```
/*
 * This file contains an ECC algorithm from Toshiba that detects and
 * corrects 1 bit errors in a 256 byte block of data.
 *
 * drivers/mtd/nand/nand_ecc.c
 *
 * Copyright (C) 2000–2004 Steven J. Hill (sjhill@realitydiluted.com)
 *                                  Toshiba America Electronics Components, Inc.
 *
 * Copyright (C) 2006 Thomas Gleixner <tglx@linutronix.de>
 *
 * This file is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the
 * Free Software Foundation; either version 2 or (at your option) any
 * later version.
 *
 * This file is distributed in the hope that it will be useful, but WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Public License along
 * with this file; if not, write to the Free Software Foundation, Inc.,
 * 59 Temple Place, Suite 330, Boston, MA 02111–1307 USA.
 *
 * As a special exception, if other files instantiate templates or use
 * macros or inline functions from these files, or you compile these
 * files and link them with other works to produce a work based on these
 * files, these files do not by themselves cause the resulting work to be
 * covered by the GNU General Public License. However the source code for
 * these files must still be made available in accordance with section (3)
 * of the GNU General Public License.
 *
 * This exception does not invalidate any other reasons why a work based on
 * this file might be covered by the GNU General Public License.
 */

#include <linux/types.h>
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/mtd/nand_ecc.h>

/*
 * Pre–calculated 256–way 1 byte column parity
 */
static const u_char nand_ecc_precalc_table[] = {
        0x00, 0x55, 0x56, 0x03, 0x59, 0x0c, 0x0f, 0x5a, 0x5a, 0x0f, 0x0c, 0x59, 0x03, 0x56, 0x55, 0x00,
        0x65, 0x30, 0x33, 0x66, 0x3c, 0x69, 0x6a, 0x3f, 0x3f, 0x6a, 0x69, 0x3c, 0x66, 0x33, 0x30, 0x65,
        0x66, 0x33, 0x30, 0x65, 0x3f, 0x6a, 0x69, 0x3c, 0x3c, 0x69, 0x6a, 0x3f, 0x65, 0x30, 0x33, 0x66,
        0x03, 0x56, 0x55, 0x00, 0x5a, 0x0f, 0x0c, 0x59, 0x59, 0x0c, 0x0f, 0x5a, 0x00, 0x55, 0x56, 0x03,
        0x69, 0x3c, 0x3f, 0x6a, 0x30, 0x65, 0x66, 0x33, 0x33, 0x66, 0x65, 0x30, 0x6a, 0x3f, 0x3c, 0x69,
        0x0c, 0x59, 0x5a, 0x0f, 0x55, 0x00, 0x03, 0x56, 0x56, 0x03, 0x00, 0x55, 0x0f, 0x5a, 0x59, 0x0c,
        0x0f, 0x5a, 0x59, 0x0c, 0x56, 0x03, 0x00, 0x55, 0x55, 0x00, 0x03, 0x56, 0x0c, 0x59, 0x5a, 0x0f,
        0x6a, 0x3f, 0x3c, 0x69, 0x33, 0x66, 0x65, 0x30, 0x30, 0x65, 0x66, 0x33, 0x69, 0x3c, 0x3f, 0x6a,
        0x6a, 0x3f, 0x3c, 0x69, 0x33, 0x66, 0x65, 0x30, 0x30, 0x65, 0x66, 0x33, 0x69, 0x3c, 0x3f, 0x6a,
        0x0f, 0x5a, 0x59, 0x0c, 0x56, 0x03, 0x00, 0x55, 0x55, 0x00, 0x03, 0x56, 0x0c, 0x59, 0x5a, 0x0f,
        0x0c, 0x59, 0x5a, 0x0f, 0x55, 0x00, 0x03, 0x56, 0x56, 0x03, 0x00, 0x55, 0x0f, 0x5a, 0x59, 0x0c,
        0x69, 0x3c, 0x3f, 0x6a, 0x30, 0x65, 0x66, 0x33, 0x33, 0x66, 0x65, 0x30, 0x6a, 0x3f, 0x3c, 0x69,
        0x03, 0x56, 0x55, 0x00, 0x5a, 0x0f, 0x0c, 0x59, 0x59, 0x0c, 0x0f, 0x5a, 0x00, 0x55, 0x56, 0x03,
        0x66, 0x33, 0x30, 0x65, 0x3f, 0x6a, 0x69, 0x3c, 0x3c, 0x69, 0x6a, 0x3f, 0x65, 0x30, 0x33, 0x66,
        0x65, 0x30, 0x33, 0x66, 0x3c, 0x69, 0x6a, 0x3f, 0x3f, 0x6a, 0x69, 0x3c, 0x66, 0x33, 0x30, 0x65,
        0x00, 0x55, 0x56, 0x03, 0x59, 0x0c, 0x0f, 0x5a, 0x5a, 0x0f, 0x0c, 0x59, 0x03, 0x56, 0x55, 0x00
};
```

```
/**
 * nand_calculate_ecc - [NAND Interface] Calculate 3-byte ECC for 256-byte block
 * @mtd:        MTD block structure
 * @dat:        raw data
 * @ecc_code:   buffer for ECC
 */
int nand_calculate_ecc(struct mtd_info *mtd, const u_char *dat,
                       u_char *ecc_code)
{
        uint8_t idx, reg1, reg2, reg3, tmp1, tmp2;
        int i;

        /* Initialize variables */
        reg1 = reg2 = reg3 = 0;

        /* Build up column parity */
        for(i = 0; i < 256; i++) {
                /* Get CP0 - CP5 from table */
                idx = nand_ecc_precalc_table[*dat++];
                reg1 ^= (idx & 0x3f);

                /* All bit XOR = 1 ? */
                if (idx & 0x40) {
                        reg3 ^= (uint8_t) i;
                        reg2 ^= ~((uint8_t) i);
                }
        }

        /* Create non-inverted ECC code from line parity */
        tmp1 = (reg3 & 0x80) >> 0; /* B7 -> B7 */
        tmp1 |= (reg2 & 0x80) >> 1; /* B7 -> B6 */
        tmp1 |= (reg3 & 0x40) >> 1; /* B6 -> B5 */
        tmp1 |= (reg2 & 0x40) >> 2; /* B6 -> B4 */
        tmp1 |= (reg3 & 0x20) >> 2; /* B5 -> B3 */
        tmp1 |= (reg2 & 0x20) >> 3; /* B5 -> B2 */
        tmp1 |= (reg3 & 0x10) >> 3; /* B4 -> B1 */
        tmp1 |= (reg2 & 0x10) >> 4; /* B4 -> B0 */

        tmp2 = (reg3 & 0x08) << 4; /* B3 -> B7 */
        tmp2 |= (reg2 & 0x08) << 3; /* B3 -> B6 */
        tmp2 |= (reg3 & 0x04) << 3; /* B2 -> B5 */
        tmp2 |= (reg2 & 0x04) << 2; /* B2 -> B4 */
        tmp2 |= (reg3 & 0x02) << 2; /* B1 -> B3 */
        tmp2 |= (reg2 & 0x02) << 1; /* B1 -> B2 */
        tmp2 |= (reg3 & 0x01) << 1; /* B0 -> B1 */
        tmp2 |= (reg2 & 0x01) << 0; /* B7 -> B0 */

        /* Calculate final ECC code */
#ifdef CONFIG_MTD_NAND_ECC_SMC
        ecc_code[0] = ~tmp2;
        ecc_code[1] = ~tmp1;
#else
        ecc_code[0] = ~tmp1;
        ecc_code[1] = ~tmp2;
#endif
        ecc_code[2] = ((~reg1) << 2) | 0x03;

        return 0;
}
EXPORT_SYMBOL(nand_calculate_ecc);

static inline int countbits(uint32_t byte)
```

```
{
        int res = 0;

        for (;byte; byte >>= 1)
                res += byte & 0x01;
        return res;
}

/**
 * nand_correct_data - [NAND Interface] Detect and correct bit error(s)
 * @mtd:        MTD block structure
 * @dat:        raw data read from the chip
 * @read_ecc:   ECC from the chip
 * @calc_ecc:   the ECC calculated from raw data
 *
 * Detect and correct a 1 bit error for 256 byte block
 */
int nand_correct_data(struct mtd_info *mtd, u_char *dat,
                      u_char *read_ecc, u_char *calc_ecc)
{
        uint8_t s0, s1, s2;

#ifdef CONFIG_MTD_NAND_ECC_SMC
        s0 = calc_ecc[0] ^ read_ecc[0];
        s1 = calc_ecc[1] ^ read_ecc[1];
        s2 = calc_ecc[2] ^ read_ecc[2];
#else
        s1 = calc_ecc[0] ^ read_ecc[0];
        s0 = calc_ecc[1] ^ read_ecc[1];
        s2 = calc_ecc[2] ^ read_ecc[2];
#endif
        if ((s0 | s1 | s2) == 0)
                return 0;

        /* Check for a single bit error */
        if( ((s0 ^ (s0 >> 1)) & 0x55) == 0x55 &&
            ((s1 ^ (s1 >> 1)) & 0x55) == 0x55 &&
            ((s2 ^ (s2 >> 1)) & 0x54) == 0x54) {

                uint32_t byteoffs, bitnum;

                byteoffs = (s1 << 0) & 0x80;
                byteoffs |= (s1 << 1) & 0x40;
                byteoffs |= (s1 << 2) & 0x20;
                byteoffs |= (s1 << 3) & 0x10;

                byteoffs |= (s0 >> 4) & 0x08;
                byteoffs |= (s0 >> 3) & 0x04;
                byteoffs |= (s0 >> 2) & 0x02;
                byteoffs |= (s0 >> 1) & 0x01;

                bitnum = (s2 >> 5) & 0x04;
                bitnum |= (s2 >> 4) & 0x02;
                bitnum |= (s2 >> 3) & 0x01;

                dat[byteoffs] ^= (1 << bitnum);

                return 1;
        }

        if(countbits(s0 | ((uint32_t)s1 << 8) | ((uint32_t)s2 <<16)) == 1)
                return 1;
```

```
190             return -EBADMSG;
    }
    EXPORT_SYMBOL(nand_correct_data);

    MODULE_LICENSE("GPL");
195 MODULE_AUTHOR("Steven J. Hill <sjhill@realitydiluted.com>");
196 MODULE_DESCRIPTION("Generic NAND ECC support");
```