MLX83203-2,MLX83100 Automotive Pre-Driver

Custom SPI Interface for Access to EEPROM in Application

Stefan Poels

SEPTEMBER 14, 2016

Transportstraat 1

3980 Tessenderlo

Phone: +32 13 67 07 95

Mobile: +32 491 15 74 18

Fax: +32 13 67 07 70

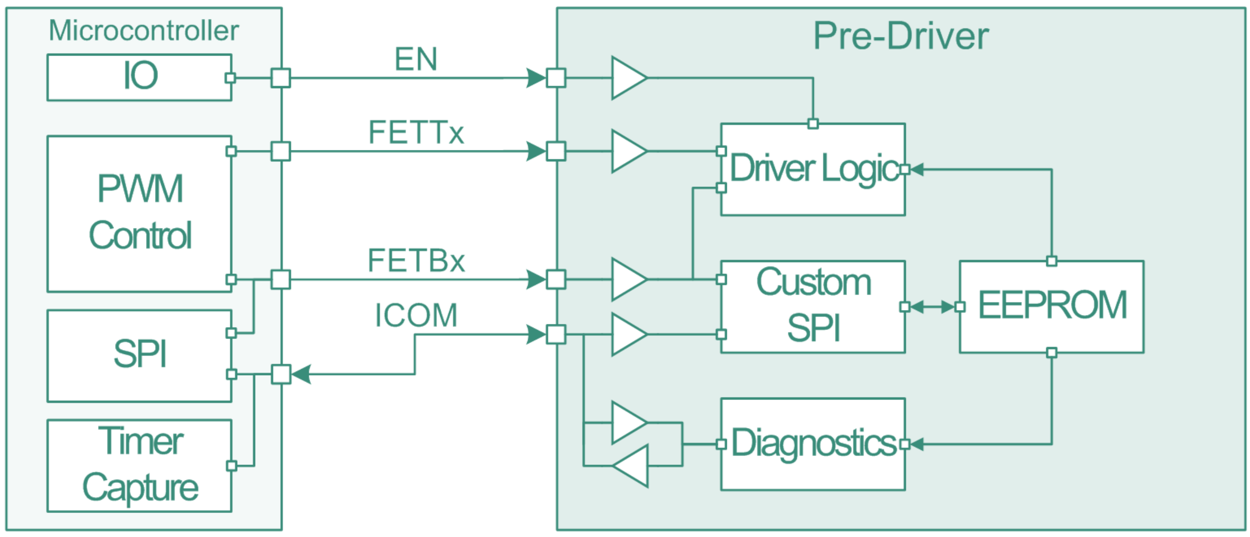
www.melexis.com

Contents

1. Scope

This application note is to be used in combination with the corresponding pre-driver datasheet.

The goal of this application note is to describe the custom SPI interface for access to EEPROM. This interface allows the user to check the default configuration by reading the different bytes. It is also explained how the user can change the default configuration and can verify correct communication. In the end some examples with scope plots are available to show correct and incorrect communication.



2. Description

The MLX83203-02 (MLX83100) are three (two) phase pre-drivers, also called ‘bridge’ or ‘gate’ driver, IC with integrated current sense amplifier. This device is used to drive brushless (brushed) DC motors in combination with a microcontroller and six (four) discrete power N-FETs.

The device is able to control six (four) external N-FETs. The high side gate drivers are supplied via bootstrap circuits. The trickle charge pump allows 100% PWM operation despite the use of bootstrap capacitors.

The device comprises various monitoring and protection functions, including under voltage and over voltage detection at multiple internal voltage nodes, over temperature detection, drain-source and gate-source voltage monitoring of the external N-FETs.

An integrated fast, high-bandwidth, low offset current sense amplifier allows for precise torque control with programmable gain selection.

The pre-driver provides an EEPROM for configurability, avoiding the need for a high pin-count package and/or external components for configuration. The configuration allows the user to optimize the pre-driver’s operation for different applications by configurability of the current sense amplifier and protection and diagnostic functions. This application note describes how the configuration can be done in the application.

3. Custom SPI to Read/Program EEPROM

The pre-driver provides an EEPROM for configuration of the current sense amplifier and protection and diagnostic functions for optimization to the application requirements. The configuration can be done at customer production by using the PTC-04, or in the application by the microcontroller via a custom program interface.

The EEPROM is composed of 6 bytes for user configurability. The first two bytes are not used for the internal configuration of the pre-driver, and can thus be used by the user for traceability purposes. The other 4 bytes are used for configuration of the current sense amplifier and configuration of the diagnostics.

3.1. "SPI Program Mode"

The EEPROM memory can be accessed through a custom SPI interface. It allows the user to read/program the EEPROM by the microcontroller in the application. This custom interface re-uses the low-side driver pins.

Since the same pins are used for both reading/writing the EEPROM and for controlling the motor, the EEPROM is only accessible when the motor is not running. Furthermore it is necessary to apply a certain sequence of conditions before the pre-driver will enter the "SPI Program Mode". Once in this mode, the EEPROM can be accessed for reading and writing, until the IC enters "Normal Mode" again and motor operation is possible.

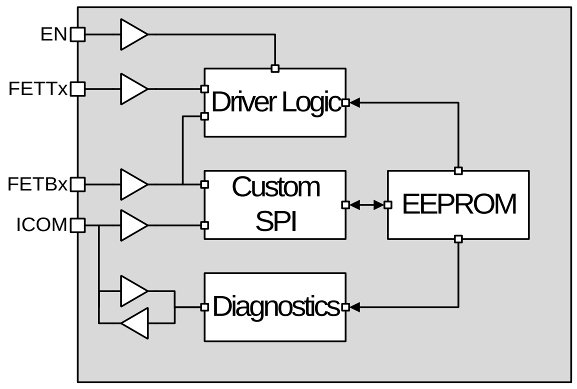


Figure 3-1 Custom SPI Read/Program Interface

3.1.1. SPI Signals

When the pre-driver is in SPI mode the EEPROM is accessible via the SPI port. The 16 bit SPI shift register is controlled by 4 signals:

SPI Signal	Pre-Driver Pin	Color on scope plots
CSB	ICOM	Yellow
MOSI	FETB3	Blue
CLK	FETB2	Purple
MISO	FETB1 (MISO)	Green

Table 3-1 SPI signals

3.2. Entering “SPI Program Mode”

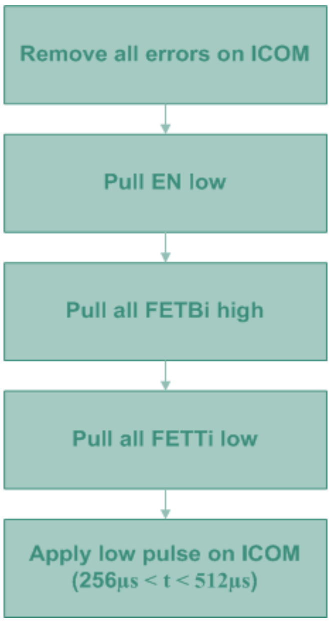


Figure 3-2 Entering the "SPI Program Mode"

The pre-driver will enter from “Normal Mode” into “SPI Program Mode” when all below conditions are satisfied.

Since ICOM is used as CSB signal, ICOM should not be communicating any errors. So first any pending errors have to be acknowledged.

In order not to have the SPI communication disturb the motor, all gate driver outputs should be disabled by pulling EN low. Next the driver inputs need to be disabled by pulling all PWM input signals for the high side MOSFETs low (FETTx), and PWM input signals for the low side MOSFETs high (FETBx).

If all these conditions are satisfied, a low pulse of time t_{SPI_ISU} (specified in the datasheet) on ICOM will force the pre-driver to enter the “SPI Program Mode”.

After applying this pulse, the SPI communication can be checked by transmitting a (dummy) MOSI frame and checking if the pre-driver responds on the MISO line.

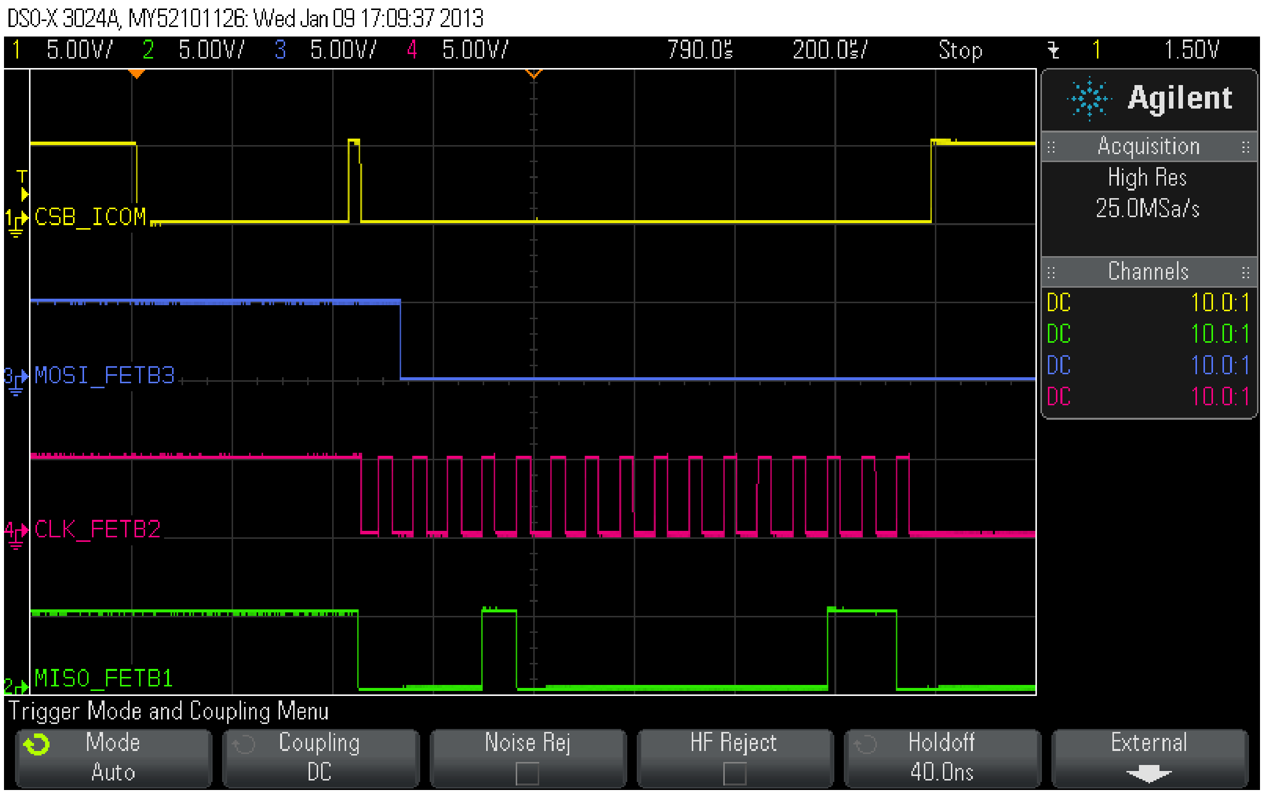


Figure 3-3 Scope plot of the Pre-Driver Entering the “SPI Program Mode”

3.3. Exiting “SPI Program Mode”

The pre-driver will exit the “SPI Program Mode” when the enable input EN is pulled high. Similar to when the pre-driver comes out of POR, after leaving the “SPI Program Mode” the pre-driver will be blocked until the data has been copied to the registers. This means before entering “Normal Mode” any ongoing EEPROM write will be completed and the EEPROM will be copied into the registers. During this time ICOM will be kept low. ICOM returning to its default high state signals when the pre-driver is ready for normal operation.

3.4. Protocol

Once the IC is in “SPI Program Mode” the microcontroller can read/program the EEPROM, following the protocol depicted below.

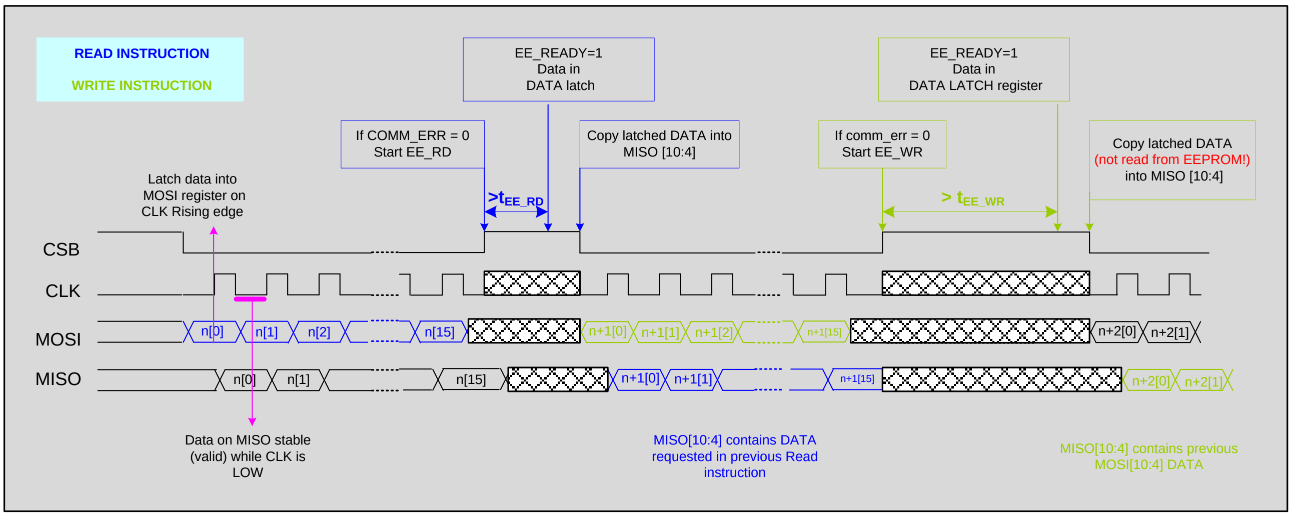

READ INSTRUCTION
WRITE INSTRUCTION
EE_READY=1 Data in DATA latch
EE_READY=1 Data in DATA LATCH register
If COMM_ERR = 0 Start EE_RD
Copy latched DATA into MISO [10:4]
If comm_err = 0 Start EE_WR
Copy latched DATA (not read from EEPROM!) into MISO [10:4]
Latch data into MOSI register on CLK Rising edge
>tEE_RD
> tEE_WR
CSB
CLK
MOSI
MISO
n[0]
n[1]
n[2]
n[15]
n+1[0]
n+1[1]
n+1[2]
n+1[15]
n+2[0]
n+2[1]
Data on MISO stable (valid) while CLK is LOW
MISO[10:4] contains DATA requested in previous Read instruction
MISO[10:4] contains previous MOSI[10:4] DATA

Figure 3-4 SPI Protocol (LSB first)

3.5. Registers Description

MOSI [15:0]							
Bit [15]	Bit [14]	Bit [13]	Bit [12]	Bit [11]	Bit [10]	Bit [9]	Bit [8]
MOSI_PAR	x	x	CMD [1:0]		MOSI_DATA [7:5]		
Bit [7]	Bit [6]	Bit [5]	Bit [4]	Bit [3]	Bit [2]	Bit [1]	Bit [0]
MOSI_DATA [4:1]				x	ADDRESS [2:0]		
MISO[15:0]							
Bit [15]	Bit [14]	Bit [13]	Bit [12]	Bit [11]	Bit [10]	Bit [9]	Bit [8]
MISO_PAR	COMM_ERR	EE_READY	CMD [1:0]		MISO_DATA [7:5]		
Bit [7]	Bit [6]	Bit [5]	Bit [4]	Bit [3]	Bit [2]	Bit [1]	Bit [0]
MISO_DATA [4:1]				x	ADDRESS [2:0]		

Table 3-2 MOSI / MISO Registers: Frame Description

3.5.1. MOSI-frame

The 16-bit MOSI frame transmitted by the microcontroller to the pre-driver consists of the 7-bit data that needs to be written in EEPROM (not important for read command), the 3-bit address that needs to be read/written and the 2-bit read/write command. The MOSI frame is then completed with some dummy bits and by calculating and adding an odd parity bit in the end.

3.5.2. MISO-frame

The MISO frame is transmitted by the pre-driver to the microcontroller. When the previous instruction was a write instruction, the 16-bit MISO frame contains the 7-bit data that needed to be written. This can be used as a first validation of the write cycle. When the previous instruction was a read instruction, the 16-bit MISO frame contains the 8-bit data read from EEPROM.

In both cases the MISO frame contains the address it used to read/write EEPROM and the read/write command itself. Further the MISO frame gives some communication diagnostics, see below Table 3-3.

Bit	Description
ADDRESS	Address of the byte in EEPROM that needs to be read/programmed.
MOSI_DATA[7:1]	For write command, the data that needs to be written. Don't care for any read command.
MISO_DATA[7:1]	After a write it returns the written data, after a read instruction the data read from EEPROM.
CMD [1:0]	Read/Write command 00: EE_RD: Read command 01: EE_WR: Write command 10: EE_RDAW1 11: EE_RDAW2
EE_READY	Reading/writing the EEPROM takes a certain time, specified by t_{EE_RD} and t_{EE_WR} respectively. These times define the minimum time CSB (ICOM) has to remain high between two SPI-frames in order to finish the read/ write action. As soon as the read/write action starts, the EE_READY bit is reset. After completion of the read/write action the bit is set. If the read/write delay between SPI-frames was long enough to execute the read/write action, the EE_READY bit will thus be set, signaling the read/write action was finished. If the time was too short, the bit will still be '0'.
COMM_ERR	This bit indicates if the previous MOSI-frame was received correctly. If no communication error occurred the bit will be reset, and the read/write action was started as soon as CSB was pulled high. If a communication error occurred in the previous MOSI-frame the read/write command was not executed. Possible communication errors are: Odd parity bit was not correct Number of clock periods was not equal to 16
MOSI_PAR, MISO_PAR	Odd parity bit of the current MOSI/MISO frame.

Table 3-3 MOSI/MISO Registers: Bit Description

3.6. Communication

3.6.1. Transceiving a MOSI/MISO-frame

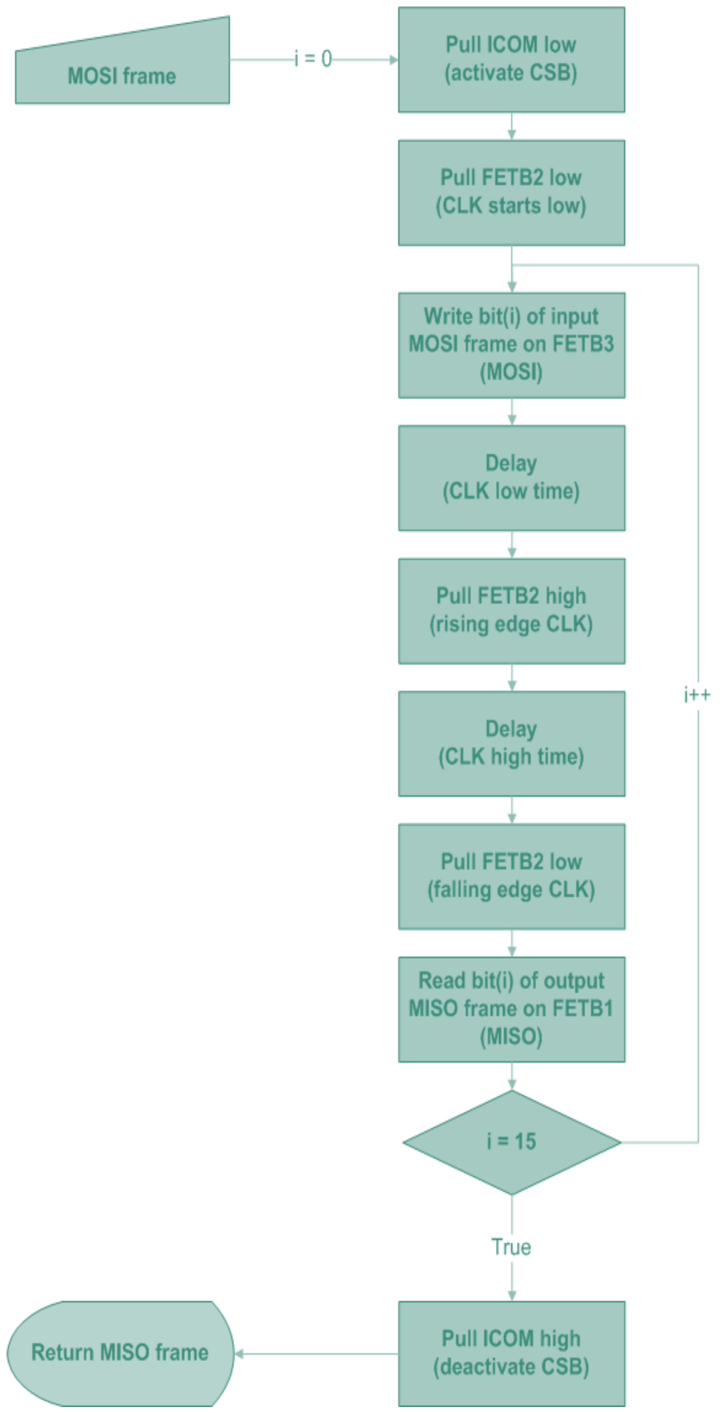

MOSI frame
i = 0
Pull ICOM low (activate CSB)
Pull FETB2 low (CLK starts low)
Write bit(i) of input MOSI frame on FETB3 (MOSI)
Delay (CLK low time)
Pull FETB2 high (rising edge CLK)
Delay (CLK high time)
Pull FETB2 low (falling edge CLK)
Read bit(i) of output MISO frame on FETB1 (MISO)
i = 15
i++
True
Pull ICOM high (deactivate CSB)
Return MISO frame

Figure 3-5 Transceiving a MOSI/MISO frame

Once the microcontroller prepared the MOSI frame (see paragraph 3.5.1) it can transmit it to the pre-driver.

During each SPI cycle CSB (ICOM) needs to be pulled low. The clock (FETB2) starts with a low clock signal.

The SPI communication starts by transmission of the first bit of the MOSI frame on FETB3.

Next the clock (FETB2) is pulled high and from this moment the first MOSI bit is valid and will be read by the pre-driver.

The clock is kept high for half a clock cycle to give the pre-driver the time to read the first bit.

Then the clock is pulled low again and kept low for another half a clock cycle. During this low time, the first MISO bit from the pre-driver is valid and can be read by the MCU.

This sequence of writing the MOSI and reading the MISO, while setting the clock accordingly, is repeated for the other 15 bits of the SPI frame.

Once all 16 bits of the MOSI frame are transmitted and all 16 bits of the MISO frame are received the SPI cycle is over and CSB needs to be deactivated by releasing ICOM.

On the rising edge of ICOM the reading/writing of EEPROM will start, if no communication error occurred.

The received MISO frame needs to be decoded, according to paragraph 3.5.2, and can then be used to validate the read/write instruction.

3.6.2. Read Instruction

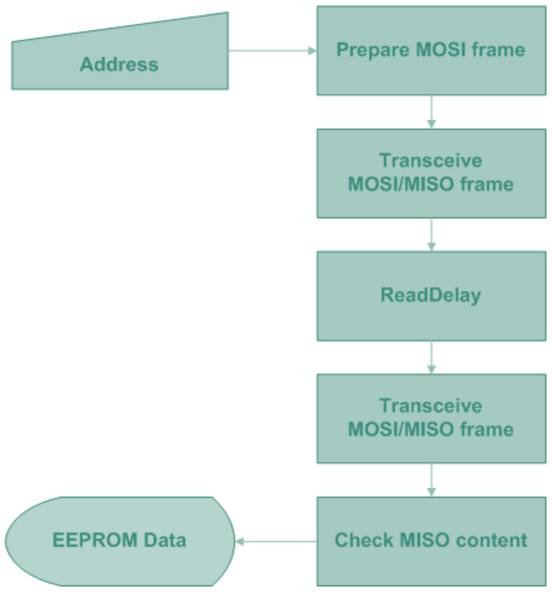


Figure 3-6 Read Instruction

In order to read one or more bytes of the EEPROM, the microcontroller has to transceive at least two SPI frames. The first frame is to send the read command, the second frame is for the pre-driver to send back the result of this read instruction.

The first MOSI(N)-frame is to be composed of the read command and address to read from, completed with the correct odd parity bit , according to paragraph 3.5.1. Next this MOSI(N)-frame is transmitted to the pre-driver to start the read instruction.

After receive of the read instruction in MOSI(N), and if no communication error is detected, the pre-driver starts the read instruction at the specific address as soon as CSB (ICOM) is pulled high. If CSB is kept high long enough for the pre-driver to execute and finish the read action, it will transmit the read data on the next MISO(N+1)-frame. Otherwise it will report the read instruction is still on-going. This EEPROM read delay is specified by t_{EE_RD} in the datasheet.

During the second the SPI-frame the pre-driver will send the result of the read instruction on MISO(N+1). The MOSI(N+1) message that is send is not important. This means that if the user wants to read a single byte of the EEPROM, the same MOSI(N)-frame can be used. However if more bytes need to be read, the MCU can send the read instruction for the next byte to minimize the total time required for reading the whole EEPROM.

The microcontroller can check if the received MISO(N+1) frame is valid to ensure the data is not corrupted. The data is valid if:

- COM_ERR = 0:
  No communication error occurred during the previous MOSI(N) frame, meaning the read command was received correctly.
- EE_READY = 1:
  The read delay was long enough for the pre-driver to finish the read instruction before the falling edge of CSB.
- MISO_PAR = correct:
  The odd parity bit in the MISO(N+1) frame is correct.

To verify correct programming of a byte in EEPROM two different read commands have to checked sequentially:

- EE_RDAW1
- EE_RDAW2

Note that to read a byte from EEPROM one read cycle with the normal EE_RD command is sufficient.

3.6.3. Write Instruction

The pre-driver provides different configuration options through the EEPROM memory. In order to program one of the EEPROM bytes, MOSI(N) frame is to be composed according to paragraph 3.5.1 with the address and data it wants to write, the write command and the correct odd parity bit.

After transmission of this MOSI(N)-frame the pre-driver will start the execution of the write instruction as soon as CSB (ICOM) is pulled high and no communication error has occurred. In order for the pre-driver to finish the write instruction it needs a minimum write delay, specified by t_{EE_WR} in the datasheet.

On the first MISO(N+1)-frame after the write command, it can be checked if the write command was received correctly, and the status of write cycle. The next two SPI-frames are used to validate the write execution.

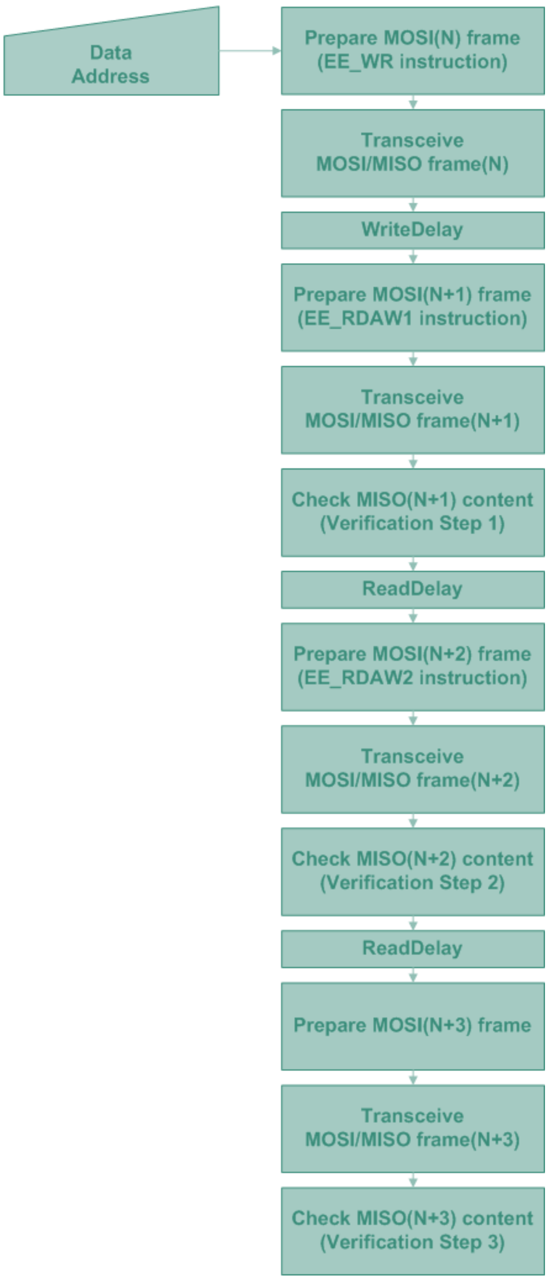

Data Address
Prepare MOSI(N) frame (EE_WR instruction)
Transceive MOSI/MISO frame(N)
WriteDelay
Prepare MOSI(N+1) frame (EE_RDAW1 instruction)
Transceive MOSI/MISO frame(N+1)
Check MISO(N+1) content (Verification Step 1)
ReadDelay
Prepare MOSI(N+2) frame (EE_RDAW2 instruction)
Transceive MOSI/MISO frame(N+2)
Check MISO(N+2) content (Verification Step 2)
ReadDelay
Prepare MOSI(N+3) frame
Transceive MOSI/MISO frame(N+3)
Check MISO(N+3) content (Verification Step 3)

Figure 3-7 Write Instruction

Verification step 1: Using MISO(N+1):

The MISO(N+1) frame is valid if:

- Correct odd parity bit in MISO(N+1)

The write instruction was received correctly if:

- COM_ERR = 0:

The write instruction was finished if:

- EE_READY = 1:

The correct data is used to write EEPROM if:

- MISO_data(N+1) = MOSI_data(N)

Verification step 2: Using MISO(N+2):

After EE_RDAW1 instruction in MOSI(N+1), the data in EEPROM is returned in MISO(N+2). This data is valid if:

- The odd parity bit in MISO(N+2) is correct.
- COM_ERR = 0

The read instruction was finished if:

- EE_READY = 1:

The correct data is in EEPROM if:

- MISO_data(N+2) = MOSI_data(N)

Verification step 3: Using MISO(N+3):

After EE_RDAW2 instruction in MOSI(N+2), the data in EEPROM is returned in MISO(N+3). This data is valid if:

- The odd parity bit in MISO(N+3) is correct.
- COM_ERR = 0

The read instruction was finished if:

- EE_READY = 1:

The correct data is in EEPROM if:

- MISO__data(N+3) = MOSI_data(N)

4. Examples

4.1. Successful Read Instruction

An example of a successful read instruction is shown in Figure 4-1. The received EEPROM data in this example is valid because:

- MISO[14] : COM_ERR = 0: the read command was received correctly
- MISO[13] : EE_READY = 1: the read instruction was completed and thus the MISO-frame contains the requested EEPROM data
- MISO[15] : the odd parity bit is correct: no bit errors in the MISO-frame

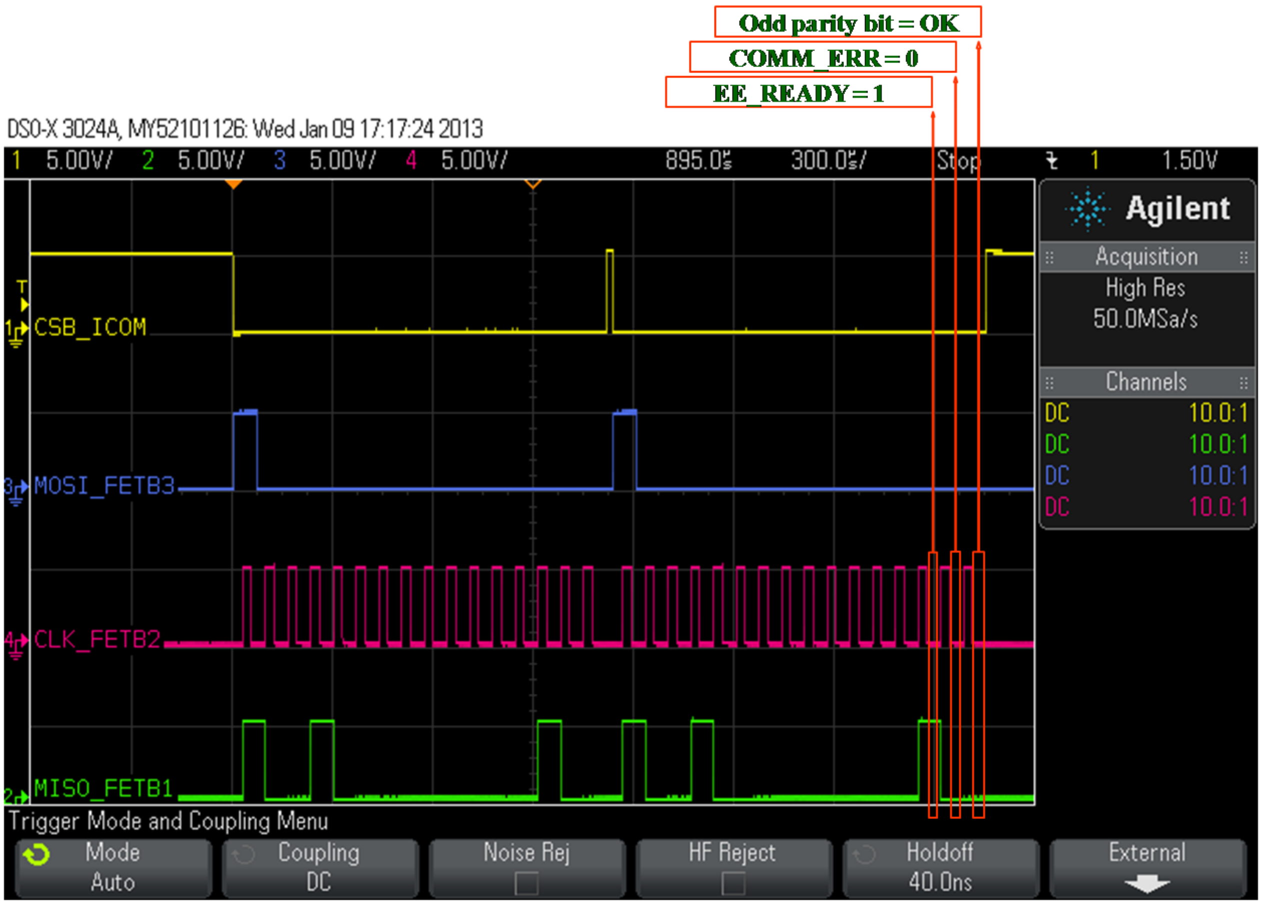


Figure 4-1 Scope plot of a successful SPI read instruction

4.2. Read Delay too Small

If the EEPROM read delay between the first and second SPI cycle is too small, the pre-driver does not have enough time to finish the read instruction. The data transmitted in the MISO(N+1) frame is thus not the requested EEPROM data. This is signaled via:

- MISO[13] : EE_READY = 0: the read instruction is still on-going

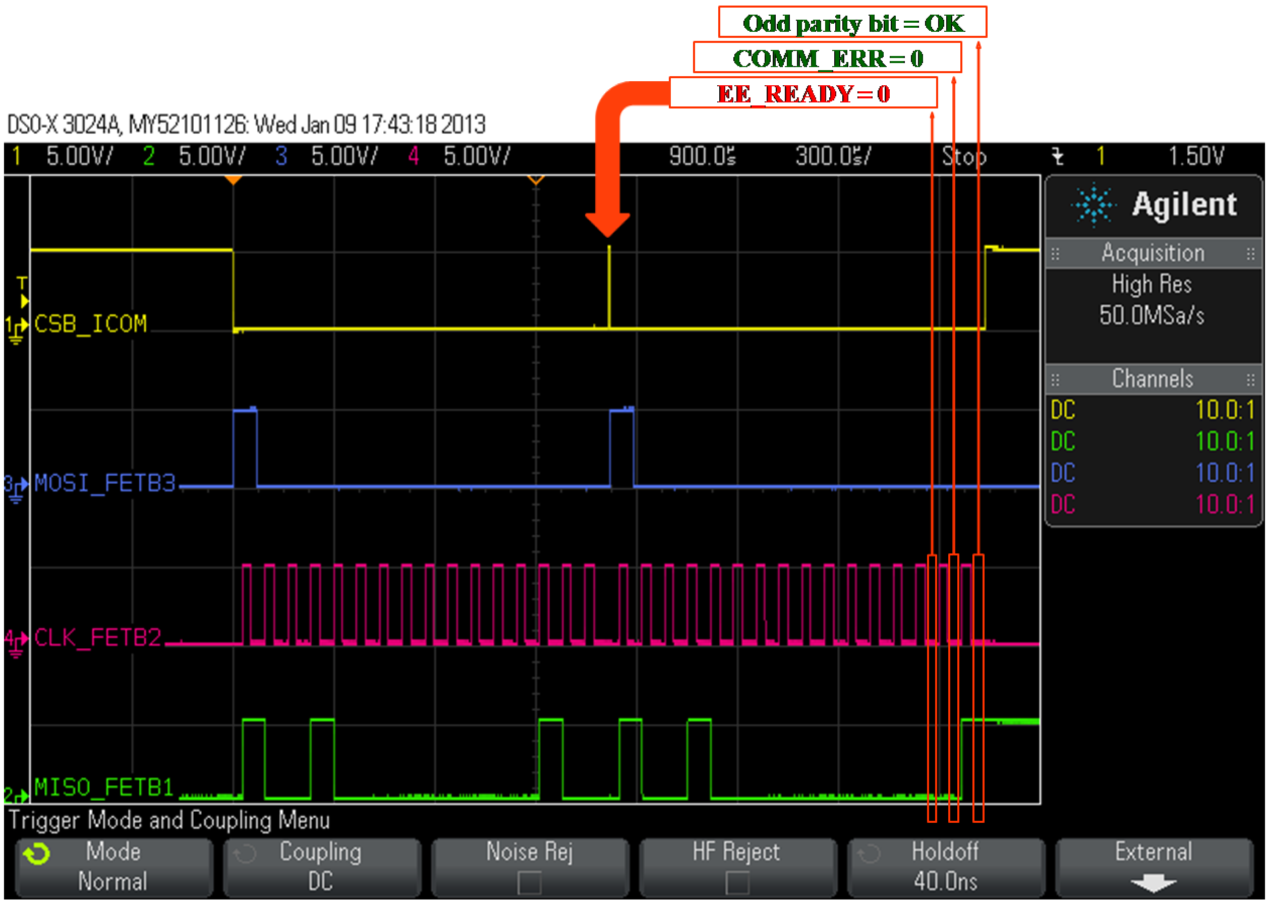


Figure 4-2 Scope plot of an SPI read instruction where the EEPROM read delay is set too small

4.3. Odd Parity Bit Error

One of the bits in a MOSI frame is the odd parity bit. If this odd parity bit is not calculated correctly, the pre-driver will see this as a communication error. The read command will not be executed. This error will be communicated back to the microcontroller in the next MISO(N+1) frame.

- MISO[14] : COM_ERR = 1: the read command was not received correctly and thus the read instruction is not started

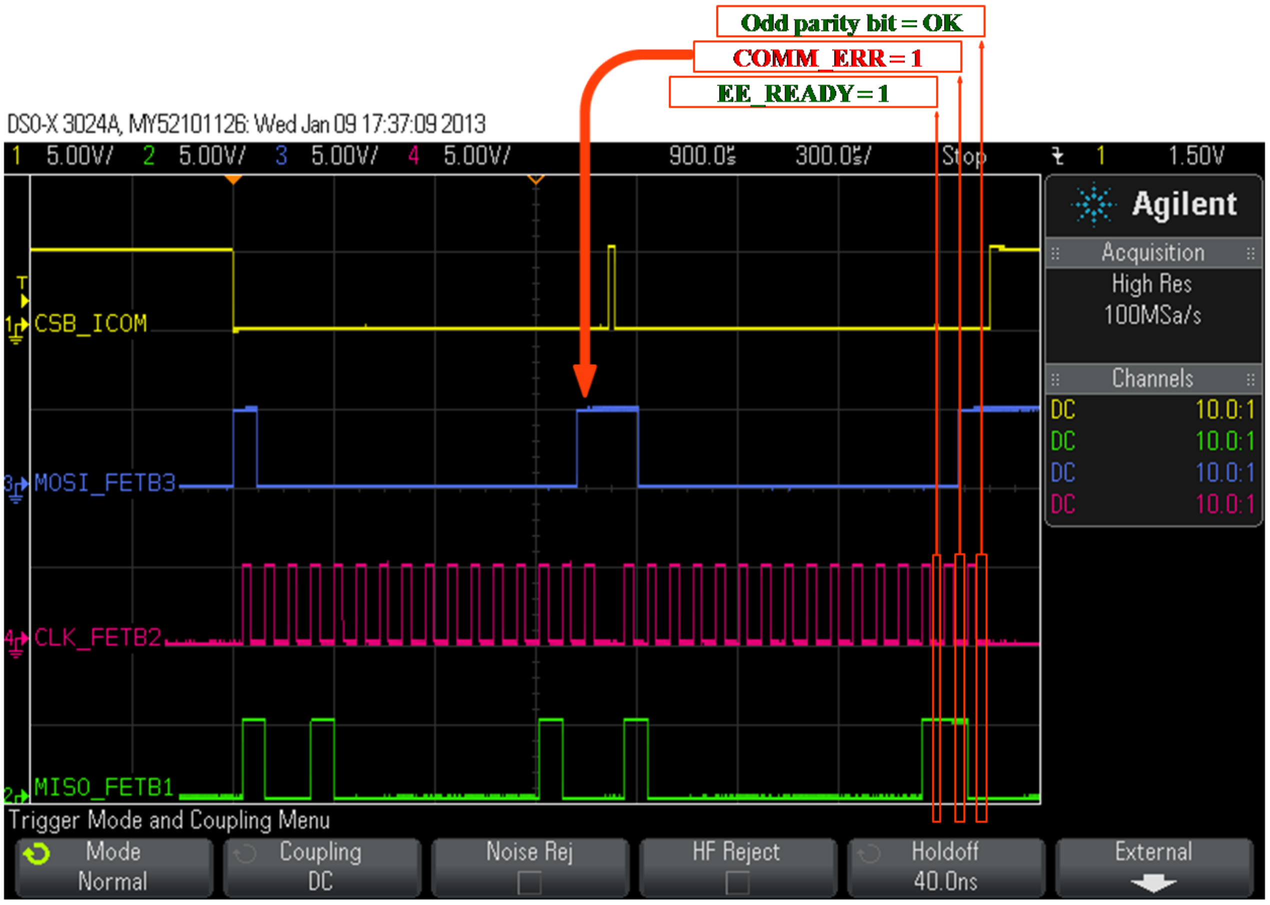


Figure 4-3 Scope plot of an SPI read instruction with wrong odd parity bit

4.4. Wrong Number of Clock Periods

Similar to the previous case, where the odd parity bit was not set correctly, a wrong number of clock periods will again cause a communication error. This error is communicated back to the MCU in the next MISO(N+1) frame:

- MISO[14] : COM_ERR = 1: the read command was not received correctly and thus the read instruction is not started

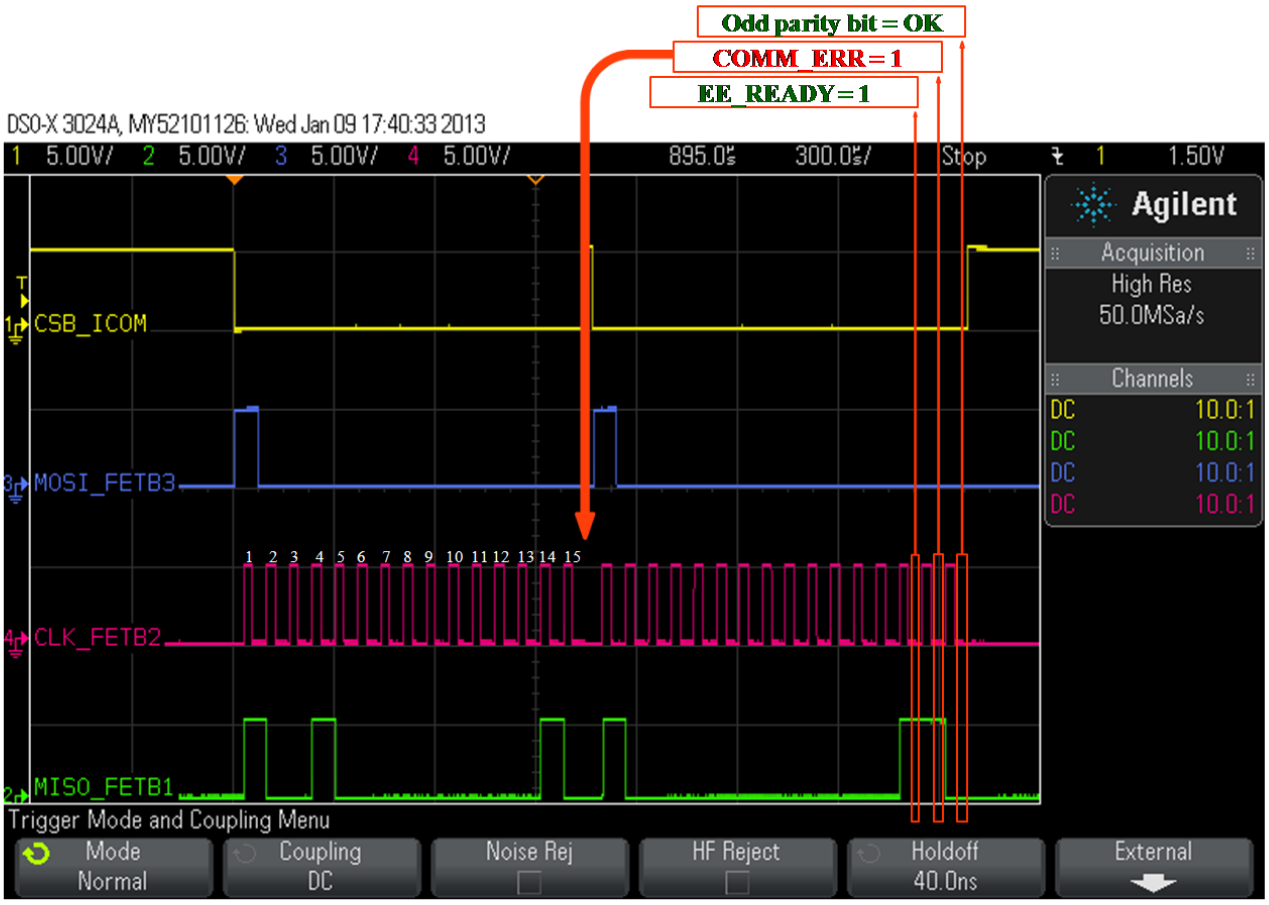


Figure 4-4 Scope plot of an SPI read instruction with a wrong number of clock periods

4.5. Successful Write Instruction

An example of a successful write instruction is shown in Figure 4-5.

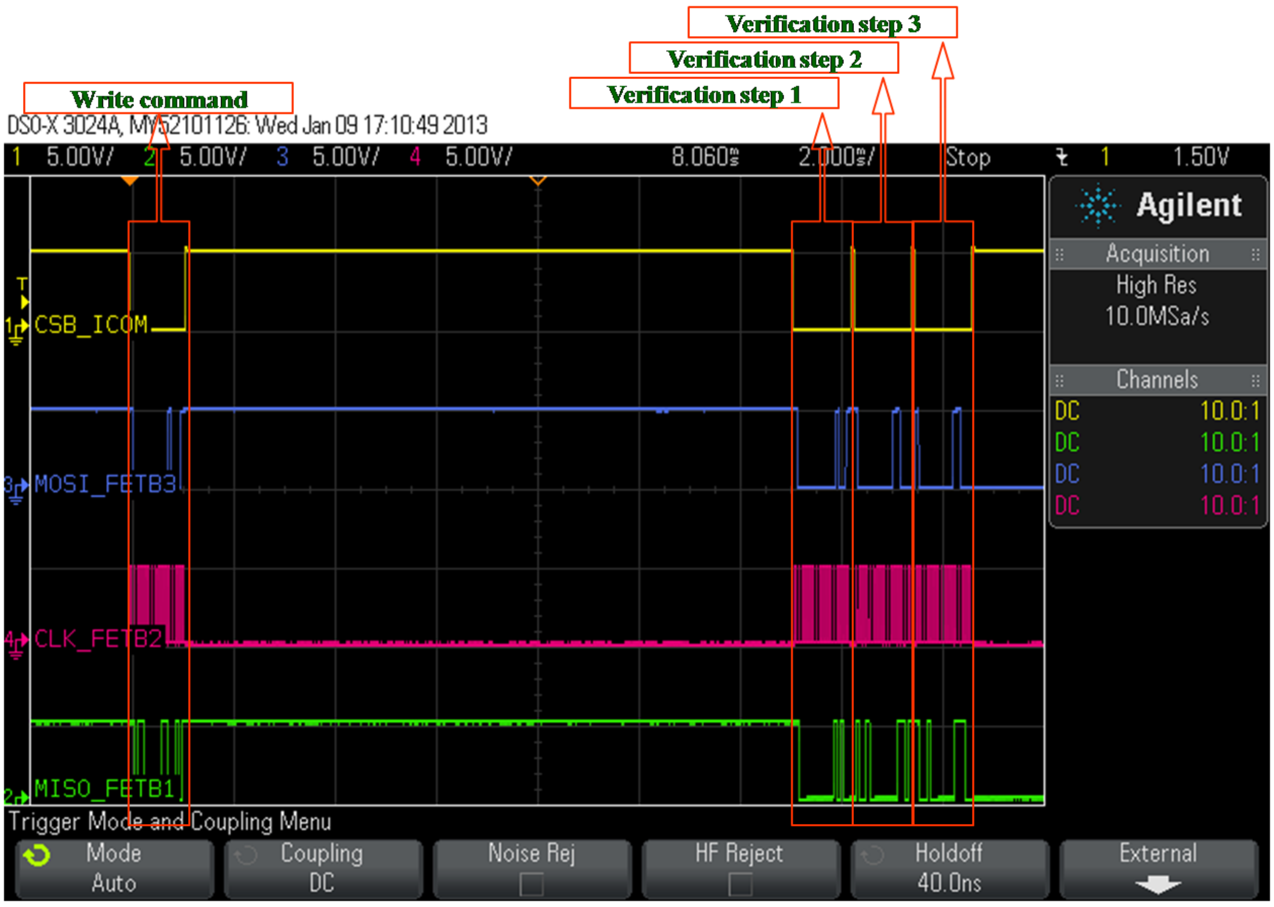


Figure 4-5 Scope plot of a successful SPI write instruction

5. Revision History

Revision	Date	Description
2.0	26-02-13	First release
3.0	22-07-14	General update according to new template
4.0	14-09-16	New Melexis branding

Table 5-1 Revision history

6. Disclaimer