SPI Communication - Hercules Microcontroller with TPS65381

Peter Steffan Microcontroller Products

ABSTRACT

This application report summarizes the necessary steps to setup the SPI / MibSPI of the Hercules safety microcontrollers to interface the TPS65381 multi rail power supply through the Serial Peripheral Interface (SPI). Detailed code examples give guidelines to do the different setup steps. All examples are based on the Hardware Abstraction Layer Code Generator (HalCoGen) tool for Hercules devices.

Table 1 Abbreviations

Abbreviation/ Acronym	Description
SPI	Serial Peripheral Interface
MibSPI	Multi-Buffered Serial Peripheral Interface
SOMI	SPI / MibSPI Slave-output master-input (microcontroller)
SIMO	SPI / MibSPI Slave-input master output (microcontroller)
NCS	SPI Chip select (active low)
SPICLK	SPI clock
SDI	SPI Serial Data In (TPS65381)
SDO	SPI Serial Data Out (TPS65381)
LSB	Least Significant Bit
MSB	Most Significant Bit
tVCLK	Peripheral clock (VCLK) period
tDELAY	Delay-time between 2 SPI transfers
HalCoGen	Hardware Abstraction Layer Code Generator
LBIST	Logic Build In Self-Test
ABIST	Analog Build In Self-Test

Preliminary

Contents

1 Wiring Diagram 3
2 SPI / MibSPI Initialization 4
2.1 TPS65381 SPI Down-Time 4
2.2 SPI Initialization 4
2.2.1 Generate SPI Software Functions with HalCoGen 4
2.2.2 Data Format 5
2.2.3 Signal Timing 7
2.3 MibSPI Initialization 9
2.3.1 Generate MibSPI Software Functions with HalCoGen 9
2.3.2 Transfer Group and Multi-Buffered RAM Setup 10
3 TPS65381 SPI Status Information 13
4 SPI Communication Examples 13
4.1 Reading the Device Revision Number 14
4.2 Set Watchdog Open Window Duration 15
4.3 Send Open Window Watchdog Tokens 16
5 Multiple SPI Slave Network 17

Figure 1 Basic Wiring (Interconnect) 3
Figure 2 HalCoGen SPI Selection 5
Figure 3 SPI Data Format 6
Figure 4 SPI Delay Timings 8
Figure 5 HalCoGen MibSPI Selection 9
Figure 6 MibSPI Transfer Group Setup Example 11
Figure 7 SPI Read Transfer 14
Figure 8 SPI Write Transfer 15
Figure 9 Watchdog Token Sequence 16

Table 1 Abbreviations 1
Table 2 Basic Wiring (Required Signals) 3
Table 3 Data Format Parameter Settings 5
Table 4 Signal Timing Parameter Settings 7
Table 5 Transfer Group Parameter Settings 10
Table 6 RAM Transfer Buffer Parameter Settings 10

Preliminary

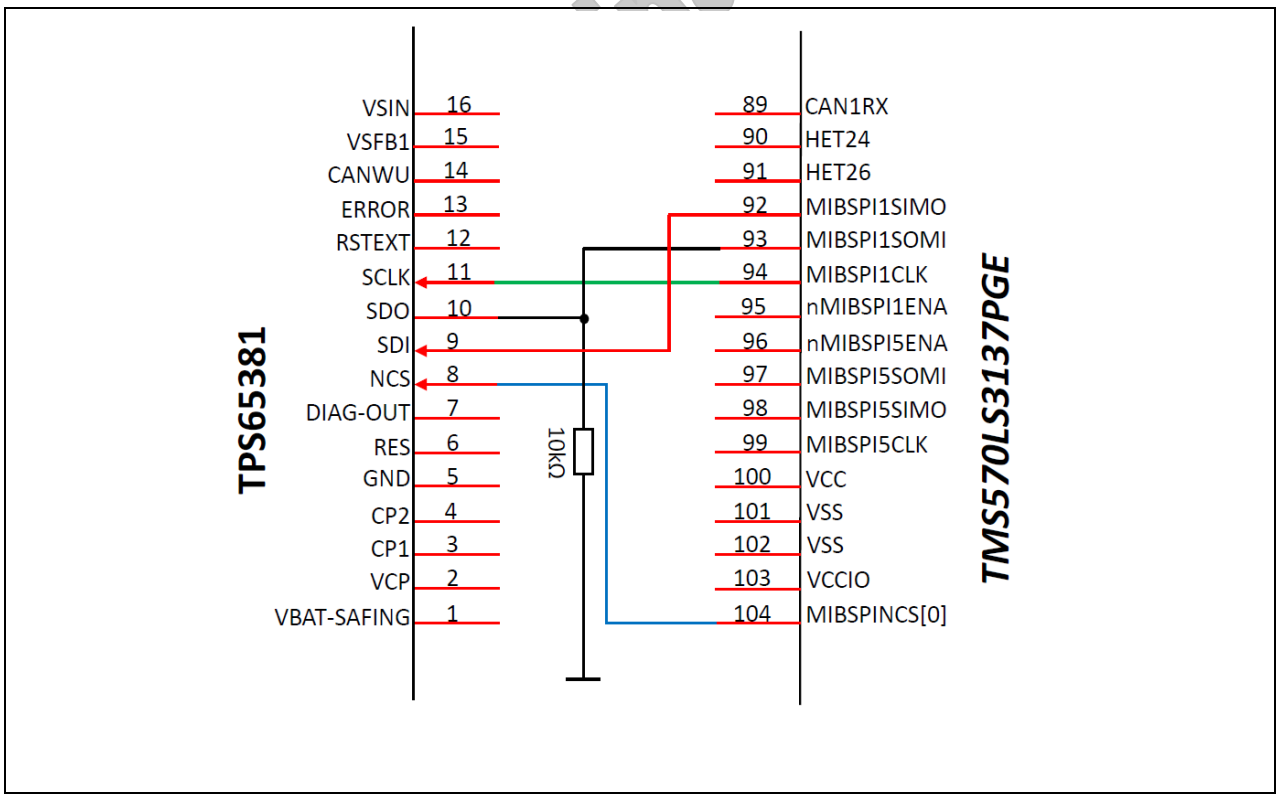
1 Wiring Diagram

This chapter describes the SPI / MibSPI hardware wiring between a Hercules microcontroller and the TPS65381, which is required to setup a working communication between the devices through SPI. The following examples use the MibSPI1 of the Hercules device, represented by the TMS570LS3137

Table 2 Basic Wiring (Required Signals)

TMS570LS3137 Microcontroller			TPS65381	
Pin	Pin# on 144PGE package	Pin# on 337ZWT package	Pin	TPS65381
MIBSPI1NCS[0]	105	R2	NCS	8
MIBSPI1CLK	95	F18	SCLK	11
MIBSPI1SOMI	94	G18	SDO	10
MIBSPI1SIMO	93	F19	SDI	9

Figure 1 Basic Wiring (Interconnect)



It is relevant that the SDI pin of the TPS65381 is connected to the SIMO pin of the Hercules device and the SDO pin of the TPS65381 is connected to the SOMI pin of the Hercules device. The connected pull-down resistor (i.e. 10k) between SDO and SOMI prevents the possible incorrect occurrence of SPI SDO error flags during a SPI communication.

The Hercules device acts as the master and provides the SPI clock signal (SPICLK) to the TPS65381.

The chip select signal NCS generated by the SPI master device enables the communication to the TPS65381. It also allows having several SPI slaves connected to the SPI master, given that each SPI slave can be enabled by a separate chip select signal from the SPI master.

2 SPI / MibSPI Initialization

This chapter describes how to setup the master SPI on the microcontroller side to establish communication with the TPS65381. In addition, steps how to setup the multi-buffered SPI (MibSPI) of the Hercules device are described. Using the MibSPI reduces microcontroller CPU load and simplifies the communication i.e. for the TPS65381 register initialization or the watchdog token handling for the question and answer watchdog.

For generating the SPI / MibSPI initialization code, the HalCoGen (Hardware Abstraction Layer Code Generator) graphical code generation tool version 03.05.00 was used. This tool can be downloaded from the TI webpage (www.ti.com/hercules). Examples of the different code generation steps are shown in the following chapters.

Preliminary

2.1 TPS65381 SPI Down-Time

During each power-up sequence the TPS65381 performs an internal logic build in selftest (LBIST) and an analog build in selftest (ABIST). This is also the case on every transition through TPS65381reset state unless the automatic BIST feature is deactivated in SAFETY_BIST_CTRL register. For details about the different TPS65381 states please refer to the TPS65381 datasheet.

The typical LBIST run time is 4.2ms (+/- 5%). The LBIST execution is followed by a 16ms wait period to fill the digital filters covered by the LBIST. In parallel to this 16ms wait period, the ABIST is performed. This gives a total BIST time (LBIST and ABIST) of around 21ms.

During this 21ms BIST time a SPI read or write access should be avoided, as during internal BIST time the TPS65381 SPI module is not able to provide correct responses or accept any SPI write access.

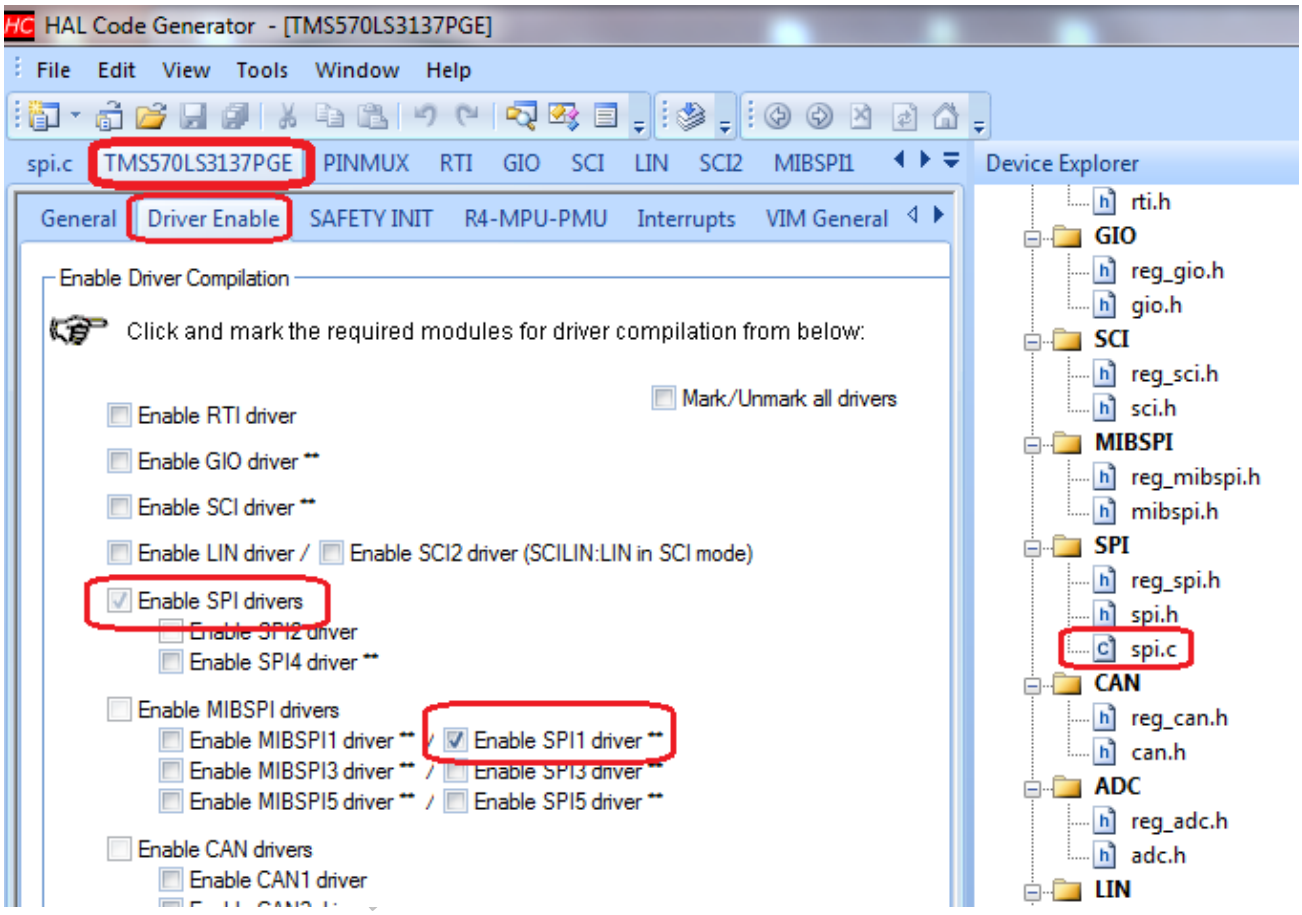
2.2 SPI Initialization

In order to setup a proper SPI communication between the Hercules device and the TPS65381, the master SPI (Hercules device) needs a minimum SPI setup, which is described in the following chapters.

2.2.1 Generate SPI Software Functions with HalCoGen

The basic HalCoGen setting as shown in Figure 2 is required to generate SPI software drivers:

Figure 2 HalCoGen SPI Selection



The generated MibSPI software drivers can be found in the source file 'spi.c'.

2.2.2 Data Format

The data format of the master SPI needs following parameter setup, assuming a peripheral clock (VCLK) frequency of 80MHz:

Table 3 Data Format Parameter Settings

Parameter	***Setting***	***Comment***
Frame size	16 bits	16 bit data word length
SPI clock frequency	1 MHz	TPS65381 supports up to 5 MHz
Shift direction	MSB first	
Clock Polarity	0	Master data is output on the rising edge, slave data on falling edge
Clock Phase	0	

Parity	No Parity	
WDELAY	62	Delay-time between 2 transfers: tDelay = 788ns

Calculation of parameter WDELAY:

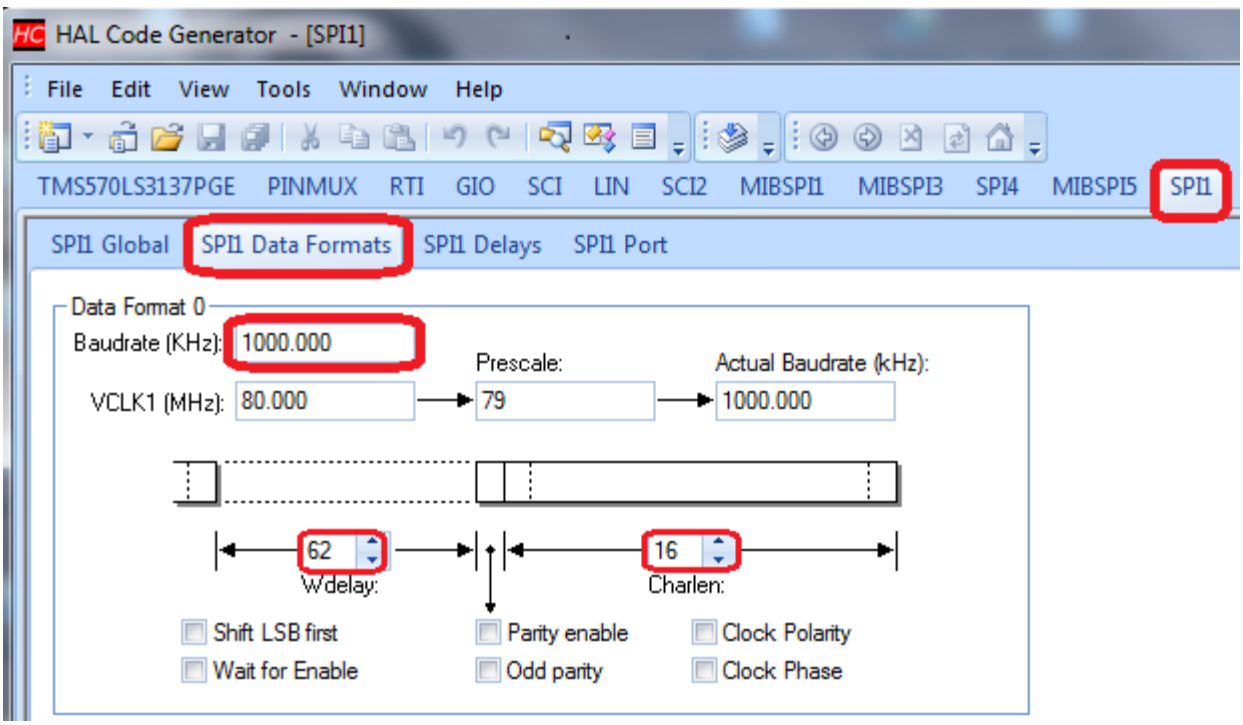
$$tDelay = WDELAY * tVCLK + 2 * tVCLK$$

$$WDELAY = \frac{tDelay - 2*tVCLK}{tVCLK} = \frac{788ns - (2*12.5ns)}{12.5ns} = 61.04$$

$$\text{tDelay} = 788ns$$

$$tVCLK = \frac{1}{80MHz} = 12.5ns$$

Figure 3 SPI Data Format



The following code sequence is generated by the HalCoGen Tool:

```
/** - Data Format 0 */
spiREG1->FMT0 = (62U << 24U) /* wdelay */
              | (0U << 23U)  /* parity Polarity */
              | (0U << 22U)  /* parity enable */
              | (0U << 21U)  /* wait on enable */
              | (0U << 20U)  /* shift direction */
              | (0U << 17U)  /* clock polarity */
              | (0U << 16U)  /* clock phase */
              | (79U << 8U)  /* baudrate prescale */
              | 16U;         /* data word length */
```

2.2.3 Signal Timing

For the chip select signal, the TPS65381 requires specific signal timings as described in Table 4.

Table 4 Signal Timing Parameter Settings

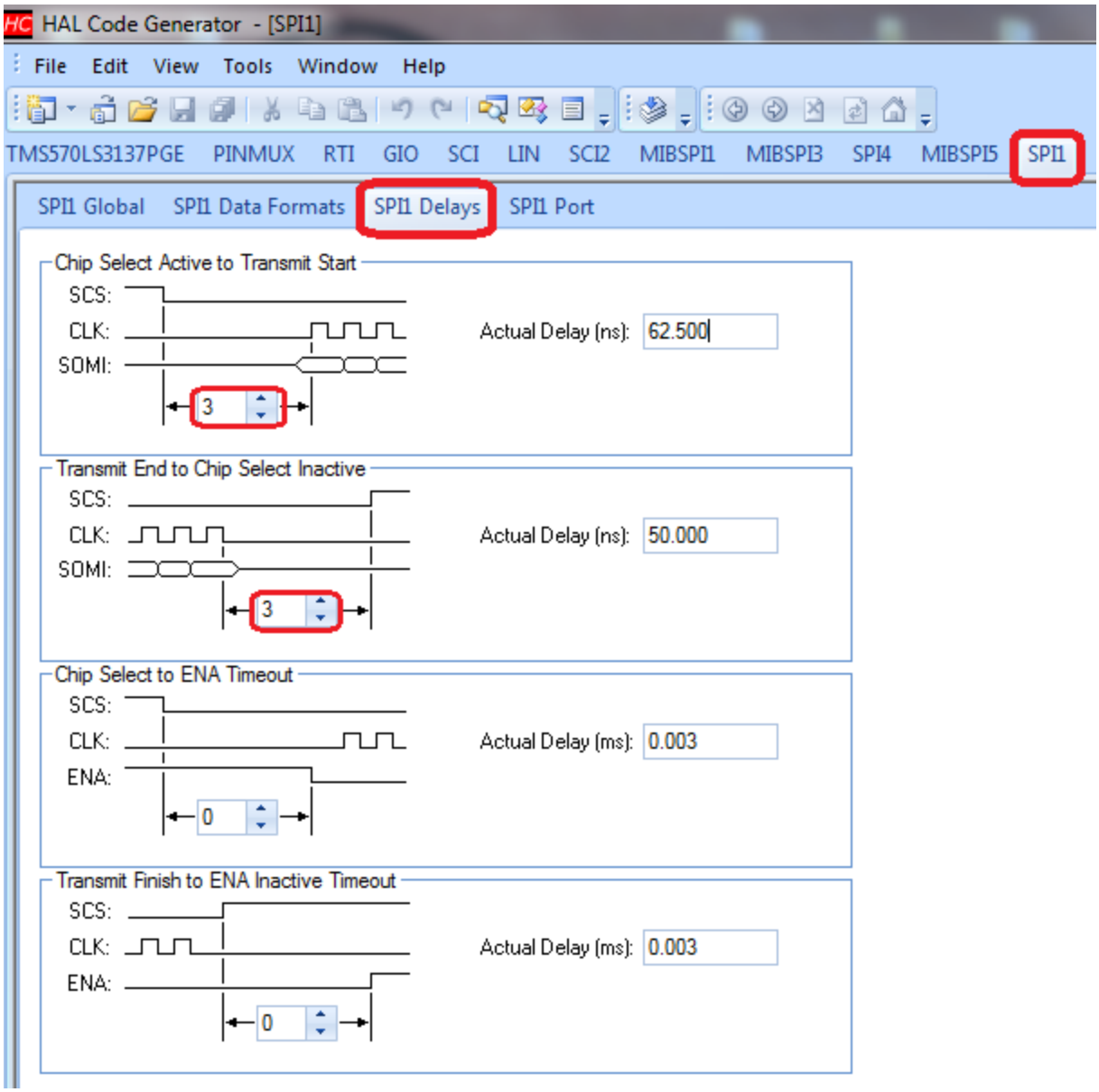
Parameter	***Setting***	***Comment***
Chip Select setup time	62.5ns	Time between falling edge of NCS and rising edge of the SPICLK clock. . Min. 45ns allowed
Chip Select hold time	50.0ns	Time between the falling edge of the SPICLK clock and rising edge of NCS. Min. 45ns allowed

Preliminary

NOTE:

During TPS65381 SPI communication the NCS signal has to be active low. An inactive high NCS signal keeps the SPI slave interface of the TPS65381 in reset state and its SDO output is tri-stated.

Figure 4 SPI Delay Timings



The following code sequence is generated by the HalCoGen Tool:

```
/** - Delays */
    spiREG1->DELAY = (3U << 24U) /* C2TDELAY */
                   | (3U << 16U) /* T2CDELAY */
                   | (0U << 8U)  /* T2EDELAY */
                   | 0U;         /* C2EDELAY */
```

2.3 MibSPI Initialization

For reducing CPU load on the microcontroller side, the usage of a multibuffered SPI (MibSPI) can be useful as soon as multiple bytes need to be transferred to the TPS65381. This can be the case for example during device initialization or for sending the calculated watchdog tokens (see chapter 4.3).

In order to establish a MibSPI communication between a Hercules microcontroller and the TPS65381, further MibSPI initialization steps are required:

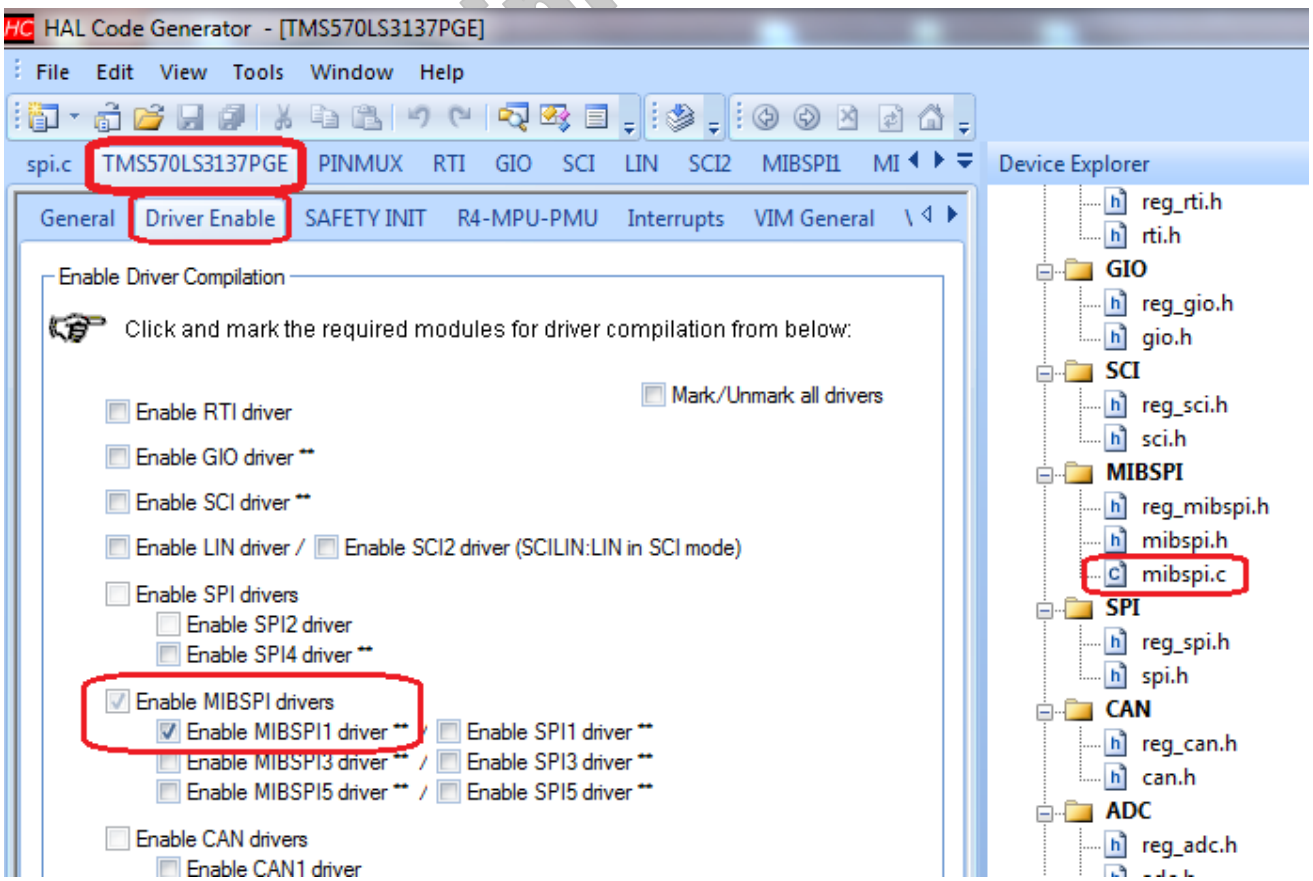
1. Enable multi-buffered mode
2. Setup a transfer group
3. Initialize the MibSPI

The data format and signal timing setup can be adopted from the SPI initialization of chapter 2.2. The HalCoGen Tool can be used to generate the required MibSPI software functions.

2.3.1 Generate MibSPI Software Functions with HalCoGen

The following HalCoGen setting is needed to make sure, MibSPI software drivers are generated:

Figure 5 HalCoGen MibSPI Selection



The generated MibSPI software drivers can be found in the source file ‘mibspi.c’.

2.3.2 Transfer Group and Multi-Buffered RAM Setup

The Table 5 describes the Transfer Group Control Register setup, Table 6 the control part of the Transmission Data Register of each used buffer:

Table 5 Transfer Group Parameter Settings

Parameter	***Setting***	***Comment***
Oneshot Transfer	enabled	initiates a Transfer Group transfer every time a trigger event occurs
Transfer Group Pointer Reset	disabled	Ignore trigger during ongoing transfer
Trigger Event	always	Transfer Group can be triggered by software
Trigger Source	disabled	Transfer Group can be triggered by software

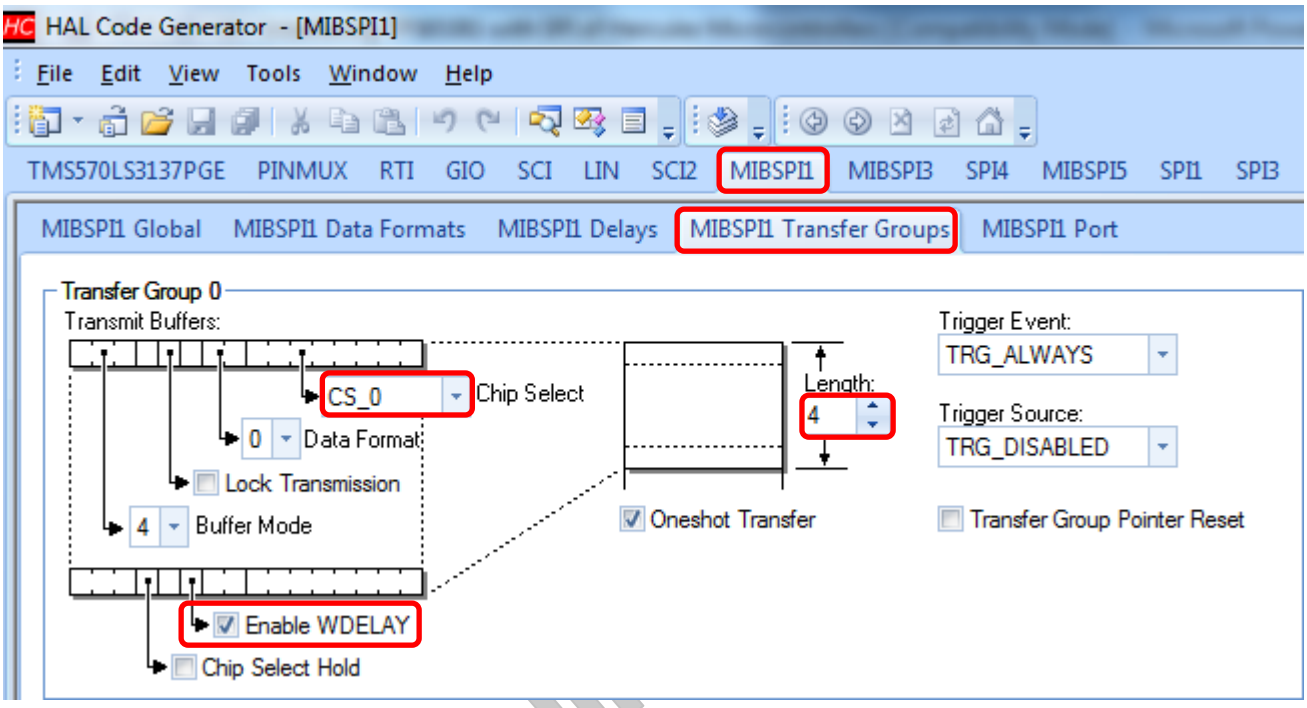
Table 6 RAM Transfer Buffer Parameter Settings

Parameter	***Setting***	***Comment***
Chip Select	0	Use MibSPI chip select 0. If other chip select should be used, the selection has to be accordingly
Data Format	0	Data format 0 is used (see Figure 2 SPI Data Format)
Lock Transmission	disabled	Chip select gets inactive after group transfer
Buffer Mode	always	Transfer Group is transferred on trigger, even if buffers are not updated
Enable WDELAY	yes	Uses WDELAY timing (see Figure 2 SPI Data Format)
Chip Select Hold	disabled	Chip select gets inactive after each buffer

Preliminary

The setup example as shown in following Figure 6 uses 1 transfer group with a length of 4 buffers.

Figure 6 MibSPI Transfer Group Setup Example



This example uses Transfer Group 0. This means the first MibSPI buffer to be transferred is buffer 0. Since the Transfer Group length is set to 4, buffer 0 to buffer 3 gets assigned to Transfer Group 0 and will be transferred, as soon as Transfer Group 0 gets triggered.

For more detailed information about buffer assignment to MibSPI transfer groups, please refer to the Technical Reference Manual (TRM) of the used Hercules device.

```
    /** - initialize transfer groups */
    mibspiREG1->TGCTRL[0U] = (1U << 30U)        /* oneshot */
                         | (0U << 29U)          /* pcurrent reset */
                         | (TRG_ALWAYS << 20U)  /* trigger event */
                         | (TRG_DISABLED << 16U)  /* trigger source */
                         | (0U << 8U);          /* start buffer */

...

    /** - initialize buffer ram */
    {
        i = 0U;

        if (4U > 0U)
        {
            while (i < (4U-1U))
            {
                mibspiRAM1->tx[i].control = (4U << 13U)  /* buffer mode */
                                          | (0U << 12U)     /* chip select hold */
                                          | (0U << 11U)     /* lock transmission */
                                          | (0U << 8U)      /* data format */
                                          |(uint32)((~(uint32)((0xFFU) ^ CS_0) & 0xFFU));  /* chip select */


                i++;
            }
            mibspiRAM1->tx[i].control = (4U << 13U)  /* buffer mode */
                                        | (0U << 12U)   /* chip select hold must be zero for last buffer */
                                        | (1U << 10U)   /* enable WDELAY */
                                        | (0U << 8U)    /* data format */
                                        |(uint32)((~(uint32)((0xFFU) ^ CS_0) & 0xFFU));  /* chip select */


            i++;
        }
```

Preliminary

The above code is automatically generated by HalCoGen. As current HalCoGen versions only allow to control the WDELAY setting for the last transfer buffer, a WDELAY setting for all other transfer buffers must be added to be able to communicate with the SPI of the TPS65381.

The following example code shows the required code sequence to perform the additional WDELAY settings. The code needs to be added to function ‘mibspiInit()’ in the source file ‘mibspi.c’.

```
/* USER CODE BEGIN (3) */

/* =========== Start of necessary HalCoGen changes to setup WDELAY ==========*/

/* Stop MIBSPI1 */
mibspiREG1->GCR1 &= ~(1U << 24U);

/* Enable WDELAY in the TX Ram buffers */
for(i=0; i < 3; i++)
{
        mibspiRAM1->tx[i].control |= (1U << 10U);    /* enable WDELAY */
}

/* Restart MIBSPI1 */
mibspiREG1->GCR1 = (mibspiREG1->GCR1 & 0xFEFFFFFFU) | (1U << 24U);

/* =========== End of necessary HalCoGen changes to setup WDELAY ==========*/

/* USER CODE END */
```

3 TPS65381 SPI Status Information

Preliminary

The TPS65381 responses to each SPI transfer with an 8 bit status response in the next SPI transfer cycle. This status information is located in the first byte of the SPI response. The following chapters 4.1 and 4.2 show TPS65381 responses to different SPI transfers.

NOTE: The status response is always valid for the previous SPI transfer. The status response bits, which are part of the very first SPI transfer after TPS65381 reset, should be ignored.

The SPI status response (STAT) is defined as follows:

- Bit[7] : 1
- Bit[6] : 0
- Bit[5] : 1
- Bit[4] : 0
- Bit[3] : SPI write access (during previous SPI frame command phase)
- Bit[2] : SPI SDO error (during previous SPI frame)
- Bit[1] : Data phase parity (during previous SPI frame)
- Bit[0] : Invalid SPI transfer

4 SPI Communication Examples

This chapter shows some read and write accesses to the TPS65381 using SPI or MibSPI.

Each SPI transfer is based on an 8-bit SPI command followed by an 8 bit data for write accesses or an 8 bit dummy for read accesses. The according responses of the TPS65381 are shown by screenshots and are explained in detail.

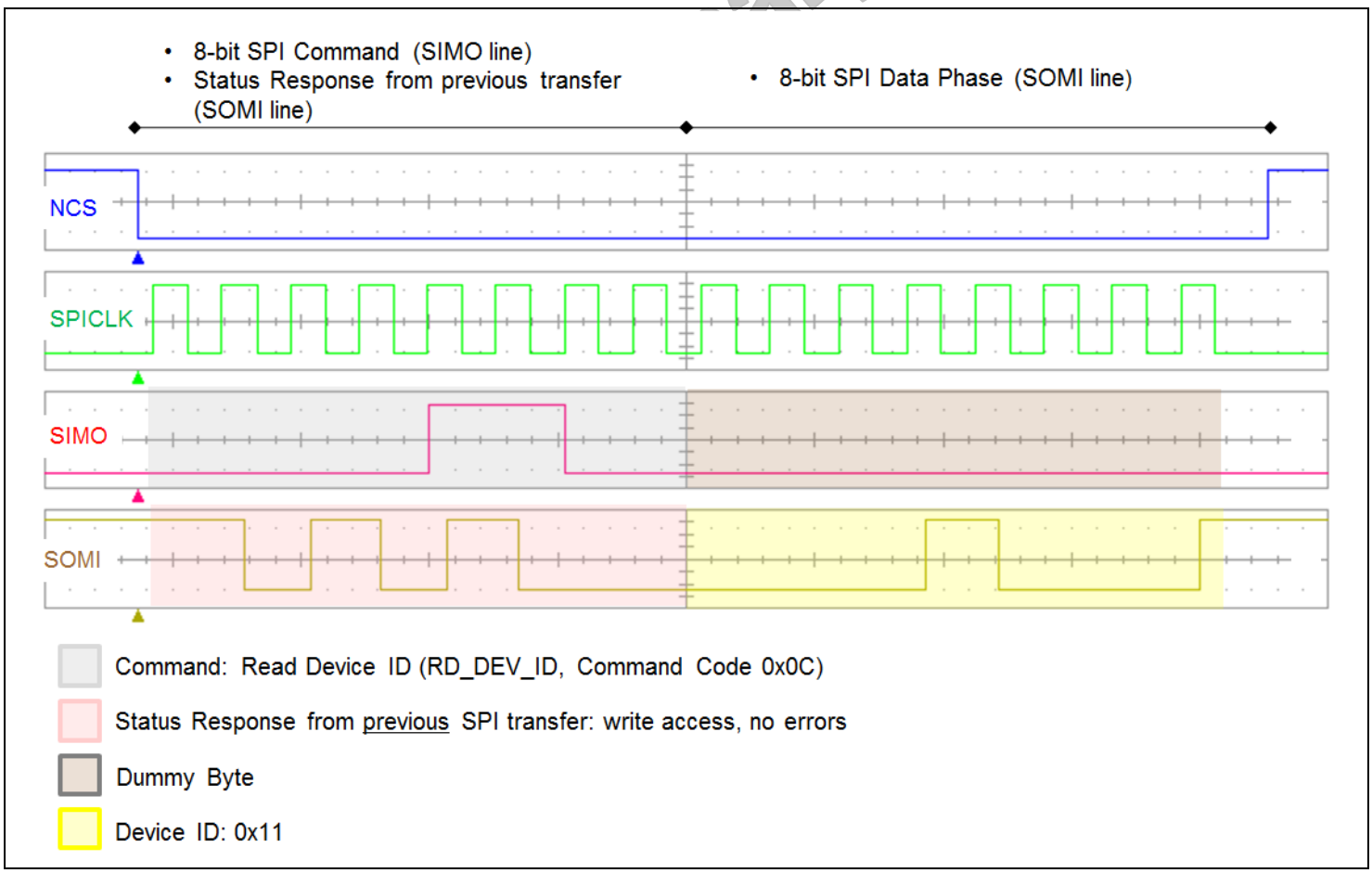
For more information concerning specific details, like TPS65381 SPI command codes or the status responses, please refer to the TPS65381 documentation.

4.1 Reading the Device Revision Number

The 16 bit SPI read transfer to read out the device revision number can be divided in following byte (8 bit) portions:

- First byte:
  - SIMO line (master): Command code (0x0C) for reading the device revision register (RD_DEV_REV)
  - SOMI line (slave): Status response (here: 0xA8) from the previous SPI transfer
- Second byte:
  - SIMO line (master): Dummy data word (0x00)
  - SOMI line (slave): Device revision (0x11)

Figure 7 SPI Read Transfer

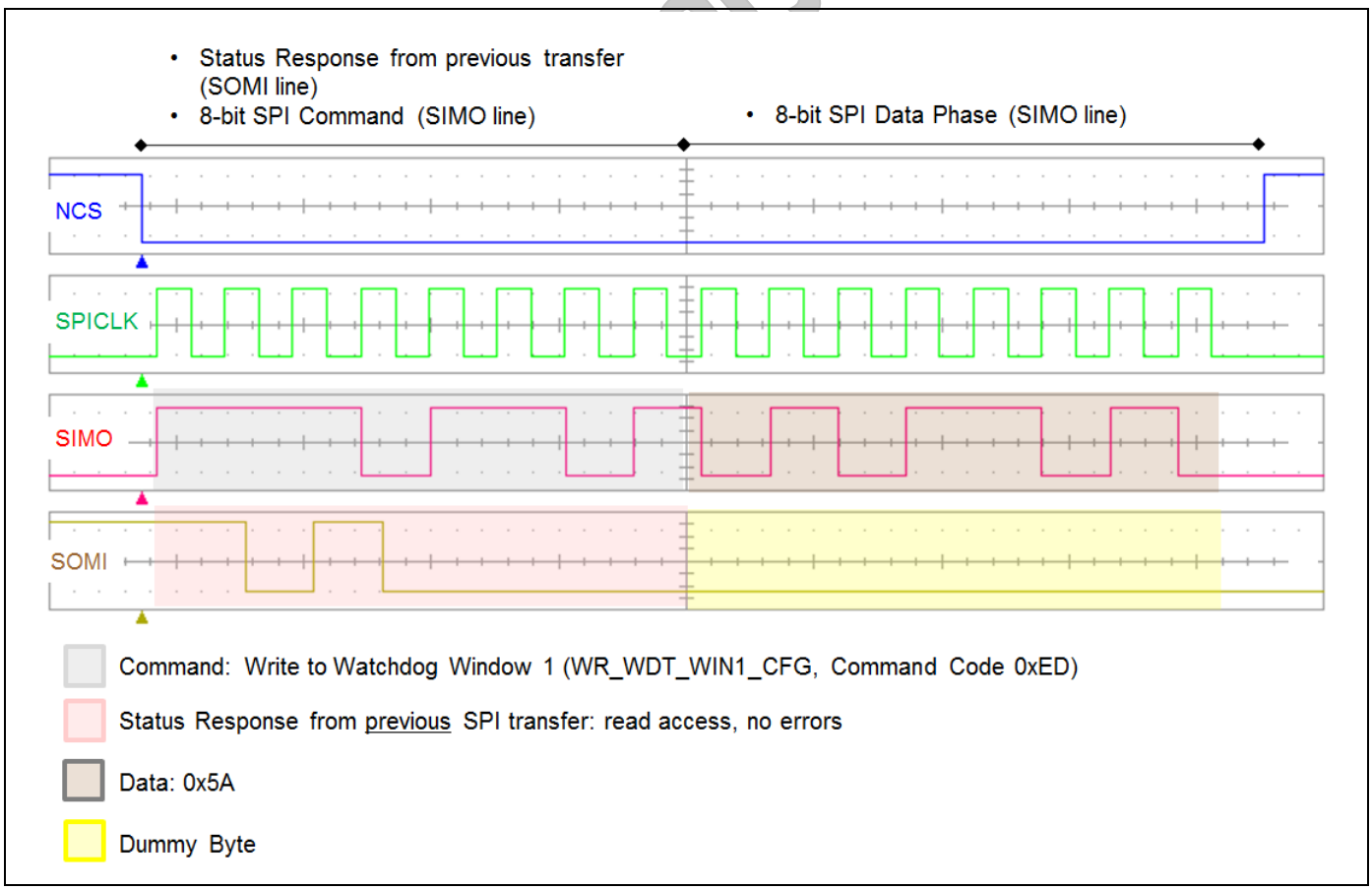


4.2 Set Watchdog Open Window Duration

The 16 bit SPI write transfer to set the watchdog open window duration can be divided in following byte (8 bit) portions:

- First byte
  - SIMO line (master): Command code (0xED) for writing to Watchdog Time Window1 Configuration Register (WR_WDT_WIN1_CFG)
  - SOMI line (slave): Status response (here: 0xA8) from the previous TPS65381 SPI transfer
- Second byte:
  - SIMO line (master): Data (0x5A) to be written to the Watchdog Time Window1 Configuration Register
  - SOMI line (slave): Dummy response (0x00)

Figure 8 SPI Write Transfer



4.3 Send Open Window Watchdog Tokens

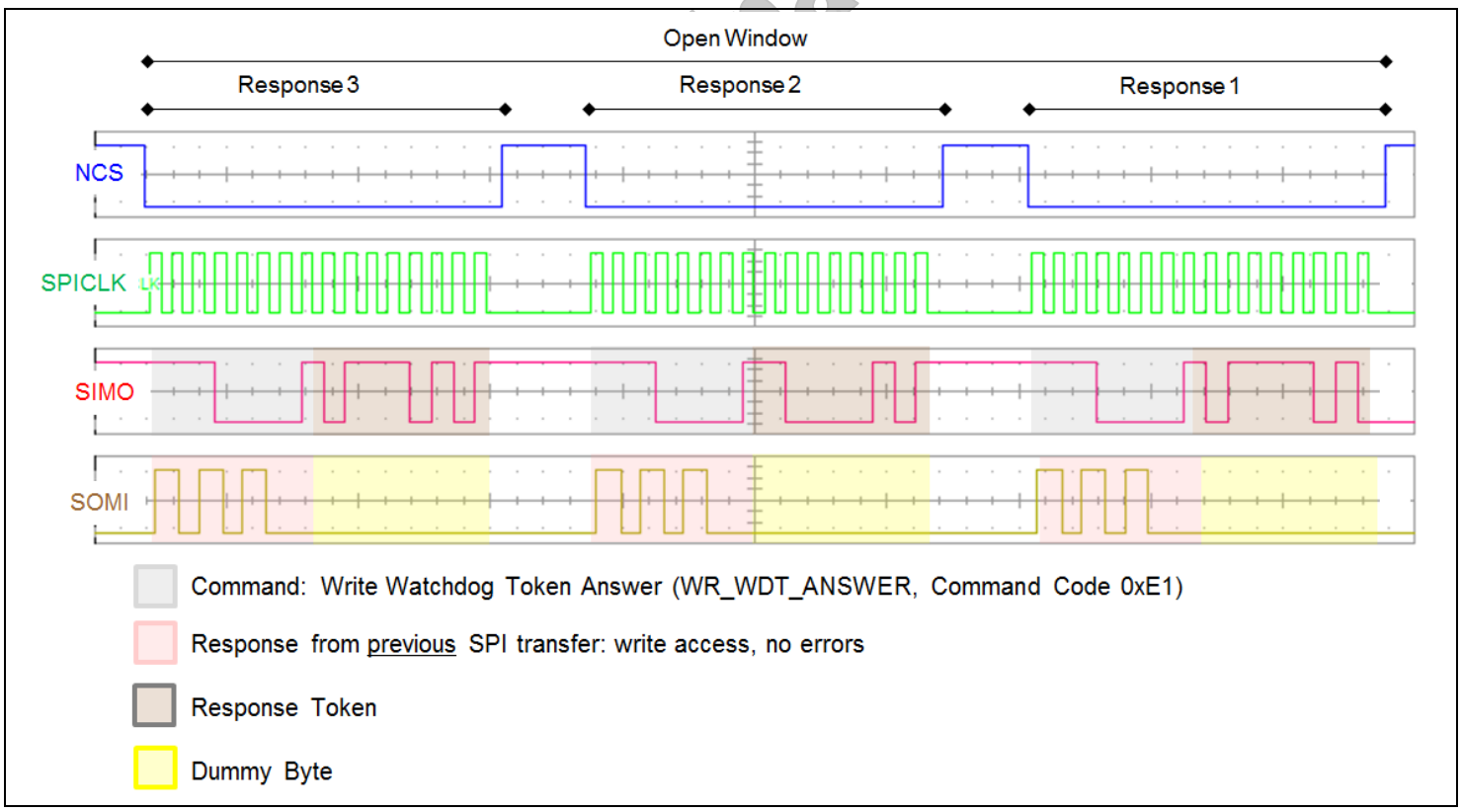
The total windowed watchdog token sequence consists of 3 SPI frames of 2 bytes each.

For reducing the CPU load on the microcontroller side, the calculated 3 watchdog tokens for the open window can be sent by using one MibSPI Transfer Group with a length of 3 to cover the 3 SPI frames.

Each SPI frame has the same structure and includes the following information:

- First byte
  - SIMO line (master): Command code (0xE1) for writing to Watchdog Token Answer Register (WR_WDT_ANSWER)
  - SOMI line (slave): Status response (here: 0xA8) from the previous TPS65381 SPI transfer
- Second byte:
  - SIMO line (master): Data 0x(XX) the current token, to be written to the Watchdog Token Answer Register
  - SOMI line (slave): Dummy response (0x00)

Figure 9 Watchdog Token Sequence



5 Multiple SPI Slave Network

The TPS65381 can be used in a multiple SPI slave network. The SPI master must drive the chip select signal to the TPS65381 active low during communication. If the chip select is high, the TPS65381 ignores data traffic on the SIMO line.

NOTE:
Due to a limitation the internal SPI error detection of the TPS65381 flags an SDO error on SPI frame occurrence, if the chip select signal is inactive high. This results in a SDO error flag in the first SPI frame of the next SPI transfer with the TPS65381. This SDO error flag needs to be ignored by the SPI master software.

Preliminary