ARC

SD

支持	www.armrunc.com http://armrunc.taobao.com

Change History

版本号	更改描述	作者	修改时间
1.0.0	第一版	armrunc	2012-3-7

目录

第14章 SD

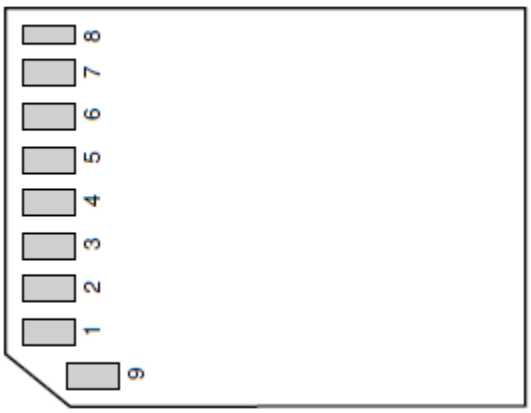
14.1 SD 简介

SD 卡（Secure Digital Memory Card）是一种基于半导体快闪记忆器的新一代记忆设备，它在小型消费类市场上越来越流行，例如数码相机、个人数码助理(PDA)和多媒体播放器等。SD Card 体积只有一般邮票大小，重量轻约 2g ，携带方便，兼容性佳，适用于各类电子商品。 SD Card 是由 SanDisk 、Matsushita Electronic (松下电器) 与 Toshiba (东芝) 根据 MMC 为基础所联合开发。因此几乎所有使用 SD 规格的存取设备都对 MultiMedia Card 具有兼容性。

14.1.1 SD 卡电气接口

SD 卡的电气接口如下：

Pin	Name	Function (SD Mode)	Function (SPI Mode)
1	DAT3/CS	Data Line 3	Chip Select/Slave Select (SS)
2	CMD/DI	Command Line	Master Out Slave In (MOSI)
3	VSS1	Ground	Ground
4	VDD	Supply Voltage	Supply Voltage
5	CLK	Clock	Clock (SCK)
6	VSS2	Ground	Ground
7	DAT0/DO	Data Line 0	Master In Slave Out (MISO)
8	DAT1/IRQ	Data Line 1	Unused or IRQ
9	DAT2/NC	Data Line 2	Unused



本章介绍 SPI 模式的使用。

14.2 SD 应用实例 ----- SD 卡类型判断及块读取

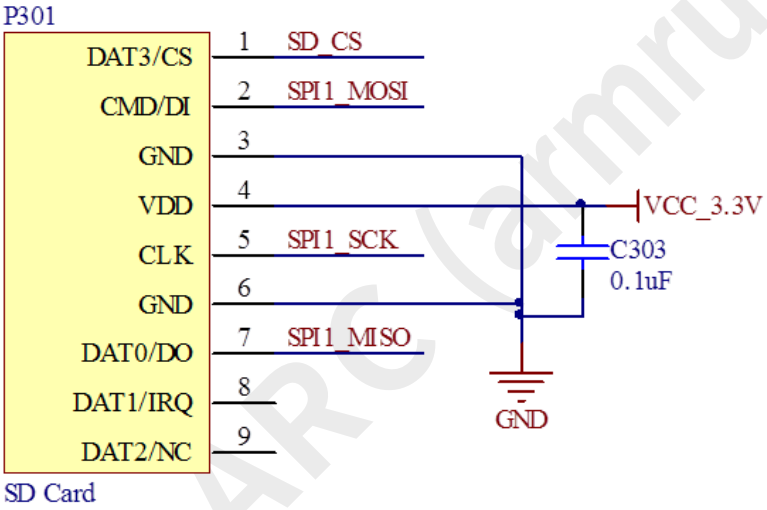
14.2.1 实例描述

本实例将 SD 卡初始化为 SPI 模式，判断 SD 卡类型，随后读取一块内容并通过串口显示。

14.2.2 硬件设计

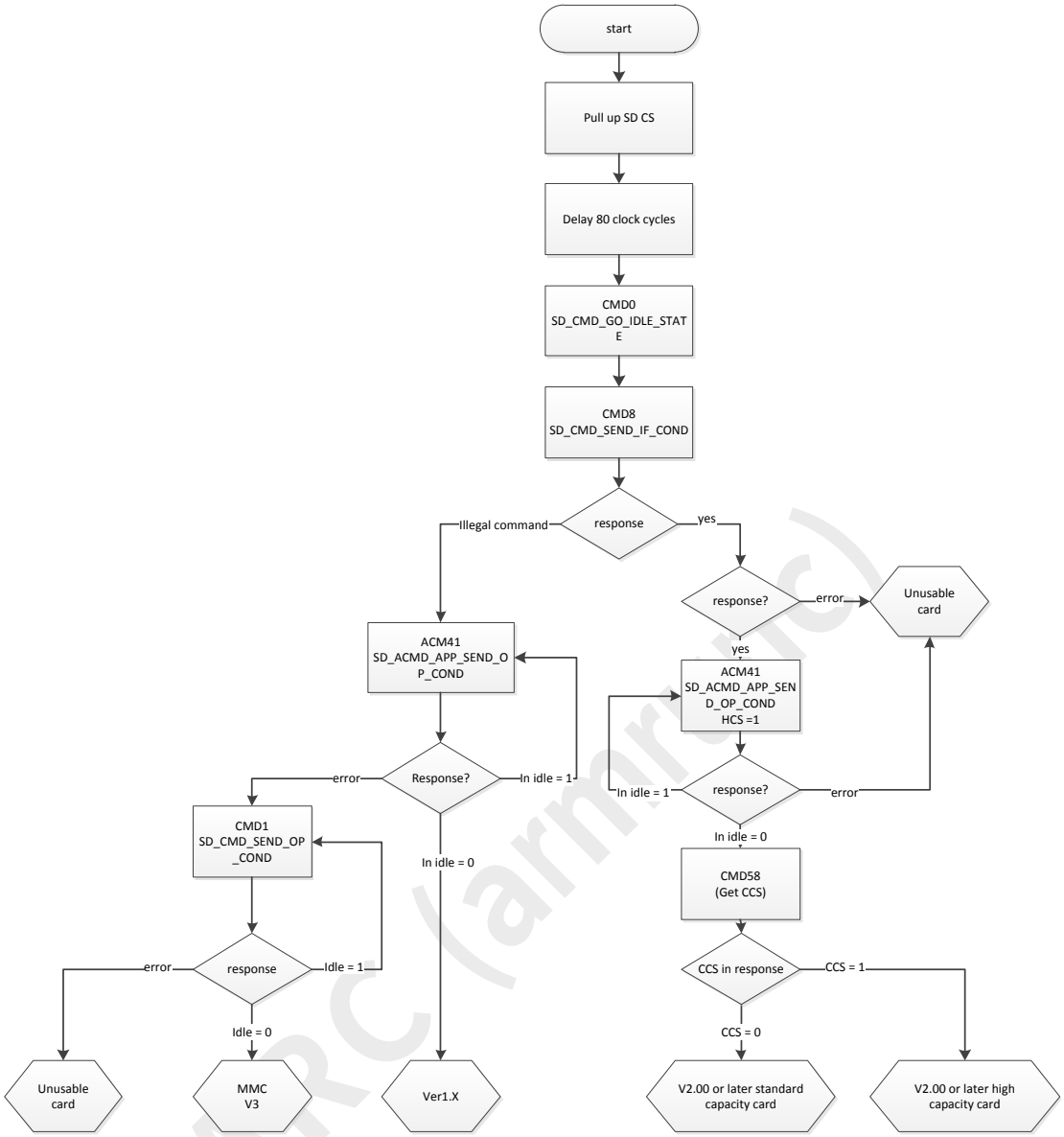
SD 卡 GPIO 分配和硬件电路图如下，SD 卡和 SPI flash 占用同一个 SPI1。

SD_CS	PA8
SPI1_SCK	PA5
SPI1_MISO	PA6
SPI1_MOSI	PA7



14.2.3 软件设计

首先初始化 SysTick 用于精确延迟，然后初始化串口用来输出 SD 卡数据。再初始化 SD 卡，SD 卡被初始化为 SPI 模式，在初始化函数内，我们可以得到该卡的类型。SD 卡初始化完成以后，我们读取 SD 的一个块的数据，通过串口显示出来。下面主要介绍 SD 卡如何进入 SPI 模式，请参考下面的流程图，具体实现代码请阅读函数 ARC_SD_Start()。



该实例的主要代码如下:

文件 SD_main.c：

```
/**
   * @brief   Main program, SD card SPI interface read example.
   * @param   None
   * @retval None
   */
int main(void)
{
      uint16_t i;
      uint8_t resp;
      uint8_t sd_response[5];
```

```
    uint8_t RxData[512];
    SD_Card_Type SD_CT = SD_Unknown;
    ARC_SysTick_Init();
    ARC_COM_Init();
    USART_Cmd(USART1, ENABLE);

    ARC_SD_SPI_Init();

    SPI_Cmd(SPI1, ENABLE); /*!< SD_SPI enable */

    ARC_DMA1_RCC_Init();

    for (i = 0; i < 3; i++)
    {
        SD_CT = ARC_SD_SPI_Start();
        if (SD_CT != SD_Unknown)
            break;
    }

    if(SD_CT == SD_Unknown)
    {
        printf("Unknown SD card or SD card not present\n");
    }
    else
    {
        printf("SD card type: 0x%X\n", (int)SD_CT);
    }

    while (1)
    {
        resp = ARC_sd_send_command(SD_CMD_READ_SINGLE_BLOCK,
0, R1, sd_response);
        if (resp == 0) /* READ_SINGLE_BLOCK */
        {
            resp = ARC_SD_SPI_ReadBlock(RxData, 512);
            if (resp)
            {
                for(i = 0; i < 512; i++)
                    printf("0x%2X ", RxData[i]);
                printf("\n");
            }
            else
            {
                printf("Failed to read block\n");
```

```
                }
            }
            else
            {
                    printf("Failed to send command\n");
            }
            ARC_SysTick_Delay(1000);
        }
}
```

文件 ARC_SD.c：

```
static SD_Card_Info SDCardInfo;
/**
   * @brief    get the pointer to SD card information struct.
   * @param    None
   * @retval pointer to the struct.
   */
SD_Card_Info * ARC_SD_SPI_GetCardInof(void)
{
      return &SDCardInfo;
}

/**
   * @brief    Wait for the SD SPI bus ready.
   * @param    None
   * @retval None
   */
__inline uint8_t ARC_SD_CSReady(void)
{
      uint8_t i = 0;
      uint16_t tmp = 0;
      do
      {
            tmp = ARC_SPI_SendByte(SPI1, SD_DUMMY_BYTE);

      }while(tmp != 0xFF && tmp != 0x00 && ++i < 0xFFFE);
      if(i >= 0xFFFE)
            return 0;
      return 1;
}

/**
   * @brief    Wait for card is not busy.
   * @param    None
```

```
  * @retval None
  */
__inline void ARC_SD_CardBusy(void)
{
    while(ARC_SPI_SendByte(SPI1, SD_DUMMY_BYTE) == 0);
}

/**
  * @brief   Send 6 bytes command to the SD card.
  * @param   Cmd: The command send to SD card.
  * @param   argument: The command argument.
  * @param   response_type: the SPI command response type.
  * @param   *response: the SPI response returned.
  * @retval The SD Response.
  */
uint8_t ARC_sd_send_command(uint8_t cmd, uint32_t argument,
                                    SD_Response response_type, uint8_t
*response)
{
    int32_t i = 0;
    uint8_t crc = 0x01;
    int8_t response_length = 0;
    uint8_t tmp;
    uint8_t Frame[6];

    if (cmd & 0x80)   /* Send a CMD55 prior to ACMD<n> */
    {
        cmd &= 0x7F;
        ARC_sd_send_command(SD_CMD_APP_CMD, 0, R1, response);
        if (response[0] > 0x01)
        {
            ARC_SD_CS_HIGH();
            ARC_SPI_SendByte(SPI1, SD_DUMMY_BYTE);
            return response[0];
        }
    }
    ARC_SD_CS_LOW();
    while(!ARC_SD_CSReady());
    if(cmd == SD_CMD_GO_IDLE_STATE)
        crc = 0x95;

    if(cmd == SD_CMD_SEND_IF_COND)
        crc = 0x87;
```

```
/* All data is sent MSB first, and MSb first */
/* cmd Format:
cmd[7:6] : 01
cmd[5:0] : command */

Frame[0] = ((cmd & 0x3F) | 0x40); /*!< Construct byte 1 */

Frame[1] = (uint8_t)(argument >> 24); /*!< Construct byte 2 */

Frame[2] = (uint8_t)(argument >> 16); /*!< Construct byte 3 */

Frame[3] = (uint8_t)(argument >> 8); /*!< Construct byte 4 */

Frame[4] = (uint8_t)(argument); /*!< Construct byte 5 */

Frame[5] = (uint8_t)(crc); /*!< Construct CRC: byte 6 */

for (i = 0; i < 6; i++)
{
    ARC_SPI_SendByte(SPI1, Frame[i]); /*!< Send the Cmd bytes */
}

switch (response_type)
{
    case R1:
    case R1B:
        response_length = 1;
        break;
    case R2:
        response_length = 2;
        break;
    case R3:
    case R7:
        response_length = 5;
        break;
    default:
        break;
}

/* Wait for a response. A response can be recognized by the start bit (a
zero) */
i = 0xFF;
do
{
```

```
        tmp = ARC_SPI_SendByte(SPI1, SD_DUMMY_BYTE);
    }while ((tmp & 0x80) && --i);

    response[0] = tmp;

    /* Just bail if we never got a response */
    if ((i > 0) && ((response[0] & SD_ILLEGAL_COMMAND) !=
SD_ILLEGAL_COMMAND))
    {
        i = 1;
        while(i < response_length)
        {
            tmp = ARC_SPI_SendByte(SPI1, SD_DUMMY_BYTE);
            response[i] = tmp;
            i++;
        }

        /* If the response is a "busy" type (R1B), then there's some
          * special handling that needs to be done. The card will
          * output a continuous stream of zeros, so the end of the BUSY
          * state is signaled by any nonzero response. The bus idles
          * high.
          */
        if (response_type == R1B)
        {
            do
            {
                tmp = ARC_SPI_SendByte(SPI1, SD_DUMMY_BYTE);;
            }while (tmp != 0xFF);

            ARC_SPI_SendByte(SPI1, SD_DUMMY_BYTE);
        }
    }

    ARC_SD_CS_HIGH();
    ARC_SPI_SendByte(SPI1, SD_DUMMY_BYTE);
    return response[0];
}

/**
  * @brief   send buffer to SPI via DMA.
  * @param   *buff, the memory address to be sent to the SPI device
  * @param   byteTransfer, the number to be sent to the SPI device
  * @retval None
```

```
   */
void ARC_SD_SPI_DMASend(const uint8_t *buff, uint32_t byteTransfer)
{
      ARC_DMA1_Ch3_Param_Init(buff, byteTransfer,
DMA_MemoryInc_Enable);

      /* Enable SPI_MASTER DMA Tx request */
      SPI_I2S_DMACmd(SPI1, SPI_I2S_DMAReq_Tx, ENABLE);

      /* Enable DMA channels */
      DMA_Cmd(DMA1_Channel3, ENABLE);

      /* Transfer complete */
      while(!DMA_GetFlagStatus(DMA1_FLAG_TC3));

      /* Enable DMA channels */
      DMA_Cmd(DMA1_Channel3, DISABLE);

      /* Disable SPI_MASTER DMA Tx request */
      SPI_I2S_DMACmd(SPI1, SPI_I2S_DMAReq_Tx, DISABLE);
}

/**
   * @brief   read buffer from SPI device via DMA.
   * @param   *buff, the memory address holding the content from SPI device
   * @param   byteTransfer, the number to be read from the SPI device
   * @retval None
   */
void ARC_SD_SPI_DMAReceive(uint8_t *buff, uint32_t byteTransfer)
{
      uint8_t dummyByte = SD_DUMMY_BYTE;
      ARC_DMA1_Ch2_Param_Init(buff, byteTransfer);
      ARC_DMA1_Ch3_Param_Init(&dummyByte, byteTransfer,
DMA_MemoryInc_Disable);

      /* Enable SPI_MASTER DMA Rx request */
      SPI_I2S_DMACmd(SPI1, SPI_I2S_DMAReq_Rx, ENABLE);
      /* Enable SPI_MASTER DMA Tx request */
      SPI_I2S_DMACmd(SPI1, SPI_I2S_DMAReq_Tx, ENABLE);

      /* Enable DMA channels */
      DMA_Cmd(DMA1_Channel2, ENABLE);
      /* Enable DMA channels */
      DMA_Cmd(DMA1_Channel3, ENABLE);
```

```
	/* Transfer complete */
	while(!DMA_GetFlagStatus(DMA1_FLAG_TC2));

	/* Disable DMA channels */
	DMA_Cmd(DMA1_Channel2, DISABLE);
	/* Enable DMA channels */
	DMA_Cmd(DMA1_Channel3, DISABLE);

	/* Disable SPI_MASTER DMA Rx request */
	SPI_I2S_DMACmd(SPI1, SPI_I2S_DMAReq_Rx, DISABLE);
	/* Disable SPI_MASTER DMA Tx request */
	SPI_I2S_DMACmd(SPI1, SPI_I2S_DMAReq_Tx, DISABLE);
}

/**
  * @brief   read a block from SPI device via DMA.
  * @param   *buff, the memory address holding the content from SPI device
  * @param   byteTransfer, the number to be read from the SPI device
  * @retval None
  */
uint8_t ARC_SD_SPI_ReadBlock(uint8_t *buff, uint32_t byteTransfer)
{
	uint16_t expire_count = 0xFFFF;
	uint8_t token, ret = 1;
	ARC_SD_CS_LOW();
	do
	{
		token = ARC_SPI_SendByte(SPI1, SD_DUMMY_BYTE);
	}while(token != 0xFE && --expire_count);
	if(token != 0xFE)
	{
		ret = 0;
	}
	else
	{
		#if 1
		ARC_SD_SPI_DMAReceive(buff, byteTransfer);
		#else
		while(byteTransfer--)
		{
			*buff = ARC_SPI_SendByte(SPI1, SD_DUMMY_BYTE);
			buff++;
		}
```

```
            #endif
            ARC_SPI_SendByte(SPI1, SD_DUMMY_BYTE);
            ARC_SPI_SendByte(SPI1, SD_DUMMY_BYTE);
        }
        ARC_SD_CS_HIGH();
        return ret;
}

/**
   * @brief   send a block to SPI via DMA.
   * @param   *buff, the memory address to be sent to the SPI device
   * @param   token
   * @retval successful or not
   */
uint8_t ARC_SD_SPI_WriteBlock(const uint8_t *buff, uint8_t token)
{
        uint8_t resp, ret = 1;
        #if 0
        uint16_t i = 0;
        #endif
        ARC_SD_CS_LOW();
        while(!ARC_SD_CSReady());
        ARC_SPI_SendByte(SPI1, token); /* transmit data token */
        if (token != 0xFD) /* Is data token */
        {

            #if 1
            ARC_SD_SPI_DMASend( buff, 512 );
            #else
            while(i < 512)
            {
                ARC_SPI_SendByte(SPI1, *buff);
                buff++;
                i++;
            }
            #endif

            ARC_SPI_SendByte(SPI1, 0xFF);                   /* CRC (Dummy)
*/
            ARC_SPI_SendByte(SPI1, 0xFF);                   /* CRC (Dummy)
*/
            resp = ARC_SPI_SendByte(SPI1, 0xFF);            /* Receive data
response */
```

```
        if ((resp & 0x1F) != 0x05)                    /* If not accepted, return
with error */
                ret = 0;
    }
    ARC_SD_CardBusy();
    ARC_SD_CS_HIGH();
    return ret;
}

/**
  * @brief   start sd card.
  * @param   None
  * @retval The SD card type.
  */
SD_Card_Type ARC_SD_SPI_Start(void)
{
    uint8_t i = 0;
    uint16_t retry_times = 0;
    uint8_t sd_response[5];
    SD_Card_Info *sdCardInfo;
    sdCardInfo = ARC_SD_SPI_GetCardInof();

    if (sdCardInfo->sd_stat & SD_Status_NoDisk)
        return SD_Unknown;

    SPI_BaudRateConfig(SPI1, ARC_SPI_MIN_SPEED);

    /*!< SD chip select high */
    ARC_SD_CS_HIGH();

    ARC_SysTick_Delay(100);

    /*!< Send dummy byte 0xFF, 10 times with CS high */
    /*!< Rise CS and MOSI for 80 clocks cycles */
    for (i = 0; i < 10; i++)
    {
        /*!< Send dummy byte 0xFF */
        ARC_SPI_SendByte(SPI1, SD_DUMMY_BYTE);
    }
    i = 20;
    do
    {
        ARC_sd_send_command(SD_CMD_GO_IDLE_STATE, 0, R1,
sd_response);
```

```
    }while(sd_response[0] != SD_IN_IDLE_STATE && --i);

    if(sd_response[0] == SD_IN_IDLE_STATE)
    {
        ARC_sd_send_command(SD_CMD_SEND_IF_COND, 0x1AA, R7,
sd_response);
        if(sd_response[0] == SD_IN_IDLE_STATE)/* SDv2? */
        {
            if(sd_response[3] == 0x01 && sd_response[4] == 0xAA)
            {
                retry_times = 0xFFF;
                do
                {

ARC_sd_send_command(SD_ACMD_APP_SEND_OP_COND, 1UL << 30,
R1, sd_response);
                }while(sd_response[0] && --retry_times);

                if(retry_times > 0)
                {
                    ARC_sd_send_command(SD_CMD_READ_OCR, 0x0,
R3, sd_response);
                    if(sd_response[1] & 0x80)
                    {
                        sdCardInfo->sd_stat &= ~SD_Status_NotInit;
                        sdCardInfo->sd_ct = (sd_response[1] & 0x40) ?
SD_SDHC : SD_SDSC;   /* SDv2 */
                    }
                    else
                    {
                        //printf("SD in power down status\n");
                    }
                }
            }
        }
        else /* SDv1 or MMCv3 */
        {
            ARC_sd_send_command(SD_ACMD_APP_SEND_OP_COND,
0x0, R1, sd_response);
            if(sd_response[0] <= 1)
            {
                sdCardInfo->sd_stat &= ~SD_Status_NotInit;
                sdCardInfo->sd_ct = SD_Ver1;
                retry_times = 0xFFF;
```

```
                    do
                    {

ARC_sd_send_command(SD_ACMD_APP_SEND_OP_COND, 0x0, R1,
sd_response);
                    }while(sd_response[0] && --retry_times);
                }
                else
                {
                    retry_times = 0xFFF;
                    do
                    {

ARC_sd_send_command(SD_CMD_SEND_OP_COND, 0x0, R1,
sd_response);
                    }while(sd_response[0] && --retry_times);
                }
                if (retry_times > 0)
                {
                    sdCardInfo->sd_stat &= ~SD_Status_NotInit;
                    sdCardInfo->sd_ct = SD_MMC_Ver3;
                    ARC_sd_send_command(SD_CMD_SET_BLOCKLEN, 512,
R1, sd_response);
                }

            }
        }

        ARC_SD_CS_HIGH();
        SPI_BaudRateConfig(SPI1, ARC_SPI_DEFAULT_SPEED);
        return sdCardInfo->sd_ct;
}

/**
   * @brief    Initialize SD parameters.
   * @param    None
   * @retval None:
   */
void ARC_SD_SPI_Param_Init(void)
{
        SD_Card_Info *sdCardInfo;
        sdCardInfo = ARC_SD_SPI_GetCardInof();
        sdCardInfo->sd_ct = SD_Unknown;
        sdCardInfo->sd_stat = SD_Status_NotInit;
```

```
}

/**
   * @brief    Initialize SD card.
   * @param    None
   * @retval None:
   */
void ARC_SD_SPI_Init(void)
{
      ARC_SPI_Init();
      ARC_SD_SPI_Param_Init();
}
```