AN194

Application Note

HOW TO PROGRAM THE HASH TABLE FILTER IN THE CS8900A AND CS8920A

In some instances you may want to accept more addresses than a single individual address. Multicast addresses would be one use. You also need to have additional addresses when doing bridging functions.

You can accept additional addresses using the Hash Table Filter. The Hash Table Filter is a 64-bit register where each bit corresponds to the hashed value of an Ethernet address.

For example, you need to accept two individual addresses. You program the first address into the individual address register.

The second address requires a little more work. First you run the address through the same hash algorithm that the CS89xxA uses. Based on that result you set the correct bit in the Hash Table Filter.

Now you should know if this is a multicast address or just another individual address. If the additional address is a multicast address, you set the MulticastA bit in the RxCTL register. If the address is an individual address you set IAHash bit in the RxCTL register.

IMPORTANT: Each bit in the Hash Table Filter corresponds to many possible addresses. So, it is possible that once you set the right bits the chip will accept the frames you want and also frames you don’t want. This is because the some frames you don’t want have addresses that hash to the same bit as the address you want.

So, to ensure you only pass on the correct frames to your operating system or program you have to do an additional software check on the destination address. You do this by keeping a list of additional addresses in the software driver that you want to accept. When a frame comes in and the IAHash or Hashed status bits set in the Receive Event status register then you look for a match in your list. If the destination address has a match in your list then you would pass that frame on. If there is no match then it was just a random frame that happened to match a bit you set in the Hash Table Filter. You simply discard it.

Here is an algorithm that you can use to determine which bit to set in the CS89xxA hash table to receive a particular address

This algorithm can also be found by going to the drivers page for the CS8900A and downloading hash.zip.

```
#define BYTE unsigned char
#define WORD int
#define DWORD int long
#include <stdio.h>

BYTE CalculateHashIndex(BYTE *pMulticastAddr);
void main();
void updatecrc( int bit );


int crc_poly[] = {1,1,1,0,  1,1,0,1,
 1,0,1,1,  1,0,0,0,
                1,0,0,0,  0,0,1,1,
                0,0,1,0,  0,0,0,0
                },
     CRC[33];


void main ()
{

  BYTE hash_index;
  BYTE multicastaddr[5];

 /* just a made up address
    0x4d in byte 0 should hash to bit 63
    of hash filter.
    Changing the first byte to 0x85 will
    hash to bit 0 of the hash filter
    */
  multicastaddr[0] = 0x4d;
  multicastaddr[1] = 0x00;
  multicastaddr[2] = 0x00;
  multicastaddr[3] = 0x00;
  multicastaddr[4] = 0x00;
  multicastaddr[5] = 0x00;

  hash_index = CalculateHashIndex( multicastaddr );
  printf("%d\n",hash_index);
  (void) getchar();
```

```
}
/********************************************************************************
*
* CalculateHashIndex()
*
********************************************************************************/

BYTE CalculateHashIndex( BYTE *pMulticastAddr )
{
   BYTE  HashIndex;
   BYTE  AddrByte;
   int   Byte;
   int   Bit, j;

   /* Prime the CRC */
   for (j = 0; j < 32; j++ ) CRC[j] = 1;

   /* For each of the six bytes of the multicast address */
   for ( Byte=0; Byte<6; Byte++ )
   {
      /*
      printf("\n%2.2x", *pMulticastAddr);
      (void) getchar();
      */
      AddrByte = *pMulticastAddr++;

      /* For each bit of the byte */
      for ( Bit=0; Bit<8; Bit++ )
      {
         updatecrc( (AddrByte >> Bit) & 1 );
      }
   }

   /* Take the least significant six bits of the CRC and copy them */
   /* to the HashIndex in reverse order.                           */
   HashIndex = 0;
   for( Bit=0,HashIndex=0; Bit<6; Bit++ )
   {
      HashIndex = (HashIndex << 1) + CRC[Bit];
   }

   return HashIndex;

} /* end of CalculateHashIndex() */

void updatecrc( int bit )
{
int j;

   /* >> 1 the crc, use high bit (now CRC[32]) as control bit */
   for (j = 32; j > 0; j--) CRC[j] = CRC[j - 1];
   CRC[0] = 0;

   /* if bit ^ (control bit) = 1, set CRC = CRC ^ polynomial */
   if (bit ^ CRC[32])
   {
   for ( j = 0; j < 32; j++)
      {
         CRC[j] ^= crc_poly[j];
      }
   }
}
```

Here is a sample list of addresses and the corresponding hash bit. This is only a sample -- many addresses can hash to a particular bit.

Address	Hash Table Filter Bit
85 00 00 00 00 00	0
A5 00 00 00 00 00	1
E5 00 00 00 00 00	2
C5 00 00 00 00 00	3
45 00 00 00 00 00	4
65 00 00 00 00 00	5
25 00 00 00 00 00	6
05 00 00 00 00 00	7
2B 00 00 00 00 00	8
0B 00 00 00 00 00	9
4B 00 00 00 00 00	10
6B 00 00 00 00 00	11
EB 00 00 00 00 00	12
CB 00 00 00 00 00	13
8B 00 00 00 00 00	14
BB 00 00 00 00 00	15
C7 00 00 00 00 00	16
E7 00 00 00 00 00	17
A7 00 00 00 00 00	18
87 00 00 00 00 00	19
07 00 00 00 00 00	20
27 00 00 00 00 00	21
67 00 00 00 00 00	22
47 00 00 00 00 00	23
69 00 00 00 00 00	24
49 00 00 00 00 00	25
09 00 00 00 00 00	26
29 00 00 00 00 00	27
A9 00 00 00 00 00	28
89 00 00 00 00 00	29
C9 00 00 00 00 00	30
E9 00 00 00 00 00	31

Address	Hash Table Filter Bit
21 00 00 00 00 00	32
01 00 00 00 00 00	33
41 00 00 00 00 00	34
71 00 00 00 00 00	35
E1 00 00 00 00 00	36
C1 00 00 00 00 00	37
81 00 00 00 00 00	38
A1 00 00 00 00 00	39
8F 00 00 00 00 00	40
BF 00 00 00 00 00	41
EF 00 00 00 00 00	42
CF 00 00 00 00 00	43
4F 00 00 00 00 00	44
6F 00 00 00 00 00	45
2F 00 00 00 00 00	46
0F 00 00 00 00 00	47
63 00 00 00 00 00	48
43 00 00 00 00 00	49
03 00 00 00 00 00	50
23 00 00 00 00 00	51
A3 00 00 00 00 00	52
83 00 00 00 00 00	53
C3 00 00 00 00 00	54
E3 00 00 00 00 00	55
CD 00 00 00 00 00	56
ED 00 00 00 00 00	57
AD 00 00 00 00 00	58
8D 00 00 00 00 00	59
0D 00 00 00 00 00	60
2D 00 00 00 00 00	61
6D 00 00 00 00 00	62
4D 00 00 00 00 00	63

SMART Analog™