Clean room analysis of the PMS150C programming sequence

Thanks to everyone involved in investigating the Padauk microcontrollers on the EEV and µC.net forums. Please see here for further background and work leading to this analysis:

http://www.eevblog.com/forum/blog/eevblog-1144-padauk-programmer-reverse-engineering

https://www.mikrocontroller.net/topic/461002

Clean Room Analysis Disclaimer

- This report is solely based on analyzing datalogs of the programming sequence as provided here: http://www.eevblog.com/forum/blog/eevblog-1144-padauk-programmer-reverse-engineering/msg2096917/#msg2096917

- No attempts have been made by me to reverse engineer any of the software provided by Padauk.

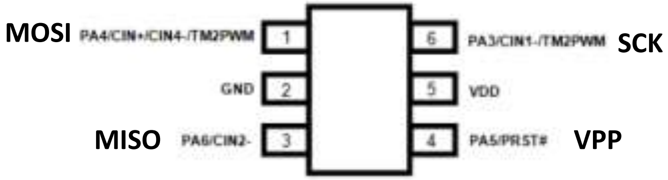
- The information in this document is provided “as is”, without warranty of any kind.

- No attempts have been made so far to verify any of the observations on actual hardware. Please update where neccessary.

Initial analysis of data logs.

From a first glance, the programming interface of the PMS150C seems to be a straight forward SPI interface. MSB first, data is valid on rising edge of clk.

The pinout is as follows:



PMS150C-U06 (SOT23-6 60mil)

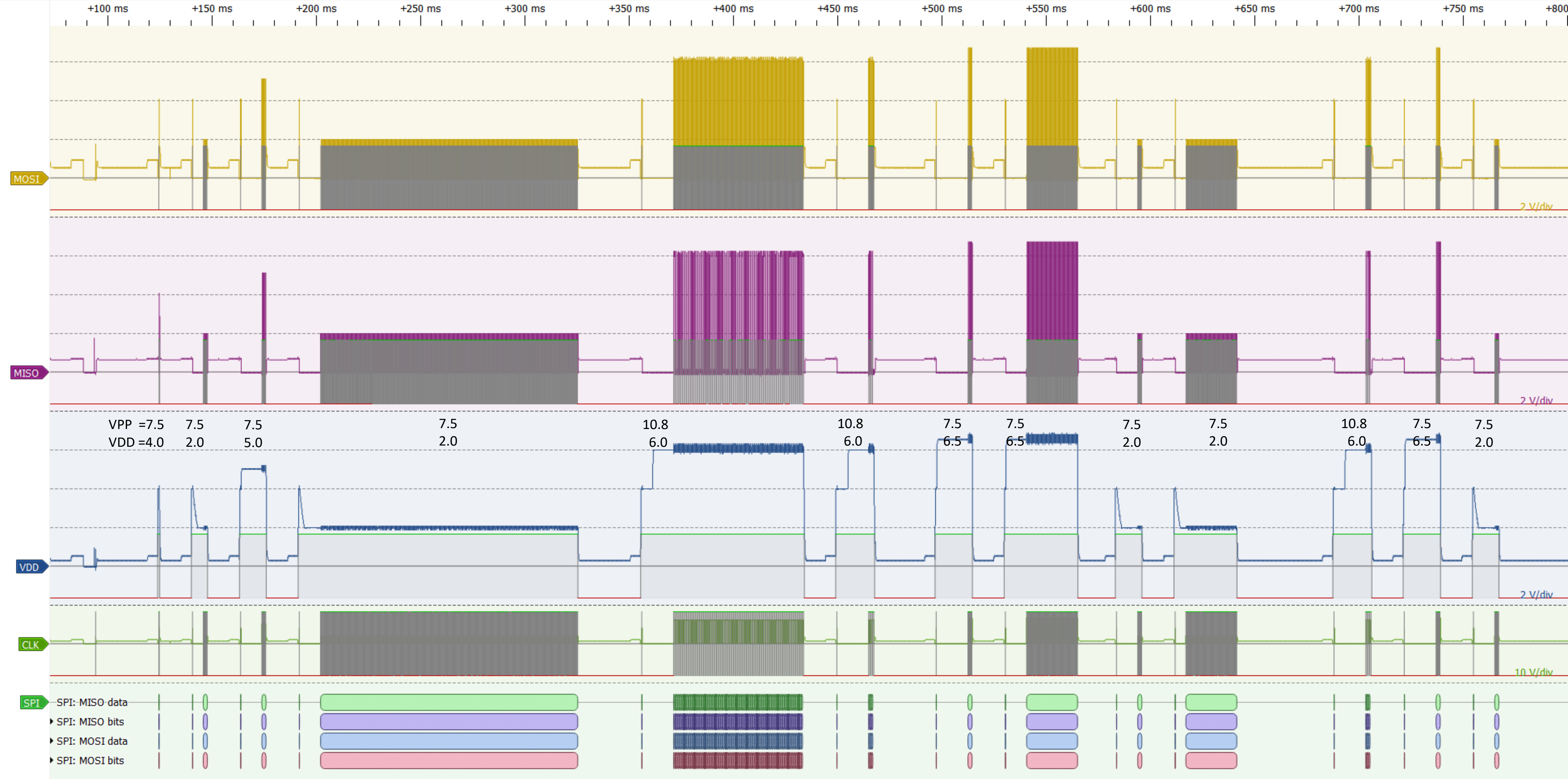
- PA3: SCK/Clock (driven by writer/master)
- PA4: MOSI/Data in (driven by writer/master, data is valid on rising edge of clk. Data is set at arbitrary times due to random timing of writer software)
- PA5: VPP
- PA6: MISO/Data out (driven by MCU, data is valid on rising edge of clk. Data is set on falling edge since the MCU does not have an internal clk)
- Furthermore, the programmer needs to control VDD to reset the MCU.

VPP is 7.5V during read and 10.8 V during writing.

VDD is 6.0 V during programming, 4 V during entry and 6.5 V/2 V for verification. I may be sufficient to keep VDD at 5V if you don't want to verify all corner cases.

Note: The PMS105C is a device with 13x1kbit memory and 13 bit instruction encoding.

Overview of dump2 – writing to previously programmed device



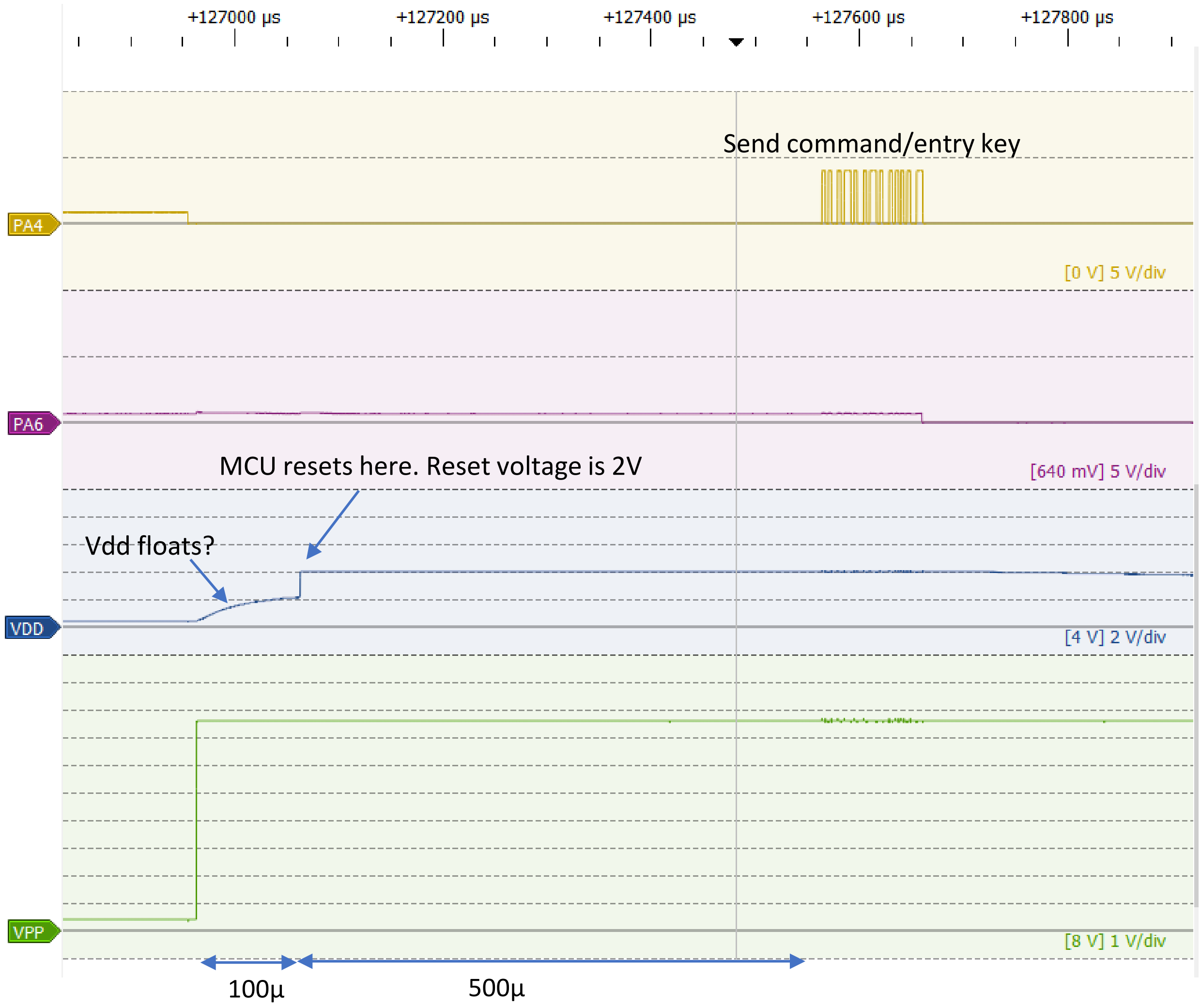
Command	A7	A6	A6	A6	A7	A7	A6	A6	A6	A6	A7	A6	A6
Phase	0	1	2	3	4	5	6	7	8	9	10	11	12

Summary of all phases of the programming sequence (Dump 2)

Phase	Command	VDD	VPP	Description
0	A7 (Write)	4.0 V	7.5 V	Read device ID. This is achieved by initiating a dummy write that is aborted before starting the actualy OTP programming
1	A6 (Read)	2.0 V	7.5 V	Read instruction memory words 0x03F0-0x3FF at low VDD voltage corner. This region contains calibration data.
2	A6 (Read)	5.0 V	7.5 V	Read instruction memory words 0x03F0-0x3FF at standard voltage corner. This region contains calibration data. (Should this be VDD=6.5V? May be a bug)
3	A6 (Read)	2.0 V	7.5 V	Read instruction memory words 0x000-0x3EF at low VDD voltage corner. Dump of full memory.
4	A7 (Write)	6.0 V	10.8 V	Write main memory region between 0x000-0x3EF. Only memory cells used by the program are written to.
5	A7 (Write)	6.0 V	10.8 V	Write to 0x3F6/0x3F8/0x3FC/0x3FE. Housekeeping?
6	A6 (Read)	6.5 V	7.5 V	Read instruction memory words 0x03F0-0x3FF at high voltage corner for verification.
7	A6 (Read)	6.5 V	7.5 V	Read main memory region between 0x000-0x3EF at high voltage corner for verification. Only previously written memory is read.
8	A6 (Read)	2.0 V	7.5 V	Read instruction memory words 0x03F0-0x3FF at low voltage corner for verification.
9	A6 (Read)	2.0 V	7.5 V	Read main memory region between 0x000-0x3EF at low voltage corner for verification. Only previously written memory is read.
10	A7 (Write)	6.0 V	10.8 V	Write to 0x3F6/0x3F8/0x3FC/0x3FE to store clock calibration data and code checksum.
11	A6 (Read)	6.5 V	7.5 V	Read instruction memory words 0x03F0-0x3FF at high voltage corner for verification.
12	A6 (Read)	2.0 V	7.5 V	Read instruction memory words 0x03F0-0x3FF at low voltage corner for verification.

- Note: For a fresh device, clock calibration takes place between steps 9 and 10.
  Two additional phases are inserted (see dump 4).

Enter programming mode



Send command/entry key

MCU resets here. Reset voltage is 2V

Vdd floats?

100µ

500µ

Each phase of the programming sequence is as follows:

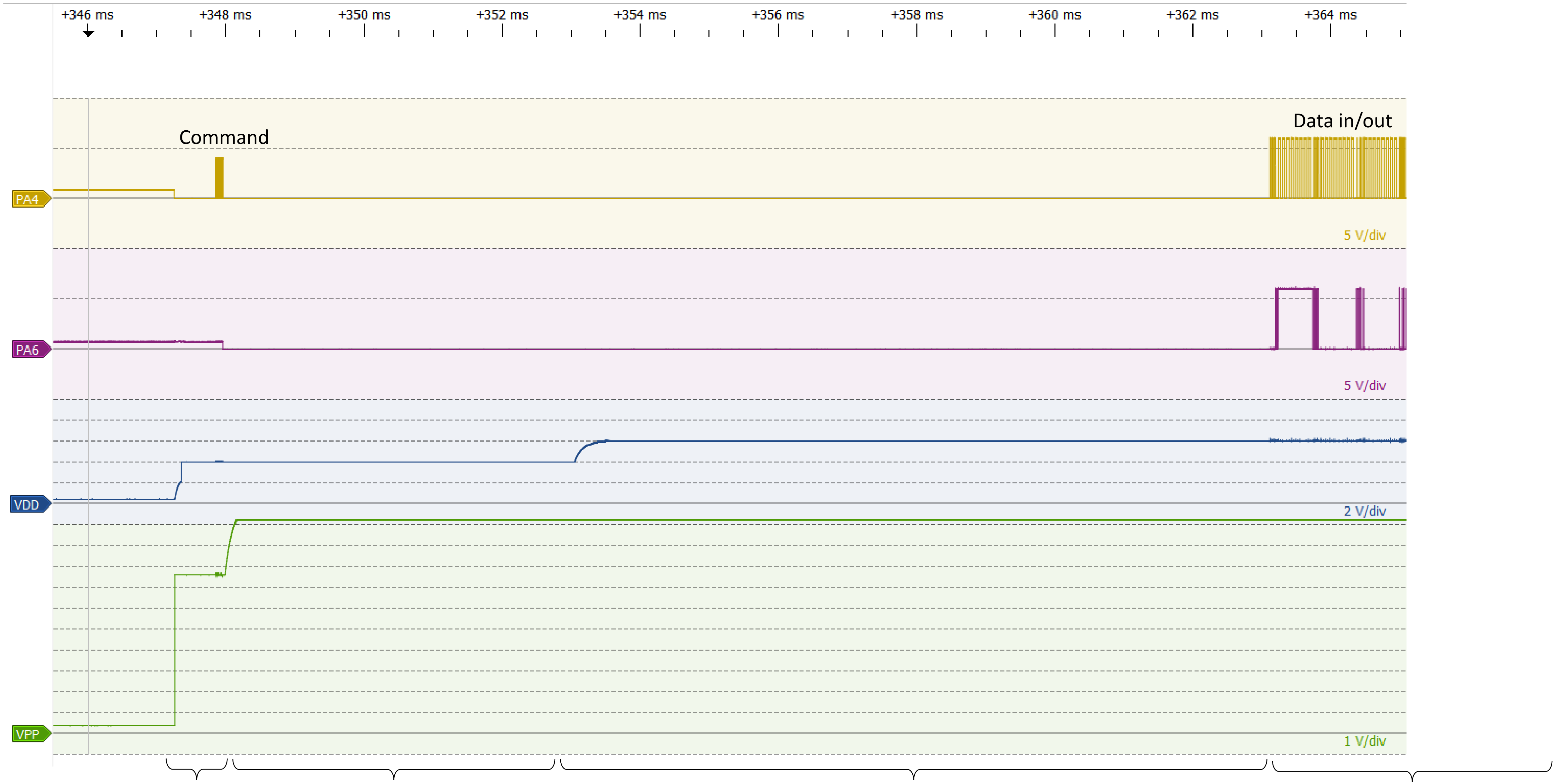
1. Set all pins to GND
2. Drive VPP to 7.5V (possibly >6V is threshold?)
3. wait 100µs
4. Drive VDD to ~4V
5. wait 500µs
6. Send key/command 0xA5A5A5AX (X=6 for read, X=7 for write)
7. Ramp to target VPP (7.5 V for reading, 10.8V for writing)
8. Wait for 5 ms
9. Ramp to target VDD
10. Wait for 10 ms
11. *Perform read or write operation (see later slides)*
12. Pull VDD and VPP to GND

Programming mode is always entered with
Vpp=7.5 V
Vdd=4 V

Voltages are only adjusted to final target after sending command (step 6).
Steps 7-10 can be skipped if initial voltages are kept.

Note: Logic levels scale with Vdd. If varying Vdd is implemented, this needs to be considered in the SPI interface hardware (buffer).

Enter programming mode with voltage adjustment and read/write phase



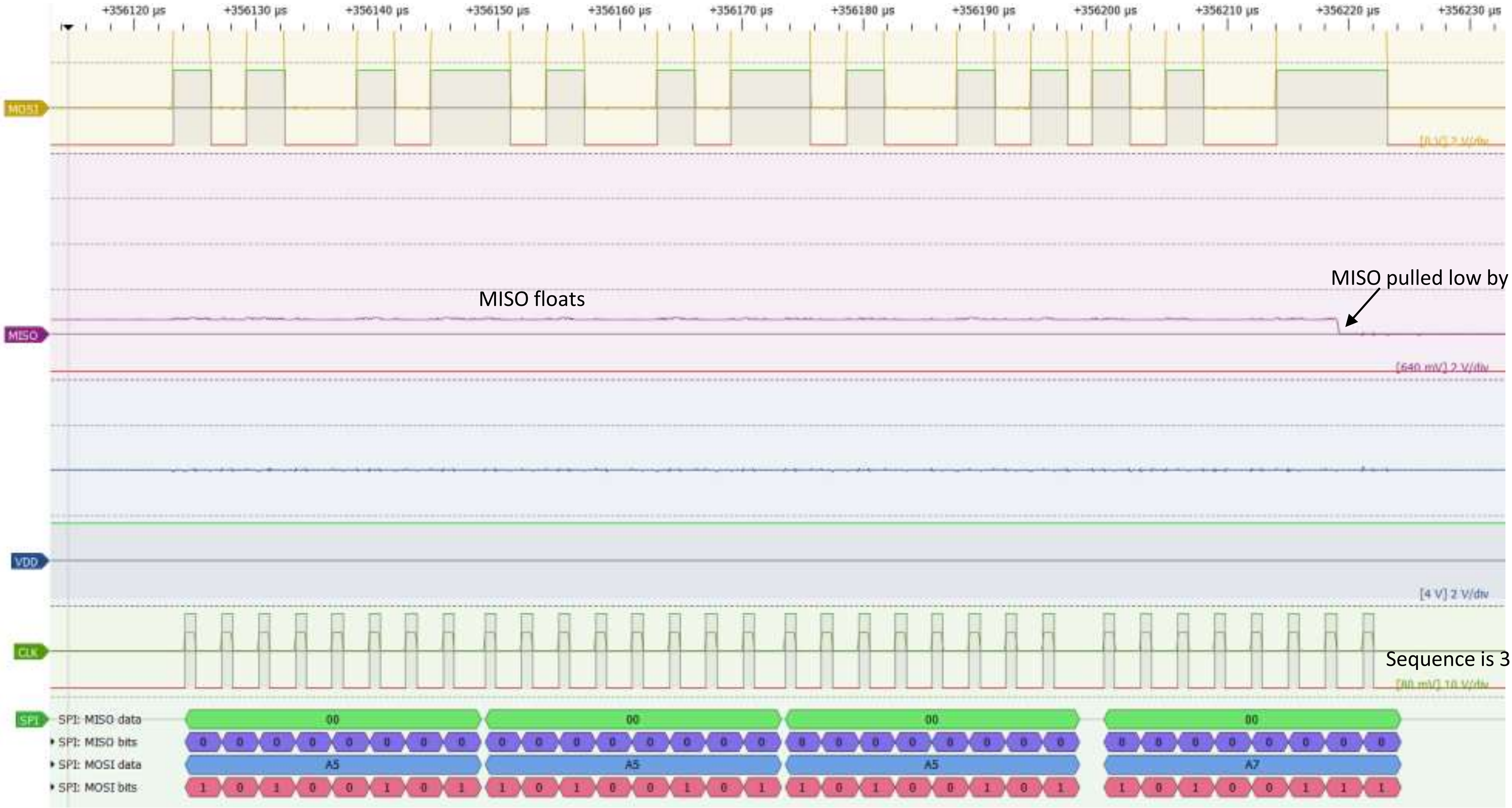
Entry sequence

Adjust VPP
(Wait time for VPP is 5 ms)

Adjust VDD
(Wait time for VDD is 10 ms)

Read / Write sequence

0xA5A5A5A7 key/command – write



MISO floats

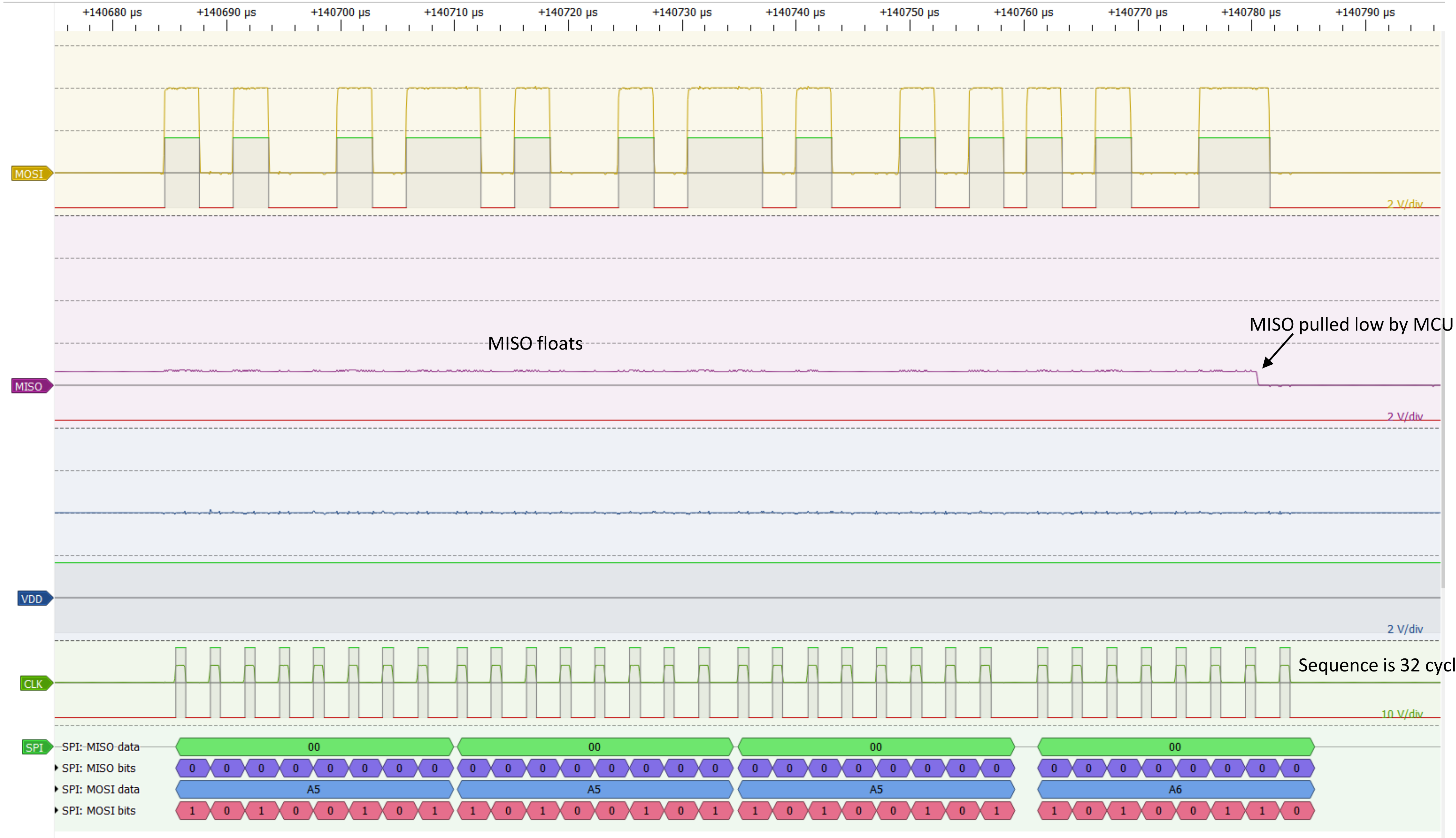
MISO pulled low by MCU after cycle 31

Sequence is 32 cycles in total

Not on PulseView: SPI decoder uses VDD as CS (Active high). This will reset the bitcount when the MCU is reset and ensures proper decoding for magic word

Analog signal were converted to logic by using a threshold of 1.8V (3.3V logic) to also capture the regions with Vdd=2V

0xA5A5A5A6 key/command - read



```
SPI: MISO data   00                00                00                00
SPI: MISO bits   0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0
SPI: MOSI data   A5                A5                A5                A6
SPI: MOSI bits   1 0 1 0 0 1 0 1   1 0 1 0 0 1 0 1   1 0 1 0 0 1 0 1   1 0 1 0 0 1 1 0
```

MISO floats

MISO pulled low by MCU after cycle 31

Sequence is 32 cycles in total

Command / Entry key encoding

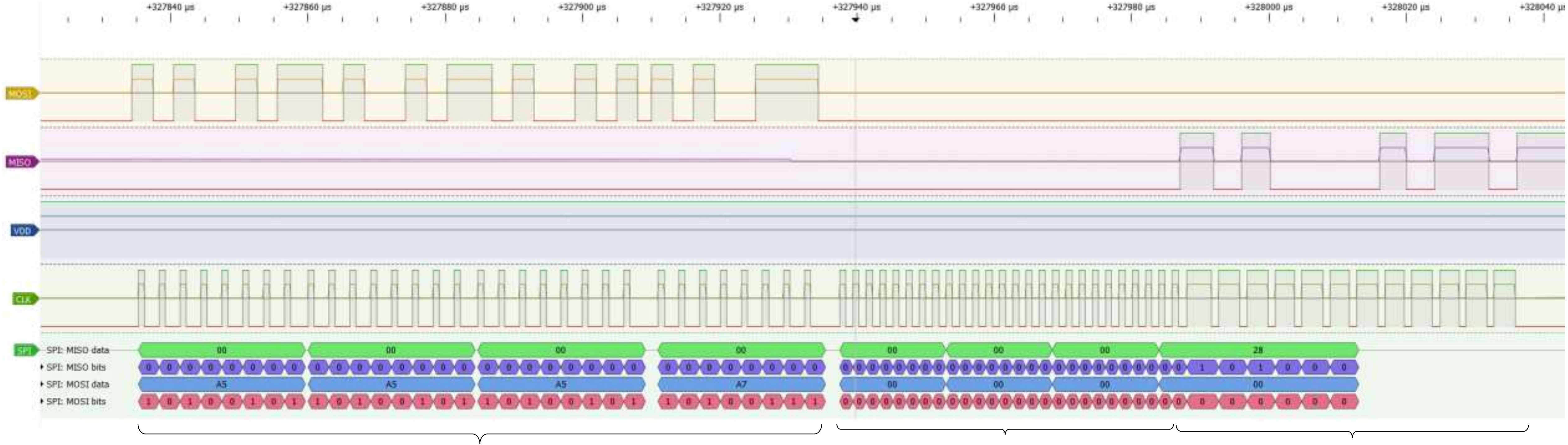
- The entry key is 32 bit and is sent by the master directly after entering programming mode.

- A6 key: 0XA5A5A5A6 -> Initiate reading
- A7 key: 0XA5A5A5A7 -> Initiate writing

- No other codes were found.

- The slave (MCU) pulls MISO down after clock 31. The pin floats before, which could suggest that the programming logic is activated after 31 clocks. This may also suggest that only the LSB is actually used for commands.

Phase 0 – Check device ID - Key-A7, Vdd=4 V, Vpp=7.5 V



32 bits magic key

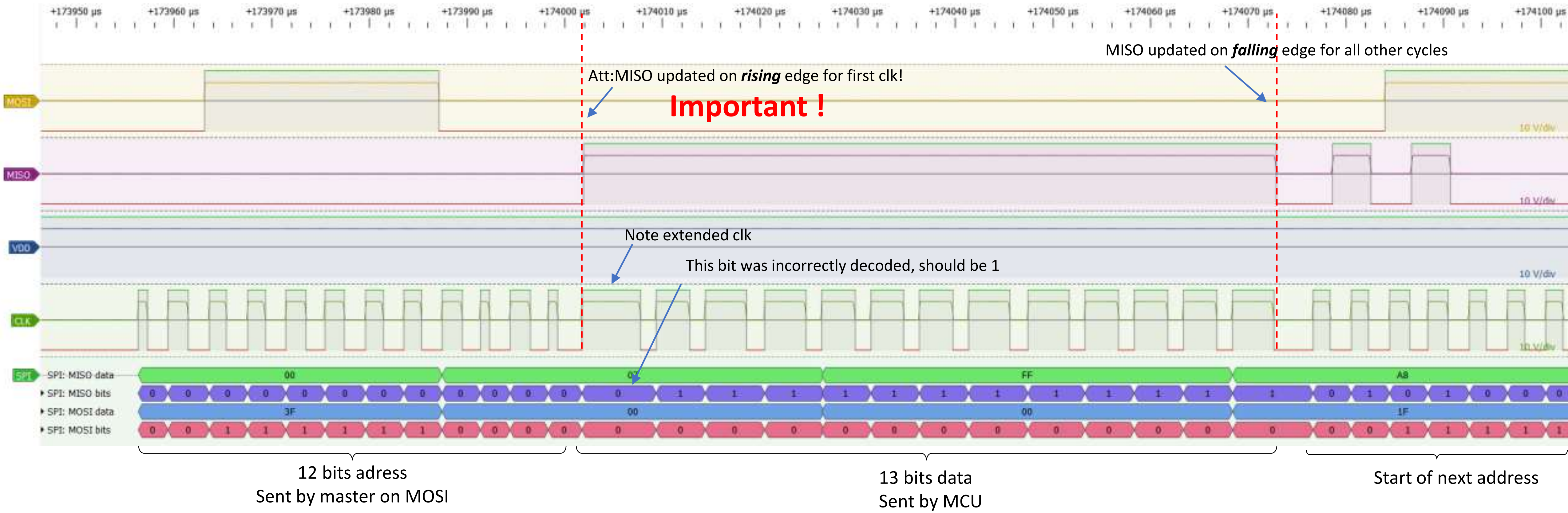
Master sends 26 Bits „0“
=> 2x13 bit empty data word

Master sends 12 bit „0“
=> Adress 0x000
Slave sends 12 bit response
0b010100001011
= 0x50B device ID

Not that this is one clockcycle less than a write-operation. Hence the write operation is not started.

- Cycle 0 is basically an incomplete write of 0x0000/0x0000 to 0x000. The last clock cycle is omitted to prevent the dummy data from actually being written.
  The device ID is clocked out of MISO during the adress phase of the write.
- In principle, the device ID could also be read during a read phase.
- Sequence is identical for both previously programmed and clean device (logs 2 and 4)
- Idcode is updated on falling edge! Delay 240-320ns.

Read sequence



MISO updated on ***falling*** edge for all other cycles

Att:MISO updated on ***rising*** edge for first clk!

Important !

Note extended clk

This bit was incorrectly decoded, should be 1

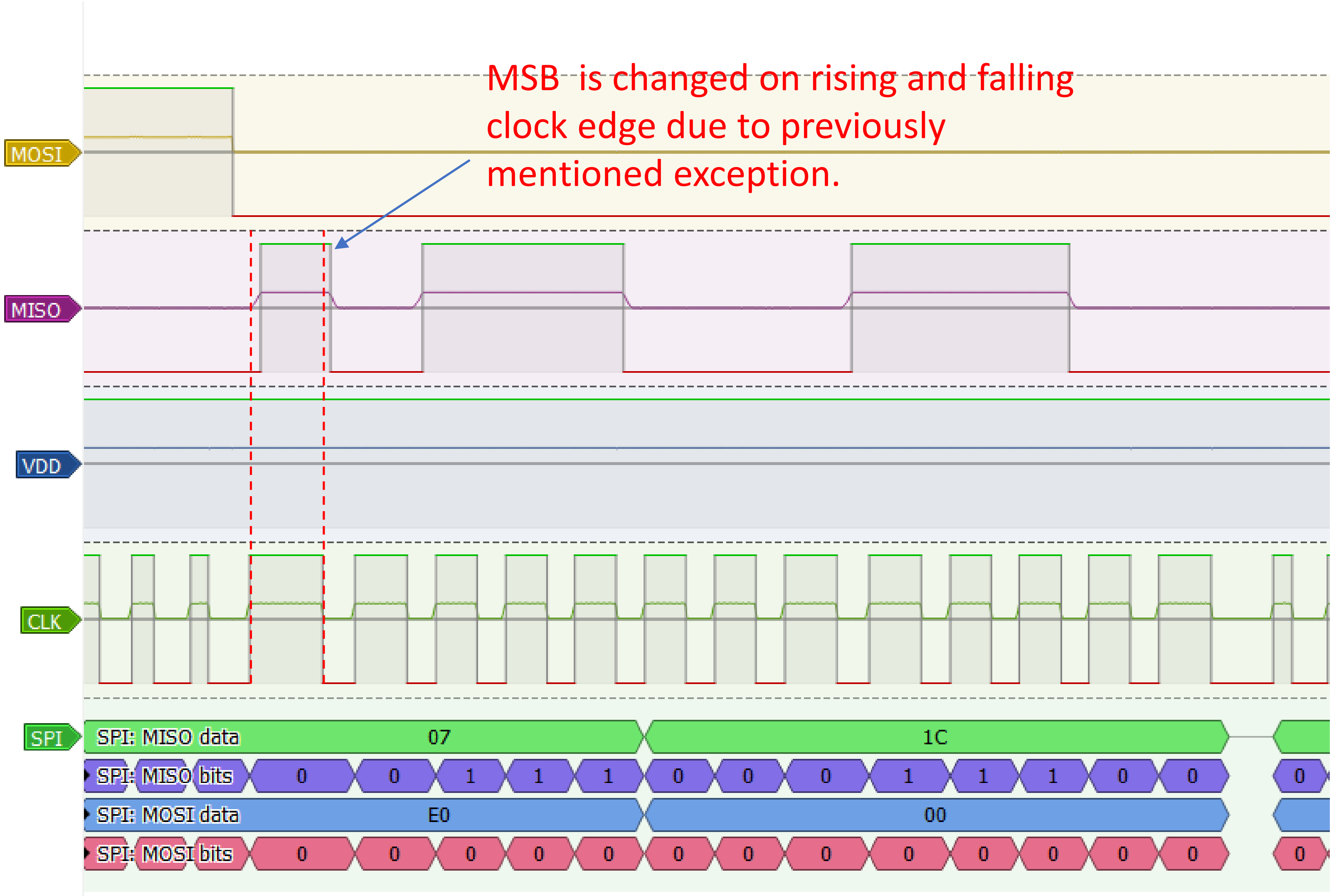
12 bits adress
Sent by master on MOSI

13 bits data
Sent by MCU

Start of next address

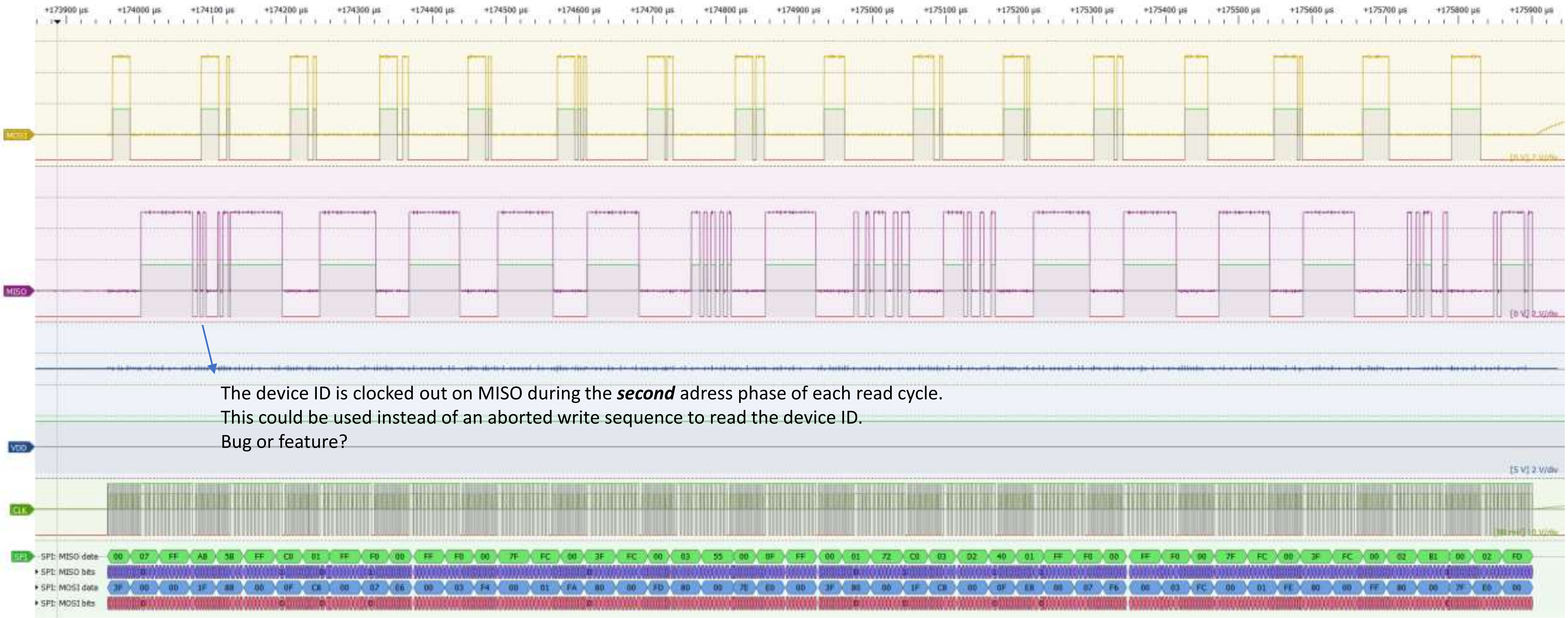
- Read sequence is straight forward:
  - Master sends 12 bit adress, slave sends 13 bits of data in subsequent cycles.
- There is one apparent oddity:
  Usually the data on MISO is always updated on the falling clk cycle. However, the MSB of the data is updated with a delay of half a clock cycle, on the rising edge of the clock.
- => For the MSB, it is therefore necessary to set clk high and read MISO after ~2µs before setting clk low again.
- What is the reason for this behavior? No idea, it could be remnant of a direction switching sequence for a bidirectional port.

Another example of MISO MSB exception during the read sequence.



MSB is changed on rising and falling clock edge due to previously mentioned exception.

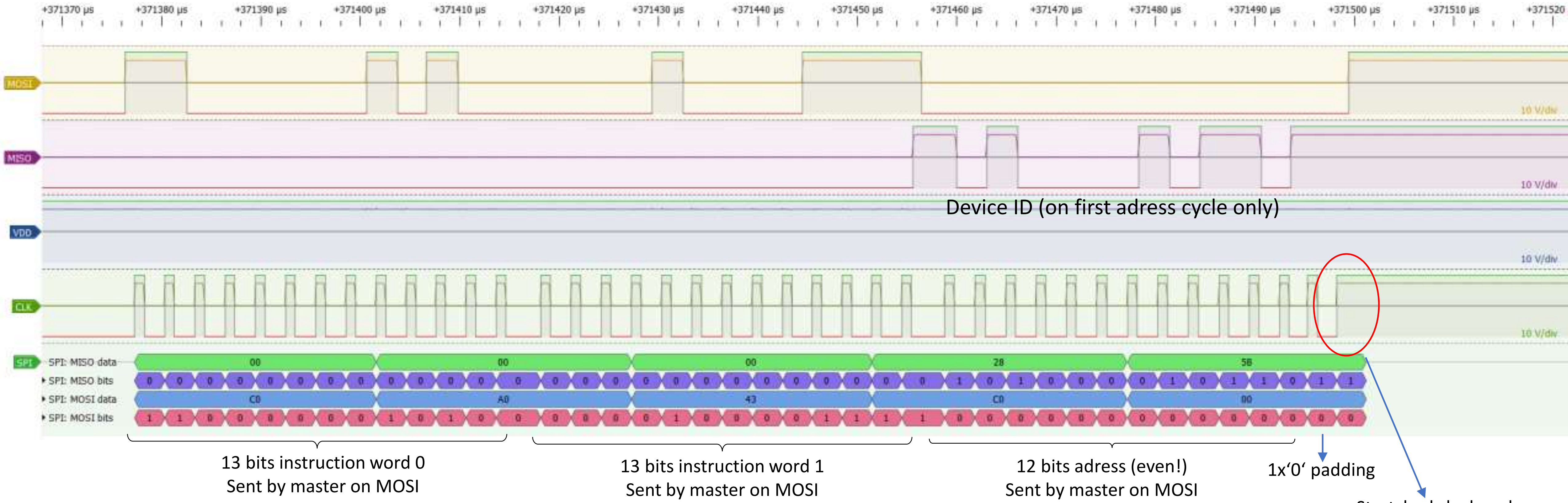
Phase 1 – overview – Read 0x3F0-0x3FF (system area)



The device ID is clocked out on MISO during the ***second*** adress phase of each read cycle.
This could be used instead of an aborted write sequence to read the device ID.
Bug or feature?

- The MCU seems to send out information during adress cycles as well:
  - During the second adress cycle the device ID is sent.
  - Occasionally the MSB is set on MISO in subsequent adress cycles. Glitch? Parity information?

Write Sequence Part I



Device ID (on first adress cycle only)

13 bits instruction word 0
Sent by master on MOSI

13 bits instruction word 1
Sent by master on MOSI

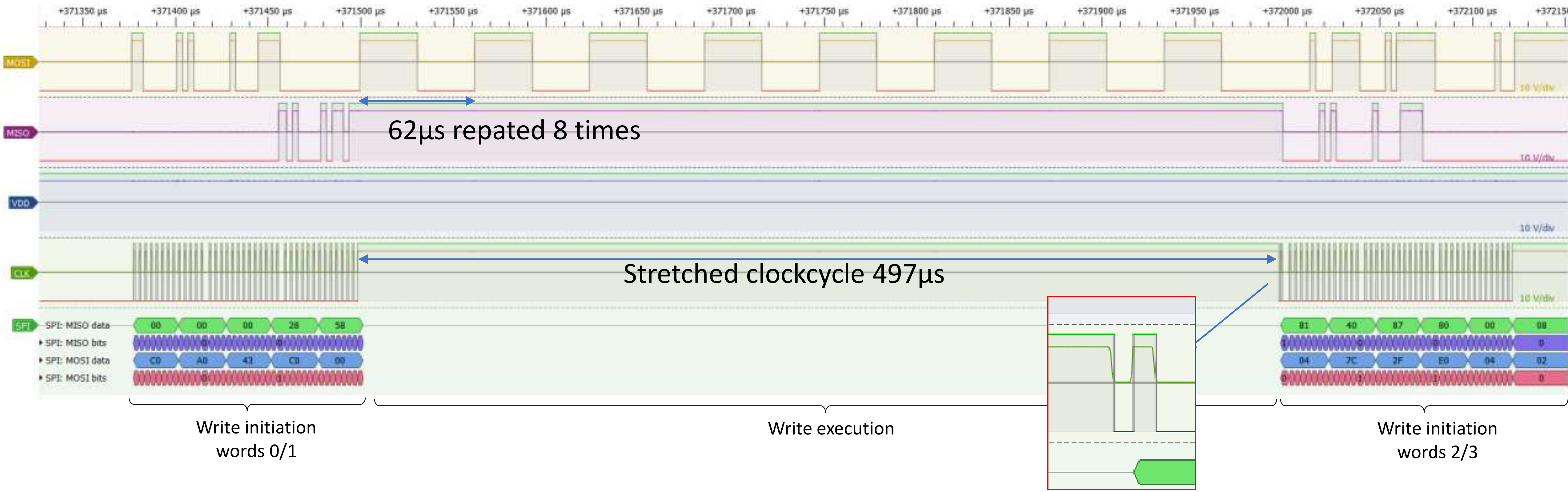
12 bits adress (even!)
Sent by master on MOSI

1x‘0‘ padding

Stretched clock cycle
-> write takes place here

- The PMS150C seems to write two words of memory per write cycle.
- Each write cycle consists of the the following initiation sequence:
  - Send 2x 13 bit instruction words
  - Send 12 bit adress word. (Needs to be dividable by two ?).
  - Send a single „0“ bit. The write cycle seems to be aborted if the device is powered down before sending this bit.
  - The next low-> high transition of the clk seems to initiate the write.

Write Sequence Part II



62µs repated 8 times

Stretched clockcycle 497µs

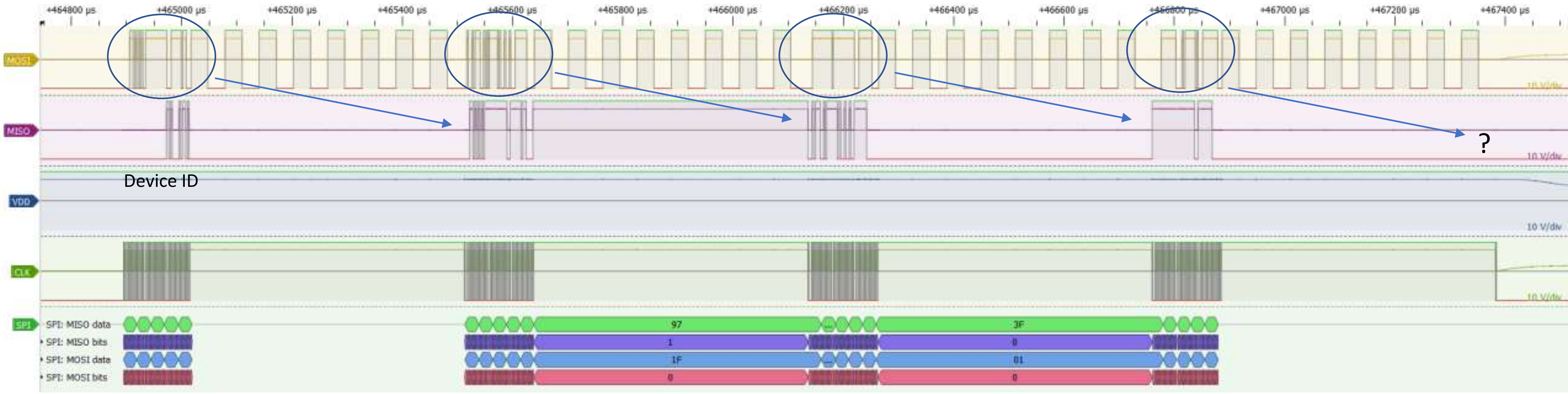
Write initiation words 0/1

Write execution

Write initiation words 2/3

- The write execution cycles takes approximately 500µs.
  - The last clockcycle of the write initiation sequence is stretched during the write execution sequence. The SCLK H->L transition preceeds the next write initiation cycle.
  - It appears a secondary clock signal of 16kHz is provided on MOSI. Very odd.

Write Sequence Part III – full sequence overview of phase 5



- The last write execution cycle simply ends with the H->L transition of SCLK.
- The MCU will also output data on MISO during the write initiation cycle.
  - First adress cycle: The device ID is clocked out.
  - Subsequent cycles: The data and adress words of the previous cycle are repeated.
    This may be a feature to verify the correctness of the written data without a separate read phase. However, it seems that the current software is not making use of it since the information of the last write cycle is discarded.

Revisions

- V0.1 – Jan 7th, 2019 – cpldcpu. Initial report.
- V0.11 - Jan 7th, 2019 – cpldcpu. Updated up clean room disclaimer and front matter.