```
/* taken from Spansion S25FL032P data sheet */

/* read */
#define SPIF_CMD_READ            0x03
#define SPIF_CMD_FAST_READ       0x0B
#define SPIF_CMD_RDID            0x9F
#define SPIF_CMD_READ_ID 0x90

/* write control */
#define SPIF_CMD_WREN            0x06
#define SPIF_CMD_WRDI            0x04

/* erase */
#define SPIF_CMD_P4E             0x20    /* 4kB parameter sector
erase */
#define SPIF_CMD_P8E             0x40    /* 4kB parameter sector
erase */
#define SPIF_CMD_SE              0xD8    /* 64kB sector erase */
#define SPIF_CMD_BE              0x60    /* bulk erase */

/* program */
#define SPIF_CMD_PP              0x02    /* page program */

/* status & config */
#define SPIF_CMD_RDSR            0x05    /* read status register
*/
#define SPIF_CMD_WRR             0x01    /* write status + cfg
register */
#define SPIF_CMD_RCR             0x35    /* read config register
*/
#define SPIF_CMD_CLSR            0x30    /* reset the erase and
program fail flag */

/* power saving */
#define SPIF_CMD_DP              0xB9
#define SPIF_CMD_RES             0xAB

/* otp */
#define SPIF_CMD_OTPP            0x42    /* otp program */
#define SPIF_CMD_OTPR            0x4B    /* otp read */


static const AT91PS_SPI pSPI = AT91C_BASE_SPI;

void spiflash_write_protect(int on)
{
        if (on)
                AT91F_PIO_ClearOutput(AT91C_BASE_PIOA,
PIO_SPIF_nWP);
        else
                AT91F_PIO_SetOutput(AT91C_BASE_PIOA, PIO_SPIF_nWP);
}
```

```
#define SPI_PERIPHA (PIO_SPIF_SCK|PIO_SPIF_MOSI|PIO_SPIF_MISO|
PIO_SPIF_nCS)

static __ramfunc void spi_irq(void)
{
        u_int32_t status = pSPI->SPI_SR;

        AT91F_AIC_ClearIt(AT91C_BASE_AIC, AT91C_ID_SPI);
}

static const u_int8_t chipid_s25fl032p[3] = { 0x01, 0x02, 0x15 };

static u_int8_t chip_id[3];
static u_int32_t otp_supported;

void spiflash_init(void)
{
        DEBUGP("spiflash_init\r\n");

        /* activate and enable the write protection */
        AT91F_PIO_CfgPullupDis(AT91C_BASE_PIOA, PIO_SPIF_nWP);
        AT91F_PIO_CfgOutput(AT91C_BASE_PIOA, PIO_SPIF_nWP);
        spiflash_write_protect(1);

        /* Configure PIOs for SCK, MOSI, MISO and nCS */
        AT91F_PIO_CfgPeriph(AT91C_BASE_PIOA, SPI_PERIPHA, 0);

        AT91F_SPI_CfgPMC();
        /* Spansion flash in v1.0p only supprts Mode 3 or Mode 0 */
        /* Mode 3: CPOL=1 nCPHA=0 CSAAT=0 BITS=0(8) MCK/2 */
        AT91F_SPI_CfgCs(AT91C_BASE_SPI, 0, AT91C_SPI_CPOL |
                                           AT91C_SPI_BITS_8 |
                                           (64 << 8));

        /* SPI master mode, fixed CS, CS = 0 */
        AT91F_SPI_CfgMode(AT91C_BASE_SPI, AT91C_SPI_MSTR |
                                          AT91C_SPI_PS_FIXED |
                                          (0 << 16));

        /* configure interrupt controller for SPI IRQ */
        AT91F_AIC_ConfigureIt(AT91C_BASE_AIC, AT91C_ID_SPI,
                              OPENPCD_IRQ_PRIO_SPI,
                              AT91C_AIC_SRCTYPE_INT_HIGH_LEVEL,
&spi_irq);
        //AT91F_AIC_EnableIt(AT91C_BASE_AIC, AT91C_ID_SPI);

        /* Enable the SPI Controller */
        AT91F_SPI_Enable(AT91C_BASE_SPI);
        AT91F_SPI_EnableIt(AT91C_BASE_SPI, AT91C_SPI_MODF |
                                           AT91C_SPI_OVRES |
                                           AT91C_SPI_ENDRX |
                                           AT91C_SPI_ENDTX);

        spiflash_get_id(chip_id);
```

```
        if (!memcmp(chip_id, chipid_s25fl032p, sizeof(chip_id)))
                otp_supported = 1;
}

static int spi_transceive(const u_int8_t *tx_data, u_int16_t tx_len,
                  u_int8_t *rx_data, u_int16_t *rx_len, u_int16_t
rx_skip)
{
        u_int16_t tx_cur = 0;
        u_int16_t rx_len_max = 0;
        u_int16_t rx_cnt = 0;
        u_int8_t tmp;

        DEBUGPSPI("spi_transceive: enter(tx_len=%u) ", tx_len);

        if (rx_len) {
                rx_len_max = *rx_len;
                *rx_len = 0;
        }

        /* make sure we drain any remaining rx data */
        tmp = pSPI->SPI_RDR;

        //AT91F_SPI_Enable(pSPI);
        while (1) {
                u_int32_t sr = pSPI->SPI_SR;
                if (sr & AT91C_SPI_RDRF) {
                        tmp = pSPI->SPI_RDR;
                        rx_cnt++;
                        if (rx_cnt > rx_skip &&
                            rx_len && *rx_len < rx_len_max)
                                rx_data[(*rx_len)++] = tmp;
                }
                if (sr & AT91C_SPI_TDRE) {
                        if (tx_len > tx_cur)
                                pSPI->SPI_TDR = tx_data[tx_cur++];
                        else
                                pSPI->SPI_TDR = 0;
                }
                if (rx_len) {
                        if (*rx_len >= rx_len_max)
                                break;
                } else {
                        if (tx_len <= tx_cur)
                                break;
                }
        }
        /* make sure we drain any remaining rx data */
        tmp = pSPI->SPI_RDR;

        //AT91F_SPI_Disable(pSPI);
        if (rx_data) {
                int i;
```

```
                DEBUGPSPI("leave(): ");
                for (i = 0; i < *rx_len; i++)
                        DEBUGPSPI("0x%02x ", rx_data[i]);
                DEBUGPSPI("\r\n");
        } else
                DEBUGPSPI("leave()\r\n");

        return 0;
}

void spiflash_get_id(u_int8_t *id)
{
        const u_int8_t tx_data[] = { SPIF_CMD_RDID, 0, 0, 0 };
        u_int8_t rx_data[] = { 0,0,0,0 };
        u_int16_t rx_len = sizeof(rx_data);

        spi_transceive(tx_data, sizeof(tx_data), rx_data, &rx_len,
1);
        DEBUGPSPI("SPI ID: %02x %02x %02x\r\n",
                rx_data[0], rx_data[1], rx_data[2]);
        memcpy(id, rx_data, 3);
}

int spiflash_read_status(void)
{
        const u_int8_t tx_data[] = { SPIF_CMD_RDSR, 0 };
        u_int8_t rx_data[1];
        u_int16_t rx_len = sizeof(rx_data);

        spi_transceive(tx_data, sizeof(tx_data), rx_data, &rx_len,
1);

        DEBUGPSPI("SPI Flash status: 0x%02x\r\n", rx_data[0]);

        return rx_data[0];
}

void spiflash_clear_status(void)
{
        const u_int8_t tx_data[] = { SPIF_CMD_CLSR };

        spi_transceive(tx_data, sizeof(tx_data), NULL, 0, 0);
}

int spiflash_write_enable(int enable)
{
        u_int8_t tx_data[1];

        if (enable)
                tx_data[0] = SPIF_CMD_WREN;
        else
                tx_data[0] = SPIF_CMD_WRDI;

        spi_transceive(tx_data, sizeof(tx_data), NULL, 0, 0);
```

```
        return 0;
}

int spiflash_otp_read(u_int32_t otp_addr, u_int8_t *out, u_int16_t
rx_len)
{
        u_int8_t tx_data[] = { SPIF_CMD_OTPR, 0, 0, 0, 0 };

        if (!otp_supported) {
                DEBUGP("OTP not supported!\r\n");
                return -1;
        }

        tx_data[1] = (otp_addr >> 16) & 0xFF;
        tx_data[2] = (otp_addr >> 8) & 0xFF;
        tx_data[3] = (otp_addr) & 0xFF;

        spi_transceive(tx_data, sizeof(tx_data), out, &rx_len, 5);

        DEBUGPSPI("OTP READ(0x%x): ", otp_addr);
        int i;
        for (i = 0; i < rx_len; i++)
                DEBUGPSPI("%02x ", out[i]);
        DEBUGPSPI("\r\n");

        return rx_len;
}

int spiflash_otp_write(u_int32_t otp_addr, u_int8_t data)
{
        u_int8_t tx_data[] = { SPIF_CMD_OTPP, 0, 0, 0, 0 };

        if (!otp_supported) {
                DEBUGP("OTP not supported!\r\n");
                return -1;
        }

        tx_data[1] = (otp_addr >> 16) & 0xFF;
        tx_data[2] = (otp_addr >> 8) & 0xFF;
        tx_data[3] = (otp_addr) & 0xFF;
        tx_data[4] = data;

        spiflash_write_enable(1);
        spi_transceive(tx_data, sizeof(tx_data), NULL, 0, 0);
        spiflash_write_enable(0);

        if (spiflash_read_status() & (1 << 6))
                return -1;

        return 0;
}

static int otp_region2addr(u_int8_t region)
```

```
{
        /* see Figure 10.1 of S25FL032P data sheet */
        if (region > 31 || region < 1)
                return -1;
        else if (region > 24)
                return 0x215;
        else if (region > 16)
                return 0x214;
        else if (region > 8)
                return 0x113;
        else
                return 0x112;
}

static int otp_region2bit(u_int8_t region)
{
        /* see Figure 10.1 of S25FL032P data sheet */
        if (region > 31 || region < 1)
                return -1;
        else if (region > 24)
                return region - 25;
        else if (region > 16)
                return region - 17;
        else if (region > 8)
                return region - 9;
        else
                return region - 1;
}

int spiflash_otp_get_lock(u_int8_t region)
{
        u_int32_t addr;
        u_int8_t bit, data;

        if (region > 31 || region < 1)
                return -1;

        bit = otp_region2bit(region);
        addr = otp_region2addr(region);

        spiflash_otp_read(addr, &data, 1);

        if (!(data & (1 << bit)))
                return 1;
        else
                return 0;
}

int spiflash_otp_set_lock(u_int8_t region)
{
        u_int32_t addr;
        u_int8_t bit;

        if (region > 31 || region < 1)
```

```
                return -1;

        bit = otp_region2bit(region);
        addr = otp_region2addr(region);

        /* clear the respective bit */
        return spiflash_otp_write(addr, ~(1 << bit));
}
```