

Hi,

There seems to be an issue with the SPI peripheral in the STM32F10x family of devices.

I have the same issue with multiple devices (STM32F103ZGT8, STM32F107VCT6)

Here is the issue:

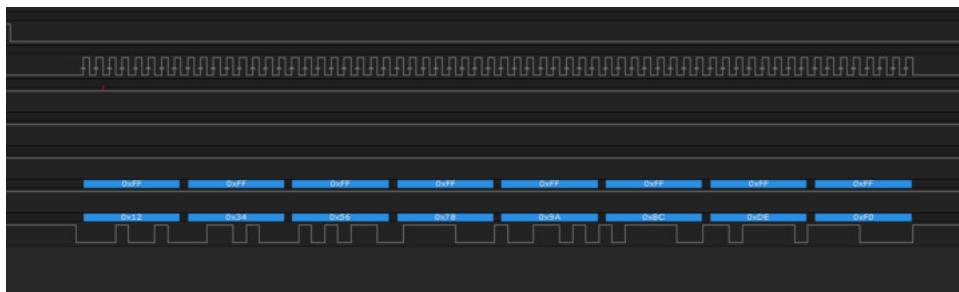
When I try to read in master mode, I must send data for the peripheral to send out a clock so I can read the data.

When I am reading data, I am reading back what I sent.

I have used the SPI in interrupt and with DMA and have the same issue.

When I try to write/read, I read the first byte properly, but then I read back what I placed in the DR register

Using a logic analyzer, I was able to see that my master and slave are transmitting the bytes properly.



From this stream, I would expect to read 0xFF 8 times.

However, my RX buffer will contain: 0xFF 0x12 0x34 0x56 0x78 0x9A 0xBC 0xDE

When using breakpoints in interrupt mode, here is the sequence of events that I have seen.

1. When we start the transfer, we place the first byte in the DR register
2. We then enable the TXE interrupt.
3. This generates an interrupt.
4. In the ISR, we place the next byte in the DR register while the previous one is transmitted.
5. Some time later, we receive a byte which triggers the RXNE interrupt
6. In the ISR, we read back the byte from the DR register.
7. An interrupt is generated when the last byte is transferred
8. We repeat these from step 4 until all bytes have been transmitted/read

The odd thing is that we get a TXE interrupt (step 2) even though we are still transferring the first byte. This means that we have not read single byte but we have already given the SPI peripheral 2 bytes.

Here is the hypothesis I have arrived to:

It seems that the DR register overwrites the data in the RX buffer if I write to it too soon. This makes the use of the DMA for reading data useless.

I was able to resolve the issue by following the steps below (this only works in interrupt mode):

1. Place first byte in the DR register
2. Wait for RXNE interrupt.
3. Read data in DR.
4. Place next byte in DR register.
5. Repeat from step 2

I originally used the HAL that comes with STMcubemx but had to write my own code to properly manage the SPI transfers.

Is this a known issue with the STM32F10x devices? Or am I missing something?

Below is my init code with DMA:

```
// To transfer data with the SPI in DMA mode, I use this line of code:
HAL_StatusTypeDef status = HAL_SPI_TransmitReceive_DMA(&hSpi, txData, rxData, size);

/**
 * @brief This function initializes the SPI peripheral
 * @param None
 * @retval None
 */
void SPI_Init(void)
{
    spiEventHandle = xEventGroupCreateStatic(&spiEvent);

    hSpi.Instance = SPI1;
    hSpi.Init.Mode = SPI_MODE_MASTER;
    hSpi.Init.Direction = SPI_DIRECTION_2LINES;
    hSpi.Init.DataSize = SPI_DATASIZE_8BIT;
    hSpi.Init.CLKPolarity = SPI_POLARITY_LOW;
    hSpi.Init.CLKPhase = SPI_PHASE_1EDGE;
    hSpi.Init.NSS = SPI_NSS_SOFT;
    hSpi.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_256;
    hSpi.Init.FirstBit = SPI_FIRSTBIT_MSB;
    hSpi.Init.TIMode = SPI_TIMODE_DISABLE;
    hSpi.Init.CRCCalculation = SPI_CRCALCULATION_DISABLE;
    hSpi.Init.CRCPolynomial = 10;
    if (HAL_SPI_Init(&hSpi) != HAL_OK)
    {
```

```

        _Error_Handler(__FILE__, __LINE__);
    }

/***
 * @brief This function initializes the low level peripherals used by the SPI
 * @param SPI_HandleTypeDef * - pointer to the SPI handle
 * @retval None
 */
void HAL_SPI_MspInit(SPI_HandleTypeDef *spiHandle)
{
    // Enable SPI1 clock
    __HAL_RCC_SPI1_CLK_ENABLE();

    // SPI1 GPIO Configuration
    GPIO_InitPin(SPI_CS);
    GPIO_InitPin(SPI_SCK);
    GPIO_InitPin(SPI_MOSI);
    GPIO_InitPin(SPI_MISO);

    // SPI1 DMA Init
    // SPI1_RX Init
    hDmaSpiRx.Instance = DMA1_Channel2;
    hDmaSpiRx.Init.Direction = DMA_PERIPH_TO_MEMORY;
    hDmaSpiRx.Init.PeriphInc = DMA_PINC_DISABLE;
    hDmaSpiRx.Init.MemInc = DMA_MINC_ENABLE;
    hDmaSpiRx.Init.PeriphDataAlignment = DMA_PDATAALIGN_BYTE;
    hDmaSpiRx.Init.MemDataAlignment = DMA_MDATAALIGN_BYTE;
    hDmaSpiRx.Init.Mode = DMA_NORMAL;
    hDmaSpiRx.Init.Priority = DMA_PRIORITY_LOW;
    if (HAL_DMA_Init(&hDmaSpiRx) != HAL_OK)
    {
        _Error_Handler(__FILE__, __LINE__);
    }

    __HAL_LINKDMA(spiHandle, hdmarx, hDmaSpiRx);

    // SPI1_TX Init
    hDmaSpiTx.Instance = DMA1_Channel3;
    hDmaSpiTx.Init.Direction = DMA_MEMORY_TO_PERIPH;
    hDmaSpiTx.Init.PeriphInc = DMA_PINC_DISABLE;
    hDmaSpiTx.Init.MemInc = DMA_MINC_ENABLE;
    hDmaSpiTx.Init.PeriphDataAlignment = DMA_PDATAALIGN_BYTE;
    hDmaSpiTx.Init.MemDataAlignment = DMA_MDATAALIGN_BYTE;
    hDmaSpiTx.Init.Mode = DMA_NORMAL;
    hDmaSpiTx.Init.Priority = DMA_PRIORITY_LOW;
    if (HAL_DMA_Init(&hDmaSpiTx) != HAL_OK)
    {
        _Error_Handler(__FILE__, __LINE__);
    }

    __HAL_LINKDMA(spiHandle, hdmatx, hDmaSpiTx);
}

/***
 * @brief This function initializes the DMA peripheral
 * @param None
 * @retval None
 */
void DMA_Init(void)
{
    // DMA controller clock enable
    __HAL_RCC_DMA1_CLK_ENABLE();
}

/***
 * @brief This function enables the interrupts used by the DMA
 * @param None
 * @retval None
 */
void DMA_EnableIRQ(void)
{
    // DMA interrupt init
    // DMA1_Channel2_IRQHandler interrupt configuration
    HAL_NVIC_SetPriority(DMA1_Channel2_IRQn, 7, 0);
    HAL_NVIC_EnableIRQ(DMA1_Channel2_IRQn);

    // DMA1_Channel3_IRQHandler interrupt configuration
    HAL_NVIC_SetPriority(DMA1_Channel3_IRQn, 7, 0);
    HAL_NVIC_EnableIRQ(DMA1_Channel3_IRQn);
}

/***
 * @brief This function handles DMA1 channel2 global interrupt.
 */
void DMA1_Channel2_IRQHandler(void)
{
    HAL_DMA_IRQHandler(&hDmaSpiRx);
}

/***
 * @brief This function handles DMA1 channel3 global interrupt.
 */
void DMA1_Channel3_IRQHandler(void)
{
    HAL_DMA_IRQHandler(&hDmaSpiTx);
}

/***
 * @brief This function is a callback used to notify when a transfer is completed
 * @note This function is called by the HAL
 * @param SPI_HandleTypeDef * - pointer to the SPI handle
 * @retval None
 */

```

Thank you in advance for the help.

Michel