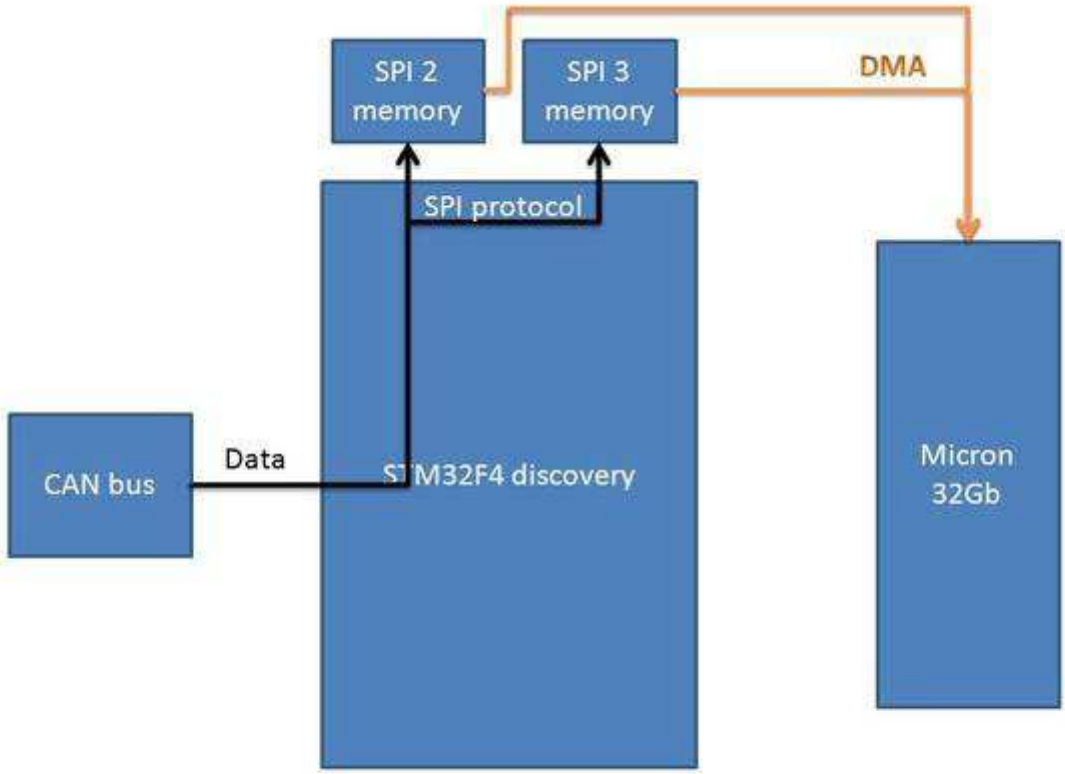
Hello all,

We need your help. We want to make DMA transfer between 2 SPI memories (SST25VF064C)(one in SPI2, other in SPI3) and FSMC memory (Micron 32Gb)



So, we have a problem with the DMA configuration.

This is the configuration of the DMA,SPI and FSMC

```
void DMA_Config(void)
{
  // Configure DMA for transmit
DMA_InitTypeDef DMA_InitStructure;
NVIC_InitTypeDef NVIC_InitStructure;
__IO uint32_t    Timeout = TIMEOUT_MAX;


RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_DMA1, ENABLE);


/* Reset DMA Stream registers (for debug purpose) */
DMA_DeInit(DMA1_Stream4);
```

```
/* Configure DMA Stream */
DMA_StructInit(&DMA_InitStructure);
DMA_InitStructure.DMA_Channel = DMA_Channel_0;
DMA_InitStructure.DMA_PeripheralBaseAddr = (uint32_t) (&SPI2->DR);
DMA_InitStructure.DMA_Memory0BaseAddr = 0x70000000; // FSMC address
DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralToMemory;


//DMA_InitStructure.DMA_BufferSize = SPI_BUFFER_SIZE;
DMA_InitStructure.DMA_BufferSize = 4;
// DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Enable;
DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_Byte;
DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_Byte;
DMA_InitStructure.DMA_Mode = DMA_Mode_Normal;
DMA_InitStructure.DMA_Priority = DMA_Priority_VeryHigh;
DMA_InitStructure.DMA_FIFOMode = DMA_FIFOMode_Enable;
DMA_InitStructure.DMA_FIFOThreshold = DMA_FIFOThreshold_HalfFull;
DMA_InitStructure.DMA_MemoryBurst = DMA_MemoryBurst_Single;
DMA_InitStructure.DMA_PeripheralBurst = DMA_PeripheralBurst_Single;
DMA_Init(DMA1_Stream4, &DMA_InitStructure);


/* Enable DMA Stream Transfer Complete interrupt */
DMA_ITConfig(DMA1_Stream4, DMA_IT_TC, ENABLE);




// Enable SPI2
SPI_Cmd(SPI2, ENABLE);
SPI_I2S_DMACmd (SPI2, SPI_I2S_DMAReq_Tx, ENABLE);


/* DMA Stream enable */
DMA_Cmd(DMA1_Stream4, ENABLE);




/* Check if the DMA Stream has been effectively enabled.
The DMA Stream Enable bit is cleared immediately by hardware if there is an
error in the configuration parameters and the transfer is no started (ie. when
wrong FIFO threshold is configured ...) */
Timeout = TIMEOUT_MAX;
while ((DMA_GetCmdStatus(DMA1_Stream4) != ENABLE) && (Timeout-- > 0))
{
}


/* Check if a timeout condition occurred */
if (Timeout == 0)
{
/* Manage the error: to simplify the code enter an infinite loop */
while (1)
```

```
{
}
 }


// Configure DMA1 Stream4 interrupt
NVIC_InitStructure.NVIC_IRQChannel = DMA1_Stream4_IRQn;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 6;
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStructure);







// Enable DMA request
//SPI_I2S_DMACmd(SPI2, SPI_I2S_DMAReq_Tx, ENABLE);
}


void NAND_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStructure;
  FSMC_NANDInitTypeDef FSMC_NANDInitStructure;
  FSMC_NAND_PCCARDTimingInitTypeDef  p;


  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD | RCC_AHB1Periph_GPIOE |
RCC_AHB1Periph_GPIOB, ENABLE);
//  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD | RCC_AHB1Periph_GPIOG |
RCC_AHB1Periph_GPIOE |
//                         RCC_AHB1Periph_GPIOF | RCC_AHB1Periph_GPIOB, ENABLE);


  RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);
  RCC_AHB3PeriphClockCmd(RCC_AHB3Periph_FSMC, ENABLE);

/*-- GPIO Configuration ------------------------------------------------------*/


/*!< Pin configuration  */
  GPIO_PinAFConfig(GPIOD, GPIO_PinSource14, GPIO_AF_FSMC);      // D0
  GPIO_PinAFConfig(GPIOD, GPIO_PinSource15, GPIO_AF_FSMC);      // D1
  GPIO_PinAFConfig(GPIOD, GPIO_PinSource0, GPIO_AF_FSMC);       // D2
  GPIO_PinAFConfig(GPIOD, GPIO_PinSource1, GPIO_AF_FSMC);       // D3

  GPIO_PinAFConfig(GPIOE, GPIO_PinSource7, GPIO_AF_FSMC);       // D4
  GPIO_PinAFConfig(GPIOE, GPIO_PinSource8, GPIO_AF_FSMC);       // D5
  GPIO_PinAFConfig(GPIOE, GPIO_PinSource9, GPIO_AF_FSMC);       // D6
  GPIO_PinAFConfig(GPIOE, GPIO_PinSource10, GPIO_AF_FSMC);      // D7


  GPIO_PinAFConfig(GPIOD, GPIO_PinSource4, GPIO_AF_FSMC);       // !RE
  GPIO_PinAFConfig(GPIOD, GPIO_PinSource5, GPIO_AF_FSMC);       // !WE
  GPIO_PinAFConfig(GPIOD, GPIO_PinSource6, GPIO_AF_FSMC);       // R/!B
  // GPIO_PinAFConfig(GPIOD, GPIO_PinSource7, GPIO_AF_FSMC);       // !CE
```

```
  GPIO_PinAFConfig(GPIOD, GPIO_PinSource12, GPIO_AF_FSMC);       // ALE
  GPIO_PinAFConfig(GPIOD, GPIO_PinSource11, GPIO_AF_FSMC);       // CLE


  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1 | GPIO_Pin_15 |GPIO_Pin_14 |
                        GPIO_Pin_4 | GPIO_Pin_5 | GPIO_Pin_6  |GPIO_Pin_12 |GPIO_Pin_11;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
  //GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;
  GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStructure.GPIO_PuPd  = GPIO_PuPd_UP;


  GPIO_Init(GPIOD, &GPIO_InitStructure);

  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_7 | GPIO_Pin_8 | GPIO_Pin_9 | GPIO_Pin_10 ;


  GPIO_Init(GPIOE, &GPIO_InitStructure);


  /* Configure the GPIO_ Write protect and ce2 */
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_13 | GPIO_Pin_2;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
  GPIO_Init(GPIOB, &GPIO_InitStructure);
  GPIOB->BSRRL = GPIO_Pin_13;
  GPIOB->BSRRL = GPIO_Pin_2;


  /* Configure the GPIO_ Write  ce1 pin (FSMC_CE2, here CE1? and not controled by FSMC because of 2
ce...)  */
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_7;
  GPIO_Init(GPIOD, &GPIO_InitStructure);
  GPIOD->BSRRL = GPIO_Pin_7;


  /*-- FSMC Configuration ------------------------------------------------------*/
  p.FSMC_SetupTime =  0x3;      //0x0;
  p.FSMC_WaitSetupTime = 0x6;   //0x2;
  p.FSMC_HoldSetupTime = 0x0;   //0x1;
  p.FSMC_HiZSetupTime =  0x0;   //0x0;


//  p.FSMC_SetupTime =  0x0;      //0x0;
//  p.FSMC_WaitSetupTime = 0x6;   //0x2;
//  p.FSMC_HoldSetupTime = 0x3;   //0x1;
//  p.FSMC_HiZSetupTime =  0x0;   //0x0;


//  p.FSMC_SetupTime =  0x1;      //0x0;
//  p.FSMC_WaitSetupTime = 0x6;   //0x2;
//  p.FSMC_HoldSetupTime = 0x0;   //0x1;
//  p.FSMC_HiZSetupTime =  0x0;   //0x0;


//  p.FSMC_SetupTime =  0x0F;      //0x0;
```

```
//  p.FSMC_WaitSetupTime = 0x2F;   //0x2;
//  p.FSMC_HoldSetupTime = 0x1F;   //0x1;
//  p.FSMC_HiZSetupTime =  0x0;   //0x0;


  FSMC_NANDInitStructure.FSMC_Bank = FSMC_Bank2_NAND;
  FSMC_NANDInitStructure.FSMC_Waitfeature = FSMC_Waitfeature_Enable;
  FSMC_NANDInitStructure.FSMC_MemoryDataWidth = FSMC_MemoryDataWidth_8b;
  FSMC_NANDInitStructure.FSMC_ECC = FSMC_ECC_Disable;
  FSMC_NANDInitStructure.FSMC_ECCPageSize = FSMC_ECCPageSize_2048Bytes;
  FSMC_NANDInitStructure.FSMC_TCLRSetupTime = 0x00;
  FSMC_NANDInitStructure.FSMC_TARSetupTime = 0x00;
  FSMC_NANDInitStructure.FSMC_CommonSpaceTimingStruct = &p;
  FSMC_NANDInitStructure.FSMC_AttributeSpaceTimingStruct = &p;


  FSMC_NANDInit(&FSMC_NANDInitStructure);


  /*!< FSMC NAND Bank Cmd Test */
  FSMC_NANDCmd(FSMC_Bank2_NAND, ENABLE);
}


void sFLASH_Init(void)
{
  SPI_InitTypeDef  SPI_InitStructure2;
  SPI_InitTypeDef  SPI_InitStructure3;


  sFLASH_LowLevel_Init();


  /*!< Deselect the FLASH: Chip Select high */
  sFLASH_CS2_HIGH();
  sFLASH_CS3_HIGH();


  /*!< SPI configuration */
  SPI_InitStructure2.SPI_Direction = SPI_Direction_2Lines_FullDuplex;
  SPI_InitStructure2.SPI_Mode = SPI_Mode_Master;
  SPI_InitStructure2.SPI_DataSize = SPI_DataSize_8b;
  SPI_InitStructure2.SPI_CPOL = SPI_CPOL_High;
  SPI_InitStructure2.SPI_CPHA = SPI_CPHA_2Edge;
  SPI_InitStructure2.SPI_NSS = SPI_NSS_Soft;
  //SPI_InitStructure2.SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_8;   // 5Mhz
  SPI_InitStructure2.SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_4; //10MHz
  SPI_InitStructure3.SPI_Direction = SPI_Direction_2Lines_FullDuplex;
  SPI_InitStructure3.SPI_Mode = SPI_Mode_Master;
  SPI_InitStructure3.SPI_DataSize = SPI_DataSize_8b;
  SPI_InitStructure3.SPI_CPOL = SPI_CPOL_High;
  SPI_InitStructure3.SPI_CPHA = SPI_CPHA_2Edge;
  SPI_InitStructure3.SPI_NSS = SPI_NSS_Soft;
  //SPI_InitStructure3.SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_8;   // 5Mhz
  SPI_InitStructure3.SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_4; //10MHz


  SPI_InitStructure2.SPI_FirstBit = SPI_FirstBit_MSB;
```

```
  SPI_InitStructure2.SPI_CRCPolynomial = 7;
  SPI_InitStructure3.SPI_FirstBit = SPI_FirstBit_MSB;
  SPI_InitStructure3.SPI_CRCPolynomial = 7;
  SPI_Init(sFLASH_SPI2, &SPI_InitStructure2);
  SPI_Init(sFLASH_SPI3, &SPI_InitStructure3);


  /*!< Enable the sFLASH_SPI  */
  SPI_Cmd(sFLASH_SPI2, ENABLE);
  SPI_Cmd(sFLASH_SPI3, ENABLE);
}
```

At the first time, we try to make a dma transfer between SPI2 and internal memory. But the internal memory just receive the SPI2->DR without increment the SPI memory address. So we have always the same data copied. If we make the PERIPHERALINC = ENABLE, the DR is copied in the first adress and other elements are at 0x00 whereas the memory content 0x55.
How do we increment our memory to modify the DR and do a good DMA?

After, when the fisrt SPI memory is full, we make the DMA and use the other SPI to store the data of CAN. But when this memory is full, we make the DMA of this, always to the FSMC.

Do you think this project is possible? If yes, do we have a good configuration ? (for DMA SPI->FSMC)

Thank you to not send link of this forum, we looked all posts about DMA.

Thank you so much,

Florian and Jérôme.