AN2745
Rev. 0, 07/2004

Setting Up TSEC Hash Tables

by Dana Castillo
NCSD Applications
Freescale Semiconductor, Inc.
Austin, TX

This application note describes the procedure to setup the hash tables for the three-speed Ethernet controller (TSEC). PowerQUICC™ Ethernet controllers prior to TSEC accomplished this task for the user. However, TSEC requires that the user manually set the hash table entries by setting the appropriate bits in the group and individual address registers.

This document should assist the user in successfully programming registers and creating hash tables. Included are recommended steps for determining and setting the appropriate bits in the hash tables and example code.

Contents

1 Destination Address Recognition

Figure 1 shows a flowchart for address recognition on received frames that is used to perform frame filtering using destination address (DA) recognition to determine whether to receive or discard the frame. Frames can be either individual (I) or group (G) addressed frames.

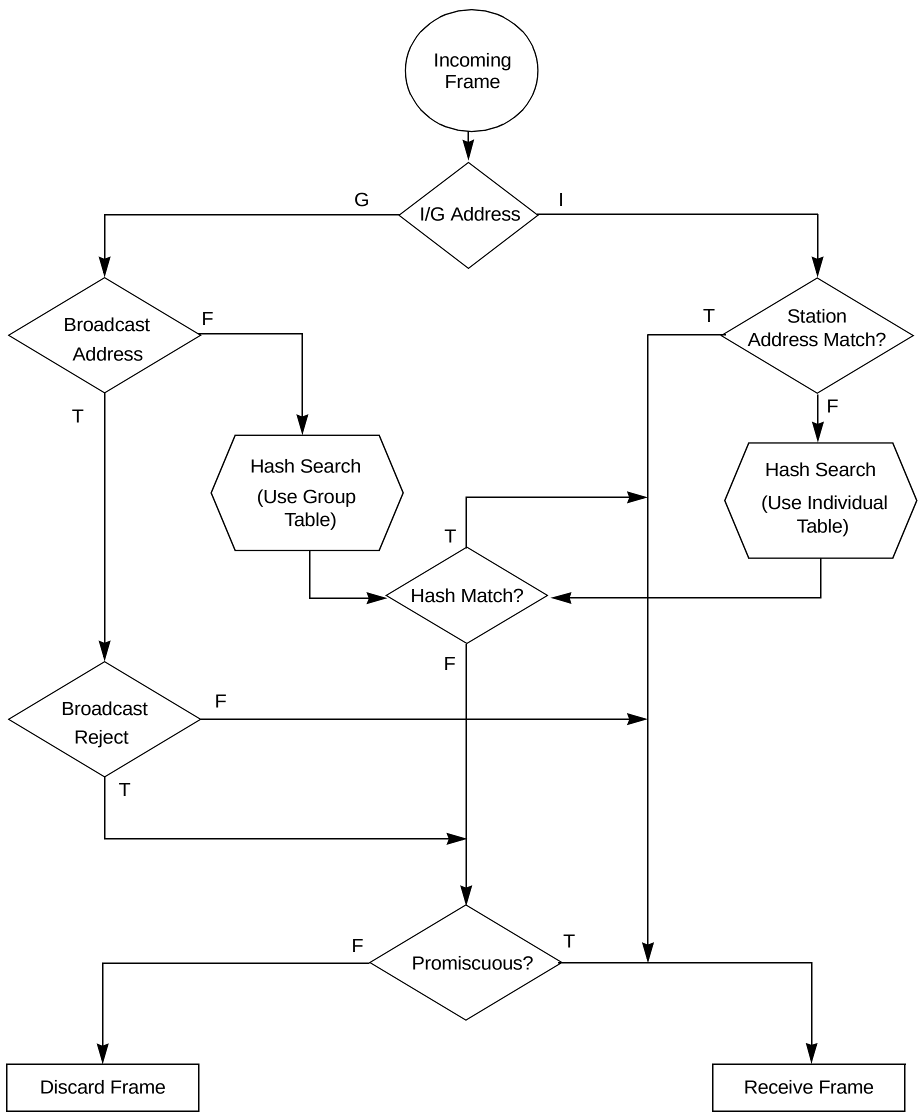

Figure 1. Ethernet Address Recognition Flowchart

The hash searches occur only when a DA does not match the station address or broadcast address. To determine whether the DA maps to a bit in the hash table, the cyclic redundancy check (CRC) of the DA is determined and the CRC value's least significant byte is complemented and bit-reversed. The resulting value is then analyzed and compared to the appropriate bit in the individual or group registers.

2 Filling the Hash Tables

The following subsections describe the IADDRs, GADDRs, and recommended procedure for setting bits in the hash tables. Additional examples are also provided in this section.

2.1 Individual Address Registers 0-7 (IADDR*n*)

The user must write the IADDR*n* registers, shown in Figure 2. These registers represent 256 entries of the individual (unicast) address hash table used in the address recognition process. While the DA field of a receive frame is processed through a 32-bit CRC generator, the 8 bits of the CRC remainder are mapped to one of the 256 entries. The user can enable a hash entry by setting the appropriate bit. A hash table hit occurs if the DA CRC result points to an enabled hash entry. Each of the eight IADDR register fields represents the 32-bit value associated with the corresponding register. IADDR0 contains the high-order 32 bits of the 256-entry hash table and IADDR7 represents the low-order 32 bits.

	0 … 31
R W	IADDR*n*
Reset	0000_0000_0000_0000_0000_0000_0000_0000
Offset	TSEC1:0x2_4800, 0x2_4804, 0x2_4808, 0x2_480C, 0x2_4810, 0x2_4814, 0x2_4818, 0x2_481C TSEC2:0x2_5800, 0x2_5804, 0x2_5808, 0x2_580C, 0x2_5810, 0x2_5814, 0x2_5818, 0x2_581C

Figure 2. IADDR*n* Register Definition

2.2 Group Address Registers 0-7 (GADDR*n*)

The user must also write the GADDR*n* registers, shown in Figure 3. Together these registers represent 256 entries of the group (multicast) address hash table used in the address recognition process. While the DA field of a receive frame is processed through a 32-bit CRC generator, the 8 bits of the CRC remainder are mapped to one of the 256 entries. The user can enable a hash entry by setting the appropriate bit. A hash table hit occurs if the DA CRC result points to an enabled hash entry. Each GADDR*n* represents the 32-bit value associated with the corresponding register. GADDR0 contains the high-order 32 bits of the 256-entry hash table and GADDR7 represents the low-order 32 bits.

	0 … 31
R W	GADDR*n*
Reset	0000_0000_0000_0000_0000_0000_0000_0000
Offset	TSEC1:0x2_4880, 0x2_4884, 0x2_4888, 0x2_488C, 0x2_4890, 0x2_4894, 0x2_4898, 0x2_489C TSEC2:0x2_5880, 0x2_5884, 0x2_5888, 0x2_588C, 0x2_5890, 0x2_5894, 0x2_5898, 0x2_589C

Figure 3. GADDR*n* Register Definition

2.3 Setting the Appropriate Bits

If the DA field of a receive frame is processed through a 32-bit CRC generator, the lower order 8 bits of the CRC remainder (1's complemented and bit-reversed) are mapped to a hash table entry. The user can enable a hash entry by setting the appropriate bit. A hash entry usually represents a set of addresses. A hash table hit occurs if the DA CRC result points to an enabled hash entry.

NOTE

The hash tables cannot be used to reject frames that match a set of selected addresses because unintended addresses can map to the same bit in the hash tables. Thus, an external CAM (content-addressable memory) or software must be used to implement this function.

The three steps to setting the appropriate bits in the IADDRs and GADDRs are as follows:

1. Compute the CRC value of the DA.
2. Bit-reverse the least significant byte of the CRC.
3. Select the appropriate register bit to set.

NOTE

The Linux implementation of the CRC algorithm requires the user to complement the least significant byte of the CRC value before you bit reverse it.

The following subsections will describe the three steps and step through an example with the following group destination address:

- Destination MAC address: DA = 0x0100_0CCC_CCCC

2.3.1 Computing the CRC

There are many algorithms for calculating the CRC value of a number. Refer to the RFC3309 standard (which can be found at http://www.faqs.org/rfcs/rfc3309.html) to compute the CRC value for the purposes of TSEC. The RFC3309 algorithm uses the following polynomial to calculate the CRC value:

$x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^{8}+x^{7}+x^{5}+x^{4}+x^{2}+x^{1}+x^{0}$ or 0x04C1_1DB7.

The algorithm results in the following CRC value using the previously given example values:

- CRC Value:

  CRC =0xA29F_4BBC

2.3.2 Bit-Reversing the CRC

The high-order 3 bits of the BR_CRC value obtained from the last step are used to select which of the eight 32-bit registersto use. The resulting value is shown below:

- Bit-Reversed CRC value:

  BR_CRC = 0x3D = 0b0011_1101

2.3.3 Selecting the Appropriate Register Bit to Set

The high-order 3-bits of the BR_CRC value obtained from the last step are used to select which 32-bit register (of the 8) to use. The on-going example maps the DA to register 1.

- High-Order 3 bits of BR_CRC:

  HO_CRC = 0b001 = 1

The low-order 5 bits will be used to select which bit to set in the given register (with a value of 0 setting 0x8000_0000 and 31 setting 0x0000_0001). Therefore, the example DA maps to bit 29 of register 1.

- Low-Order 5 bits of BR_CRC:

  LO_CRC = 0b1_1101 = 29

Therefore, GADDR1 would be ORed with the value 0x0000_0004.

2.3.4 Additional Calculated Examples

Example 1:

- Destination MAC address:
  DA = 0x0100_5E00_0128
- CRC remainder value:
  CRC = 0x821D_6CD3
- Bit-Reversed least significant byte of CRC Value:
  BR_CRC = 0xCB = 0b1100_1011
- High-Order 3 bits of BR_CRC:
  HO_CRC = 0b110 = 6
- Low-Order 5 bits of BR_CRC:
  LO_CRC = 0b0_1011 = 11
- GADDR6 |= 0x0010_0000

Example 2:

- Destination MAC address:
  DA = 0x0004_F060_4F10
- CRC remainder value:
  CRC = 0x1F5A_66B5
- Bit-Reversed least significant byte of CRC Value:
  BR_CRC = 0xAD = 0b1010_1101
- High-Order 3 bits of BR_CRC:
  HO_CRC = 0b101 = 5
- Low-Order 5 bits of BR_CRC:
  LO_CRC = 0b0_1101 = 13
- GADDR5 |= 0x0004_0000

3 Example C Code

The following example C code calculates the CRC and can be used to find the appropriate bits to set for a given DA value. The code was developed using Metrowerks™ CodeWarrior™ Development Tools for Windows, Version 8.0.

```
#define CRC32_POLY 0x04c11db7    // AUTODIN II, Ethernet, & FDDI

// CRC table (256 entries) generated by RFC3309

unsigned crctab[] = {

 0x00000000, 0x77073096, 0xee0e612c, 0x990951ba, 0x076dc419,

 0x706af48f, 0xe963a535, 0x9e6495a3, 0x0edb8832, 0x79dcb8a4,

 0xe0d5e91e, 0x97d2d988, 0x09b64c2b, 0x7eb17cbd, 0xe7b82d07,

 0x90bf1d91, 0x1db71064, 0x6ab020f2, 0xf3b97148, 0x84be41de,
```

```
0x1adad47d, 0x6ddde4eb, 0xf4d4b551, 0x83d385c7, 0x136c9856,
0x646ba8c0, 0xfd62f97a, 0x8a65c9ec, 0x14015c4f, 0x63066cd9,
0xfa0f3d63, 0x8d080df5, 0x3b6e20c8, 0x4c69105e, 0xd56041e4,
0xa2677172, 0x3c03e4d1, 0x4b04d447, 0xd20d85fd, 0xa50ab56b,
0x35b5a8fa, 0x42b2986c, 0xdbbbc9d6, 0xacbcf940, 0x32d86ce3,
0x45df5c75, 0xdcd60dcf, 0xabd13d59, 0x26d930ac, 0x51de003a,
0xc8d75180, 0xbfd06116, 0x21b4f4b5, 0x56b3c423, 0xcfba9599,
0xb8bda50f, 0x2802b89e, 0x5f058808, 0xc60cd9b2, 0xb10be924,
0x2f6f7c87, 0x58684c11, 0xc1611dab, 0xb6662d3d, 0x76dc4190,
0x01db7106, 0x98d220bc, 0xefd5102a, 0x71b18589, 0x06b6b51f,
0x9fbfe4a5, 0xe8b8d433, 0x7807c9a2, 0x0f00f934, 0x9609a88e,
0xe10e9818, 0x7f6a0dbb, 0x086d3d2d, 0x91646c97, 0xe6635c01,
0x6b6b51f4, 0x1c6c6162, 0x856530d8, 0xf262004e, 0x6c0695ed,
0x1b01a57b, 0x8208f4c1, 0xf50fc457, 0x65b0d9c6, 0x12b7e950,
0x8bbeb8ea, 0xfcb9887c, 0x62dd1ddf, 0x15da2d49, 0x8cd37cf3,
0xfbd44c65, 0x4db26158, 0x3ab551ce, 0xa3bc0074, 0xd4bb30e2,
0x4adfa541, 0x3dd895d7, 0xa4d1c46d, 0xd3d6f4fb, 0x4369e96a,
0x346ed9fc, 0xad678846, 0xda60b8d0, 0x44042d73, 0x33031de5,
0xaa0a4c5f, 0xdd0d7cc9, 0x5005713c, 0x270241aa, 0xbe0b1010,
0xc90c2086, 0x5768b525, 0x206f85b3, 0xb966d409, 0xce61e49f,
0x5edef90e, 0x29d9c998, 0xb0d09822, 0xc7d7a8b4, 0x59b33d17,
0x2eb40d81, 0xb7bd5c3b, 0xc0ba6cad, 0xedb88320, 0x9abfb3b6,
0x03b6e20c, 0x74b1d29a, 0xead54739, 0x9dd277af, 0x04db2615,
0x73dc1683, 0xe3630b12, 0x94643b84, 0x0d6d6a3e, 0x7a6a5aa8,
0xe40ecf0b, 0x9309ff9d, 0x0a00ae27, 0x7d079eb1, 0xf00f9344,
0x8708a3d2, 0x1e01f268, 0x6906c2fe, 0xf762575d, 0x806567cb,
0x196c3671, 0x6e6b06e7, 0xfed41b76, 0x89d32be0, 0x10da7a5a,
0x67dd4acc, 0xf9b9df6f, 0x8ebeeff9, 0x17b7be43, 0x60b08ed5,
0xd6d6a3e8, 0xa1d1937e, 0x38d8c2c4, 0x4fdff252, 0xd1bb67f1,
0xa6bc5767, 0x3fb506dd, 0x48b2364b, 0xd80d2bda, 0xaf0a1b4c,
0x36034af6, 0x41047a60, 0xdf60efc3, 0xa867df55, 0x316e8eef,
0x4669be79, 0xcb61b38c, 0xbc66831a, 0x256fd2a0, 0x5268e236,
```

```
 0xcc0c7795, 0xbb0b4703, 0x220216b9, 0x5505262f, 0xc5ba3bbe,
 0xb2bd0b28, 0x2bb45a92, 0x5cb36a04, 0xc2d7ffa7, 0xb5d0cf31,
 0x2cd99e8b, 0x5bdeae1d, 0x9b64c2b0, 0xec63f226, 0x756aa39c,
 0x026d930a, 0x9c0906a9, 0xeb0e363f, 0x72076785, 0x05005713,
 0x95bf4a82, 0xe2b87a14, 0x7bb12bae, 0x0cb61b38, 0x92d28e9b,
 0xe5d5be0d, 0x7cdcefb7, 0x0bdbdf21, 0x86d3d2d4, 0xf1d4e242,
 0x68ddb3f8, 0x1fda836e, 0x81be16cd, 0xf6b9265b, 0x6fb077e1,
 0x18b74777, 0x88085ae6, 0xff0f6a70, 0x66063bca, 0x11010b5c,
 0x8f659eff, 0xf862ae69, 0x616bffd3, 0x166ccf45, 0xa00ae278,
 0xd70dd2ee, 0x4e048354, 0x3903b3c2, 0xa7672661, 0xd06016f7,
 0x4969474d, 0x3e6e77db, 0xaed16a4a, 0xd9d65adc, 0x40df0b66,
 0x37d83bf0, 0xa9bcae53, 0xdebb9ec5, 0x47b2cf7f, 0x30b5ffe9,
 0xbdbdf21c, 0xcabac28a, 0x53b39330, 0x24b4a3a6, 0xbad03605,
 0xcdd70693, 0x54de5729, 0x23d967bf, 0xb3667a2e, 0xc4614ab8,
 0x5d681b02, 0x2a6f2b94, 0xb40bbe37, 0xc30c8ea1, 0x5a05df1b,
 0x2d02ef8d,
};
// crc32core: returns RFC3309 CRC value
unsigned crc32core(unsigned char *buf, int len){
    unsigned  crc = 0xffffffff;
    while(len--) {
        crc = (crc>>8) ^ crctab[(crc ^ *buf)&0xff];
        //printf("data: 0x%02x, crc: %08x\n", *buf, crc);
        buf++;
    }
    return crc;
}
// reflect: bit-reverse value, swap 0 for n, 1 for n-1 and so on
unsigned reflect(unsigned val, int nbits){
    unsigned ret = 0;
    int k;
    for (k=1; k < (nbits+1); k++){
```

```
        if (val & 1)
            ret |= 1 << (nbits-k);
        val >>= 1;
    }
    return ret;
}
//crc32hash: calculates DA's CRC and the appropriate bit in G/IADDR
void crc32hash (unsigned char *da, int len){
     unsigned *GADDR = (unsigned *) (0xFF700000 + 0x24880);
     unsigned *IADDR = (unsigned *) (0xFF700000 + 0x24800);
    unsigned crc = crc32core(da, len);
    unsigned lsb = reflect(crc, 8);     //bit-reverse 8 lsb
    unsigned bitIndex = lsb & 0x1f;     //least 5 bits for bit index
    unsigned regIndex = lsb >> 5;       //most 3 bits for register index
     if (da[0] & 0x01) {    //check for multicast group (I/G bit)
        GADDR[regIndex] |= 0x80000000 >> bitIndex;
     }
    else {
        IADDR[regIndex] |= 0x80000000 >> bitIndex;
     }
}
//main: calls crc32hash to compute DA's CRC and the corresponding I/GADDR bit
void main() {
     unsigned char da_pat1[] = { //example DA pattern
          0x01,0x00,0x5E,0x00,0x01,0x28,  //DA
     };
     crc32hash(da_pat1, 6);
}
```

4 Revision History

Table 1 provides a revision history for this application note.

Table 1. Document Revision History

Rev. No.	Date	Substantive Change(s)
0	07/29/04	Initial release.

THIS PAGE INTENTIONALLY LEFT BLANK

How to Reach Us:

USA/Europe/Locations Not Listed:
Freescale Semiconductor
Literature Distribution Center
P.O. Box 5405,
Denver, Colorado 80217
1-480-768-2130
(800) 521-6274

Japan:
Freescale Semiconductor Japan Ltd.
Technical Information Center
3-20-1, Minami-Azabu, Minato-ku
Tokyo 106-8573, Japan
81-3-3440-3569

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T. Hong Kong
852-26668334

Home Page:
www.freescale.com

AN2745
Rev. 0
07/2004