Application Note
Software

Non-Volatile Memory of dresden elektronik Radio Modules and USB Sticks

Author

Enrico Lehmann

dresden elektronik ingenieurtechnik gmbh

July 2013

Table of contents

Document history

Date	Version	Description
2011-09-29	1.0	Initial version
2012-07-09	1.1	Described details of the NV memory structure.
2013-01-28	1.2	More explanation about structure and its fields. Added location of the NV memory section based on the platform.
2013-02-11	1.3	Added recovery section of NV memory.
2013-05-17	1.4	Added CRC C-Code example section.
2013-07-19	1.5	Added module picture with MAC address

Abbreviations

Abbreviation	Description
de	dresden elektronik
EEPROM	Electrically Erasable Programmable Read-Only Memory
MAC	Medium Access Control
NV	Non-Volatile

Introduction

This Application Note will explain the Non-Volatile memory structure which is used to save the MAC address on dresden elektronik's (de) radio modules and USB sticks. This MAC address is necessary to successfully build up a wireless sensor network since it is used for initial identification. Every single radio module or USB stick is shipped with a world-wide unique MAC address. This address is saved in the module's internal flash or EEPROM in a defined place. This application note will introduce the scheme which is used and will further explain how to restore an erased MAC address.

Non-Volatile Memory Location

The Non-Volatile Memory (referred to as NV memory from now on) is located at the end of the NV memory region as illustrated in **Figure 1**. The location depends on the platform used. In **Table 1** the location of the NV memory within each dresden elektronik platform is described. The NV memory itself contains different information and is 32 byte in size. The content is described in the next section.

Contr.	Platform	Location	Description
AVR	deRFmega128-22AXX deRFmega128-22CXX deRFmega128-22MXX (4kB EEPROM)	(E2END + 1) - 32 (0xFFF + 1) - 0x20 = 0xFE0	last 32 bytes of the internal EEPROM
	deRFmega256-23MXX (8kB EEPROM)	(E2END + 1) - 32 (0x1FFF + 1) - 0x20= 0x1FE0	last 32 bytes of the internal EEPROM
ARM7	deRFarm7-X5AXX (512kB Flash)	(Flash_Start + Flash_Size) - 32 (0x100000 + 0x80000) - 0x20 = 0x17FFE0	last 32 bytes of last flash page of the internal flash
SAM3	deRFsam3-X3MXX deRFusb-X3EXX (256kB Flash)	(Flash_Start + Flash_Size) - 32 (0x400000 + 0x40000) - 0x20 = 0x43FFE0	last 32 Bytes of last flash page of the internal flash

Table 1: NV memory location

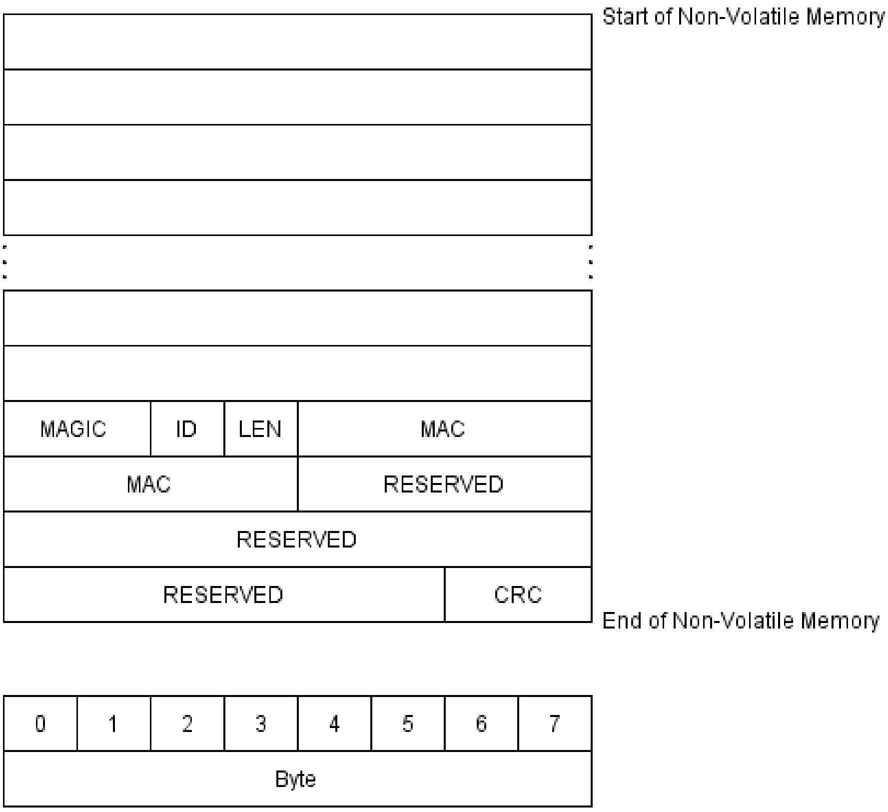

Figure 1: NV memory location and structure

Non-Volatile Memory Structure

Figure 2 illustrates the structure of the NV memory. Each field is explained in **Table 3**.

The MAGIC field is a constant value. The ID denotes the version of the NV memory structure. Its value changes with each structural change within the NV memory. The LEN field refers to the size of the actual NV memory data – currently only the MAC address. The MAC field is the actual data and contains the MAC address of the device. The RESERVED fields are reserved for future use. The CRC is calculated over MAGIC, ID, LEN and all following data bytes. All RESERVED fields are not included in the CRC calculation.

Currently two types of CRC algorithm exist. NV memories with version 1 (ID = 1) use a proprietary CRC-16 algorithm. With version 2 (ID = 2) the algorithm has been changed to CRC-16-CCITT.

```
struct
{
    uint16_t magic;
    uint8_t  id;
    uint8_t  len;
    uint8_t  mac[8];
    uint8_t  reserved[18];
    uint16_t crc;
}
```

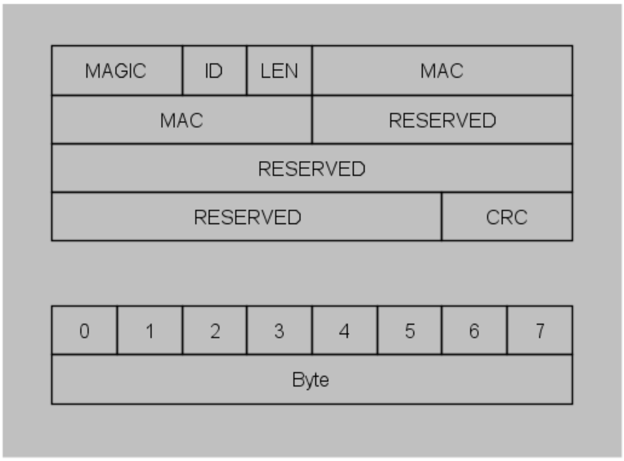

Figure 2: NV memory Structure

The left part of **Figure 2** shows a C code structure definition that can be used to access the NV memory in a structured way.

Depending on the version of the NV memory not only the structure but also the data format in which the fields are stored might change (e.g. little-endian, big-endian). To generate full transparency for each field of the NV memory the storage format for this field is given for each known version of the NV memory. Therefore, each field description contains a small table like this:

Version	Endianess
1	Big-Endian
2	Little-Endian

Table 2: Example of the data format table

Field Description

<table>
<tr><th>Field</th><th>Offset</th><th>Length (Bytes)</th><th>Description</th></tr>
<tr><td>MAGIC</td><td>0x00</td><td>2</td><td>The value is 0xDE90.
Data format:<table><tr><th>Version</th><th>Endianess</th></tr><tr><td>1</td><td>Little-Endian</td></tr><tr><td>2</td><td>Little-Endian</td></tr></table></td></tr>
<tr><td>ID</td><td>0x02</td><td>1</td><td>The version of the NV memory structure.
Data format: not required</td></tr>
<tr><td>LEN</td><td>0x03</td><td>1</td><td>The length of the data stored in this NV memory. As each structural change also changes the version this value is always related to the version.
Data length:<table><tr><th>Version</th><th>Value (data length in byte)</th></tr><tr><td>1</td><td>8</td></tr><tr><td>2</td><td>8</td></tr></table></td></tr>
<tr><td>MAC</td><td>0x04</td><td>8</td><td>The MAC address. The data format depends on the NV memory version.
Data format:<table><tr><th>Version</th><th>Endianess</th></tr><tr><td>1</td><td>Big-Endian</td></tr><tr><td>2</td><td>Little-Endian</td></tr></table></td></tr>
<tr><td>RESERVED</td><td>0x0C</td><td>18</td><td>Is reserved for future use. The content and data format is undefined.</td></tr>
<tr><td>CRC</td><td>0x1E</td><td>2</td><td>The CRC, to check if the content is valid.
Data format:<table><tr><th>Version</th><th>CRC type</th><th>Endianess</th></tr><tr><td>1</td><td>proprietary</td><td>Big-Endian</td></tr><tr><td>2</td><td>CRC-16-CCITT</td><td>Little-Endian</td></tr></table></td></tr>
</table>

Table 3: NV memory fields

C-Code for CRC Calculation

This section provides C-Code examples for CRC calculation, which is used to validate the NV-Memory structure. As described in section **Non-Volatile Memory Structure**, there are two CRC calculations used.

In **Listing 1**, the proprietary CRC calculation, which is used in Version 1, is demonstrated.

```c
/**
 * @brief Calculate a 16 bit CRC over the given buffer.
 *
 * @param pBuffer Pointer to buffer where to calculate CRC.
 * @param size    Size of buffer.
*/
static uint16_t calc_crc(uint8_t* pBuffer, uint8_t size)
{
   uint8_t tmp;
   uint8_t high, low;

   high = 0xFF;
   low = 0xFF;

   while (size > 0)
   {
      tmp = *pBuffer;

      high = high ^ tmp;
      high = high ^ ((high >> 4) & 0x0F);
      tmp = high >> 3;
      low = low ^ (tmp & 0x1F );
      low = low ^ ((tmp >> 1) & 0x0F);
      tmp = (tmp & 0x0E) ^ high;
      high = low;
      low = tmp;

      pBuffer++;
      size--;
   }

   size = high;
   size = (size << 8) + low;

   return size;
}
```

Listing 1: Proprietary CRC calculation (used in Version 1)

In **Listing 2** the CRC-16-CCITT variant, used in Version 2, is demonstrated.

```
/**
 * @brief Calculate a CRC-16-CCITT over the given buffer.
 *
 *
 * @param pBuffer Pointer to buffer where to calculate CRC.
 * @param size    Size of buffer.
*/
uint16_t calc_crc(uint8_t* pBuffer, uint16_t size)
{
   uint16_t crc = 0xFFFF; // initialize value

   for(uint16_t i = 0; i < size; i++)
   {
      crc = crc_16_ccitt(crc, *pBuffer++);
   }
   return crc;
}

/**
 *  @brief Optimized CRC-CCITT calculation.
 *
 *  Polynomial: x^16 + x^12 + x^5 + 1 (0x8408)
 *  Initial value: 0xffff
 *
 *  This is the CRC used by PPP and IrDA.
 *  See RFC1171 (PPP protocol) and IrDA IrLAP 1.1
 *
 *  Adopted from WinAVR library code
 *
 * @param crc     The initial/previous CRC.
 * @param data    The data byte for which checksum must be created
*/
static inline uint16_t crc_16_ccitt(uint16_t crc, uint8_t data)
{
   data ^= (crc & 0xFF);
   data ^= data << 4;

   return ((((uint16_t)data << 8) | (uint8_t)(crc >> 8)) ^ \
             (uint8_t)(data >> 4) ^ ((uint16_t)data << 3));
}
```

Listing 2: CRC-16-CCITT calculation (used in Version 2)

Recover Non-Volatile Memory

If the NV memory section is erased, destroyed or changed, the content is no longer valid and therefore the radio module has no valid MAC making many software stacks stop with an alert or an exception. With help of the previously explained NV memory structure the content could be recovered by writing some recovery code. A short recovery example in C code is given below.

```
#define NV_MEMORY_START /* define location of NV memory */

#define NV_HEADER_LEN   4 /* define NV Header length */

void recover_nv_memory(void)
{
   /* create the NV memory set */
   uint8_t nv_set[32];
   uint16_t crc;

   /* make sure the NV memory is cleared */
   memset(nv_set, 0, sizeof(nv_set));

   /* fill the nv memory set with content */
   nv_set[0]  = 0x90; /* set MAGIC[1] */
   nv_set[1]  = 0xDE; /* set MAGIC[0] */
   nv_set[2]  = 0x02; /* set ID */
   nv_set[3]  = 0x08; /* set LENGTH */
   nv_set[4]  = 0x53; /* set MAC[0] - LSB */
   nv_set[5]  = 0x1C; /* set MAC[1] */
   nv_set[6]  = 0x00; /* set MAC[2] */
   nv_set[7]  = 0xFF; /* set MAC[3] */
   nv_set[8]  = 0xFF; /* set MAC[4] */
   nv_set[9]  = 0x2E; /* set MAC[5] */
   nv_set[10] = 0x21; /* set MAC[6] */
   nv_set[11] = 0x00; /* set MAC[7] - MSB */

   /* calculate CRC */
   crc = calc_crc(nv_set, (nv_set[3] + NV_HEADER_LEN));

   /* set CRC to NV memory */
   nv_set[30] = (crc & 0xFF);
   nv_set[31] = ((crc >> 8) & 0xFF);

   /* write NV memory set to NV memory location */
   write_nv_memory( (nv_set,
                    (void*)NV_MEMORY_START,
                    sizeof(nv_set));
}
```

Listing 3: C code example for NV memory recovery

As demonstrated in **Listing 3** the easiest way to create the NV memory content is to create a temporary array with size of the NV memory. Set each NV memory field with the correct values, calculate the CRC and write it to the correct location. C code examples of the CRC calculations are shown in **Listing 1** and **Listing 2**. When recover the NV memory structure, the CRC-16-CCITT should be used (see **Listing 2**).

Note: The MAC address of your module should be written on the label. The address given in the example should not be used for any of your nodes.

Note: You must change this code for each node as you always have to input the MAC address of the very module the recovery code will be used for.

For better understanding of how the MAC address (on the module label) is written to the NV memory, the corresponding order is demonstrated in **Figure 3**.

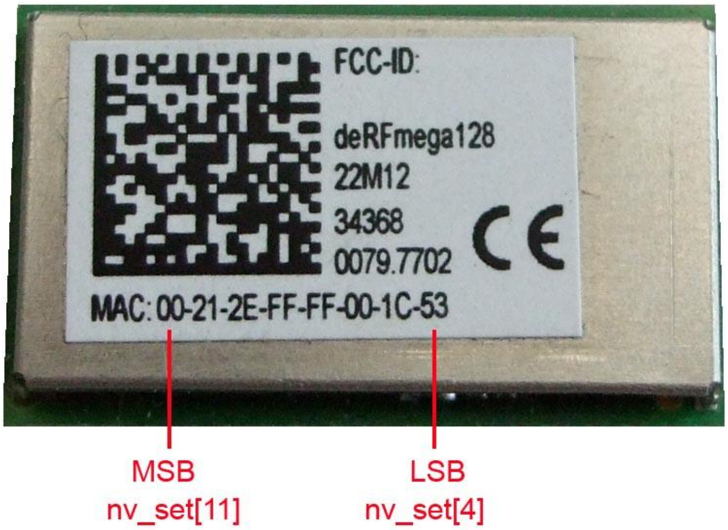

Figure 3: Module MAC Label

The example code in **Listing 3** calls `write_nv_memory()` to write the NV memory content to the NV memory location. This function must be implemented by the user. For AVR-based radio modules the `eeprom_write_block()` function is a good choice. For ARM-based radio modules flash write functions are required that might be part of the vendor supplied low level driver library.

Note: You need to set the value of `NV_MEMORY_START` to the correct address as given in **Table 1**.

Known Issues

This section will list all known issues concerning the NV memory.

CRC validation failure

During initial NV memory creation, the MAGIC field is not written as 0xDE90 to the NV memory section but as 0xDE00. This applies to all deRFmega128-22Axx and deRFmega128-22Cxx radio modules.

The CRC is calculated with the correct MAGIC value. Verifying the CRC after reading the NV memory will fail as the MAGIC read is 0xDE00 but the CRC has been calculated with a value of 0xDE90.

To prevent those CRC validation failures the NV memory should be read into an RAM array and the MAGIC field modified to 0xDE90 if it matches 0xDE00. After that, the CRC can be calculated as described above and verified against the CRC in the array.

dresden elektronik ingenieurtechnik gmbh
Enno-Heidebroek-Straße 12
01237 Dresden
GERMANY

Phone +49 351 31850-0
Fax +49 351 31850-10
Email wireless@dresden-elektronik.de

Trademarks and acknowledgements

All trademarks are registered by their respective owners in certain countries only. Other brands and their products are trademarks or registered trademarks of their respective holders and should be noted as such.

Disclaimer

This note is provided as-is and is subject to change without notice. Except to the extent prohibited by law, dresden elektronik ingenieurtechnik gmbh makes no express or implied warranty of any kind with regard to this guide, and specifically disclaims the implied warranties and conditions of merchantability and fitness for a particular purpose. dresden elektronik ingenieurtechnik gmbh shall not be liable for any errors or incidental or consequential damage in connection with the furnishing, performance or use of this guide.