NUR APPLICATION NOTE 10 (NUR AN010)

GETTING STARTED: HOW TAG SELECTION WORKS

SCOPE

This document illustrates how the tag selection for an operation (read, write, etc.) works. The selection is explained with two examples; first one is the most common type (EPC based selection) and the second one shows a more complex type of selection. The latter is “generated” for this documented but still reflects a real world situation where the tag singulation may not be that straightforward. The general singulation block information can be found in the NUR protcol documentation.

Scenario	Description
The singulation block	General about the singulation block and its behavior control
Basic tag selection using the contents of EPC	The basic, most commonly used tag selection scheme.
EPC selection build	Simple EPC-based selection mask building in C.
Advanced tag selection	Advanced tag selection procedure based on a more complex bit pattern found in the user memory.

THE SINGULATION BLOCK

In the NUR protocol the singulation block follows the common block that includes the flag byte and password (unsigned 32-bit integer). The common block's flag field has two bits that control the singulation: bit1 (mask 0x02) that, when set, tells that there is a singulation block after the common block. The second bit is the 64-bit addressing flag (bit 3, mask 0x08, "EA"): if it is set, then the singulation is expected to contain a 64-bit address instead of a 32-bit (EA = '0'). In terms of a C-structure (cast to a byte buffer to handle the variable length of the selection mask data) the common and singulation block are for example:

```
/* Block that defines the addressing modes and password usage. */
struct __packed NUR_COMMONBLOCK
{
  uint8_t flags;  /* Addressing and password usage. */
  uint32_t pwd;   /* Password value. */
};

/* Block that defines tag selection parameters: 32-bit */
struct __packed NUR_TAGSELBLOCK32
{
  uint8_t size;      /* Number of bytes to follow. */
  uint8_t bank;      /* Selection bank. */
  uint32_t bitAddr;  /* 32-bit address of the selection mask. */
  uint16_t bitLen;   /* Bit length of the selection mask. */
  uint8_t bitBuf[1]; /* Variable length bit buffer data. */
};

/* Block that defines tag selection parameters: 64-bit */
struct __packed NUR_TAGSELBLOCK64
{
  uint8_t size;      /* Number of bytes to follow. */
  uint8_t bank;      /* Selection bank. */
  uint64_t bitAddr;  /* 64-bit address of the selection mask. */
  uint16_t bitLen;   /* Bit length of the selection mask. */
  uint8_t bitBuf[1]; /* Variable length bit buffer data. */
};
```

The singulation block is always present for operations that require a tag to be accessed in a specific way based on its EPC, TID or user memory contents.

BASIC TAG SELECTION USING THE CONTENTS OF THE EPC

This is the most commonly used way to access a tag. These days growing number of UHF RFID use is based on some standard numbering system that makes the EPC contents unique thus making larger system handling a lot easier.

The EPC based selection is easy also because the bit mask's address as well as the mask's length are both aligned by 8 bits thus the data can directly be handled as bytes instead of bit shift and mask operations that may grow very complex.

The below table shows the singulation block for a tag having the 96-bit EPC

```
20 00 02 01 01 00 00 00 00 00 06 DB
```

Using 32-bit addressing the singulation block contents is:

Byte(s)	Value (HEX)	Is
0	13	Number of bytes to follow (19).
1	01	Selection bank (1 = EPC)
2…5	20 00 00 00	Selection's bit address (32, word address 2)
6…7	60 00	Selection mask's bit length (96).
8…19	20 00 02 01 01 00 00 00 00 00 06 DB	The bit mask used for selection.

BUILDING SIMPLE EPC SELECTION BLOCK IN C

A simple C-function that allocates buffer, builds an EPC selection mask and that returns a byte pointer as well as the final length of the block (expects 16-bit word aligned data):

```
uint8_t *build_epc_sel_block(
   uint8_t *epc,
   int epcLen,
   int *pBufLen
)
{
  uint8_t *pBuf;
  struct NUR_TAGSELBLOCK32 *pBlock;
  size_t szAlloc;

  if (epc == NULL  || bufLen == NULL ||
      epcLen < 2 || (epcLen % 2) != 0 || epcLen > 62) {
          return NULL;
    }

  /* -1 = bitBuf[1] */
  szAlloc = sizeof(struct NUR_TAGSELBLOCK32) - 1;
  szAlloc += epcLen;

  pBuf = (uint8_t *)malloc(szAlloc);
  /*
    Requires that memory can be addressed
    in 1 byte aligment.
  */
  pBlock = (struct NUR_TAGSELBLOCK32 *)pBuf;

  memset(pBuf, 0, szAlloc);

  pBlock->size = (uint8_t)szAlloc - 1;  /* - size */
  pBlock->bank = 1; /* EPC */
  pBlock->bitAddr = 32; /* EPC bit address. */
  pBlock->bitLen = (uint16_t)(epcLen * 8); /* -> bit length. */
  memcpy(&pBlock->bitBuf[0], epc, epcLen);

  *pBufLen = (int)szAlloc;
  return pBuf;
}
```

ADVANCED TAG SELECTION

This part of the document focuses to a bit more complex tag selection. In this example the bit pattern is not byte aligned just to illustrate how the selection bit buffer should be built.

The following 23-bit pattern is expected to be found in the user memory (bank 3) at bit address 19 (0x13) thus the bits cover bit address range 19…41:

```
1000 1100 0111 0111 0110 100
```

When the bit pattern is read from left to right and padded to the next byte we get byte mask (HEX)

```
8C 77 68
```

Table to illustrate the mask conversion:

Index	0								1								2							
Byte	8C								77								68							
Bits	1	0	0	0	1	1	0	0	0	1	1	1	0	1	1	1	0	1	1	0	1	0	0	0
	Selection mask												Padding											

The first 6 bytes (3 words, 48 bits) in the target tag's user memory is shown below. From the split we can see that the target tag has a matching pattern where it is expected to be:

Index (bits)	0 (0…7)	1 (8…15)	2 (16…23)	3 (24…31)	4 (32…39)	5 (40…47)
Byte	AC	01	71	8E	ED	14
Bits	10101100	00000001	011**10001**	**10001110**	**11101101**	**00**010100

Using 32-bit addressing the singulation block contents is:

Byte(s)	Value (HEX)	Is
0	0A	Number of bytes to follow (10).
1	03	Selection bank (3 = user memory)
2…5	13 00 00 00	Selection's bit address (19)
6…7	17 00	Selection mask's bit length (23).
8…10	8C 77 68	The bit mask used for selection: padded "to right" so that the first transmitted bit is the leftmost one in the byte array.