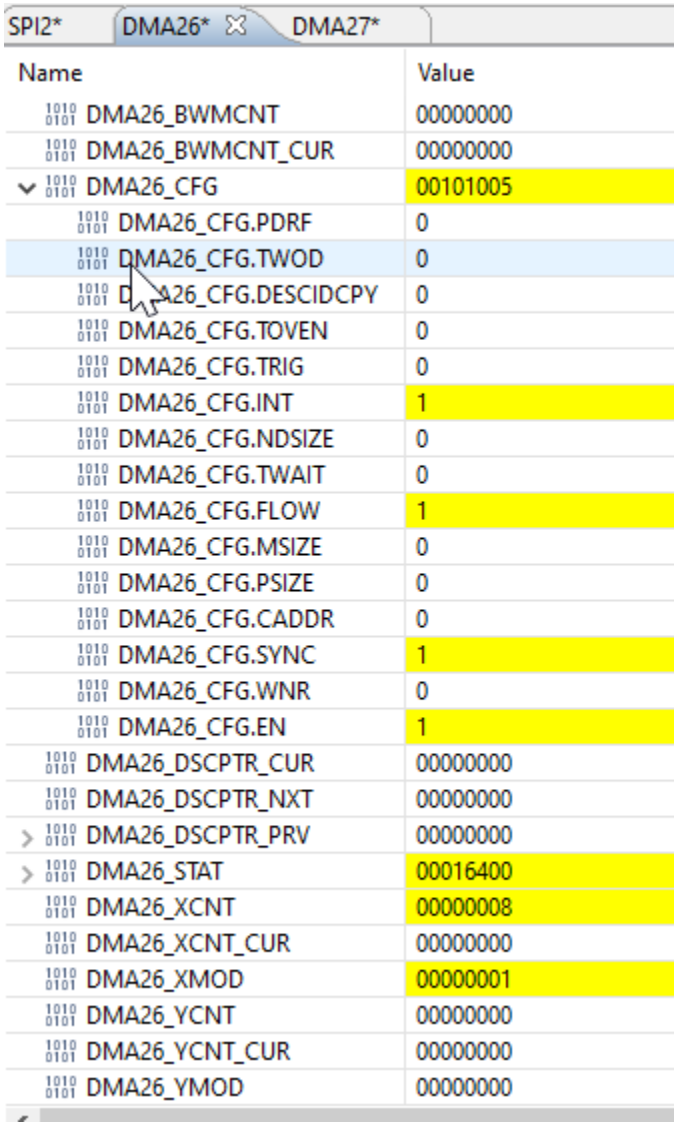
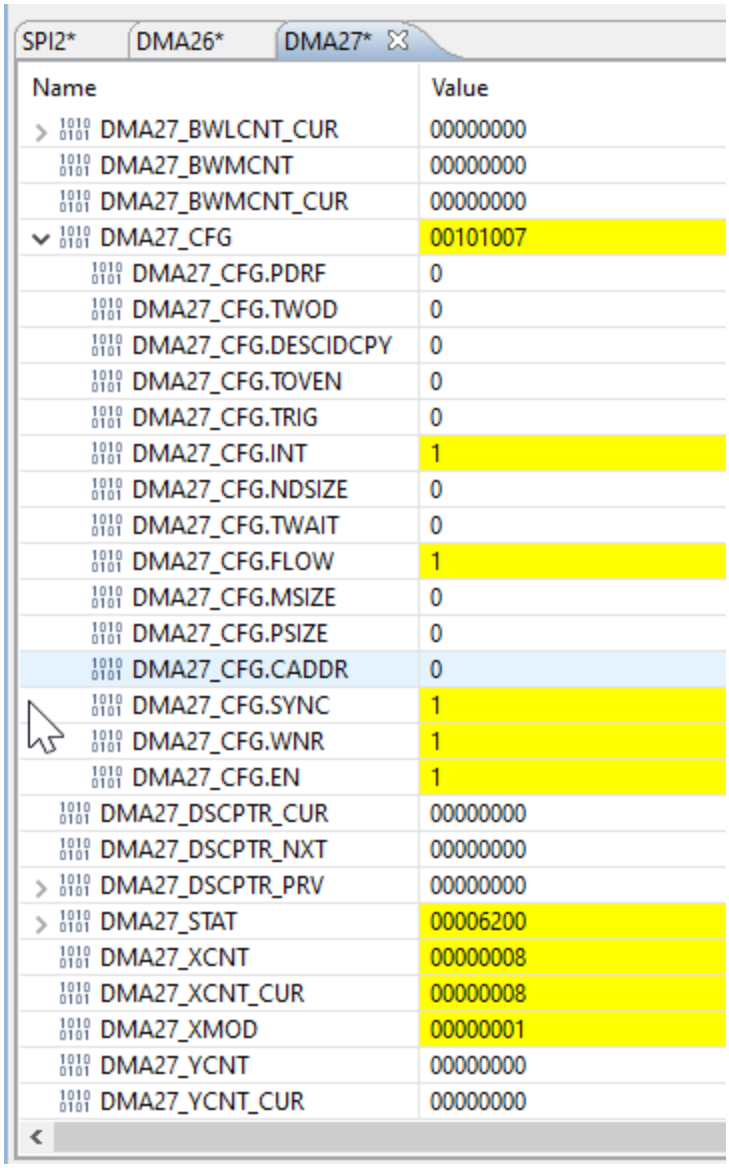
Could you please confirm whether you have enabled "DMA_CFG.SYNC" bit and "DMA_CFG.INT" bit. If not, Could you please try to enable the "DMA SYNC" bit, and "Peripheral Interrupt request" bit from "DMA_CFG" register.

Here are the DMA settings for TX(26) and RX(27) showing sync and int bits set.





1. What is the SPI Clock? Is it provided by the Host to the Slave?

Yes, provided by host (CPU) to slave (DSP).

3. Can you confirm whether the normal, Master Transmit Slave Receive(MOSI) operation functions correctly?

MOSI seems to behave correctly. When master transmits a data packet, it is received as expected in the callback function.

4. Are you experiencing the issue only in DMA Descriptor List mode? If so, could you test it in a different mode?

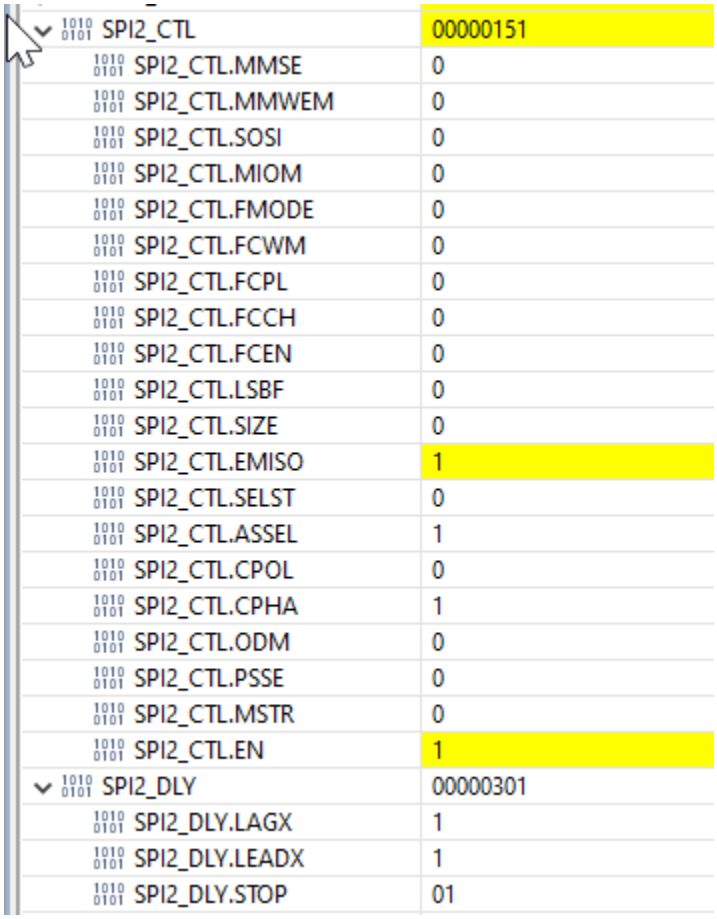
We are actually using the autobuffer mode. Descriptor list mode does not seem to work at all when using SPI2. We are using Descriptor list mode with the DSP as SPI master on a different SPI bus and it works as expected. Here is our initialization code for the SPI bus.

```
      ePDMAMode = ADI_PDMA_AUTOBUFFER_MODE;

      /* Open SPI */
      adi_spi_Open(SPI_DEVB, ADI_SPI_DIR_BIDIRECTION, SPIMemory1, ADI_SPI_BIDIR_MEMORY_SIZE,
&hSPISlave);
```

```
    /* Register SPI Callback function for DMA mode */
    adi_spi_RegisterCallback(hSPISlave, Callback, NULL);

    /* Buffer Start Address */
    Src_List.pStartAddr            = (uint8_t *)SrcData;
    /* DMA Config - only specify memory transfer size */
    Src_List.XCount                = DESC_BUFFER_SIZE; // 8
    Src_List.XModify               = DMA_MSIZE_IN_BYTES; // 1
    Src_List.Config                = ENUM_DMA_CFG_XCNT_INT;
    Src_List.pNxtDscp              = NULL;

    /* Buffer Start Address */
    Dest_List.pStartAddr           = (uint8_t *)DestData;
    /* DMA Config - only specify memory transfer size */
    Dest_List.XCount               = DESC_BUFFER_SIZE; // 8
    Dest_List.XModify              = DMA_MSIZE_IN_BYTES; // 1
    Dest_List.Config               = ENUM_DMA_CFG_XCNT_INT;
    Dest_List.pNxtDscp             = NULL;

    /* Set the Slave select for Slave 2 */
    adi_spi_EnableMISO(hSPISlave, true);
    adi_spi_DMAWrite(hSPISlave, &Src_List, 1, ePDMAMode);
    adi_spi_DMARead(hSPISlave, &Dest_List, 1, ePDMAMode);
```

5. Have you configured the SPI delay register? If so, please try setting it to zero and observe the results.

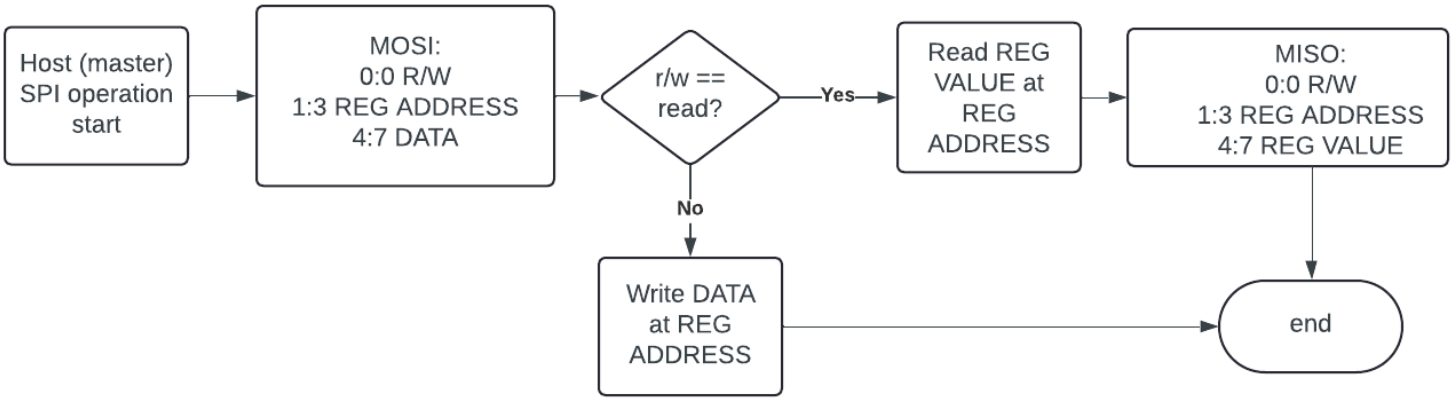
here are the SPI config settings as well as delay.



Changing the SPI2_DLY to 0 has no effect on the behavior.

7. Could you provide a detailed explanation of your application? A simple block diagram would be very helpful.

We are doing reading/writing registers created in a SigmaStudio+ application via SPI.



Our protocol, which used for both TX and RX, is an 8 byte packet where the bytes are:
0: r/w byte
1-3: 24 bit sigma studio register
4-7: 4 bytes of data

We have the following callback that is called when a SPI event happens on SPI2 of the device:

```
void Update_Parameter()
{
        uint32_trw_byte, reg_addr, data_sz, data_val   = 0;

        // DestData is the MOSI data from host.
        rw_byte  = DestData[0];
        reg_addr = ((int)DestData[3] << 16) | ((int)DestData[2] << 8) | ((int)DestData[1]);
        data_val = ((int)DestData[7] << 24) | ((int)DestData[6] << 16)| ((int)DestData[5] << 8) | ((int)DestData[4]);

        adi_gpio_Toggle(ADI_GPIO_PORT_B,ADI_GPIO_PIN_3);

        if(rw_byte == ADI_SS_WRITE)
        {
                dsp_write_register(reg_addr, 1, data_val);
        }
        else if(rw_byte == ADI_SS_READ)
        {
                // normally a read will write the current value at reg_addr into the SrcData (MISO) buffer,
                // which is what the host reads.
                // however, to demonstrate the offset between read requests and data delivered, we'll use the following
                // to show what is being read by the host after each transmission.

                static uint8_t count = 0;

                SrcData[0] = count++;
                SrcData[1] = count++;
                SrcData[2] = count++;
                SrcData[3] = count++;

                SrcData[4] = count++;
                SrcData[5] = count++;
                SrcData[6] = count++;
                SrcData[7] = count++;
        }
}
```

The result of this when performing sequential reads is as follows:

SPI READ	expected	received
1	ff ff ff ff ff ff ff ff	ff ff ff ff ff ff ff ff
2	00 01 02 03 04 05 06 07	00 00 00 00 00 00 00 00
3	08 09 0A 0B 0C 0D 0E 0F	00 00 00 00 00 00 00 00
4	10 11 12 13 14 15 16 17	00 00 00 00 00 01 02 03
5	18 19 1A 1B 1C 1D 1E 1F	04 05 06 07 08 09 0A 0B
6	20 21 22 23 24 25 26 27	0C 0D 0E 0F 10 11 12 13
7	28 29 2A 2B 2C 2D 2E 2F	14 15 16 17 18 19 1A 1B
8	30 31 32 33 34 35 36 37	1C 1D 1E 1F 20 21 22 23

The data seems to be offset by 24 bytes here which makes it awkward to use our simple protocol. We would like to have the device behave as indicated in the “expected” column.