AN2319
Application note

Using the 64-bit UID in contactless RFID applications

Introduction

All STMicroelectronics contactless memory products provide a 64-bit unique identifier (UID) for use with RFID applications. The 64-bit UID is a value written by STMicroelectronics during the manufacturing process and it is guaranteed unique for each contactless memory.

This UID can be used by the final application system to identify each tag individually and also, depending on the recognized memory type, to modify memory access models and maps.

This application note describes how STMicroelectronics handles the 64-bit UID and how applications can decode its value.

Contents

1 UID description

The 64-bit UID of STMicroelectronics provides a maximum of 2^{42} (4,398,046,511,104) unique ID tags per family.

Table 1. 64-bit unique identifier (UID) description

Bits	[63:56]	[55:48]	[47:42]	[41:0]
Size (bits)	8	8	6	42
Value	Tag registration category (0xD0)	Silicon manufacturer code (0x02)	Product code	Unique serial number

1.1 Unique serial number [41:0]

These 42 bits are the traceable unique serial number.

1.2 Product code [47:42]

These 6 bits store the STMicroelectronics product code. This code is unique for each tag family product and that is why it can be used by the contactless system to identify the product in the reader's antenna field. Once recognized, the contactless system knows the command set and available memory map. *Table 2* lists the product codes that identify current products.

Table 2. ST product code chart

Binary	Decimal	SR product
00 0010	2	SR176
00 0011	3	SRIX4K
00 0100	4	SRIX512
00 0110	6	SRI512
00 0111	7	SRI4K
00 1100	12	SRT512

1.3 Silicon manufacturer code [55:48]

This bit field identifies the Silicon manufacturer code. Value 0x02 is dedicated for STMicroelectronics in compliance with ISO 7816-6/AM1 specifications.

1.4 Tag registration category [63:56]

This bit field corresponds to the Registration category for RFID applications. Value 0xD0 is attributed to ISO14443-B standard devices.

2 Implementation

The body of a loop (see Example 1) is used by a reader to detect and identify RFID tags. This loop starts by switching the reader RF output OFF and then back ON again. This safely resets any tags in the field and puts them in a READY state.

By issuing the `INITIATE()` command, the tag enters ACTIVE (SR176) or INVENTORY (SRIxx) mode.

A non-zero response to the `INITIATE()` command indicates the presence of a tag and enables the reader to obtain the ChipID.

This ChipID is used as a link to the tag for the following `SELECT()` command.

After a successful `SELECT()` command, the tag is ready for read / write operations.

Once the value of the UID is known, the application can easily determine and set the important parameters for communication with the particular tag using the `switch()` statement shown in *Example 1: high-level mnemonic source code*.

In certain applications and when using certain tags, it is necessary to identify the tag product type. In this case, it is recommended to first read the tag UID value (which contains the product type value) as shown in *Example 2: low-level mnemonic source code*.

3 Application examples

Acquiring the UID is quite easy when using high-level SRIxx tags, but additional actions are required when using low-level SR176 memories.

After sending the `INITIATE()` and `SELECT()` command sequence to the tag(s), send a `GET_UID()` command to identify UID values.

SRIxx tags directly return the UID after receiving the `GET_UID()` command, while SR176 tags do not reply to this command.

If a tag is detected in the reader RF field but no reply is returned after a `GET_UID()` command, it is probably a SR176 tag. In this case, the reader must read a specific memory address to retrieve its UID as shown in Example 2.

The example shown in *Section 3.2* is the body of the `CRX14_Get_UID()` function used in Example 1.

3.1 Example 1: high-level mnemonic source code

This is an example of a UID acquisition loop using a high-level mnemonic source code.

```
UCHAR UID[8];
UCHAR blockSize, NoOfBlock, WriteblockAddress;


/*********************************************
****            Manage all SRxx
*********************************************/

CRX14_Set_Param(0x00);  //RF Off (accessing the Parameter Register of CRX14)
CRX14_Set_Param(0x10);  //RF On

if (CRX14_Initiate())  // (INITIATE() command)
{
   CRX14_Select(Chip_ID); (SELECT() command)
   CRX14_Get_UID(UID);
   if((UID[7]==0xd0)&&(UID[6]==0x02))
      switch (((UID[5] & 0xFC)>>2))
      {
         case 2: //SR176
         blockSize=2; NoOfBlock=16; WriteblockAddress=4;
         break;

         case 3: //SRIX4K
         blockSize=4; NoOfBlock=128; WriteblockAddress=7;
         break;
      }
}
```

3.2 Example 2: low-level mnemonic source code

This is an example of the body of a function used to acquire the UID using a low-level mnemonic source code.

```
UCHAR buff[];  //actual copy of Input/Output Frame Register of CRX14

send_get_UID_Command();  (GET_UID() command)

if(buff[0]=0x0)  //we may deal with SR176
{
   read_a(0); //common read function (READ_BLOCK() command)
   if(buff[0]!=0x0)
   {
      UID[0] = buff[1];
      UID[1] = buff[2];

      read_a(1);
      UID[2] = buff[1];
      UID[3] = buff[2];
      read_a(2);
      UID[4] = buff[1];
      UID[5] = buff[2];
      read_a(3);
      UID[6] = buff[1];
      UID[7] = buff[2];
   }
}
```

The follow-up processing of the UID is the same for all product types as shown in Example 1.

4 Revision history

Table 3. Document revision history

Date	Revision	Changes
28-Feb-2006	1	Initial release.
20-Jan-2009	2	*Table 2: ST product code chart* updated.