Sfud 移植

Contents

1、 介绍

SFUD 是一款开源的串行 SPI Flash 通用驱动库。由于现有市面的串行 Flash 种类居多，各个 Flash 的规格及命令存在差异， SFUD 就是为了解决这些 Flash 的差异现状而设计，让我们的产品能够支持不同品牌及规格的 Flash，提高了涉及到 Flash 功能的软件的可重用性及可扩展性，同时也可以规避 Flash 缺货或停产给产品所带来的风险。

暂时支持的芯片

型号	制造商	容量	最高速度	SFDP标准	QSPI 模式	备注
型号	制造商	容量	最高速度	SFDP标准	Qspi模式	备注
W25Q40BV	Winbond	4Mb	50Mhz	不支持	双线	已停产
W25Q80DV	Winbond	8Mb	104Mhz	支持	双线	
W25Q16BV	Winbond	16Mb	104Mhz	不支持	双线	by slipperstree
W25Q16CV	Winbond	16Mb	104Mhz	支持	未测试	
W25Q16DV	Winbond	16Mb	104Mhz	支持	未测试	by slipperstree
W25Q32BV	Winbond	32Mb	104Mhz	支持	双线	
W25Q64CV	Winbond	64Mb	80Mhz	支持	四线	
W25Q128BV	Winbond	128Mb	104Mhz	支持	四线	
W25Q256FV	Winbond	256Mb	104Mhz	支持	四线	
MX25L3206E	Macronix	32Mb	86MHz	支持	双线	
KH25L4006E	Macronix	4Mb	86Mhz	支持	未测试	by JiapengLi
KH25L3206E	Macronix	32Mb	86Mhz	支持	双线	
SST25VF016	Microchip	16Mb	50MHz	不支	不支	SST 已被 Microchip

B				持	持	收购
M25P40	Micron	4Mb	75Mhz	不支持	未测试	by redocCheng
M25P80	Micron	8Mb	75Mhz	不支持	未测试	by redocCheng
M25P32	Micron	32Mb	75Mhz	不支持	不支持	
EN25Q32B	EON	32Mb	104MHz	不支持	未测试	
GD25Q16B	GigaDevice	16Mb	120Mhz	不支持	未测试	by TanekLiang
GD25Q64B	GigaDevice	64Mb	120Mhz	不支持	双线	
S25FL216K	Cypress	16Mb	65Mhz	不支持	双线	
S25FL032P	Cypress	32Mb	104Mhz	不支持	未测试	by yc_911
S25FL164K	Cypress	64Mb	108Mhz	支持	未测试	
A25L080	AMIC	8Mb	100Mhz	不支持	双线	
A25LQ64	AMIC	64Mb	104Mhz	支持	支持	
F25L004	ESMT	4Mb	100Mhz	不支持	不支持	

PCT25VF016B	PCT	16Mb	80Mhz	不支持	不支持	SST 授权许可，会被识别为 SST25VF016B
AT45DB161E	ADESTO	16Mb	85MHz	不支持	不支持	ADESTO 收购 Atmel 串行闪存产品线

2、 移植过程

我们使用 FRDM-K64 板子，w25q32，将 sfud 移植到 k64 上。
硬件连接

FRDM-k64 引脚	w25q32 引脚
PTD0（复用为 GPIO）	CS
PTD1（SCK）	SCK
PTD2（MISO）	SOUT
PTD3（MOSI）	SIN

1、 下载 sfud 源码 https://github.com/armink/SFUD
2、 我们要使用的就是 sfud 这个文件夹。我们使用 k64 库函数的 dspi 的 master 的 polling 例程来进行移植。将他添加到工程中，添加头文件路径

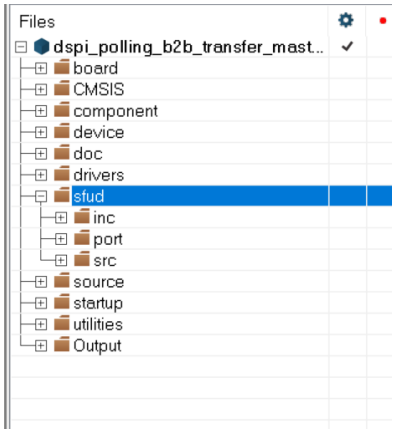

图 1.添加文件

3、 在 sfud_cfg.h 添加 SFUD_USING_FLASH_INFO_TABLE 的索引设备，SFUD_FLASH_DEVICE_TABLE 设置索引名称等基本信息

```
enum {
    SFUD_W25Q32_DEVICE_INDEX = 0,
};

#define SFUD_FLASH_DEVICE_TABLE
{
    [SFUD_W25Q32_DEVICE_INDEX] = {.name = "w25q32", .spi.name = "SPI0"},
}
```

图 2.定义设备信息

4、sfud_port.c 中，sfud_spi_port_init 添加设备 case，然后对应初始化其 IO 引脚，时钟，GPIO，片选引脚配置为 GPIO，不要配置为 SPI 的 cs，初始化 spi。我们在 main 函数里一开始调用 sfud_init 函数。这个函数实现就是获取 SFUD_FLASH_DEVICE_TABLE 中设备，然后根据索引值去初始化对应的设备。

sfud_spi_port_init 需要自己去实现的函数有，spi 时钟配置函数，引脚配置函数，spi 配置函数，最重要的是实现 spi_write_read 函数，他决定了 mcu 是否能正确读取数据。spi_lock，和 spi_unlock 实现就是开关中断。还要定义一个用户数据配置，他保存了一些 spi 基本信息，使用的是哪个 spi，片选 gpio 的基地址，片选信号是第几脚

```
static spi_user_data spi0 = { .spix = SPI0, .cs_gpiox = GPIOD, .cs_gpio_pin = PIN0_IDX };

sfud_err sfud_spi_port_init(sfud_flash *flash) {
    sfud_err result = SFUD_SUCCESS;

     */
    switch (flash->index)
    {
    case SFUD_W25Q32_DEVICE_INDEX:
      {
        spi_clock_configuration(&spi0);
        port_configuration(&spi0);
        gpio_pin_config_t cs_config = {kGPIO_DigitalOutput, 1,};
        GPIO_PinInit (GPIOD,0,&cs_config);
        spi_configuration(&spi0);

        flash->spi.wr = spi_write_read;
        flash->spi.lock = spi_lock;
        flash->spi.unlock = spi_unlock;
        flash->spi.user_data = &spi0;
        /* about 100 microsecond delay */
        flash->retry.delay = retry_delay_100us;
        /* adout 60 seconds timeout */
        flash->retry.times = 60 * 10000;
      }
      break;
    }
    return result;
}
```

图 3. sfud_spi_port_init 实现以及用户数据的定义

```
static void spi_clock_configuration(spi_user_data_t spi)
{
    if(spi->spix == SPI0)
    {
      CLOCK_EnableClock(kCLOCK_PortD);
    }
    else if(spi->spix == SPI1)
    {
    }

}
static void port_configuration(spi_user_data_t spi)
{
    if(spi->spix == SPI0)
    {
      PORT_SetPinMux(PORTD, PIN0_IDX, kPORT_MuxAsGpio);          /* PORTD0 (pin 93) is configured as SPI0_PCS0 */
      PORT_SetPinMux(PORTD, PIN1_IDX, kPORT_MuxAlt2);           /* PORTD1 (pin 94) is configured as SPI0_SCK */
      PORT_SetPinMux(PORTD, PIN2_IDX, kPORT_MuxAlt2);           /* PORTD2 (pin 95) is configured as SPI0_SOUT */
      PORT_SetPinMux(PORTD, PIN3_IDX, kPORT_MuxAlt2);           /* PORTD3 (pin 96) is configured as SPI0_SIN */
    }
    else if(spi->spix == SPI1)
    {
    }
}
```

图 4.spi 以及 port 初始化

```
static void spi_configuration(spi_user_data_t spi)
{

      if(spi->spix == SPI0)
      {
        dspi_master_config_t masterConfig;

        uint32_t srcClock_Hz;
        /* Master config */
        masterConfig.whichCtar                                = kDSPI_Ctar0;
        masterConfig.ctarConfig.baudRate                      = TRANSFER_BAUDRATE;
        masterConfig.ctarConfig.bitsPerFrame                  = 8U;
        masterConfig.ctarConfig.cpol                          = kDSPI_ClockPolarityActiveHigh;
        masterConfig.ctarConfig.cpha                          = kDSPI_ClockPhaseFirstEdge;
        masterConfig.ctarConfig.direction                     = kDSPI_MsbFirst;
        masterConfig.ctarConfig.pcsToSckDelayInNanoSec        = 1000000000U / TRANSFER_BAUDRATE;
        masterConfig.ctarConfig.lastSckToPcsDelayInNanoSec    = 1000000000U / TRANSFER_BAUDRATE;
        masterConfig.ctarConfig.betweenTransferDelayInNanoSec = 1000000000U / TRANSFER_BAUDRATE;

        masterConfig.whichPcs           = EXAMPLE_DSPI_MASTER_PCS_FOR_INIT;
        masterConfig.pcsActiveHighOrLow = kDSPI_PcsActiveLow;

        masterConfig.enableContinuousSCK        = false;
        masterConfig.enableRxFifoOverWrite      = false;
        masterConfig.enableModifiedTimingFormat = false;
        masterConfig.samplePoint                = kDSPI_SckToSin0Clock;

        srcClock_Hz = DSPI_MASTER_CLK_FREQ;
        DSPI_MasterInit(EXAMPLE_DSPI_MASTER_BASEADDR, &masterConfig, srcClock_Hz);

      }
      else if(spi->spix == SPI1)
      {

      }
}
```

图 5.spi 配置

```
static void spi_lock(const sfud_spi *spi)
{
    __disable_irq();
}

static void spi_unlock(const sfud_spi *spi)
{
    __enable_irq();
}
```

图 6.spi 锁配置

在实现 spi_write_read 函数之前，先要实现 spi 传输配置函数。

```
static void SPI_Transfer(SPI_Type* SPIx, uint8_t* masterTxData, uint8_t* masterRxData, uint8_t size)
{
    dspi_transfer_t masterXfer;
    masterXfer.txData      = masterTxData;
    masterXfer.rxData      = masterRxData;
    masterXfer.dataSize    = size;
    masterXfer.configFlags = kDSPI_MasterPcsContinuous; //kDSPI_MasterCtar0 | EXAMPLE_DSPI_MASTER_PCS_FOR_TRANSFER |
    DSPI_MasterTransferBlocking(SPIx, &masterXfer);
}


static uint8_t SPI_ReadWriteByte(SPI_Type* SPIx, uint8_t inData)
{
    uint8_t a = inData,b=0x00;
    SPI_Transfer(SPIx,&a,&b,1);
    return b;
}
```

图 7.spi 传输配置函数

```
static sfud_err spi_write_read(const sfud_spi *spi, const uint8_t *write_buf, size_t write_size, uint8_t *read_buf,
        size_t read_size)
{
#if 1
    sfud_err result = SFUD_SUCCESS;
    uint8_t send_data;
    spi_user_data_t spi_dev = (spi_user_data_t) spi->user_data;

    if (write_size)
    {
        SFUD_ASSERT(write_buf);
    }
    if (read_size)
    {
        SFUD_ASSERT(read_buf);
    }

    GPIO_PinWrite(spi_dev->cs_gpiox, spi_dev->cs_gpio_pin,0);
    /* 开始读写数据 */
    for (size_t i = 0; i < write_size + read_size; i++)
    {
        /* 先写缓冲区中的数据到 SPI 总线，数据写完后，再写 dummy(0xFF) 到 SPI 总线 */
        if (i < write_size)
        {
            send_data = *write_buf++;
            SPI_ReadWriteByte(spi_dev->spix,send_data);
        }
        else
        {
            send_data = SFUD_DUMMY_DATA;
            *read_buf++ = SPI_ReadWriteByte(spi_dev->spix,send_data);
        }
    }
    GPIO_PinWrite(spi_dev->cs_gpiox, spi_dev->cs_gpio_pin,1);
#endif
    return result;
}
```

图 8.spi_write_read 实现

5、 sfud_log_debug，sfud_log_info 里的串口打印函数要修改。

修改完以后，在 main 函数调用 sfud_init。

3、 程序效果

如果成功的话，串口会打印以下数据。只要红线数值能正常被读出来，那么基本就移植成功了。红线就是工厂 id，内存类型，容量 id，每个 flash 存储都不一样，如果读出来都是 0xff 那就有问题，得检查 io 配置是否正确，读写函数是否正常。在下载的文件 demo 里还有 sfud_demo 来测试 flash，直接复制进来用，不需要修改。完整效果如下

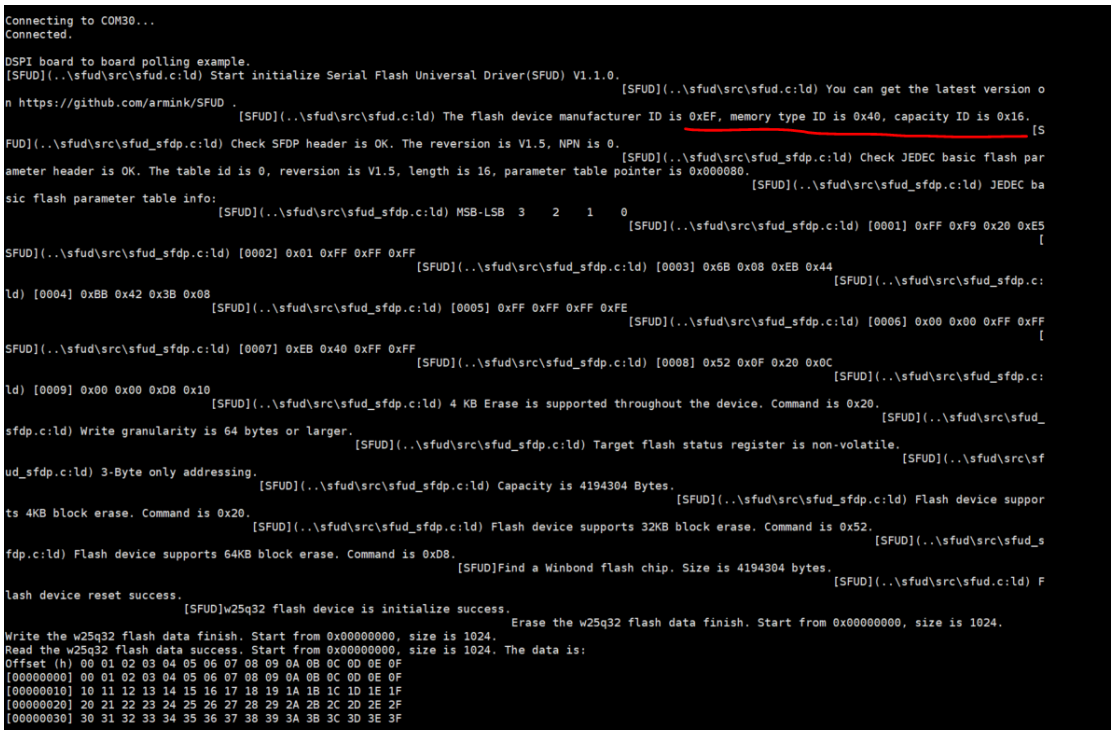

图 8.运行效果 1

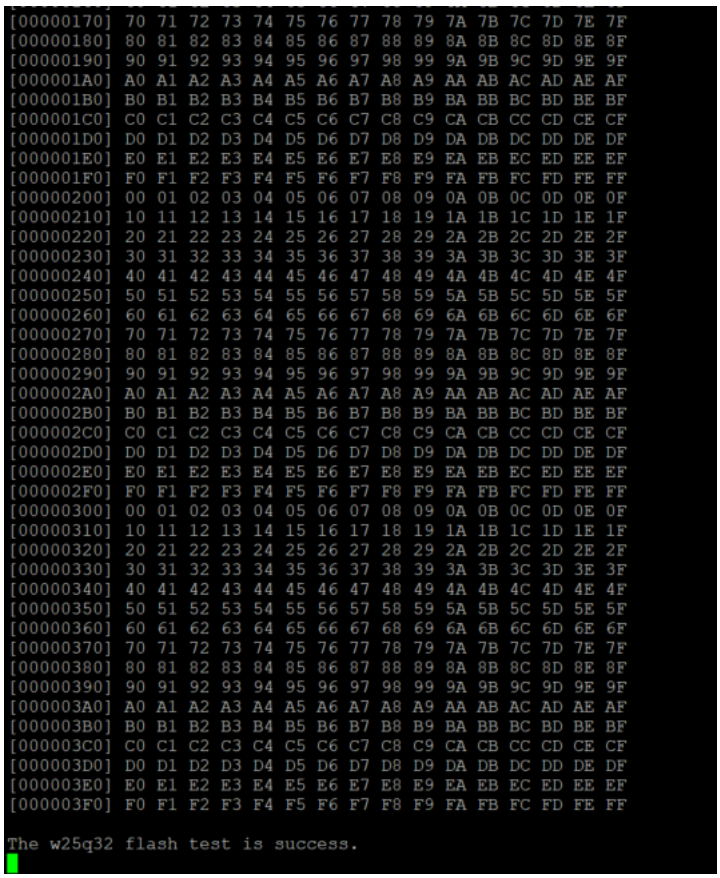

图 9.运行效果 2

4、参考

1）SFUD