | 16 bits | |

**Figure 48 — Get system information request format**

**Request parameter:**

(Optional) UID

| SOF | Flags | Error code | CRC16 | EOF |
|---|---|---|---|---|
| | 8 bits | 8 bits | 16 bits | |

**Figure 49 — Get system information response when Error_flag is set**

| SOF | Flags | Info flags | UID | DSFID | AFI | Other fields | CRC16 | EOF |
|---|---|---|---|---|---|---|---|---|
| | 8 bits | 8 bits | 64 bits | 8 bits | 8 bits | See below | 16 bits | |

**Figure 50 — Get system information response format when Error_flag is NOT set**

**Response parameter:**

Error_flag (and code if Error_flag is set)

if Error_flag is not set

Information flag

UID (mandatory)

Information fields, in the order of their corresponding flag, as defined in figure 50, if their corresponding flag is set.

**Table 10 — Information flags definition**

| Bit Nb | Flag name | State | Description |
|---|---|---|---|
| Bit 1 | DSFID | 0 | DSFID is not supported. DSFID field is not present |
| | | 1 | DSFID is supported. DSFID field is present |
| Bit 2 | AFI | 0 | AFI is not supported. AFI field is not present |
| | | 1 | AFI is supported. AFI field is present |
| Bit 3 | VICC memory size | 0 | Information on VICC memory size is not supported. Memory size field is not present. |
| | | 1 | Information on VICC memory size is supported. Memory size field is present. |
| Bit 4 | IC reference | 0 | Information on IC reference is not supported. IC reference field is not present. |
| | | 1 | Information on IC reference is supported. IC reference field is present. |
| Bit 5 | RFU | | Shall be set to 0. |
| Bit 6 | RFU | | Shall be set to 0. |
| Bit 7 | RFU | | Shall be set to 0. |
| Bit 8 | RFU | | Shall be set to 0. |

**Table 11 — VICC memory size information**

MSB LSB

| 16 14 | 13 9 | 8 1 |
|---|---|---|
| RFU | Block size in bytes | Number of blocks |

Block size is expressed in number of bytes on 5 bits, allowing to specify up to 32 bytes i.e. 256 bits. It is one less than the actual number of bytes. E.g. a value of '1F' indicates 32 bytes, a value of '00' indicates 1 byte.

Number of blocks is on 8 bits, allowing to specify up to 256 blocks. It is one less than the actual number of blocks. E.g. a value of 'FF' indicates 256 blocks, a value of '00' indicates 1 block.

The three most significant bits are reserved for future use and shall be set to zero.

The IC reference is on 8 bits and its meaning is defined by the IC manufacturer.

### 9.3.13 Get multiple block security status

**Command code = '2C'**

When receiving the Get multiple block security status command, the VICC shall send back the block security status.

The blocks are numbered from '00 to 'FF' (0 to 255).

The number of blocks in the request is one less than the number of block security status that the VICC shall return in its response.

E.g. a value of '06' in the "Number of blocks" field requests to return 7 Block security status. A value of '00' in the "Number of blocks" field requests to return a single Block security status.

| SOF | Flags | Get multiple block security status | UID | First block number | Number of blocks | CRC16 | EOF |
|---|---|---|---|---|---|---|---|
| | 8 bits | 8 bits | 64 bits | 8 bits | 8 bits | 16 bits | |

**Figure 51 — Get multiple block security status request format**

**Request parameter:**

(Optional) UID

First block number

Number of blocks minus one

| SOF | Flags | Error code | CRC16 | EOF |
|---|---|---|---|---|
| | 8 bits | 8 bits | 16 bits | |

**Figure 52 — Get multiple block security status response format when Error_flag is set**

| SOF | Flags | Block security status | CRC16 | EOF |
|---|---|---|---|---|
| | 8 bits | 8 bits | 16 bits | |
| | | Repeated as needed | | |

**Figure 53 — Get multiple block security status response format when Error_flag is NOT set**

**Response parameter:**

Error_flag (and code if Error_flag is set)

if Error_flag is not set

Block security status (repeated as per figure 53)

## 9.4 Custom commands

The format of custom command is generic and allows unambiguous attribution of custom command codes to each VICC manufacturer.

The custom command code is the combination of a custom command code and of the VICC manufacturer code.

The custom request parameters definition is the responsibility of the VICC manufacturer.

| SOF | Flags | Custom | IC Mfg code | Custom request parameters | CRC16 | EOF |
|---|---|---|---|---|---|---|
| | 8 bits | 8 bits | 8 bits | Custom defined | 16 bits | |

**Figure 54 — Custom request format**

**Request parameter:**

IC manufacturer code according to ISO/IEC 7816-6/AM1.

| SOF | Flags | Error code | CRC16 | EOF |
|---|---|---|---|---|
| | 8 bits | 8 bits | 16 bits | |

**Figure 55 — Custom response format when Error_flag is set**

| SOF | Flags | Custom response parameters | CRC16 | EOF |
|---|---|---|---|---|
| | 8 bits | Custom defined | 16 bits | |

**Figure 56 — Custom response format when Error_flag is NOT set**

**Response parameter:**

Error_flag (and code if Error_flag is set)

if Error_flag is not set

Custom parameters

## 9.5 Proprietary commands

The format of these commands is not defined in this standard

# Annex A
(informative)

# Compatibility with other card standards

This standard does not preclude the addition of other existing card standards on the VICC, such as ISO/IEC 7816 or others listed in annex B.

# Annex B
(informative)

# Bibliography of other ISO/IEC card standards.

ISO/IEC 7811, *Identification cards - Recording technique -*

ISO/IEC 7812-1:1993, *Identification cards - Identification of issuers - Part 1: Numbering system.*

ISO/IEC 7812-2:1993, *Identification cards - Identification of issuers - Part 2: Application and registration procedures.*

ISO/IEC 7813:1995, *Identification cards - Financial transaction cards.*

ISO/IEC 7816-1:1997, *Identification cards - Integrated circuit(s) cards with contacts - Part 1: Physical characteristics.*

ISO/IEC 7816-2:1998, *Identification cards - Integrated circuit(s) cards with contacts - Part 2: Dimensions and location of the contacts.*

# Annex C
(informative)

# VCD pseudo-code for anticollision

The following pseudo-code describes how the anticollision could be implemented on the VCD, using recursivity. It does not describe the collision detection mechanism.

**Algorithm for 16 slots**

```
function push (mask, address)          ; pushes on private stack
function pop (mask, address)           ; pops from private stack
function pulse_next_pause              ; generates a power pulse
function store(VICC_UID)               ; stores VICC_UID
function poll_loop (sub_address_size as integer)

; address length must be four (4) bits.
pop (mask, address)
mask = address & mask                  ; generates new mask

; send the Request
mode = anticollision
send_Request(Request_cmd, mode, mask length, mask[0])

for address = 0 to (2^sub_address_size - 1)
     if no_collision_is_detected then  ; VICC is inventoried
          store (VICC_UID)
else                                   ; remember a collision was detected
     push(mask,address)
endif

pulse_next_pause

next sub_address
; if some collisions have been detected and not yet processed,
; the function calls itself recursively to process the last
; stored collision
if stack_not_empty then poll_loop (sub_address_size)

end poll_loop

main_cycle:
mask = null
address = null
push (mask, address)
poll_loop(sub_address_size)
end_main_cycle
```

**Figure C.1 — VCD pseudo-code for anticollision**

# Annex D
(informative)

# Cyclic Redundancy Check (CRC)

## D.1 The CRC error detection method

The Cyclic Redundancy Check (CRC) is calculated on all data contained in a message, from the start of the flags through to the end of data. This CRC is used from VCD to VICC and from VICC to VCD.

**Table D.1 — CRC Definition**

| CRC Definition | | | | | |
|---|---|---|---|---|---|
| **CRC type** | **Length** | **Polynomial** | **Direction** | **Preset** | **Residue** |
| ISO/IEC 13239 | 16 bits | $X^{16} + X^{12} + X^5 + 1$ = '8408' | Backward | 'FFFF' | 'F0B8' |

To add extra protection against shifting errors, a further transformation on the calculated CRC is made. The One's complement of the calculated CRC is the value attached to the message for transmission. This transformation is included in the example below.

For checking of received messages the 2 CRC bytes are often also included in the re-calculation, for ease of use.

In this case, given the expected value for the generated CRC is the residue of 0xF0B8.

The example in Figure D.1 illustrates one method in C language of calculating the CRC on a given set of bytes comprising a message.

```
#include <stdio.h>

#define  POLYNOMIAL     0x8408         //  x^16 + x^12 + x^5 + 1
#define  PRESET_VALUE   0xFFFF
#define  CHECK_VALUE    0xF0B8
#define  NUMBER_OF_BYTES     4        // Example: 4 data bytes
#define  CALC_CRC            1
#define  CHECK_CRC           0

void main()
{
  unsigned int  current_crc_value;
  unsigned char array_of_databytes[NUMBER_OF_BYTES + 2] = {1, 2, 3, 4, 0x91, 0x39};
  int           number_of_databytes = NUMBER_OF_BYTES;
  int           calculate_or_check_crc;
  int           i, j;
  calculate_or_check_crc = CALC_CRC;
  //  calculate_or_check_crc = CHECK_CRC;  // This could be an other example
  if (calculate_or_check_crc == CALC_CRC)
  {
     number_of_databytes = NUMBER_OF_BYTES;
```

```
  }
  else    // check CRC
  {
     number_of_databytes = NUMBER_OF_BYTES + 2;
  }
  current_crc_value = PRESET_VALUE;
  for (i = 0; i < number_of_databytes; i++)
  {
     current_crc_value = current_crc_value ^ ((unsigned int)array_of_databytes[i]);
     for (j = 0; j < 8; j++)
     {
        if (current_crc_value & 0x0001)
        {
           current_crc_value = (current_crc_value >> 1) ^ POLYNOMIAL;
        }
        else
        {
           current_crc_value = (current_crc_value >> 1);
        }
     }
  }
  if (calculate_or_check_crc == CALC_CRC)
  {
     current_crc_value = ~current_crc_value;
     printf ("CRC-ISO/IEC 13239 of { 1, 2, 3, 4 } is '3991'\n");
     printf ("Generated CRC is '%04X'\n", current_crc_value);
        printf ("The Least Significant Byte (transmitted first) is: '%02X'\n",
          current_crc_value & 0xFF);
        printf( "The Most Significant Byte (transmitted second) is: '%02X'\n",
       (current_crc_value >> 8) & 0xFF);
     printf( "Executing this program when CHECK_CRC generates: 'F0B8'\n");
     // current_crc_value is now ready to be appended to the data stream
     // (first LSByte, then MSByte)
  }
  else   // check CRC
  {
     if (current_crc_value == CHECK_VALUE)
     {
        printf ("Checked CRC is ok (0x%04X)\n", current_crc_value);
     }
     else
     {
        printf ("Checked CRC is NOT ok (0x%04X)\n", current_crc_value);
     }
                                                          }
}
```

**Figure D.1 — C-example to calculate or check the CRC16 according to ISO/IEC 13239**

```
CRC-ISO/IEC 13239 of { 1, 2, 3, 4 } is '3991'
Generated CRC is '3991'
The Least Significant Byte (transmitted first) is: '91'
The Most Significant Byte (transmitted second) is: '39'
Executing this program when CHECK_CRC generates: 'F0B8'
```

**Figure D.2 — Printout of the C-program example in figure D.1**

## D.2 CRC calculation example

This example refers to a Read single block request for reading block '0B'.

The modes selected by the VCD are: single subcarrier, high VICC-to-VCD data rate, addressed.

The UID of the VICC is: 'E0 04 AB 89 67 45 23 01'

The request therefore consists of the following fields:

- the flags: '22'
- the command code: '20'
- the UID: 'E0 04 AB 89 67 45 23 01' where 'E0' is the most significant byte
- the block number: '0B'
- the CRC: 'BAE3' where 'BA' is the most significant byte.

The CRC is calculated on the request fields assembled in a frame according to the transmission rules defined in this standard:

'22' '20' '01' '23' '45' '67' '89' 'AB' '04' 'E0' '0B'

NOTE 1 The UID is transmitted least significant byte first.

NOTE 2 Table D.2 describes the different steps of the calculation.

The request is then transmitted as follows:

SOF '22' '20' '01' '23' '45' '67' '89' 'AB' '04' 'E0' '0B' 'E3' 'BA' EOF

NOTE 3 The CRC is transmitted least significant byte first.

NOTE 4 Each byte is transmitted least significant bit first.

**Table D.2 — Practical example of CRC calculation**

| Step | Input | Calculated CRC in VCD | Calculated CRC in VICC for check |
|---|---|---|---|
| 1 | '22' | 'F268' | '0D97' |
| 2 | '20' | '3EC6' | 'C139' |
| 3 | '01' | '42F5' | 'BD0A' |
| 4 | '23' | '4381' | 'BC7E' |
| 5 | '45' | '7013' | '8FEC' |
| 6 | '67' | 'C5AB' | '3A54' |
| 7 | '89' | 'F2AD' | '0D52' |
| 8 | 'AB' | '95BC' | '6A43' |
| 9 | '04' | 'C92E' | '36D1' |
| 10 | 'E0' | 'DFC3' | '203C' |
| 11 | '0B' | 'BAE3' | '451C' |
| 12 | 'E3' | | '0F3D' |
| 13 | 'BA' | | 'F0B8' |