SPI-DirectC v2021.1 User Guide

Introduction

SPI-DirectC is designed to support an embedded In-System Programming for Microchip devices. In-System Programming refers to an external processor on-board programming one of the IGLOO®2, SmartFusion®2, PolarFire® or PolarFire SoC devices using a SPI peripheral interface.

SPI-DirectC supports systems with direct and indirect access to the memory space containing the data file image. With paging support, it is possible to implement the embedded ISP using SPI-DirectC on systems with no direct access to the entire memory space containing the data. Paging support is accomplished by making modifications to the data communication functions defined in `dpuser.h`, `dpcom.c`, and `dpcom.h`.

To use SPI-DirectC v2021.1, you must make some minor modifications to the source code, add the necessary API, and compile the source code and the API together to create a binary executable. The binary executable is downloaded to the system along with the programming data file. The programming data file is a binary file that can be generated by Libero® SoC Design Suite version 11.2 or later. For more information on detailed specification of the programming file, see 4. Data File Format.

Supported Device Family

This document describes how to enable microprocessor-based embedded In-System Programming (ISP) on the supported Microchip devices. The following table lists the Microchip devices SPI-DirectC supports.

Table 1. Device Family Supported by SPI-DirectC

Device Family	Description
PolarFire®	PolarFire FPGAs deliver the industry's lowest power at mid-range densities with exceptional security and reliability.
PolarFire SoC	PolarFire SoC is the first SoC FPGA with a deterministic, coherent RISC-V CPU cluster, and a deterministic L2 memory subsystem enabling Linux and real-time applications.
SmartFusion®2	SmartFusion2 addresses fundamental requirements for advanced security, high reliability, and low power in critical industrial, military, aviation, communications, and medical applications.
IGLOO®2	IGLOO2 is a low-power mixed-signal programmable solution.

Table of Contents

1. System Overview

The system must contain the following parameters to perform the In-System Programming (ISP) for the FPGA.

- A microprocessor with at least 8192 bytes of RAM or a softcore processor implemented in another FPGA.
- SPI IP to interface with the target device. SPI Mode 3 must be used.
- Access to the data file containing the programming data.
- Memory to store and run SPI-DirectC code.

For more information on power requirements for V_{pump} and other power supplies, see your device product device datasheet.

The following table lists the memory requirements.

Table 1-1. Code Memory Requirements- SPI-DirectC Code Size on CM3 16-Bit Mode

Compile Options Enabled	Units are in Bytes		
	ROM Code[1]	ROM Data[2]	Read/Write Data[3]
ENABLE_G4_SUPPORT	16902	608	12578
ENABLE_G5_SUPPORT	20242	1570	12851
All the above	30414	1576	13639

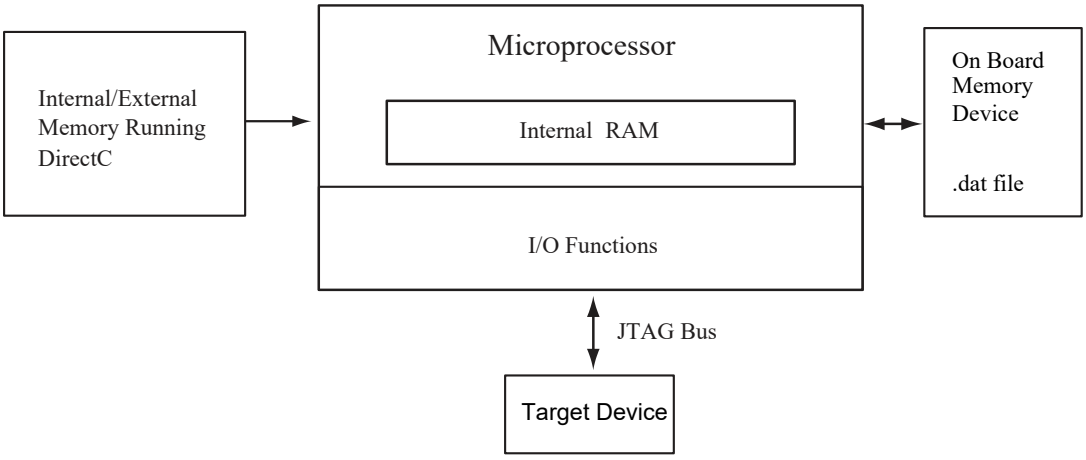
Notes:

1. ROM Code— This is the compiled code size memory requirements.
2. ROM Data— This is the block started by Symbol allocation for variables that do not yet have values, that is, uninitialized data. It is part of the overall data size.
3. Read/Write Data— This is the run time memory requirement, that is, the free data memory space required to execute the code.

1.1 Systems with Direct Access to Memory

The following figure shows the overview of a typical system with direct access to the memory space holding the data file.

Figure 1-1. System with Direct Access to Memory



The following table lists the data storage memory requirements.

Table 1-2. Data Storage Memory Requirements - Data Image Size

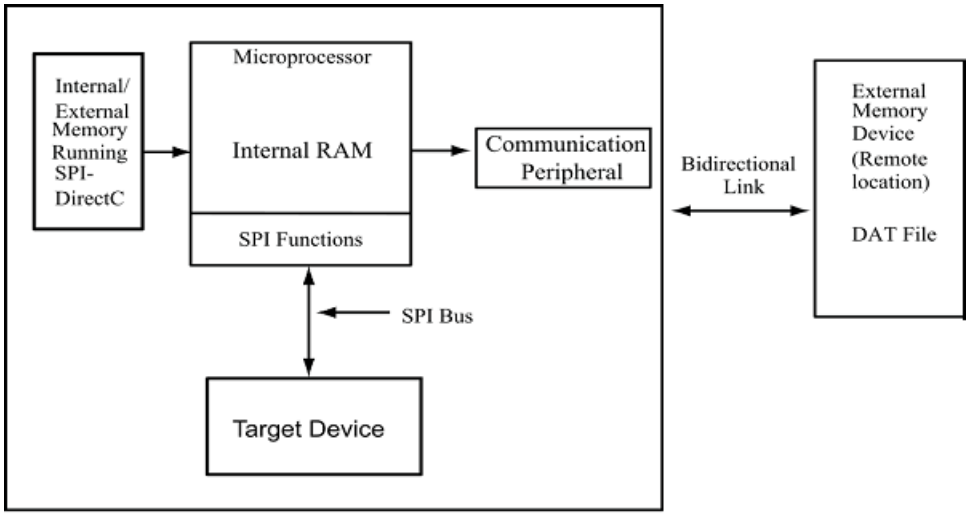
Data Image Size			
Device	**Core/FPGA Array - Encrypt (kB)**	**Embedded Flash Memory Block - Encrypt (kB)**	**Core/FPGA Array and Security - Encrypt (kB)**
M2GL005	297	133	851
M2GL010	557	267	1639
M2GL025	1197	267	2918
M2GL050	2364	267	5253
M2GL090	3564	532	8178
M2GL150	5997	531	13046
M2S005	297	137	860
M2S010	557	272	1648
M2S025	1197	272	2926
M2S050	2364	272	5261
M2S090	3564	536	8186
M2S150	5997	535	13054
MPF100	3447	N/A	N/A
MPF200	5992	N/A	N/A
MPF300	9256	N/A	N/A
MPF500	14739	N/A	N/A
MPFS250T	9542	N/A	N/A

Note: The total image size is the sum of all the corresponding enabled blocks for the specific target device.

1.2 Systems with Indirect Access to Memory

The following figure is an overview of a system with no direct access to the memory space holding the data file. For example, the programming data might be received via a communication interface peripheral that exists between the processor memory and the remote system holding the data file `dpcom.h` and `dpcom.c` must be modified to interface with the communication peripheral.

Figure 1-2. System With Indirect Access to Memory



1.3 Motorola SPI Protocol

Motorola SPI Mode 3 is required to communicate with SmartFusion2, IGLOO2, PolarFire, and PolarFire SoC devices using a dedicated system controller SPI port. See Motorola SPI standard for more information.

The Motorola SPI is a full duplex, four-wire synchronous transfer protocol, which supports programmable clock polarity (SPO) and clock phase (SPH). The state of SPO and SPH control bits decides the data transfer modes as listed in the following table.

Table 1-3. Data Transfer Modes

Data Transfer Mode	SPO	SPH
Mode 0	0	0
Mode 1	0	1
Mode 2	1	0
Mode 3	1	1

The SPH control bit determines the clock edge that captures the data.

- When SPH is low, data is captured on the first clock transition.
  - Data is captured on the falling edge of `SPI_CLK` when SPO = 1.
  - Data is captured on the rising edge of `SPI_CLK` when SPO = 0.
- When SPH is high, data is captured on the second clock transition (rising edge if SPO = 1).
  - Data is captured on the rising edge of `SPI_CLK` when SPO = 1.
  - Data is captured on the falling edge of `SPI_CLK` when SPO = 0.

The SPO control bit determines the polarity of the clock and SPS defines the slave select behavior.

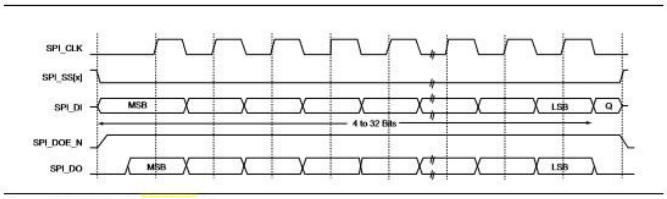
- When SPO is low and no data is transferred, `SPI_CLK` is driven to low.
- When SPO is high and no data is transferred, `SPI_CLK` is driven to high.

Table 1-4. Summary of the Clock Active Edges in Various SPI Master Modes

Mode	SPS	SPO	SPH	Clock in Idle	Sample Edge	Shift Edge	Select in Idle	Select Between Frames
Motorola	0	0	0	Low	Rising	Falling	High	Pulses between all frames
	0	1	0	High	Falling	Rising	High	
	0	0	1	Low	Falling	Rising	High	Does not pulse between back-to-back frames. Pulses if transmit FIFO empties.
	0	1	1	High	Rising	Falling	High	Does not pulse between back-to-back frames. Pulses if transmit FIFO empties.
	1	0	0	Low	Rising	Falling	High	Stays active until all the frames set by frame counter are transmitted.
	1	0	1	Low	Falling	Rising	High	
	1	1	0	High	Falling	Rising	High	
	1	1	1	High	Rising	Falling	High	

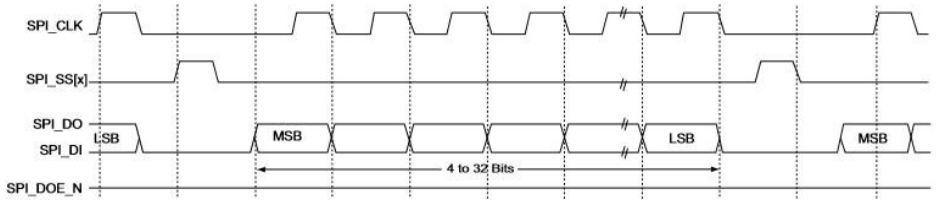
Single Frame Transfer - Mode 0: SPO = 0, SPH = 0
The following figure illustrates the single frame transfer using Mode 0 data transfer mode with programmable clock polarity 0 and clock phase 0.

Figure 1-3. Motorola SPI Mode 0



Multiple Frame Transfer - Mode 0: SPO = 0, SPH = 0
The following figure illustrates the multiple frame transfer using the Mode 0 data transfer mode with programmable clock polarity 0 and clock phase 0.

Figure 1-4. Motorola SPI Mode 0 Multiple Frame Transfer



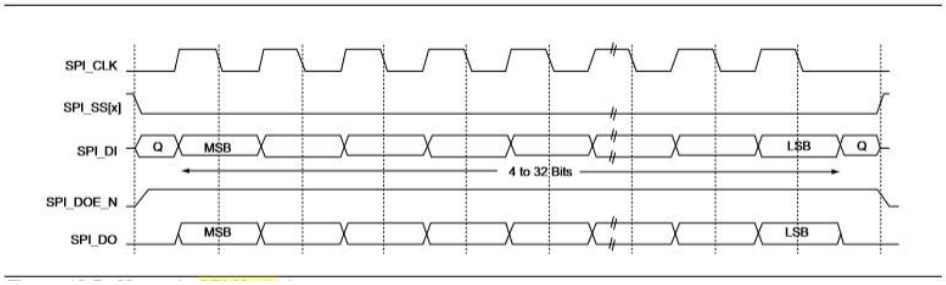
Notes:

- Between frames, the slave selects (SPI_SS[x]) a signal that is asserted for the duration of the clock pulse.
- Between frames, the clock (SPI_CLK) is low.
- Data is transferred to Most Significant Bit (MSB) first.
- The output enables (SPI_DOE_N) a signal that is asserted during the transmission and deasserted at the end of the transfer (after the last frame is sent).

Single Frame Transfer - Mode 1: SPO = 0, SPH = 1

The following figure illustrates the single frame transfer using the Mode 1 data transfer mode with programmable clock polarity 0 and clock phase 1.

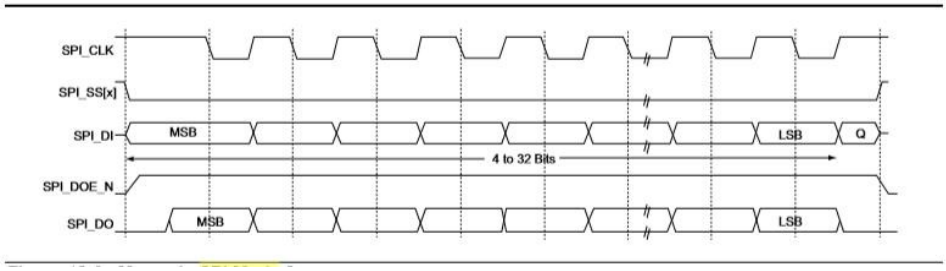
Figure 1-5. Motorola SPI Mode 1



Single Frame Transfer - Mode 2: SPO = 1, SPH = 0

The following figure illustrates the single frame transfer using the Mode 2 data transfer mode with programmable clock polarity 1 and clock phase 0.

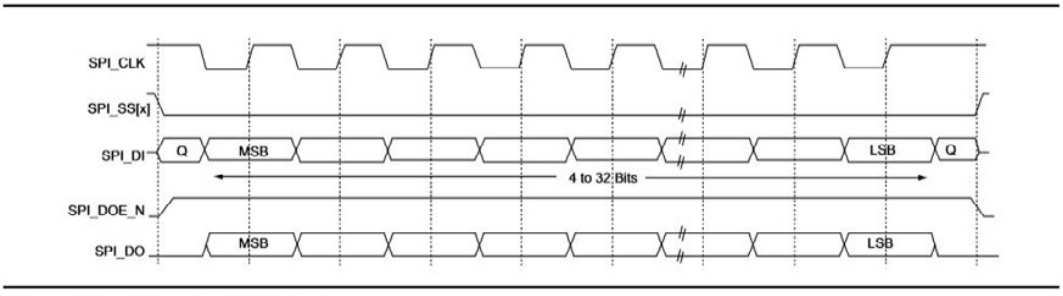
Figure 1-6. Motorola SPI Mode 2



Single Frame Transfer - Mode 3: SPO = 1, SPH = 1

The following figure illustrates the single frame transfer using the Mode 3 data transfer mode with programmable clock polarity 1 and clock phase 1.

Figure 1-7. Motorola SPI Mode 3



2. Generating Data Files and Integrating SPI-DirectC Code

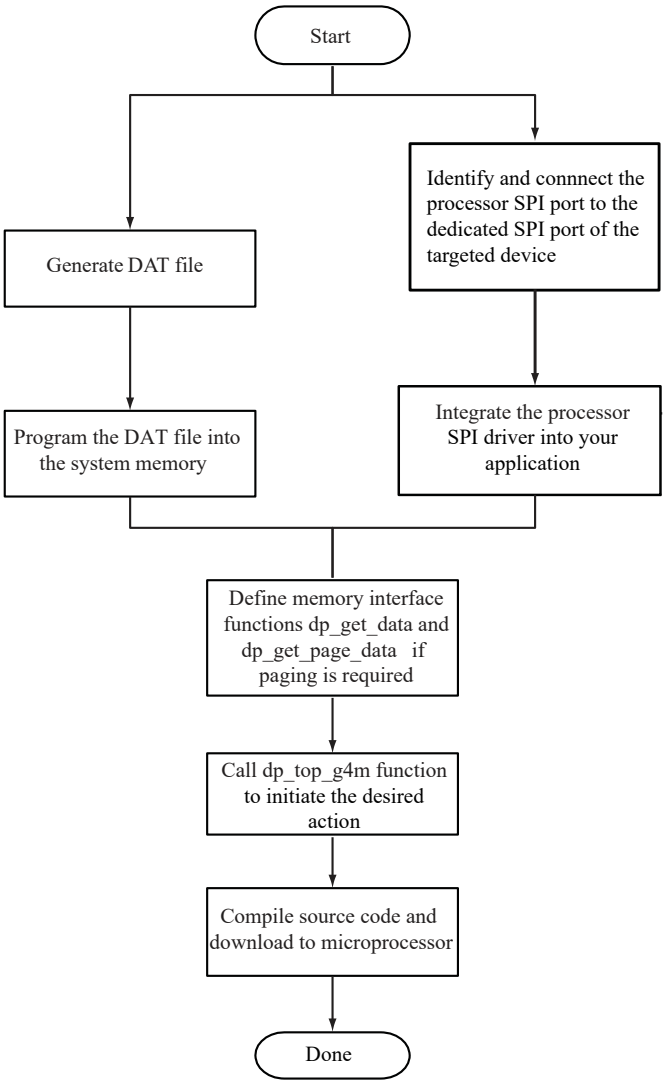
The following chapter describes the flows for data file generation and SPI-DirectC code integration. To generate the DAT file:

1. Launch the Libero SoC Design Suite and open the project.
2. Expand the **Handoff Design for Production** tree on the **Design Flow** tab.
3. Double click **Export Bitstream**. The **Export Bitstream** dialog box opens. The dialog box options depend on the device family, **Custom Security settings**, and **Permanent Locks** for the production settings. For more information on working with the **Export Bitstream**, see the Libero SoC Design Flow User Guide.
4. Program the DAT file into the storage memory.

2.1 SPI-DirectC Code Integration

The following figure shows the SPI-DirectC integration use flow.

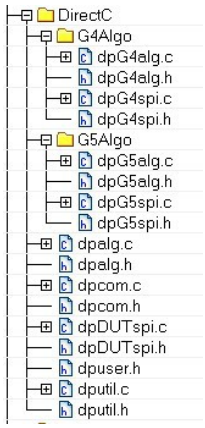
Figure 2-1. Importing SPI-DirectC Files



To use SPI-DirectC code integration:

1. Import the SPI-DirectC files into your development environment as shown in the following figure.

   Figure 2-2. SPI-DirectC Files to Import into your Development Environment



2. Modify the SPI-DirectC code.
   - Add the SPI driver (available with the processor used to run SPI-DirectC).
   - Modify the hardware interface functions (`do_SPI_SCAN_in` and `do_SPI_SCAN_out`) to use the hardware API functions designed to control the SPI port.
   - Modify memory access functions to access the data blocks within the image file programmed into the system memory. See 6. Data File Bit Orientation for more details.
   - Call `dp_top` with the action code desired.
3. Compile the source code. This creates a binary executable that is downloaded to the system for execution.

3. Required Source Code Modifications

You must modify the `dpuser.h`, `dpDUTspi.c`, `dpcom.c`, and `dputil.c` files when using the SPI-DirectC source code, which contains a short description of SPI-DirectC source code and their functions.

The following table lists the functions that must be modified.

Table 3-1. Modified Functions

Function	Source File	Purpose
`do_SPI_SCAN_in`	`dpspi.c`	Hardware interface function used to scan data in using the SPI driver
`do_SPI_SCAN_out`	`dpspi.c`	Hardware interface function used to scan data out using the SPI driver
`dp_get_page_data`	`dpcom.c`	Programming file interface function
`dp_display_text`	`dpuser.c`	Function to display text to an output device
`dp_display_value`	`dpuser.c`	Function to display value of a variable to an output device
`dp_delay`	`dputil.c`	Delay function

3.1 Compiler Switches

The following table lists the compiler switches.

Table 3-2. Compiler Switches

Function	Source File	Purpose
`USE_PAGING`	`dpuser.h`	Enables paging implementation for memory access.
`ENABLE_G4M_SUPPORT`	`dpuser.h`	Enables M2S/M2GL programming support.
`ENABLE_G5M_SUPPORT`	`dpuser.h`	Enables MPF/MPF SoC programming support.
`PERFORM_CRC_CHECK`	`dpuser.h`	Enables CRC check of the programming data prior to performing the desired action.
`ENABLE_DISPLAY`	`dpuser.h`	Enables display to hyper terminal or other output devices.

3.2 Hardware Interface Components

3.2.1 Hardware Interface Function (dpDUTspi.c)

`do_SPI_SCAN_in` and `do_SPI_SCAN_out` functions are used to interface with the SPI port to clock data into and out of the target device.

Note: These functions must use the SPI driver API available for the targeted device processor.

dp_SPI_SCAN_in Function

This function takes three arguments:

- Command: 8-bit variable holding the command value.
- Data_bits: The number of bits to clock into the device.
- input_buffer: Pointer to the buffer which holds valid data to be clocked into the device.

dp_SPI_SCAN_OUT Function
This function takes four arguments:

- Command bits: The number of bits to clock in for the command portion of the frame. This value must be 8 as all SPI commands are 8 bit long.
- Command: 8-bit variable holding the command value
- Data_bits: The number of bits to read from the device.
- Ouput_buffer: Pointer to the buffer to hold the data read from the targeted device.

3.2.2 Display Functions

Four functions, `dp_display_array`, `dp_display_array_reverse`, `dp_display_text`, and `dp_display_value` are available to display text as well as numeric values.
Note: You must modify `dp_display_text` and `dp_display_value` functions for proper operation.

3.2.3 Memory Interface Functions

All accesses to the memory blocks within the data file are done through the `dp_get_data` function within the DirectC code. This is true for all system types.

This function returns an address pointer to the byte containing the first requested bit. The `dp_get_data` function takes two arguments:

- `var_ID`: An integer variable, which contains an identifier specifying which block within the data file needs to be accessed.
- `bit_index`: The bit index addressing the bit to address within the data block specified in `Var_ID`. Upon completion of the function, it is expected that `return_bytes` indicates the total number of valid bytes available for the client of the function.

See 3.2.4 Systems with Direct Access to the Memory Containing the Data File and 3.2.5 Systems with Indirect Access to the Data File for more details.

3.2.4 Systems with Direct Access to the Memory Containing the Data File

The memory space holding the data file is accessible by the microprocessor. It can be treated as an array of unsigned characters. In this case:

1. Disable the `USE_PAGING` compiler switch.
2. Assign the physical address pointer to the first element of the data memory location (`image_buffer` defined in `dpcom.c`). `Image_buffer` is used as the base memory for accessing the information in the programming data in storage memory.

The `dp_get_data` function calculates the address offset to the requested data and adds it to `image_buffer`. `Return_bytes` is the requested data.

Example 3-1. `dp_get_data` Function Implementation

```
DPUCHAR* dp_get_data(DPUCHAR var_ID,DPULONG bit_index)
{
DPULONG image_requested_address;
if (var_ID == Header_ID)current_block_address = 0;
else
dp_get_data_block_address(var_ID);
if ((current_block_address ==0) && (var_ID != Header_ID))
{
return_bytes = 0;
return NULL;
}
/* Calculating the relative address of the data block needed within the image
*/
```

```
image_requested_address = current_block_address + bit_index / 8;
return_bytes=image_size - image_requested_address;
return image_buffer+image_requested_address;
}
```

3.2.5 Systems with Indirect Access to the Data File

These systems access programming data indirectly via a paging mechanism. Paging is a method of copying a certain range of data from the memory containing the data file and pasting it into a limited size memory buffer that DirectC can access.

To implement paging:

1. Enable the `USE_PAGING` compiler option.
2. Define `Page_buffer_size`. The minimum buffer size is 16 bytes.
3. Modify the `dp_get_page_data` function.

This function copies the requested data from the external memory device into the page buffer. See 6. Data File Bit Orientation for additional information. For correct operation:

1. Fill the entire page buffer unless the end of the image is reached. See 4. Data File Format.
2. Update `return_bytes` to reflect the number of valid bytes in the page.

Every time access to a data block within the image data file is needed, SPI-DirectC programming functions call the `dp_get_data` function. The `dp_get_data` function calculates the relative address location of the requested data and checks, if it already exists in the current page data. The paging mechanism is triggered, if the requested data is not within the page buffer.

3.2.6 dp_get_page_data Function Implementation

`dp_get_page_data` is the only function that must interface with the communication peripheral of the image data file. As the requested data blocks may not be contiguous, it must have random access to the data blocks. Its purpose is to fill the page buffer with valid data.

In addition, this function must maintain `start_page_address`, `end_page_address`, and `return_bytes`. These global variables contain the range of data currently in the page as well as the number of valid bytes.

`dp_get_page_data` takes one argument:

- `address_offset`— Contains the relative address of the needed element within the data block of the image file.

Example 3-2. `dp_get_page_data` Function Implementation

```
void dp_get_page_data(DPULONG image_requested_address)
{
DPULONG image_address_index;
start_page_address=0;
image_address_index=image_requested_address;
return_bytes = PAGE_BUFFER_SIZE;
if (image_requested_address + return_bytes > image_size)
return_bytes = image_size - image_requested_address;
while (image_address_index < image_requested_address + return_bytes)
{
page_global_buffer[start_page_address]=image_buffer[image_address_index];
start_page_address++;
image_address_index++;
}
start_page_address = image_requested_address;
end_page_address = image_requested_address + return_bytes - 1;
return;
}
```

3.2.7 Main Entry Function

The main entry function is `dp_top` defined in `dpalg.c`. It must be called to initiate the programming operation. Prior to calling the function, a global variable `Action_code` must be assigned a value as defined in `dpuser.h`. Action codes are listed in the following codeblock.

```
#define DP_DEVICE_INFO_ACTION_CODE 1
#define DP_READ_IDCODE_ACTION_CODE 2
#define DP_ERASE_ACTION_CODE 3
#define DP_PROGRAM_ACTION_CODE 4
#define DP_VERIFY_ACTION_CODE 5
#define DP_ENC_DATA_AUTHENTICATION_ACTION_CODE 6
#define DP_VERIFY_DIGEST_ACTION_CODE 7
#define DP_READ_DEVICE_CERTIFICATE_ACTION_CODE 30u
#define DP_ZEROIZE_LIKE_NEW_ACTION_CODE 31u
#define DP_ZEROIZE_UNRECOVERABLE_ACTION_CODE 32u
```

Note:
Programming of individual blocks, such as array only, eNVM only, or security only is not possible with one data file because of how the data is constructed. If you wish to use such a feature, you must generate multiple data files.

3.2.8 Data Type Definitions

Microchip uses `DPUCHAR`, `DPUSHORT`, `DPUINT`, `DPULONG`, `DPBOOL`, `DPCHAR`, `DPINT`, and `DPLONG` in the SPI-DirectC source code. Change the corresponding variable definition, if different data type names are used.

```
/*********************************************/
/* DPCHAR -- 8-bit Windows (ANSI) character */
/* that is 8-bit signed integer */
/* DPINT -- 16-bit signed integer */
/* DPLONG -- 32-bit signed integer */
/* DPBOOL -- boolean variable (0 or 1) */
/* DPUCHAR -- 8-bit unsigned integer */
/* DPUSHORT -- 16-bit unsigned integer */
/* DPUINT -- 16-bit unsigned integer */
/* DPULONG -- 32-bit unsigned integer */
\/*********************************************/
typedef unsigned char DPUCHAR;
typedef unsigned short DPUSHORT;
typedef unsigned int DPUINT;
typedef unsigned long DPULONG;
typedef unsigned char DPBOOL;
typedef char DPCHAR;
typedef int DPINT;
typedef long DPLONG;
```

3.2.9 Supported Actions

The following table lists the supported actions and devices.

Table 3-3. Supported Actions and Devices

Action	Supported Devices	Description
DP_DEVICE_INFO_ACTION	All	Displays device security settings.
DP_READ_IDCODE_ACTION	All	Reads and displays the content of the IDCODE register.
DP_ERASE_ACTION	All	Erases all supported blocks in the data file.
DP_PROGRAM_ACTION	All	Performs erase, program and verify operations for all the supported blocks in the data file.
DP_VERIFY_ACTION	All	Performs verify operation for all the supported blocks in the data file.
DP_ENC_DATA_AUTHENTICATION_ACTION	All	Performs data authentication of the bitstream within the data file.

.....continued

Action	Supported Devices	Description
DP_VERIFY_DIGEST_ACTION_CODE	All	Checks the digest of a programmed target device.
DP_READ_DEVICE_CERTIFICATE_ACTION_CODE	All	Reads and displays device certificate.
DP_ZEROIZE_LIKE_NEW_ACTION_CODE	PolarFire/PolarFire Soc	Performs zeroization action. Device is recoverable.
DP_ZEROIZE_UNRECOVERABLE_ACTION_CODE	PolarFire/PolarFire Soc	Performs zeroization action. Device is not recoverable.

4. Data File Format

The following chapter contains information related to data file format.

4.1 DAT File Description for M2GL, M2S, MPF and MPF SoC Devices

The data file contains the following sections:

Header Block
Contains information identifying the type of the binary file and data size blocks.

Table 4-1. DAT Image Description

Header Section of DAT File	
Information	**# of Bytes**
Designer Version Number	24
Header Size	1
Image Size	4
DAT File Version	1
Tools Version Number	2
Map Version Number	2
Feature Flag	2
Device Family	1

Constant Data Block
Includes device ID, silicon signature, and other information needed for programming.

Table 4-2. DAT Image Description

Constant Data Block	
Information	**# of Bytes**
Device ID	4
Device ID Mask	4
Silicon Signature	4
Checksum	2
Number of BSR Bits	2
Number of Components	2
Data Size	2
Erase Data Size	2
Verify Data Size	2
ENVM Data Size	2
ENVM Verify Data Size	2
UEK1_EXISTS	1

.....continued	
Constant Data Block	
Information	**# of Bytes**
UEK2_EXISTS	1
SEC_ERASE	1
UEK3_EXISTS (M2S, M2GL only)	1
Number of Records	1

Data Lookup Table
Contains records identifying the starting relative location of all the different data blocks used in the SPI-DirectC code and data size of each block. The following table lists the format.

Table 4-3. DAT Image Description

Look-Up-Table	
Information	**# of Bytes**
Data Identifier # 1	1
Pointer to data 1 memory location in the data block section	4
# Of bytes of data 1	4
Data Identifier # 2	1
Pointer to data 2 memory location in the data block section	4
# Of bytes of data 2	4
Data Identifier # x	1
Pointer to data x memory location in the data block section	4
# Of bytes of data x	4

Data Block
Contains the raw data for all the different variables specified in the look-up-table.

Table 4-4. DAT Image Description

Data Block	
Information	**# of Bytes**
Binary Data	Variable
CRC of the entire image	2

5. Source File Description

dpuser.h	Contains definitions of all action codes as well as possible error codes that are reported within SPI-DirectC code.
dpalg.c and dpalg.h	Contain the main entry function `dp_top` and device ID check function.
dpg4alg.c and dpg4alg.h	Contain the main entry function `dp_top_g4` and all other functions common to M2S and MGL families.
dpg5alg.c and dpg5alg.h	Contain the main entry function `dp_top_g5` and all other functions common to the MPF and MPF SoC family of devices.
dpdutspi.c and dpdutspi.h	Contain the SPI interface function declaration and definition to the target device. SPI Mode 3 must be used to program the target device. See SPI IP block used for proper initialization.
dpg4spi.c and dpg4spi.h	Contain the SPI interface function declaration and definition to the target device specific to M2S and M2GL device families.
dpg5spi.c and dpg5spi.h	Contain the SPI interface function declaration and definition to the target device specific to MPF and MPF SoC device families.
dputil.c and dputil.h	Contain utility functions needed in the SPI-DirectC code.

6. Data File Bit Orientation

This chapter specifies the data orientation of the binary data file generated by the Libero SoC Design Suite. The SPI-DirectC implementation must be in sync with the specified data orientation. The following table lists how the data is stored in the binary data file. For more information, see 4. Data File Format.

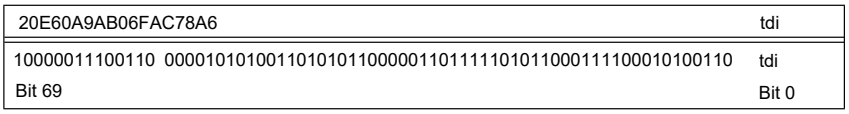
Table 6-1. Binary Data File Example

Byte 0	Byte 1	Byte 2	Byte 3	..	..	Byte N
Bit7.Bit0	Bit15.Bit8	Bit23.Bit16	Bit35.Bit24	..	..	Bit(8N+7).Bit(8N)
Valid Data	Valid Data	Valid Data	Valid Data	..	..	o <-Valid Data

If the number of bits in a data block is not a multiple of eight, the rest of the Most Significant Bit (MSB) in the last byte are filled with zeros. The following example shows a given 70-bit data to be shifted into the target shift register from the Least Significant Bit (LSB) to the Most Significant Bit (MSB).

The following figure shows a binary representation of the same data.

Figure 6-1. Binary representation of data



This data is stored in the data block section. The following table lists how the data is stored in the data block.

Table 6-2. Data Block Section Example

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	..	Byte 8
Bit7...Bit0	Bit15.Bit8	Bit23.Bit16	Bit31.Bit24	Bit43.Bit32	..	Bit71..Bit64
10100110	01111000	10101100	01101111	10110000	-	00100000
A6	78	AC	6F	B0	-	20

7. Sample Project

The sample project, IAR_SPI_SlaveDirectC.zip, available with this release of SPI-DirectC is based on IAR Embedded Workbench version 7.40. It is designed to work on M2GL_M2S-EVAL-KIT with SmartFusion2 M2S025-FGG484 device.

7.1 Project Requirements

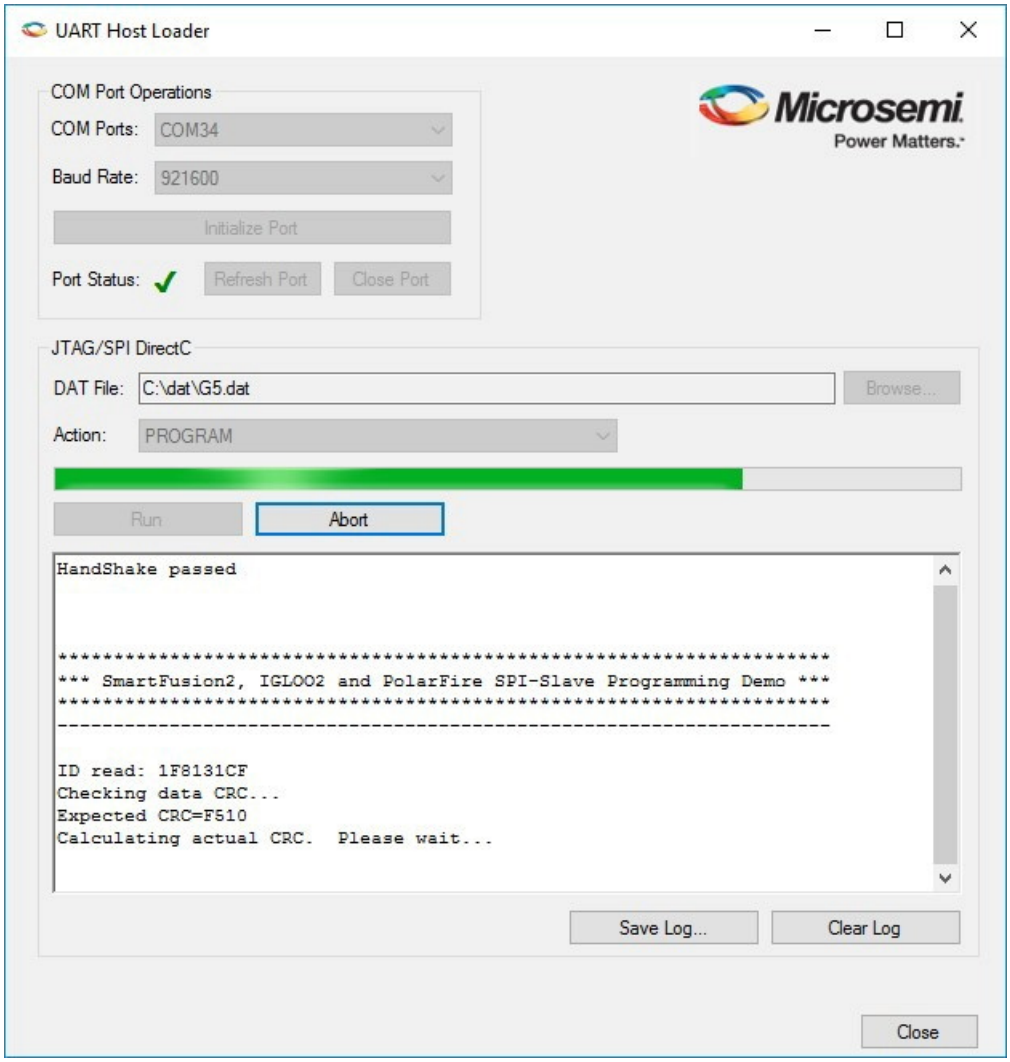
You need the following hardware and software to run the sample project:

- Hardware:
  a. SmartFusion2 Security Evaluation Kit with SmartFusion2 M2S090-FGG484 device
  b. jLink from IAR
  c. Target board with Microchip device to be programmed
- Software:
  a. IAR Embedded Workbench version 7.4
  b. UART Host Loader available with this release package

7.2 Procedure

1. Program the evaluation kit with SPI_DC_top.job file included under M2S Eval Kit Files directory. The M2S090 design connects SPI1 port to certain pins of J1 header. Although not needed for this project, it also maps out specific MSS IOs to other J1 header pins for JTAG access.
2. Connect the SPI pins as described in HeaderPinAssignment.xlsx available under M2S Eval Kit Files directory. Ignore JTAG portion of the header.
3. Connect the Mini USB (J18) to your PC. The mini-USB is connected to FTDI FT4232h device used as a USB to UART bridge.
4. Make sure the appropriate drivers are installed on your PC to communicate with this chip.
5. Run UARTHostLoader application available with this release package.
6. There should be four com ports available in the serial port setup window. Select the fourth one from the list and configure the Baud Rate as shown in the following figure. If more than four ports are available, disconnect the J18 header and refresh the com ports in the UARTHostLoader application to identify exiting ports. Reconnect the J18 header and refresh the USB ports. Select the fourth port from the newly generated port list.
7. Click **Initialize Port** to establish connection with the selected COM port.
8. Select the programming file and desired action.
9. Click **Run**. The UARTHostLoader application waits for data from the SmartFusion2 evaluation kit.
10. The programming file programmed into the evaluation kit has a SPI-DirectC sample project that supports SmartFusion2, IGLOO2, PolarFire, and PolarFire SoC devices. Resetting the board runs the embedded application and performs the action selected. To run another action or select a different programming file, select it from the UARTHostLoader. Click **Run**.
11. To make changes to the embedded project, run IAR workbench and modify the compile options as required. You can download the embedded application using jLink as follows:
    a. Connect jLink to RVI/IAR header.
    b. Set the JTAG select jumper low.
    c. Click **download** and run from IAR.

Figure 7-1. UART Host Loader



8. Error Messages and Troubleshooting Tips

The information in this chapter may help you solve or identify a problem when using SPI-DirectC code. See the following table for a description of exit codes and their solutions.

Table 8-1. Exit Code

Exit Code	Error Message	Action/Solution
0	This code does not indicate an error.	This message indicates success
2	Data processing failed.	Solution: • Check the V_{pump} level. • Try with a new device. • Measure SPI pins and noise or reflection. • Load the correct DAT file.
6	The IDCODE of the target device does not match the expected value in the DAT file image.	Possible Causes: • The data file loaded is compiled for a different device. Example: M2S010 DAT file loaded to program M2S050 device. • Noise or reflections on one or more of the SPI pins causing incorrect read-back of the SDO bits. Solution: • Choose the correct DAT file for the target device. • Cut down the extra length of ground connection.
7	Device polling error.	Solution: • Check the V_{pump} level • Try with a new device • Measure SPI pins and noise or reflection. • Load the correct DAT file.
8	FPGA failed during the Erase operation.	Possible Causes: • The device is secured, and the corresponding data file is not loaded. The device has been permanently secured and cannot be unlocked. Solution: • Load the correct DAT file.
10	Failed to program device.	Solution: • Check V_{pump} level. • Try with a new device. • Measure the SPI pins and noise or reflection.

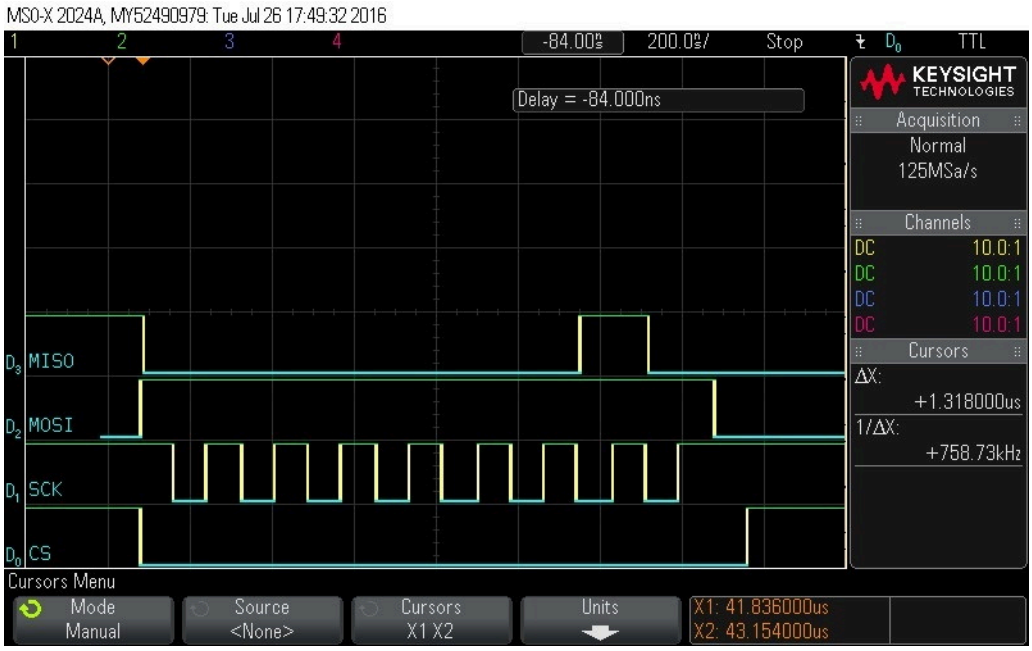
.....continued		
Exit Code	**Error Message**	**Action/Solution**
11	FPGA failed verify.	Possible Cause: • The device is secured, and the corresponding DAT file is not loaded. • The device is programmed with an incorrect design. Solution: • Load the correct DAT file. • Check V_{pump} level. • Measure the SPI pins and noise or reflection.
18	Failed to authenticate the encrypted data.	Make sure the AES key used to encrypt the data matches the AES key programmed in the device.
25	Device initialization failure.	Solution: • Check V_{pump} level. • Try with a new device. • Measure the SPI pins and noise or reflection.
100	CRC data error. Data file is corrupted or pro- gramming on system board is not successful.	Solution: • Regenerate data file. • Reprogram data file into system memory.
150	Request action is not found.	Check spelling.
151	Action is not supported because required data block is missing from the data file.	Regenerate DAT file with the needed block/feature support.

9. SmartFusion2 and IGLOO2 SPI-Slave Programming Waveform Analysis

9.1 Read ID Code Waveform

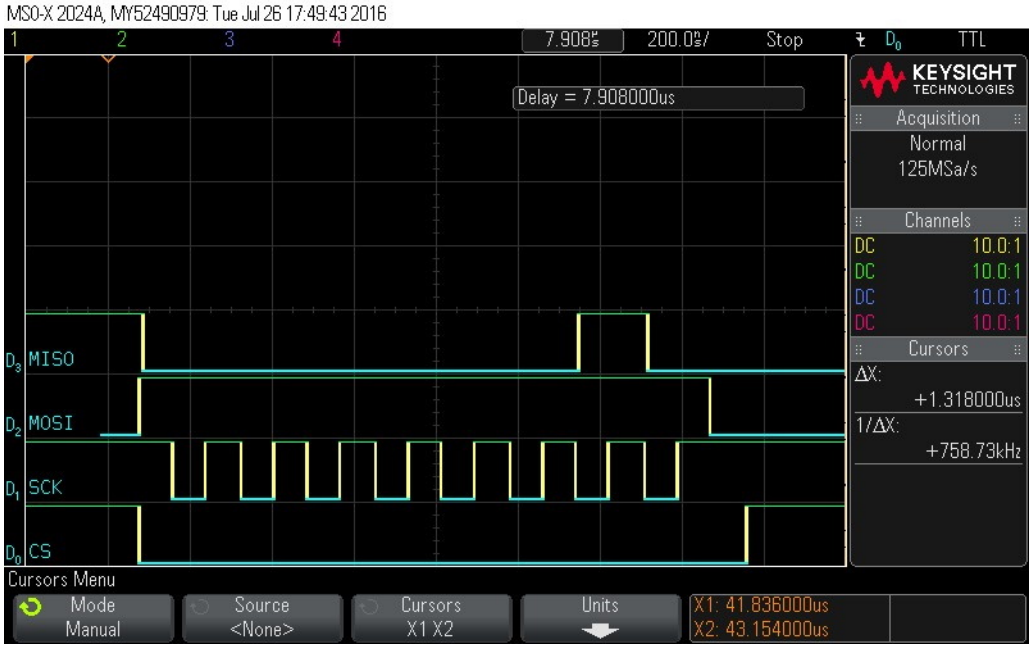
1. Checking hardware status.

Figure 9-1. Hardware Status Check



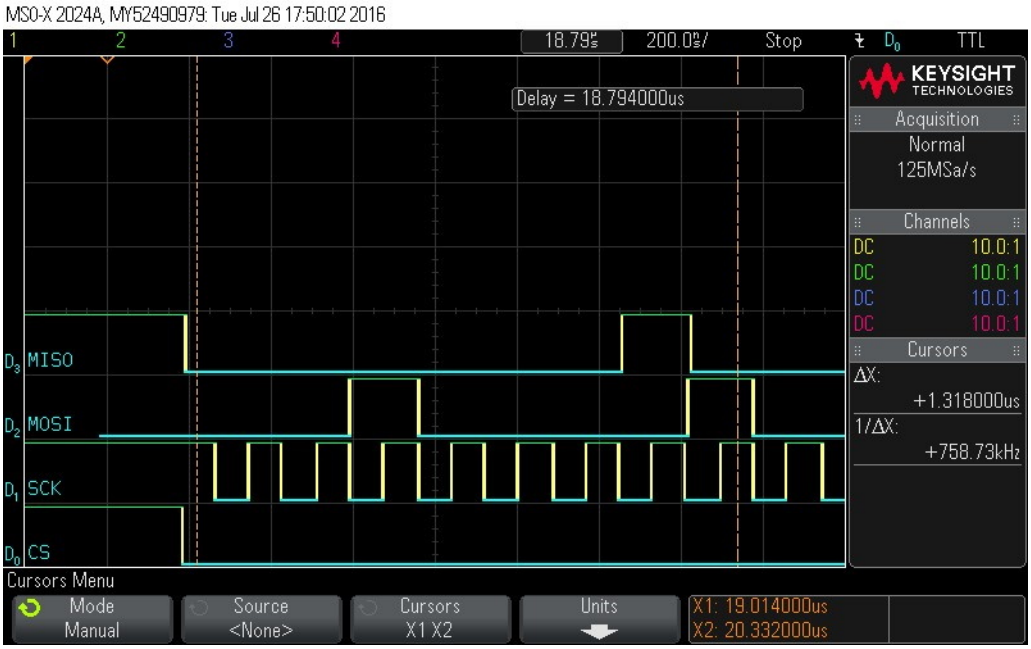
2. Checking hardware status.

Figure 9-2. Hardware Status Check



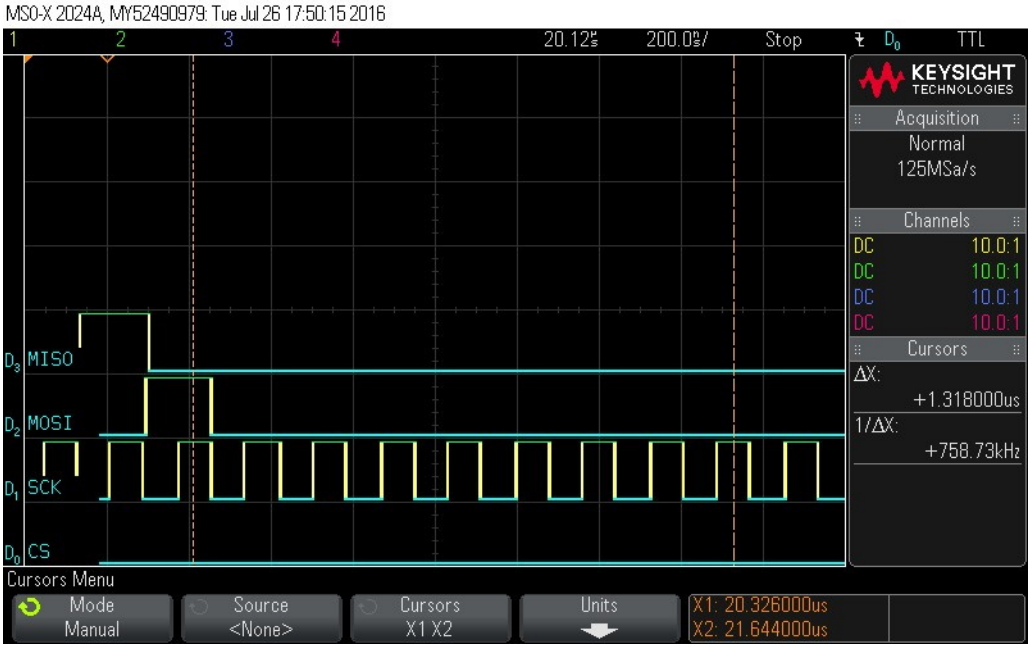
3. Clocking in `read_id` command.

Figure 9-3. Clock in read_id Command (0x21)



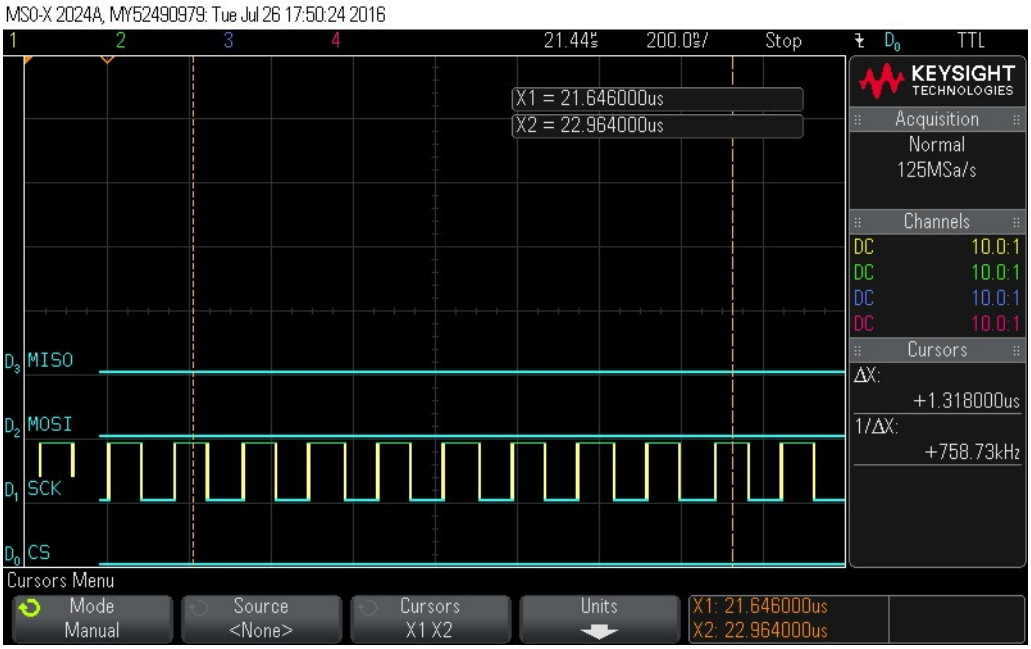
4. Clocking in 16 bytes of zero values - Byte 0

Figure 9-4. Clock in 16 Bytes of Zero Values - Byte 0



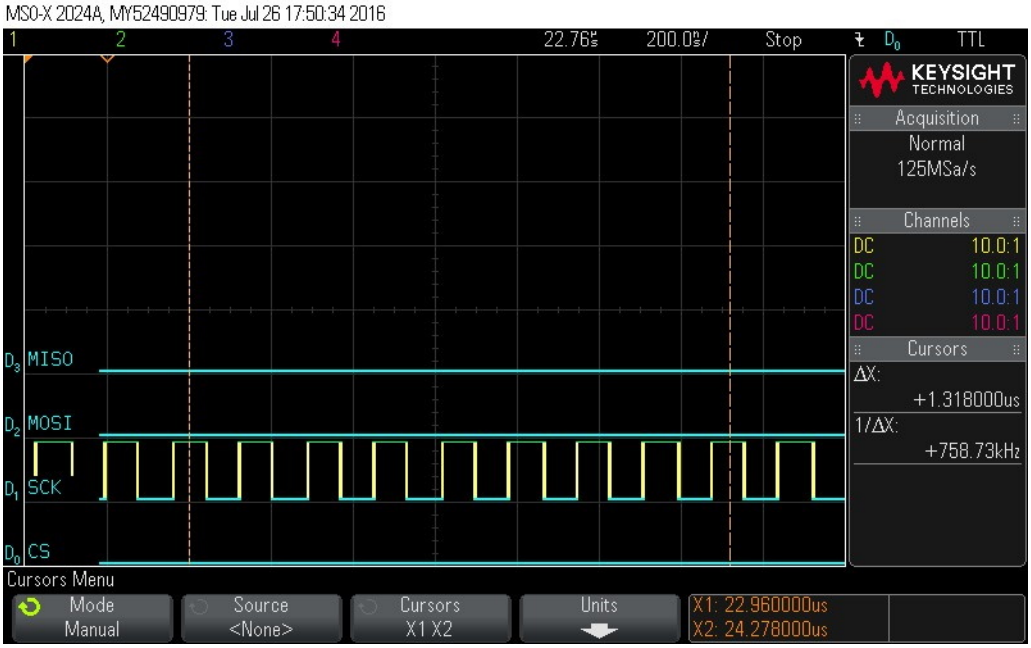
5. Clock in 16 bytes of zero values - Byte 1.

Figure 9-5. Clock in 16 Bytes of Zero Values - Byte 1



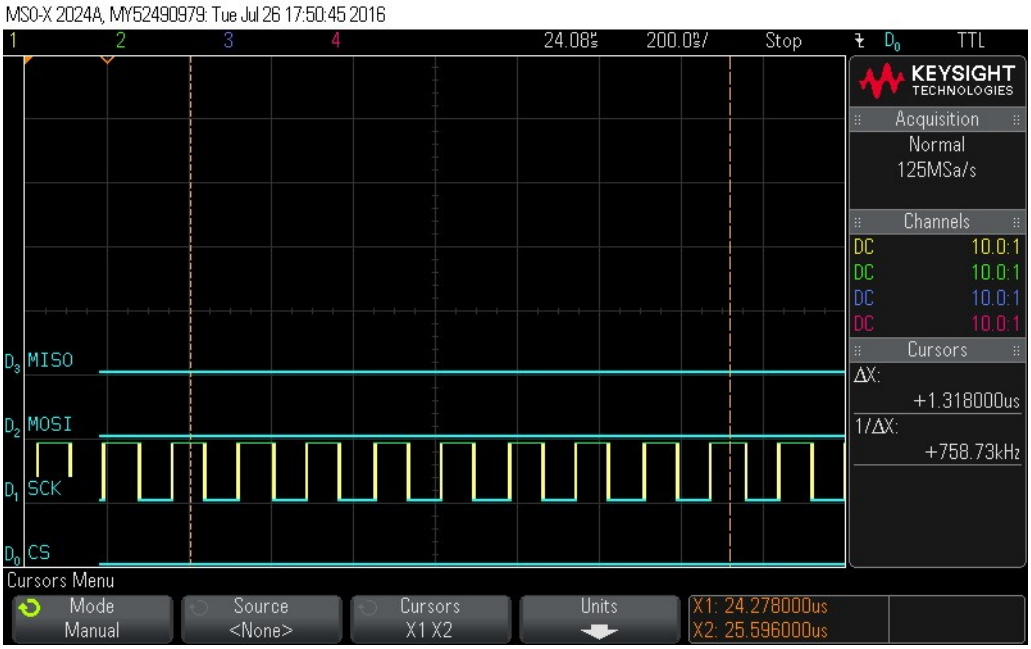
6. Clocking in 16 bytes of zero values - Byte 2.

Figure 9-6. Clock in 16 Bytes of Zero Values - Byte 2



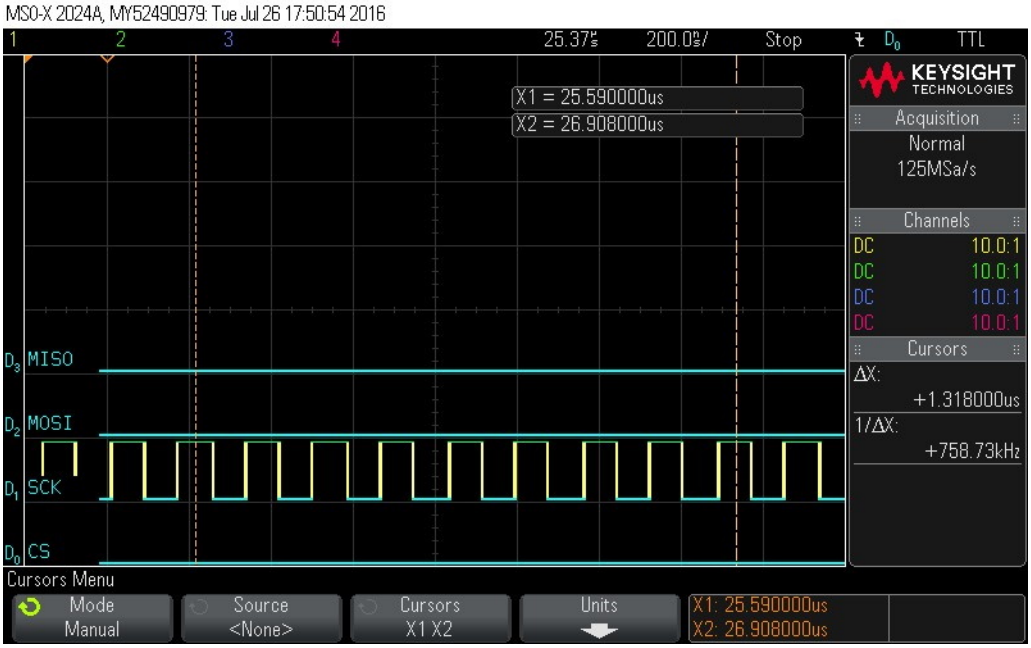
7. Clocking in 16 Bytes of Zero Values - Byte 3.

Figure 9-7. Clock in 16 bytes of zero values - Byte 3



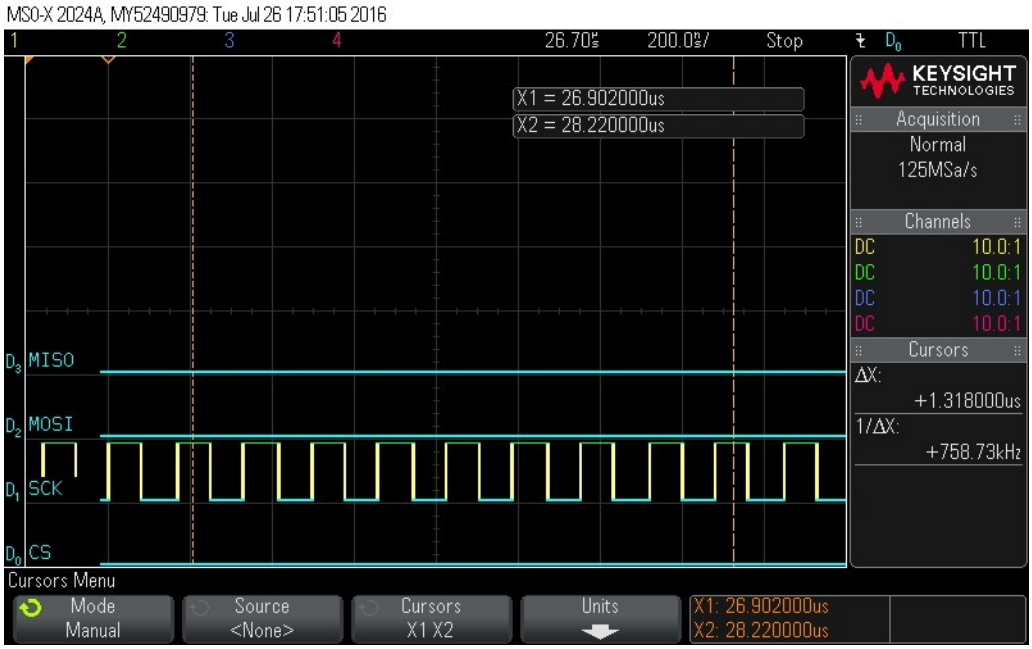
8. Clocking in 16 bytes of zero values - Byte 4.

Figure 9-8. Clock in 16 Bytes of Zero Values - Byte 4



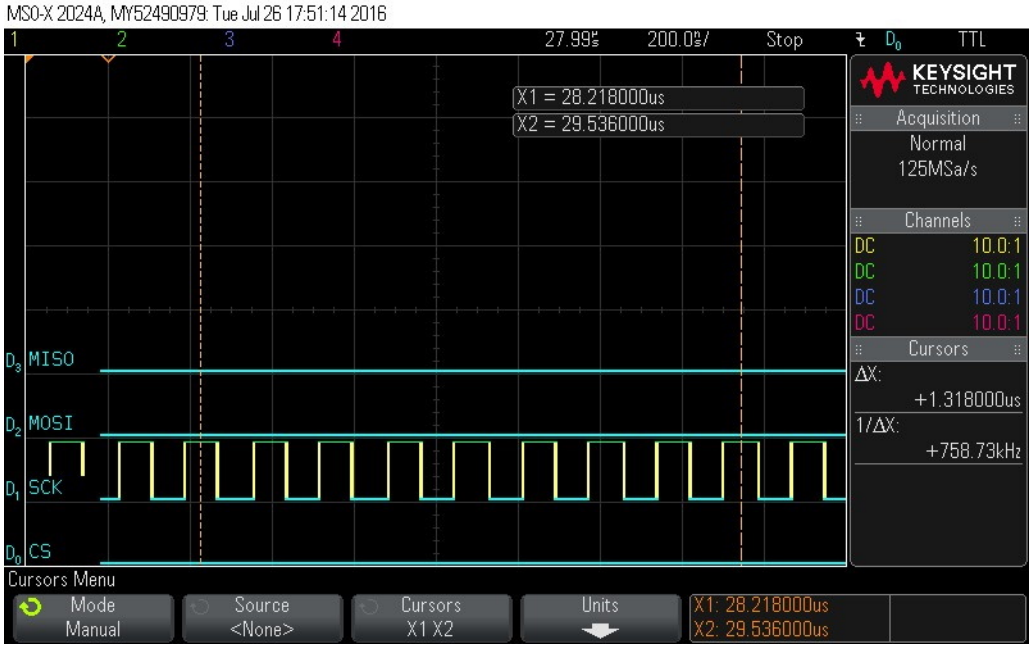
9. Clocking in 16 bytes of zero values - Byte 5.

Figure 9-9. Clock in 16 Bytes of Zero Values - Byte 5



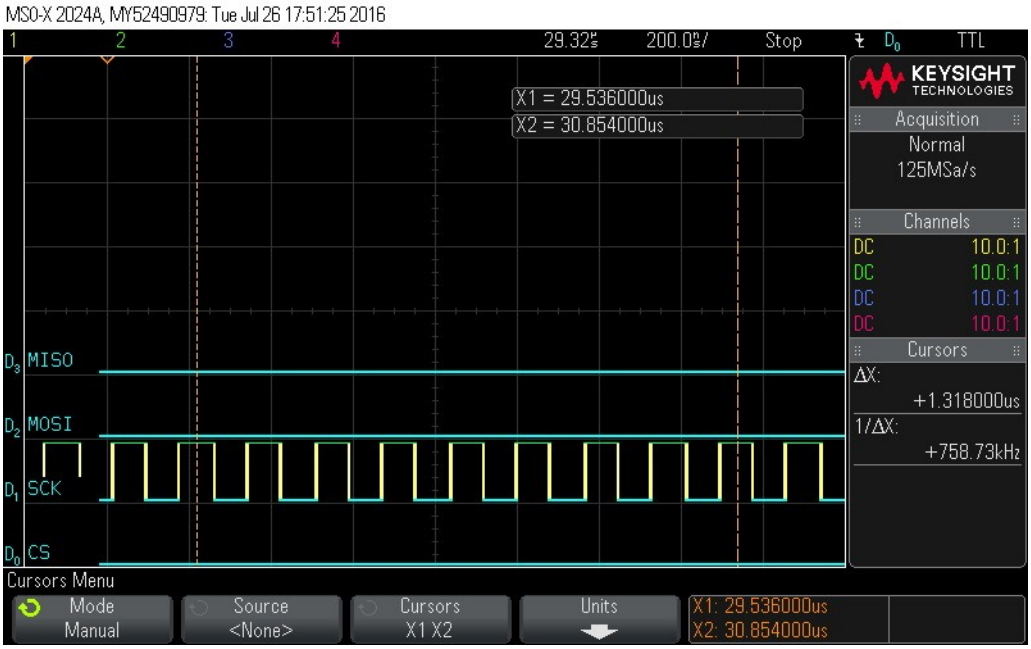
10. Clocking in 16 bytes of zero values - Byte 6.

Figure 9-10. Clock in 16 Bytes of Zero Values - Byte 6



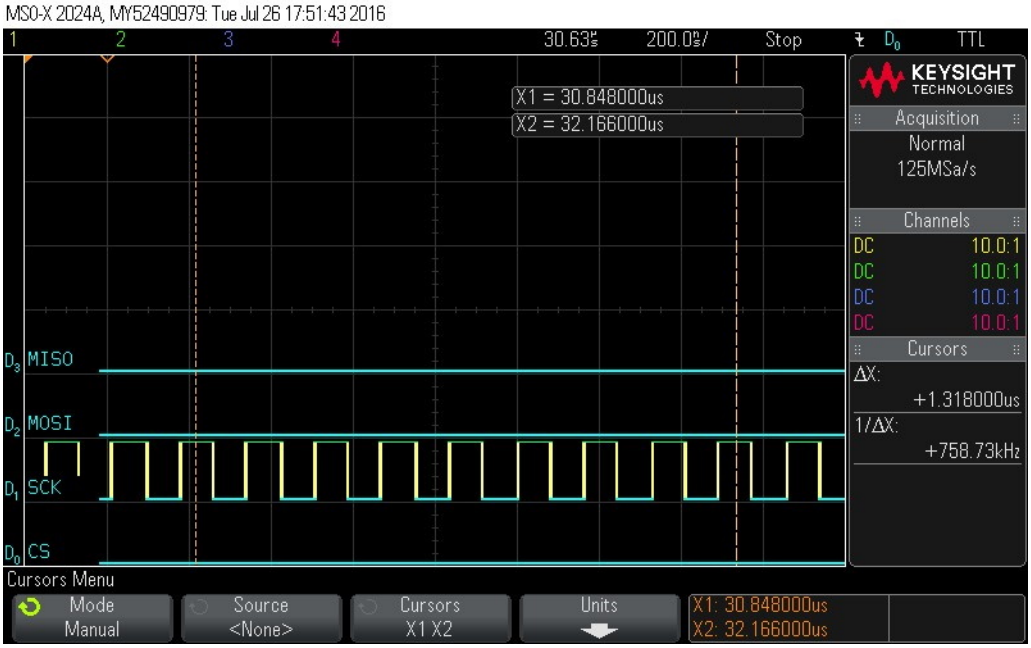
11. Clocking in 16 bytes of zero values - Byte 7.

Figure 9-11. Clock in 16 Bytes of Zero Values - Byte 7



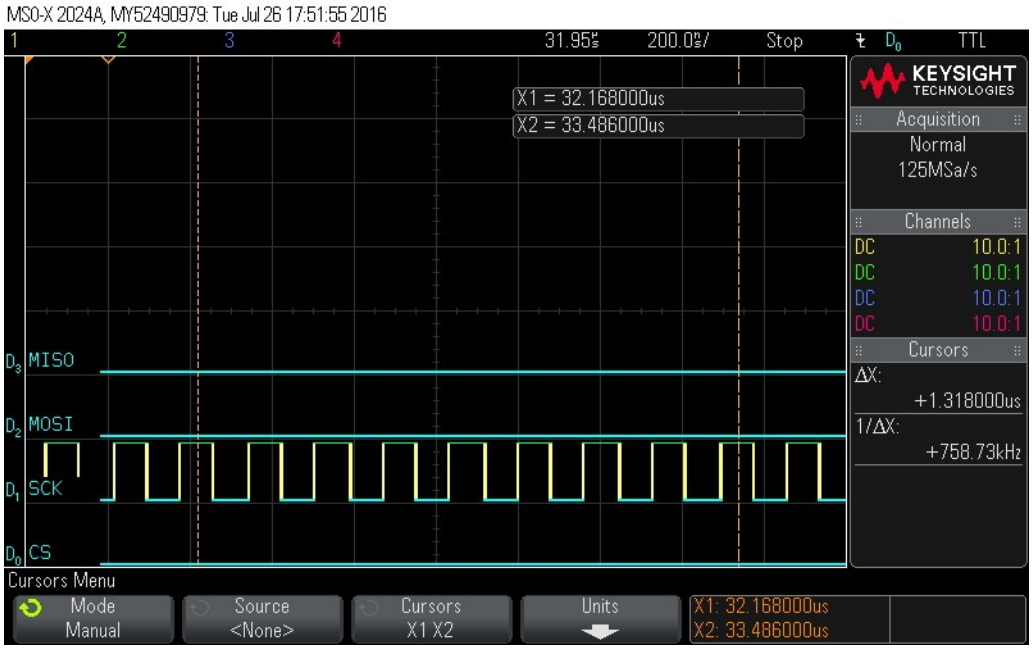
12. Clocking in 16 bytes of zero values - Byte 8.

Figure 9-12. Clock in 16 Bytes of Zero Values - Byte 8



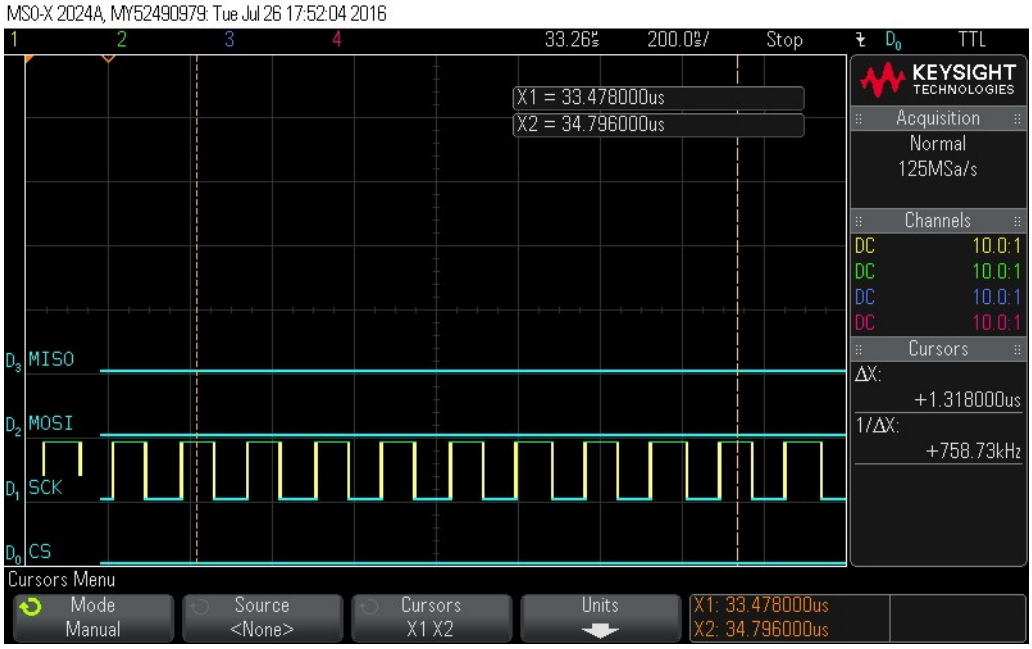
13. Clocking in 16 bytes of zero values - Byte 9.

Figure 9-13. Clock in 16 Bytes of Zero Values - Byte 9



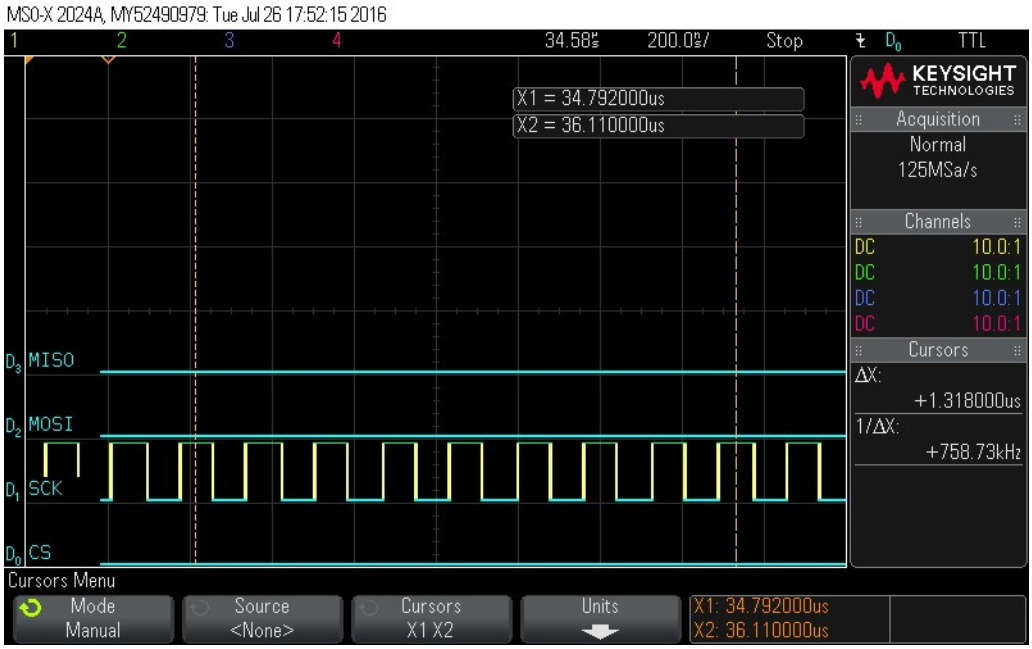
14. Clocking in 16 bytes of zero values - Byte 10.

Figure 9-14. Clock in 16 Bytes of Zero Values - Byte 10



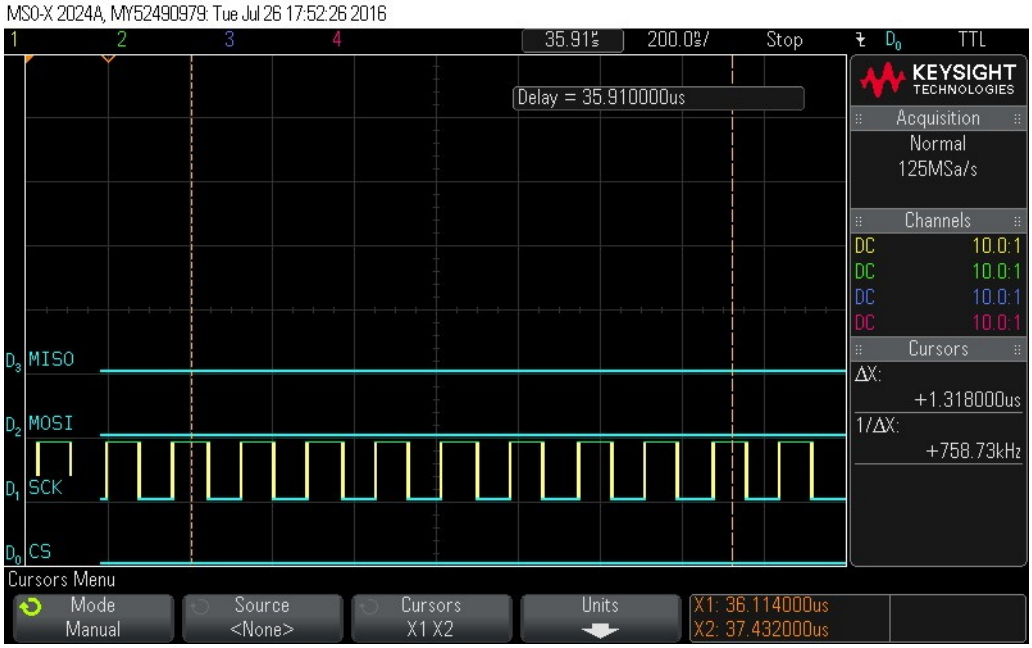
15. Clocking in 16 bytes of zero values - Byte 11.

Figure 9-15. Clock in 16 Bytes of Zero Values - Byte 11



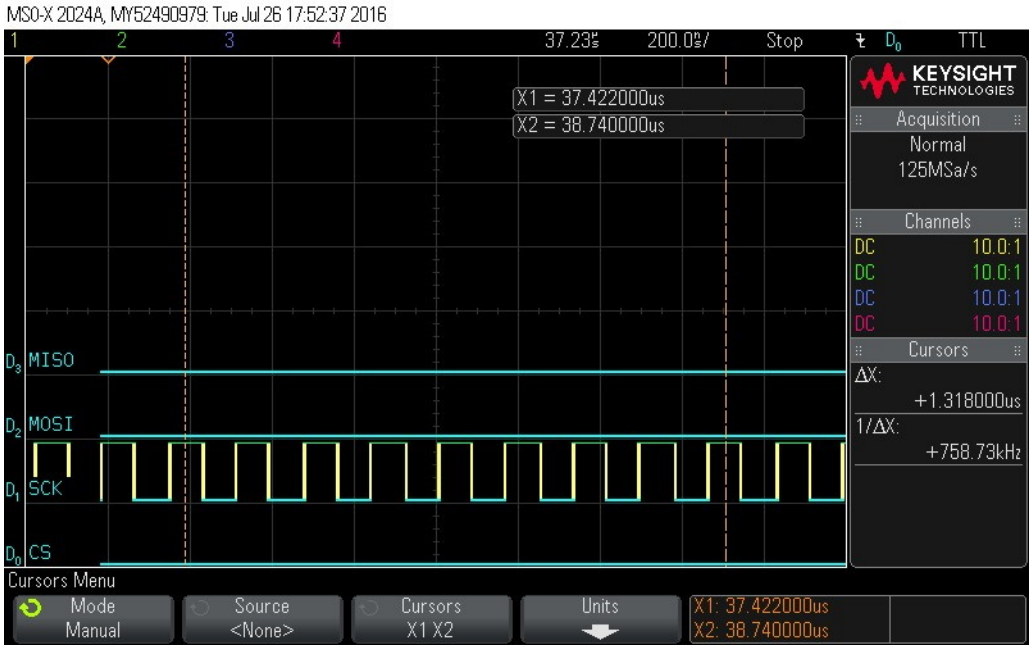
16. Clock in 16 bytes of zero values - Byte 12.

Figure 9-16. Clock in 16 bytes of Zero Values - Byte 12



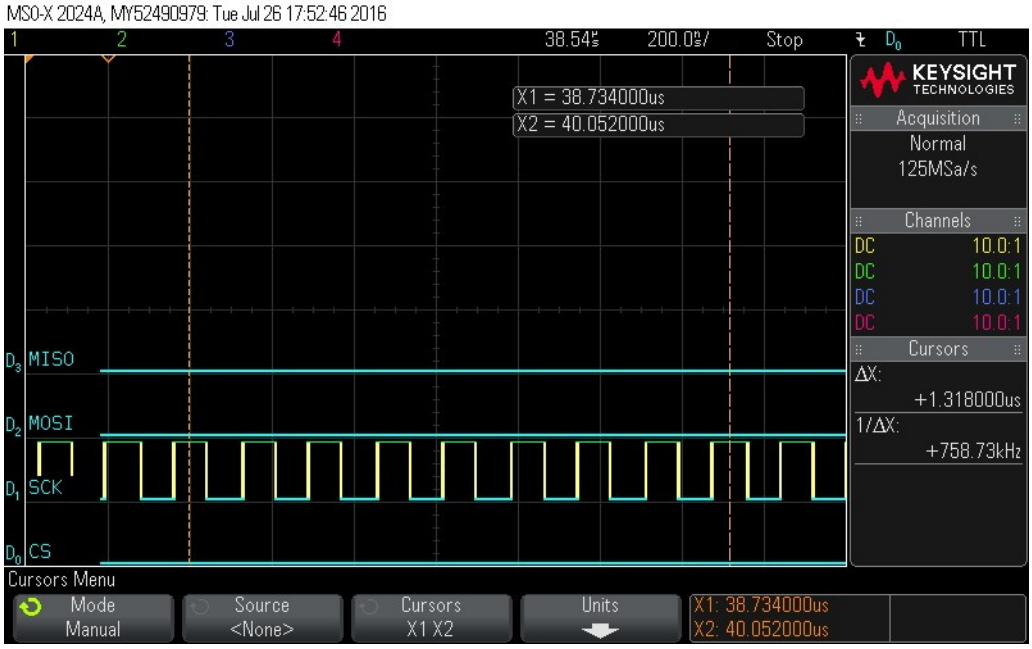
17. Clock in 16 bytes of zero values - Byte 13.

Figure 9-17. Clock in 16 Bytes of Zero Values - Byte 13



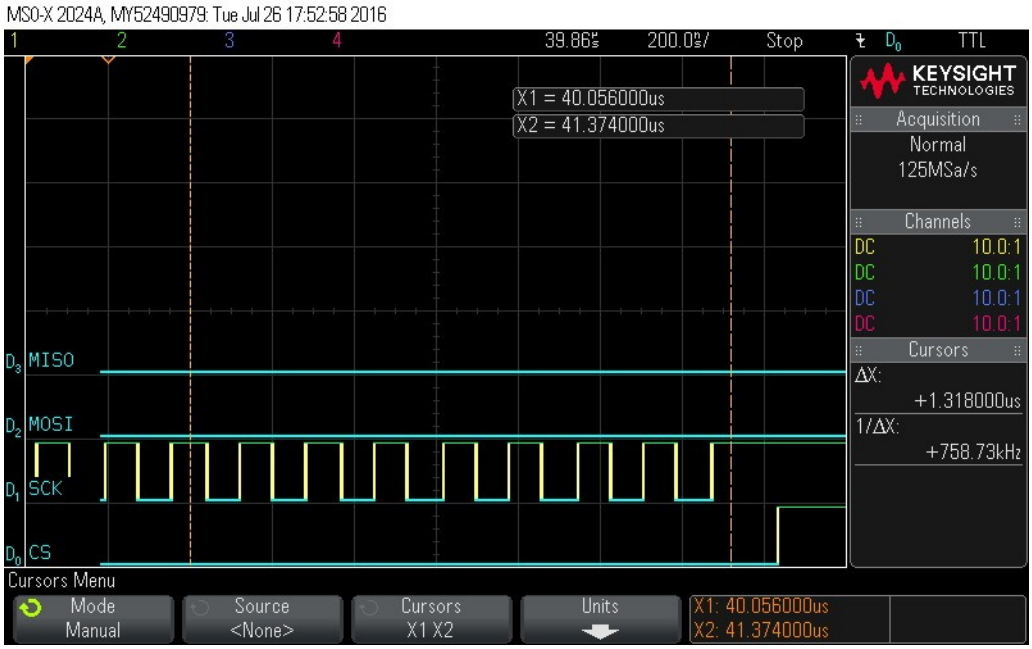
18. Clock in 16 bytes of zero values - Byte 14.

Figure 9-18. Clock in 16 Bytes of Zero Values - Byte 14



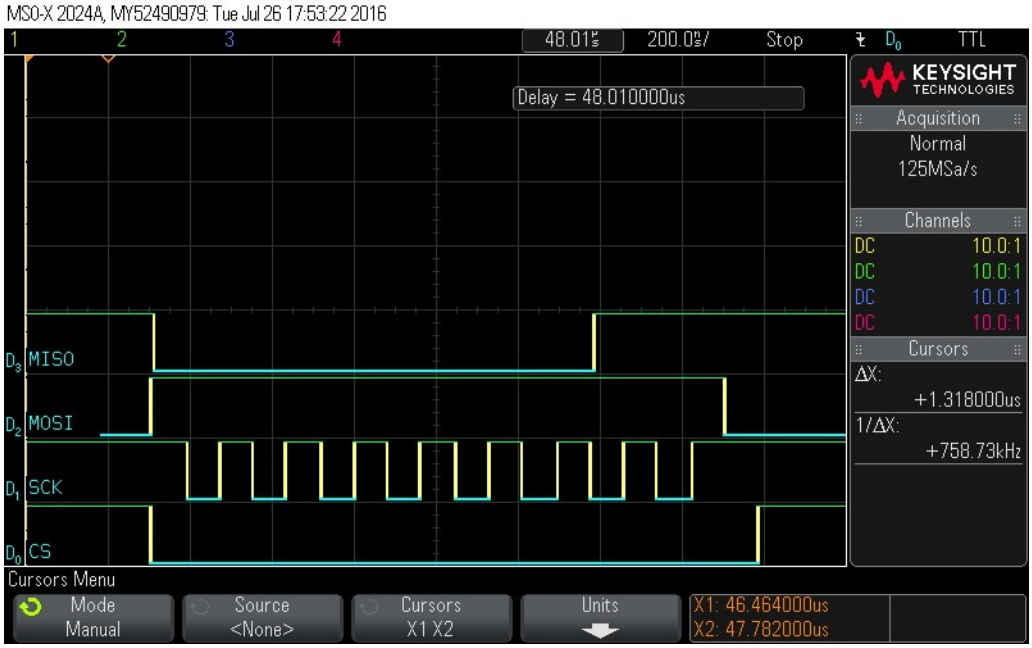
19. Clock in 16 bytes of zero values - Byte 15.

Figure 9-19. Clock in 16 Bytes of Zero Values - Byte 15



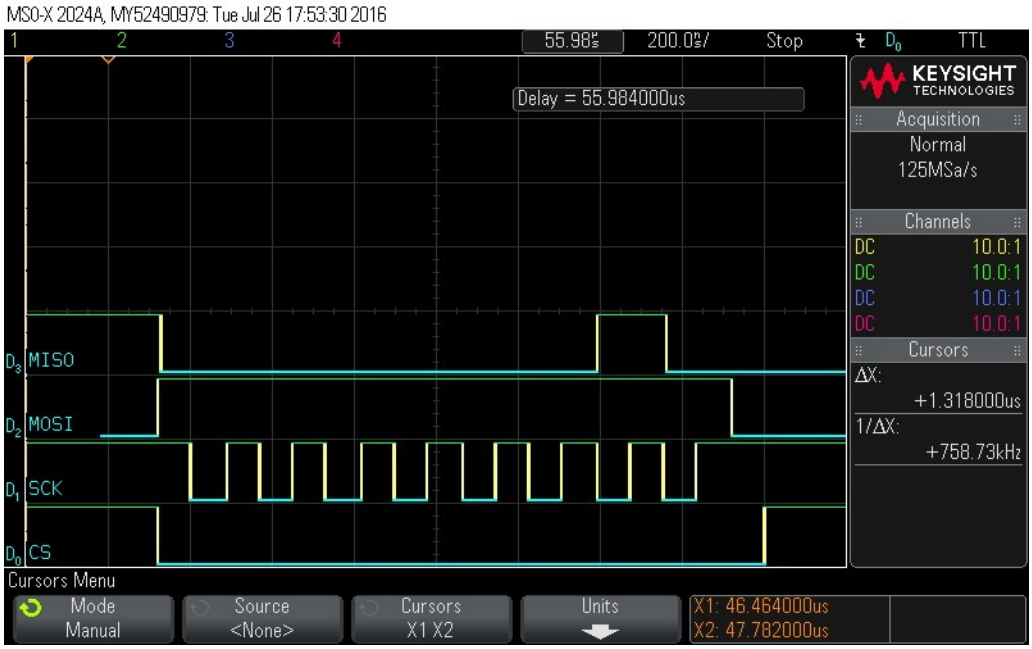
20. Checking hardware status.

Figure 9-20. Hardware Status Check



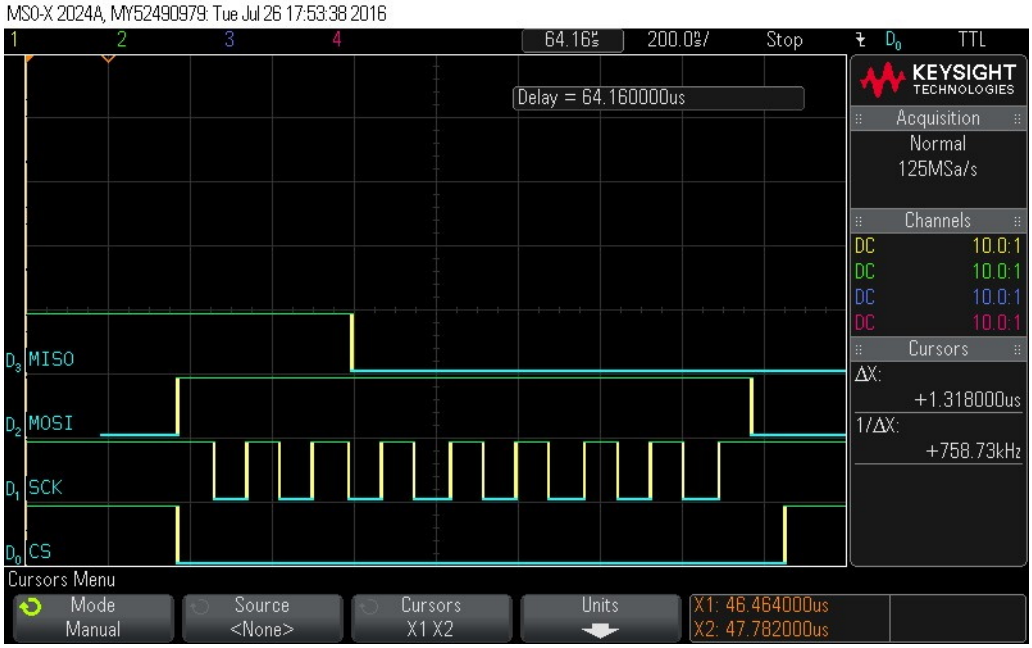
21. Checking hardware status.

Figure 9-21. Hardware Status Check



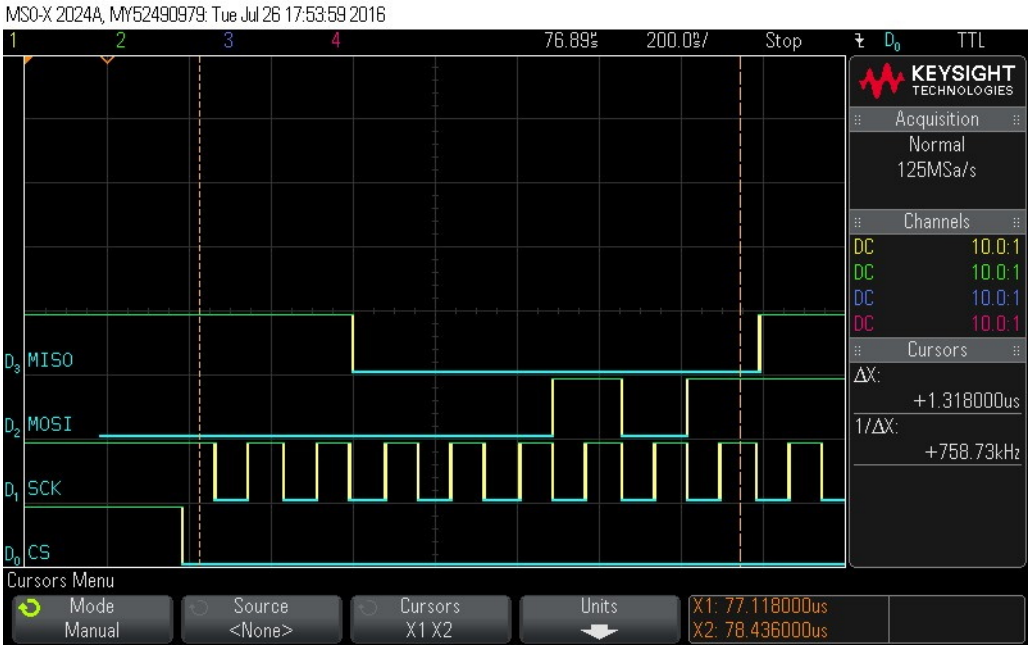
22. Checking hardware status.

Figure 9-22. Hardware Status Check



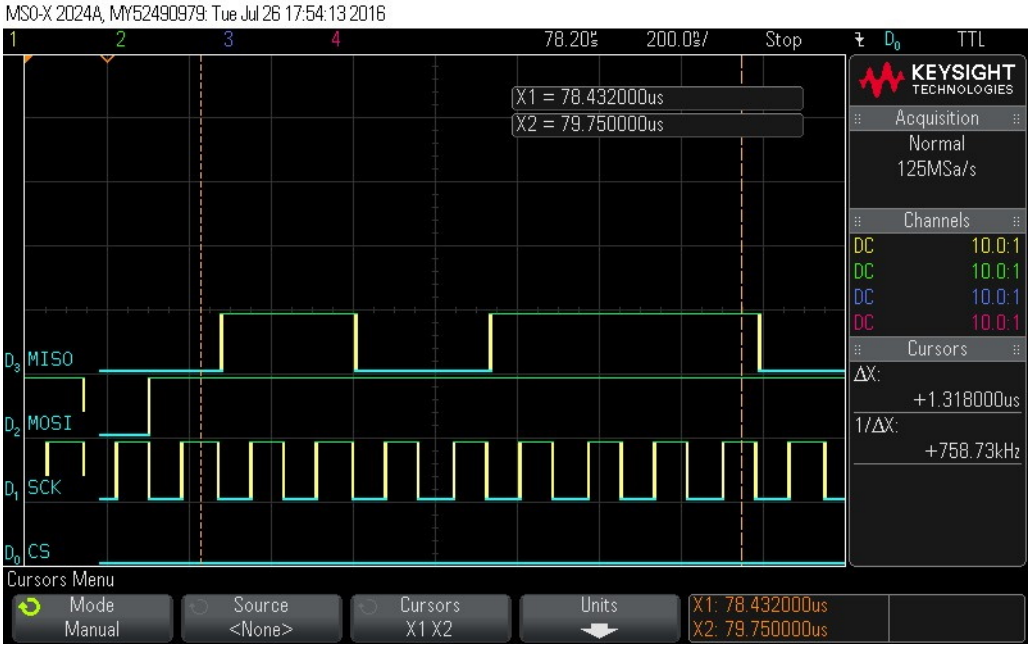
23. Clocking in read command (0x5).

Figure 9-23. Clock in Read Command (0x5)



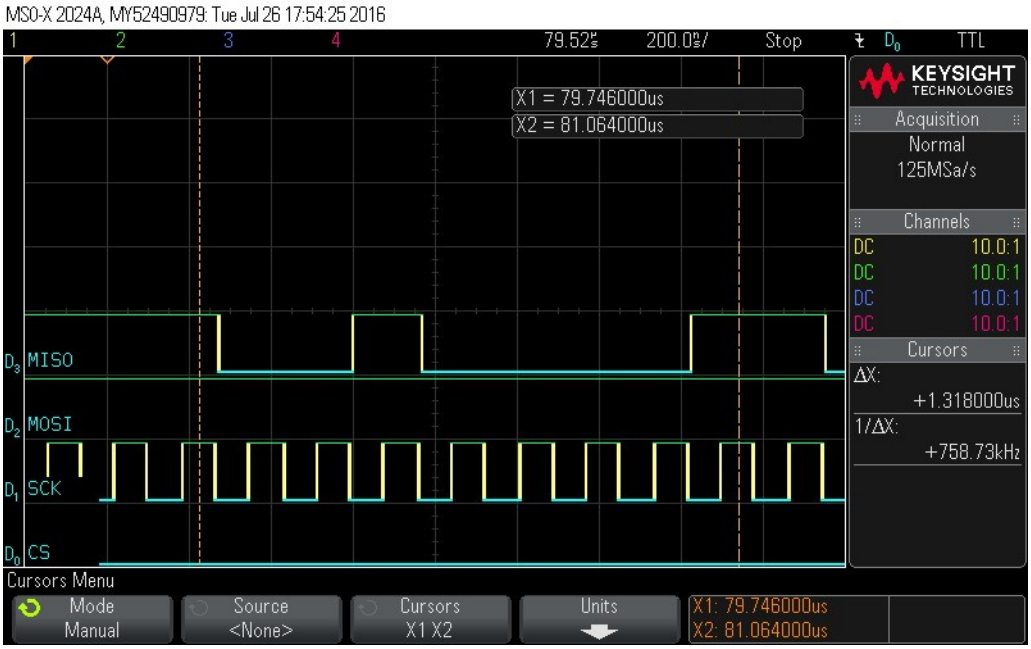
24. Reading out 16 Bytes of data – Byte 0 = 0xCF

Figure 9-24. Reading out 16 Bytes of Data – Byte 0 = 0xCF



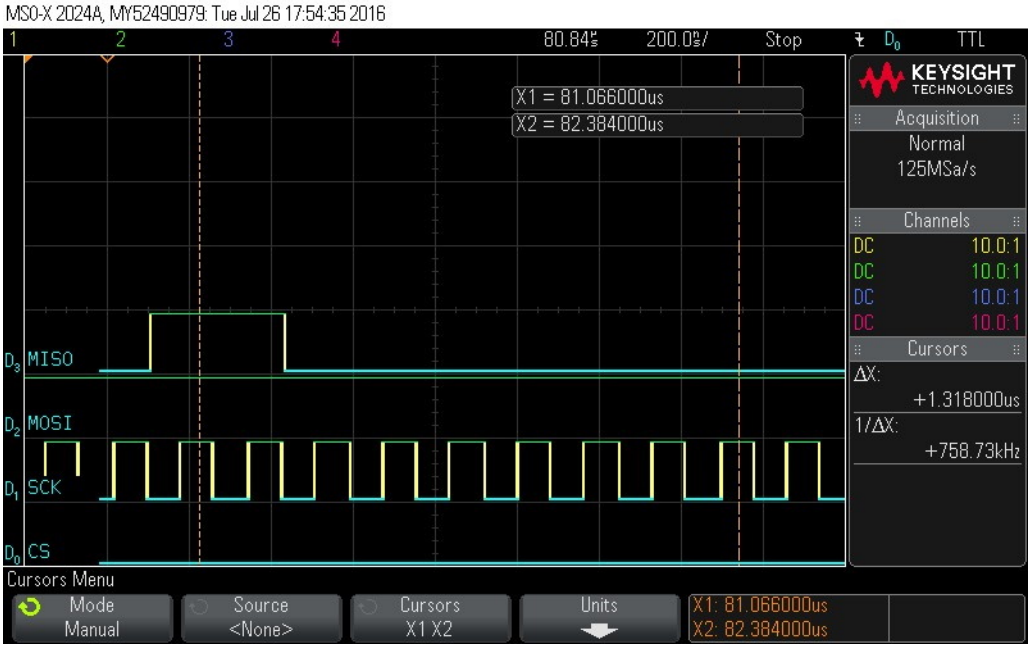
25. Reading out 16 Bytes of data – Byte 1 = 0x21.

Figure 9-25. Reading out 16 Bytes of Data – Byte 1 = 0x21



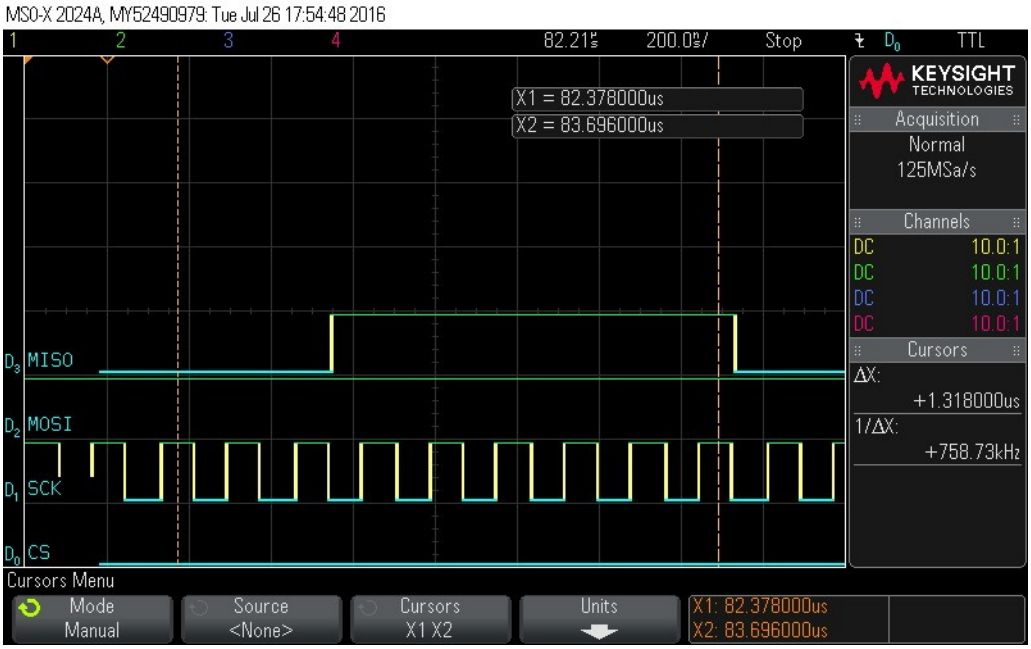
26. Reading out 16 Bytes of data – Byte 2 = 0x80.

Figure 9-26. Reading out 16 Bytes of Data – Byte 2 = 0x80



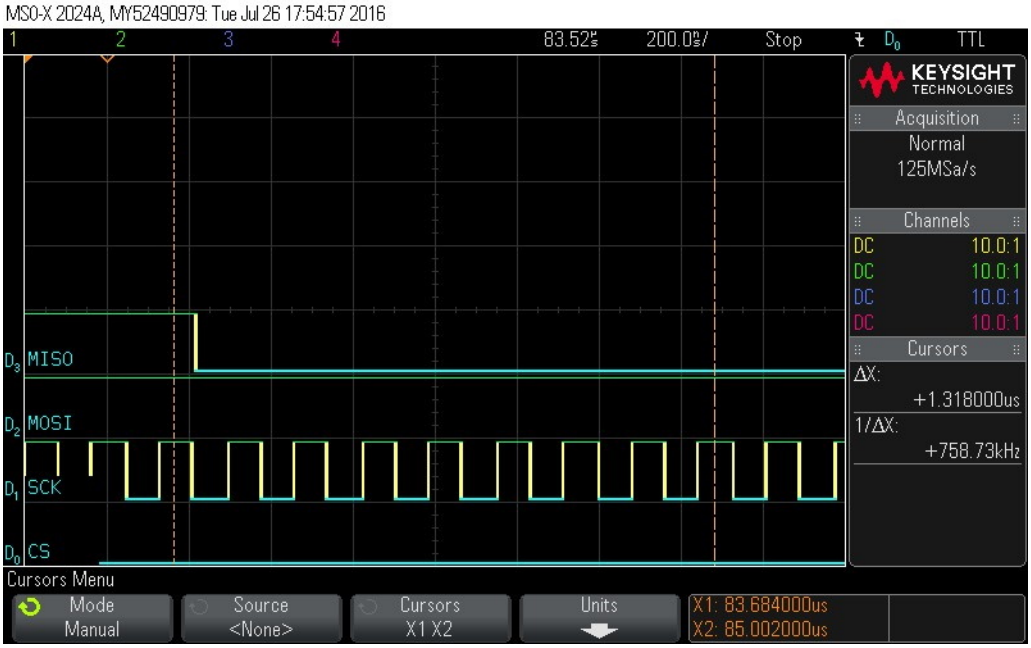
27. Reading out 16 Bytes of data – Byte 3 = 0x3F.

Figure 9-27. Reading out 16 Bytes of Data – Byte 3 = 0x3F



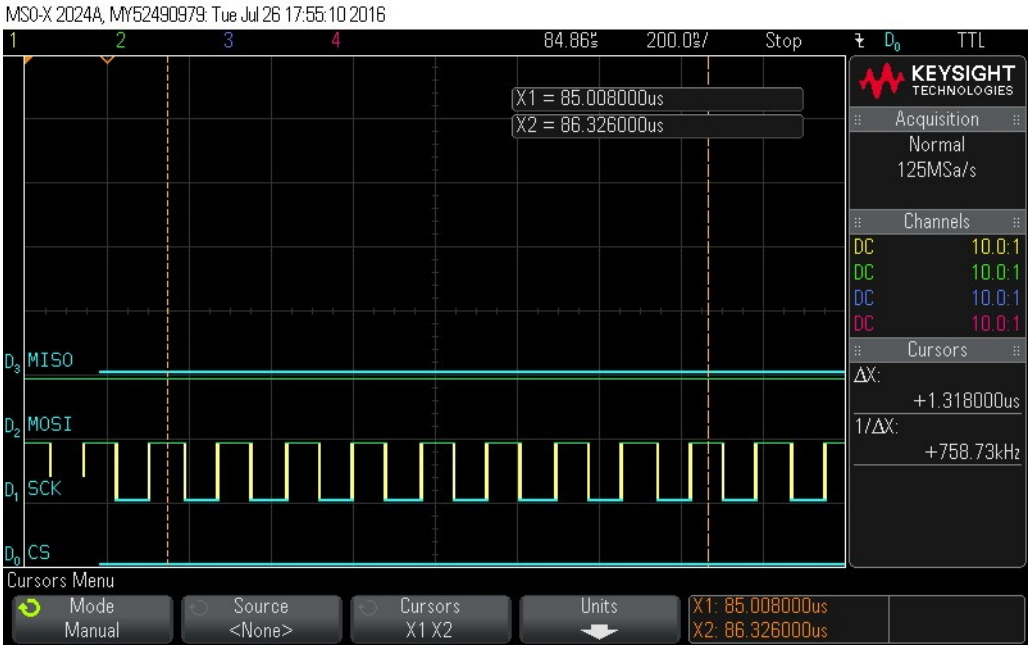
28. Reading out 16 Bytes of data – Byte 4 = 0x0.

Figure 9-28. Reading out 16 Bytes of Data – Byte 4 = 0x0



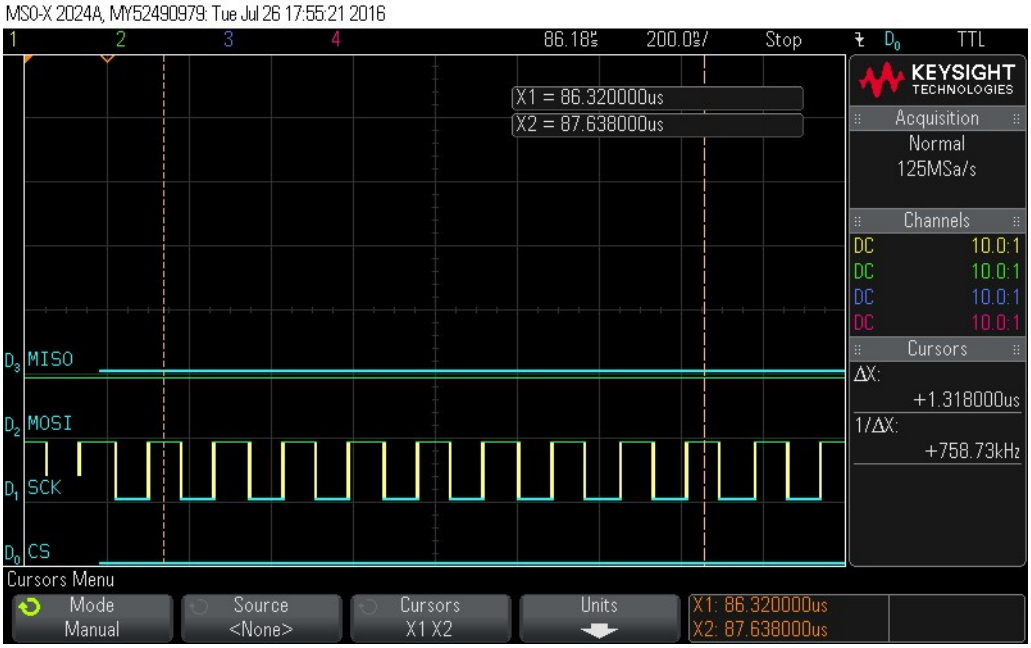
29. Reading out 16 Bytes of data – Byte 5 = 0x0.

Figure 9-29. Reading out 16 Bytes of Data – Byte 5 = 0x0



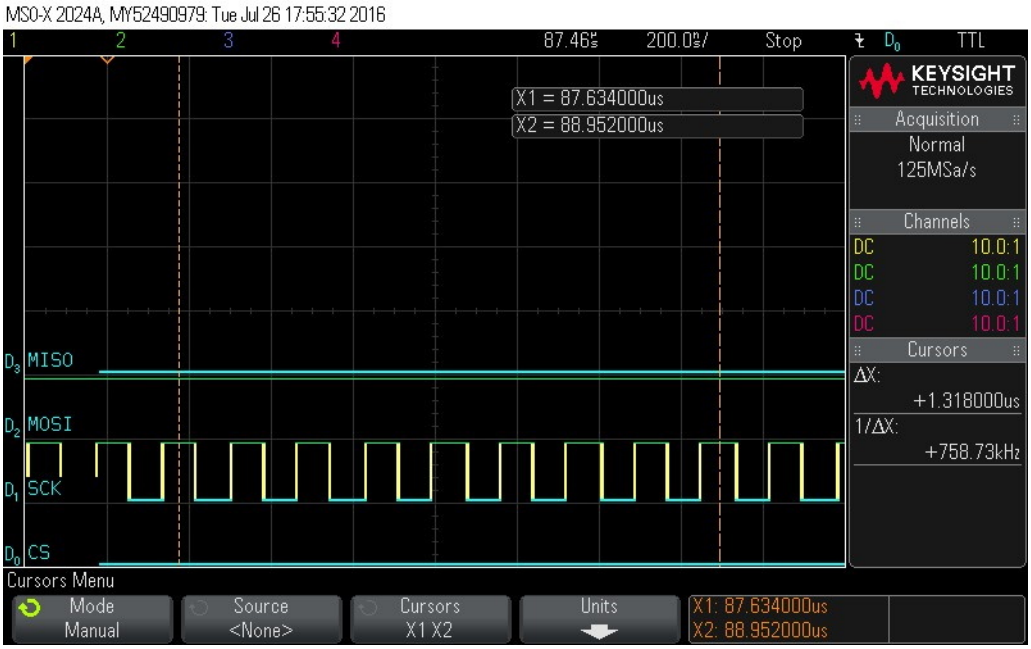
30. Reading out 16 Bytes of data – Byte 6 = 0x0.

Figure 9-30. Reading out 16 Bytes of Data – Byte 6 = 0x0



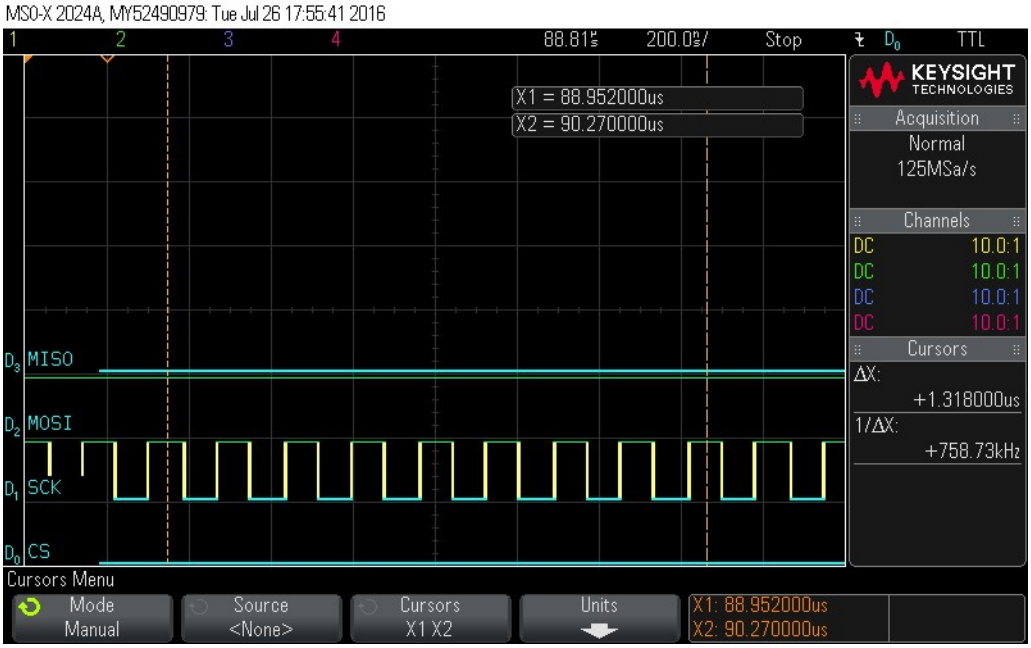
31. Reading out 16 Bytes of data – Byte 7 = 0x0.

Figure 9-31. Reading out 16 Bytes of Data – Byte 7 = 0x0



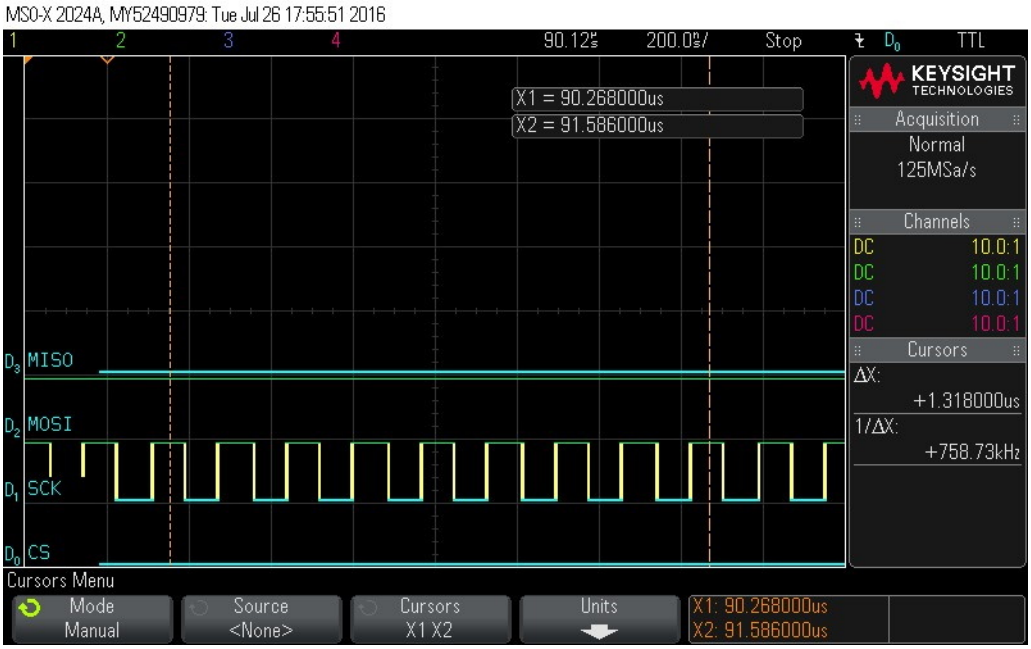
32. Reading out 16 Bytes of data – Byte 8 = 0x0.

Figure 9-32. Reading out 16 Bytes of Data – Byte 8 = 0x0



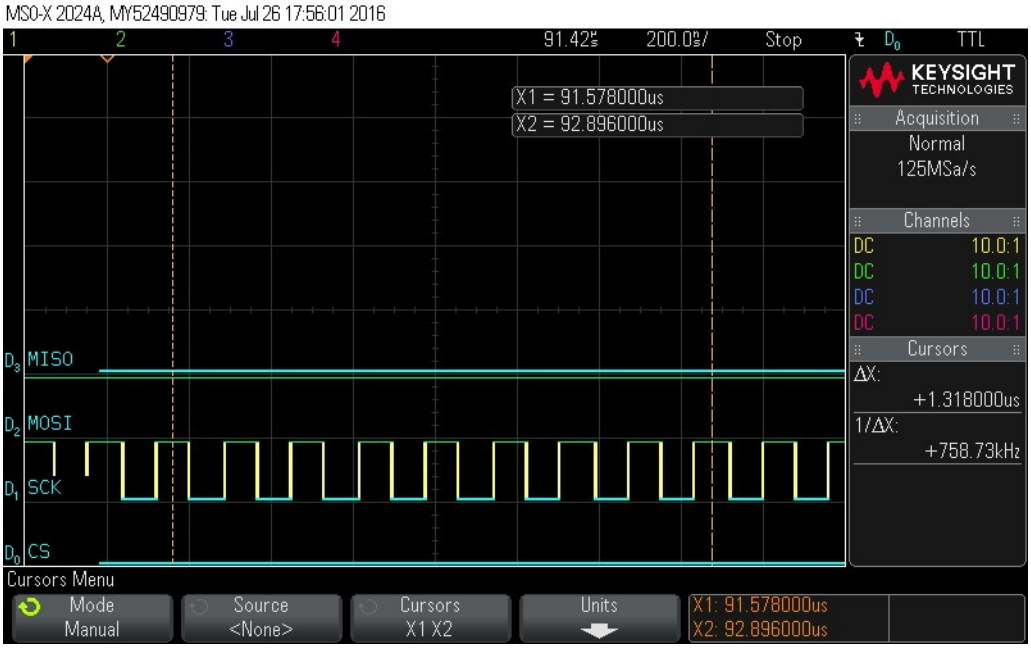
33. Reading out 16 Bytes of data – Byte 9 = 0x0.

Figure 9-33. Reading out 16 Bytes of Data – Byte 9 = 0x0



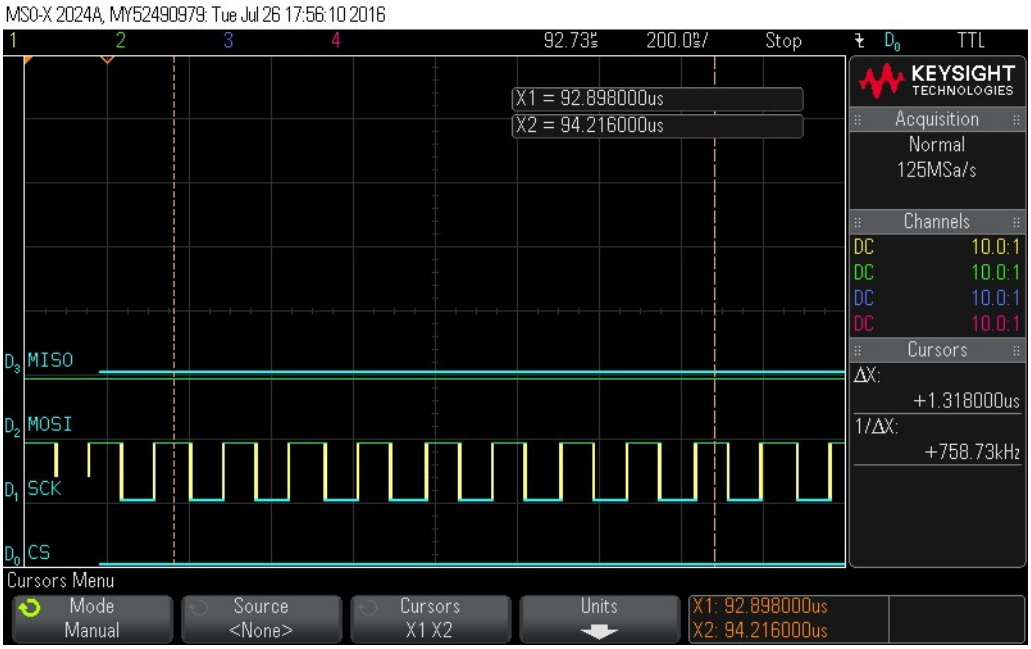
34. Reading out 16 Bytes of data – Byte 10 = 0x0.

Figure 9-34. Reading out 16 Bytes of Data – Byte 10 = 0x0



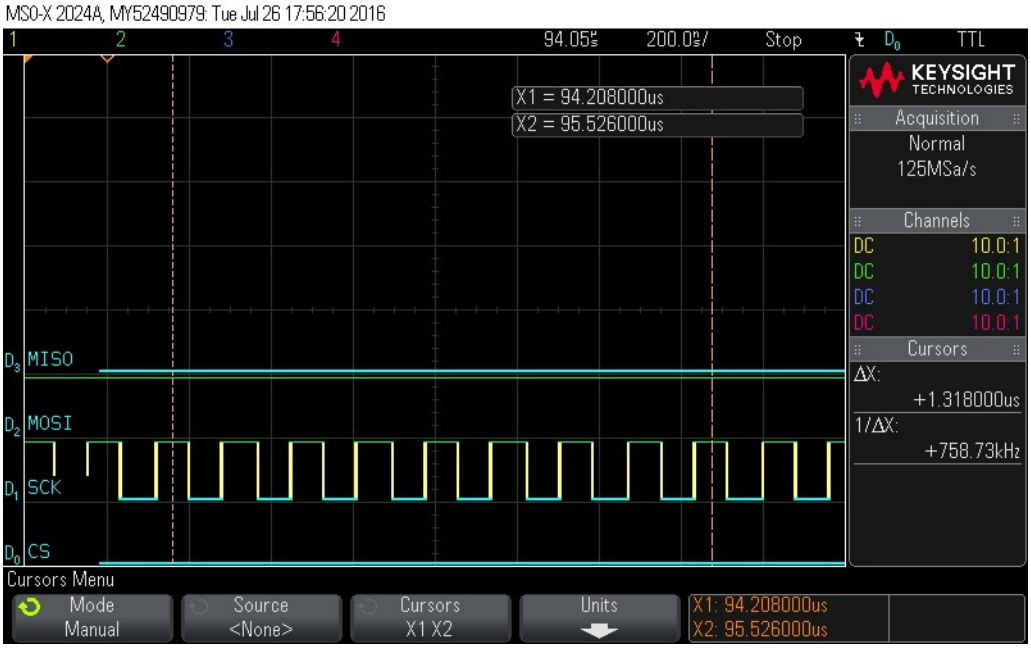
35. Reading out 16 Bytes of data – Byte 11 = 0x0.

Figure 9-35. Reading out 16 Bytes of Data – Byte 11 = 0x0



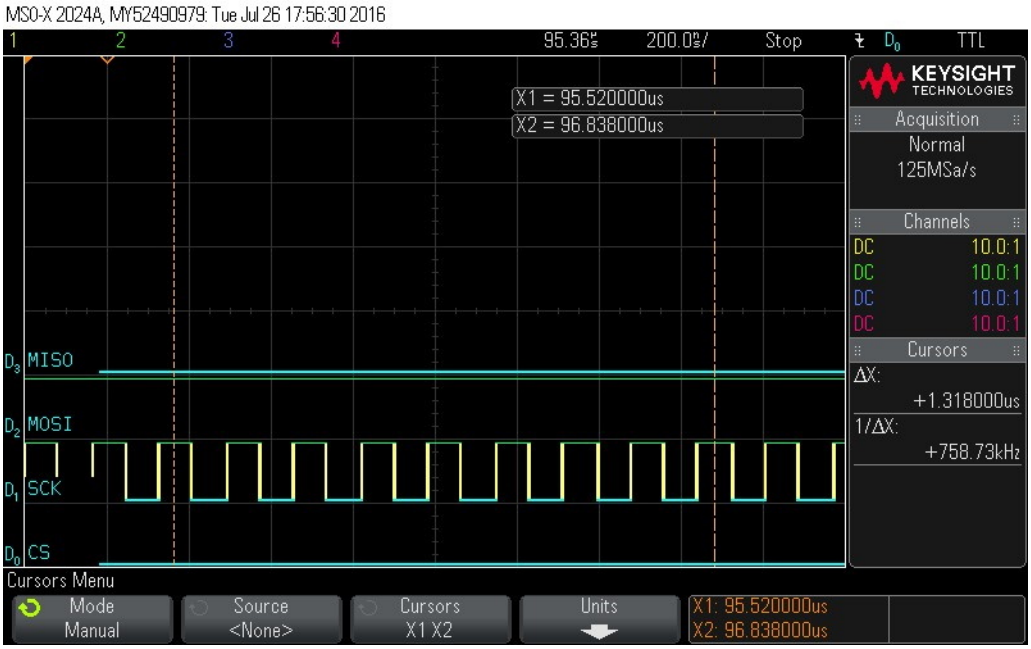
36. Reading out 16 Bytes of data – Byte 12 = 0x0.

Figure 9-36. Reading out 16 Bytes of Data – Byte 12 = 0x0



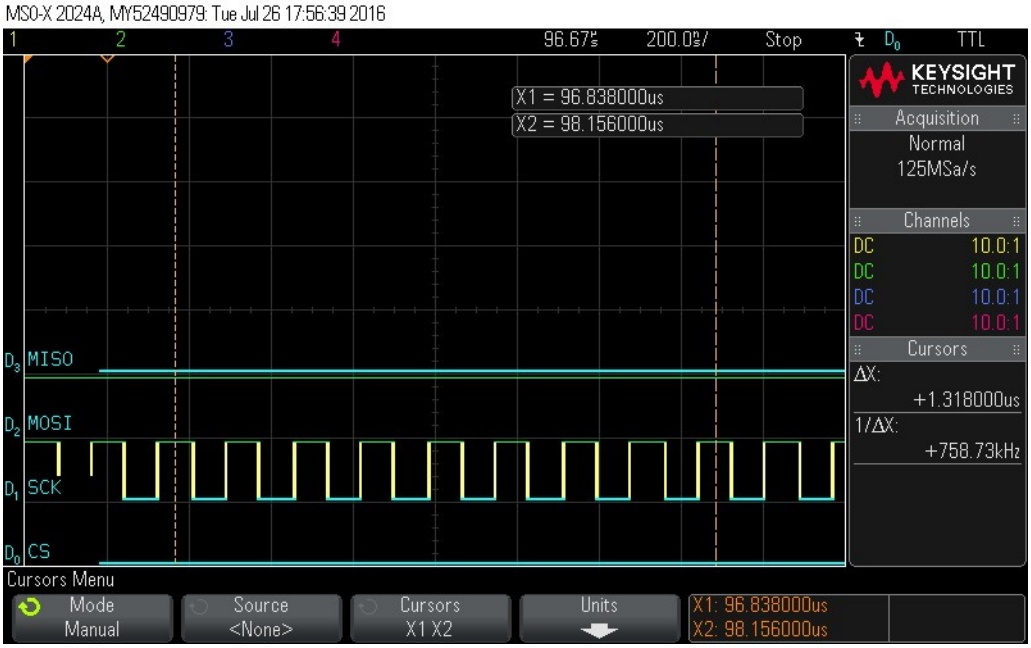
37. Reading out 16 Bytes of data – Byte 13 = 0x0.

Figure 9-37. Reading out 16 Bytes of Data – Byte 13 = 0x0



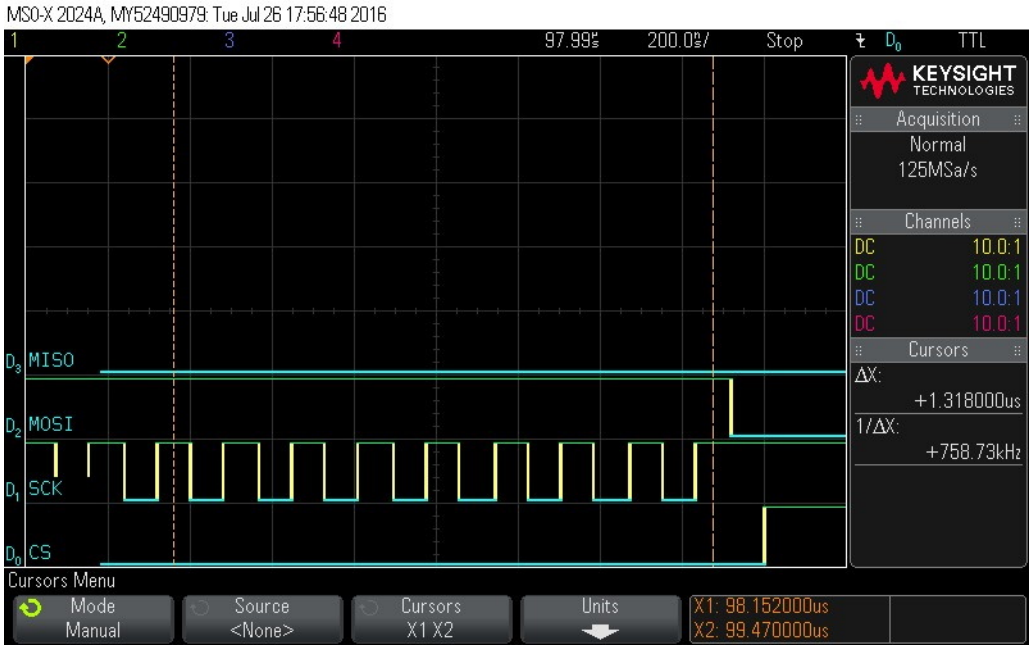
38. Reading out 16 Bytes of data – Byte 14 = 0x0.

Figure 9-38. Reading out 16 Bytes of Data – Byte 14 = 0x0



39. Reading out 16 Bytes of data – Byte 15 = 0x0.

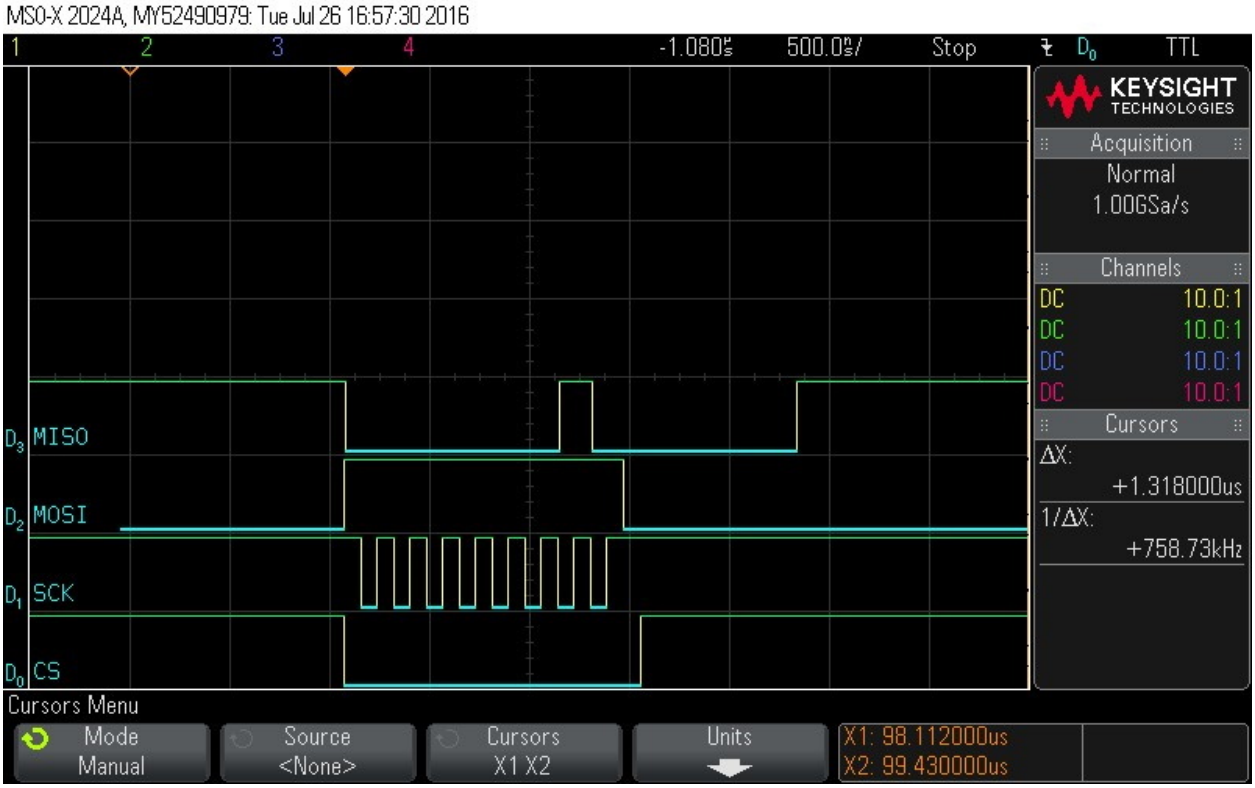
Figure 9-39. Reading out 16 Bytes of Data – Byte 15 = 0x0



9.2 Read FSN waveform

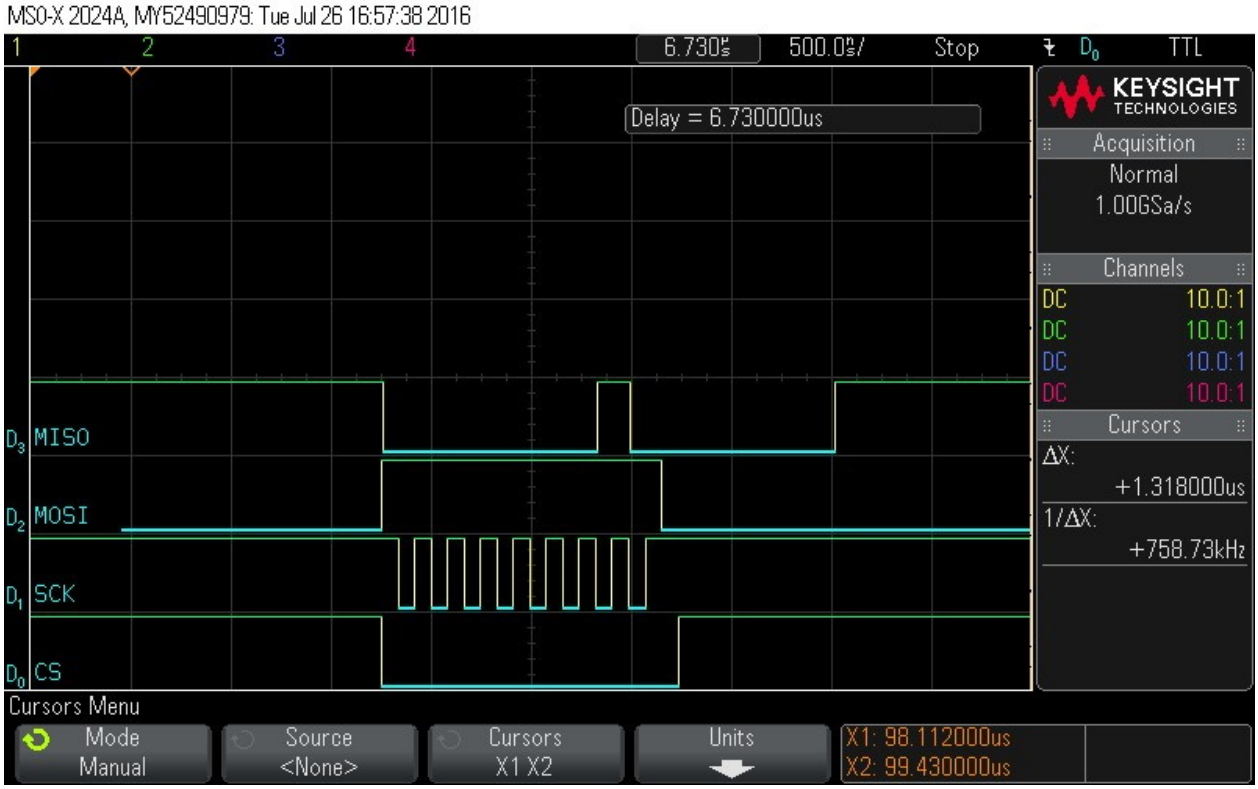
1. Checking hardware status.

Figure 9-40. Hardware Status Check



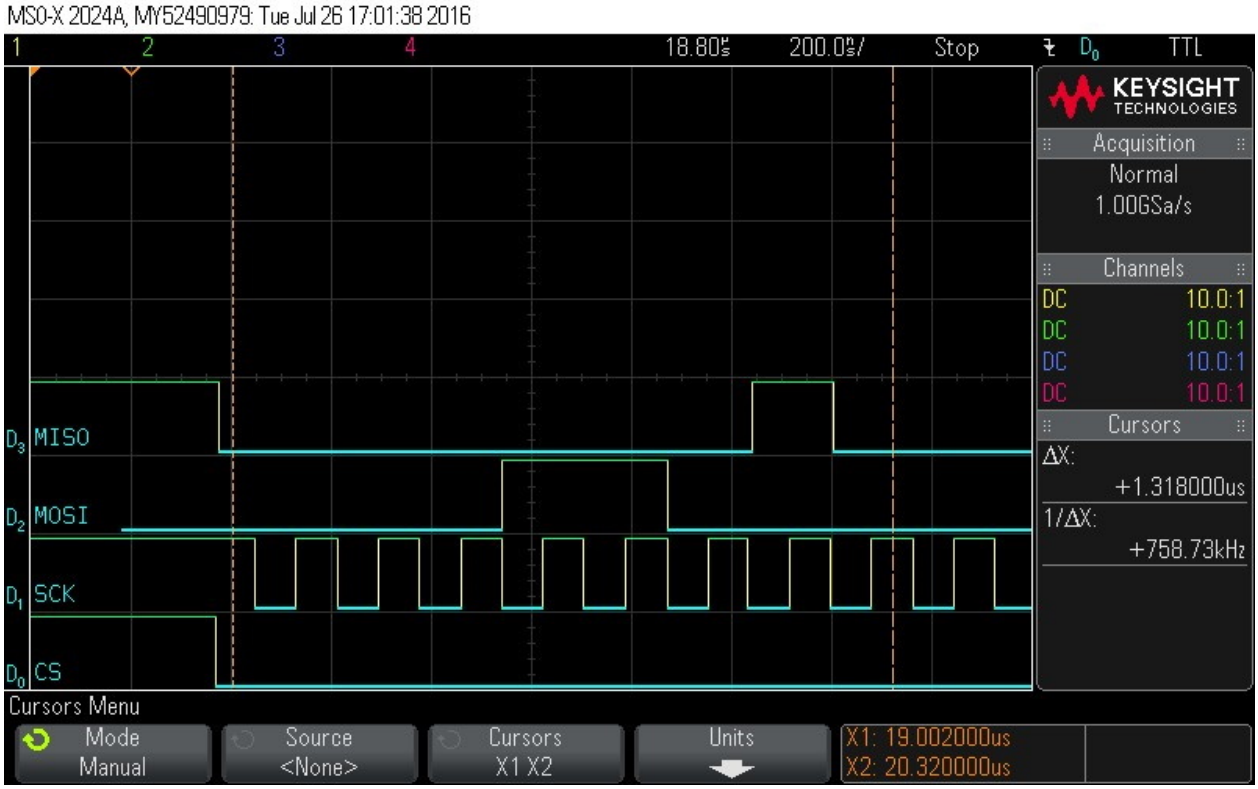
2. Checking hardware status.

Figure 9-41. Hardware Status Check



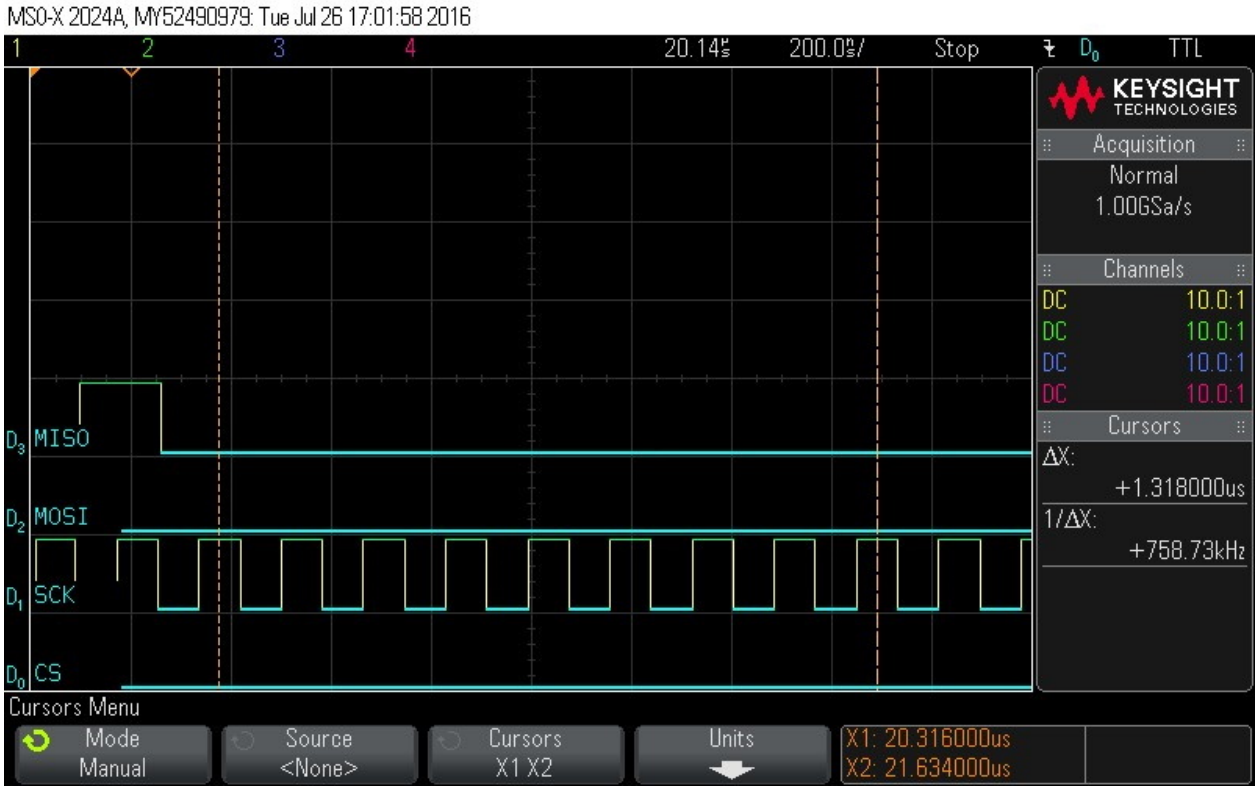
3. Clocking in `read_FSN` command (0x18).

Figure 9-42. Clock in read_FSN command (0x18)



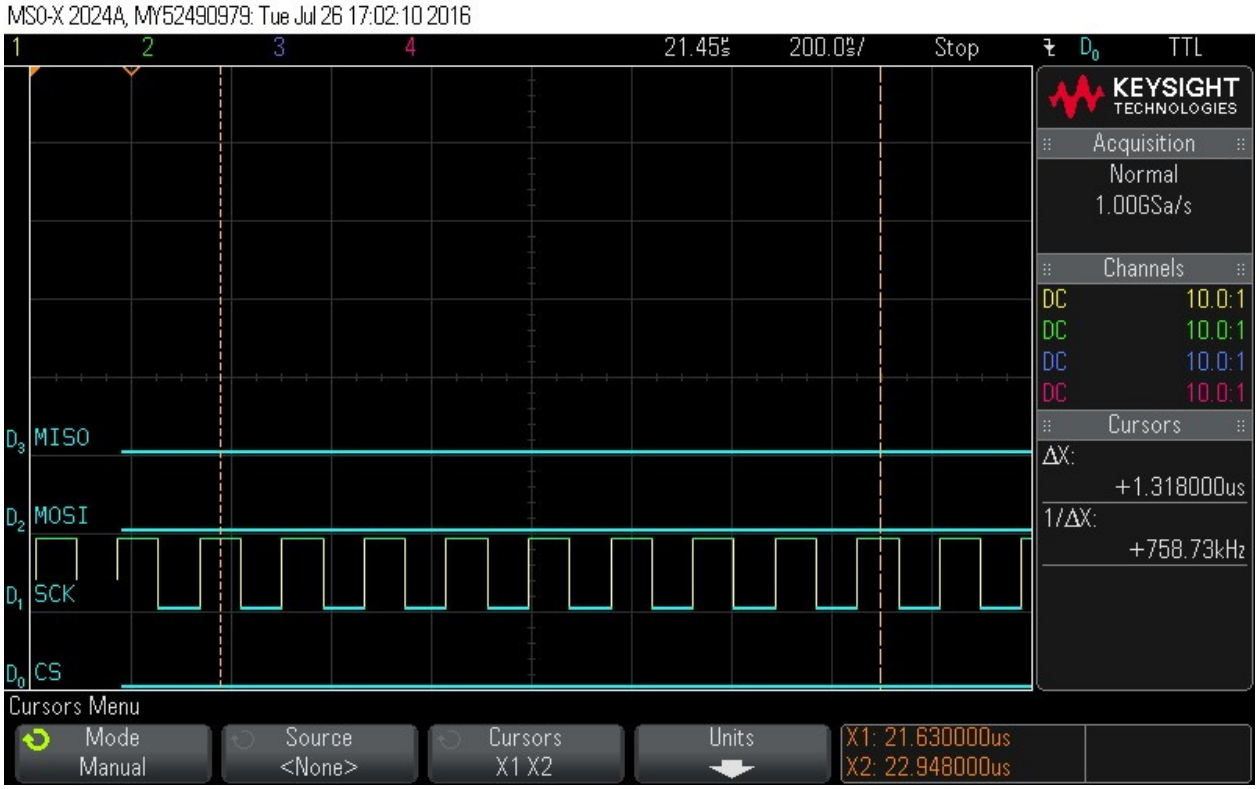
4. Clocking in 16 bytes of zero values - Byte 0.

Figure 9-43. Clock in 16 Bytes of Zero Values - Byte 0



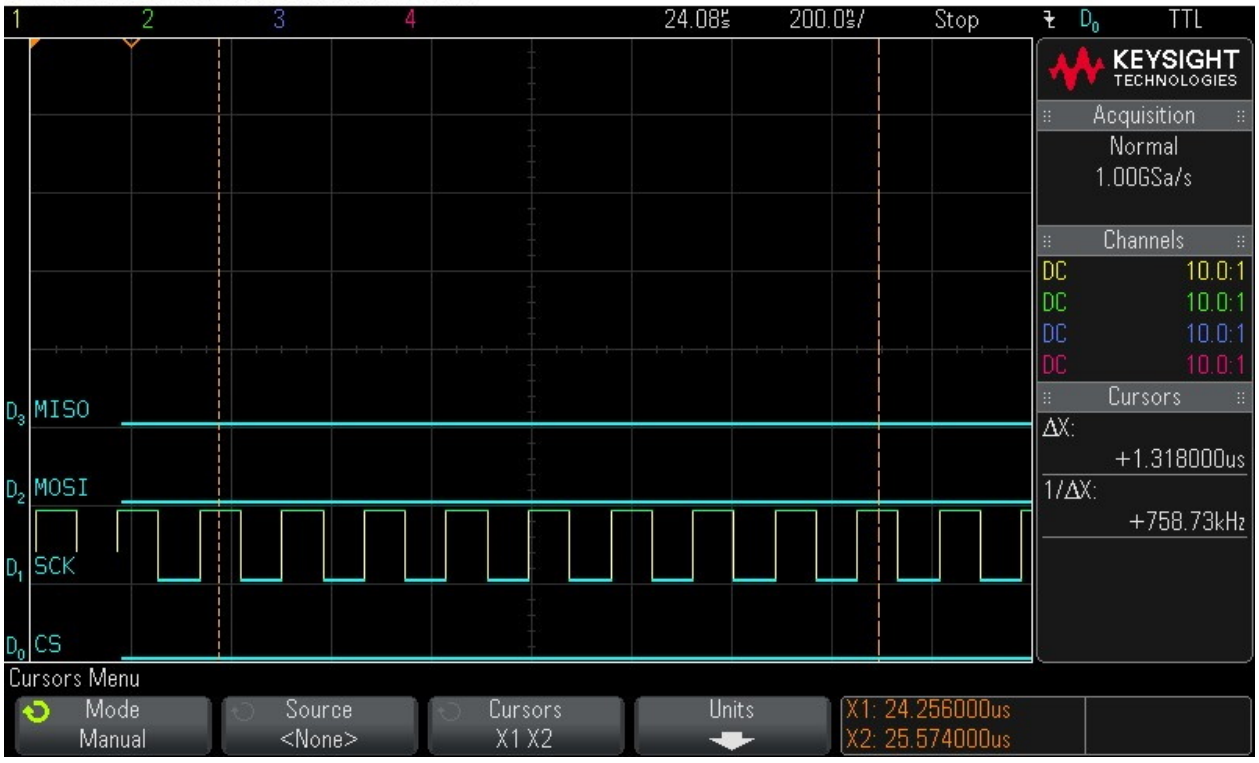
5. Clocking in 16 bytes of zero values - Byte 1.

Figure 9-44. Clock in 16 Bytes of Zero Values - Byte 1



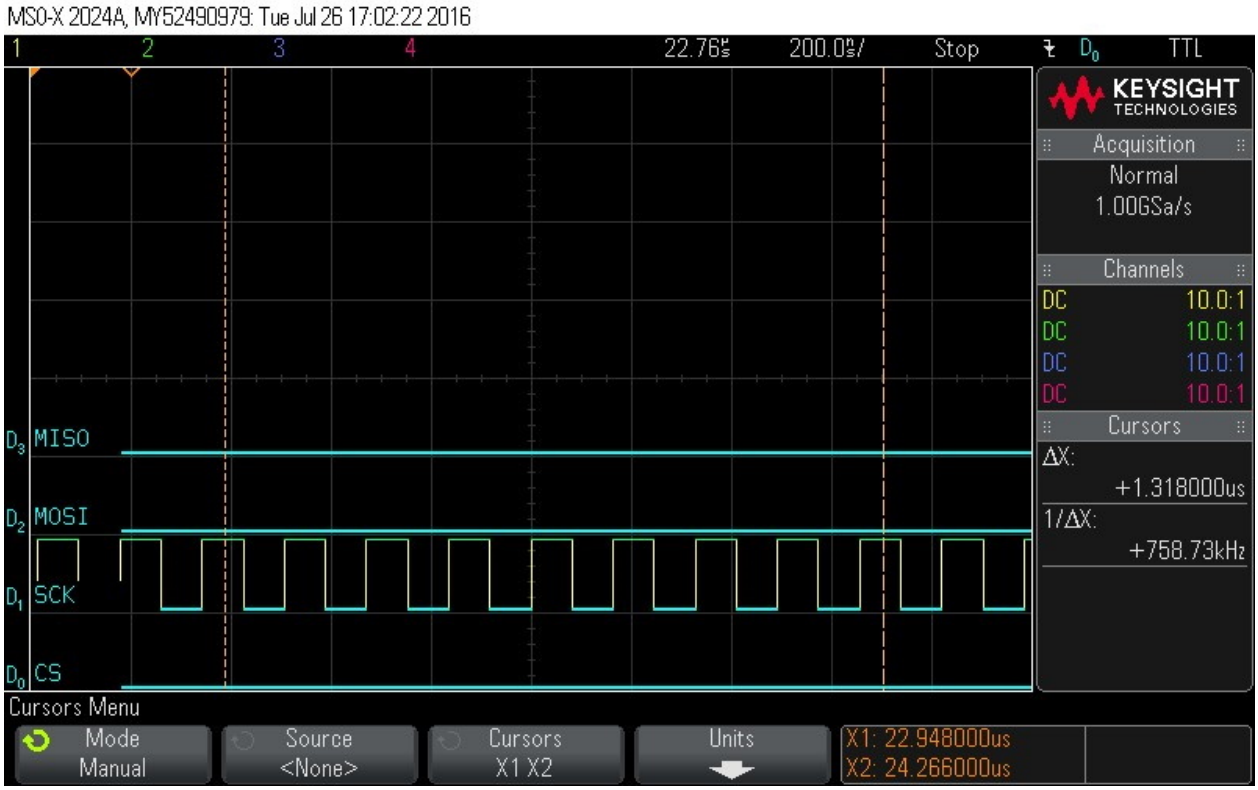
6. Clocking in 16 bytes of zero values - Byte 2.

Figure 9-45. Clock in 16 Bytes of Zero Values - Byte 2



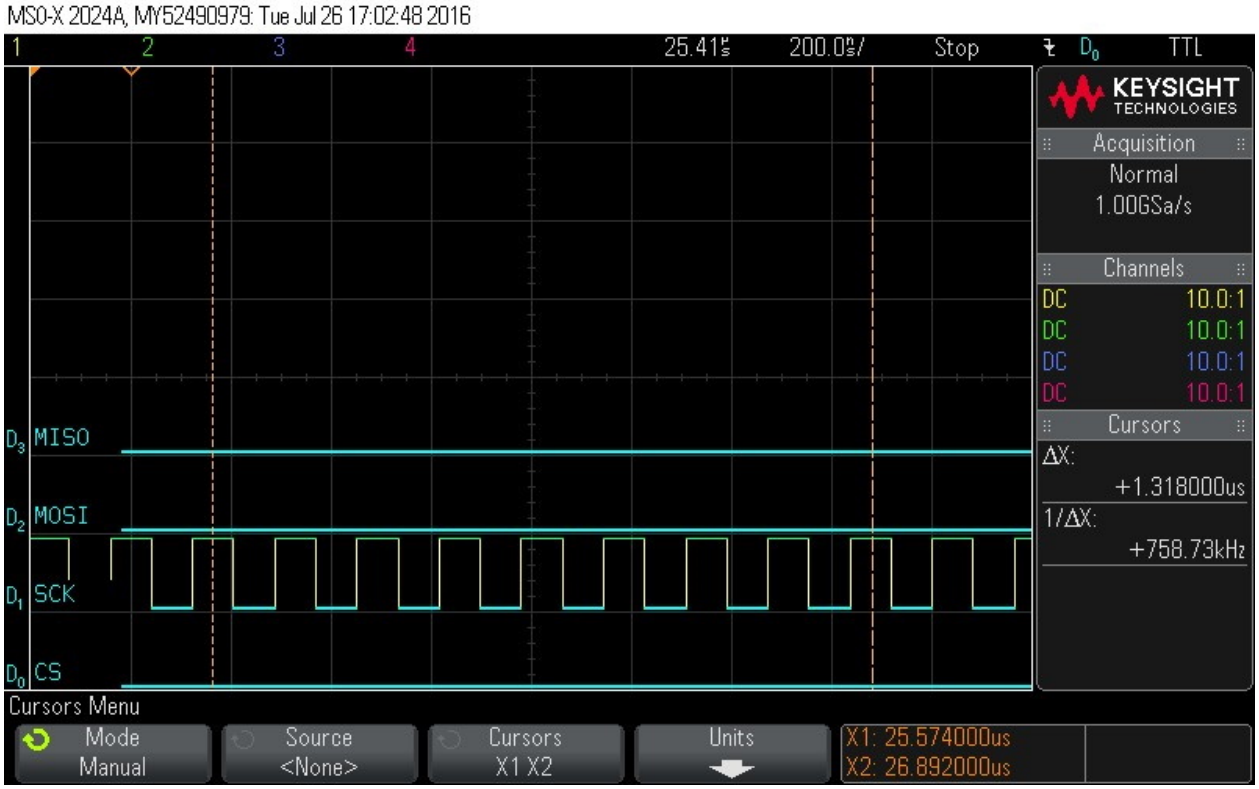
7. Clocking in 16 bytes of zero values - Byte 3.

Figure 9-46. Clock in 16 Bytes of Zero Values - Byte 3



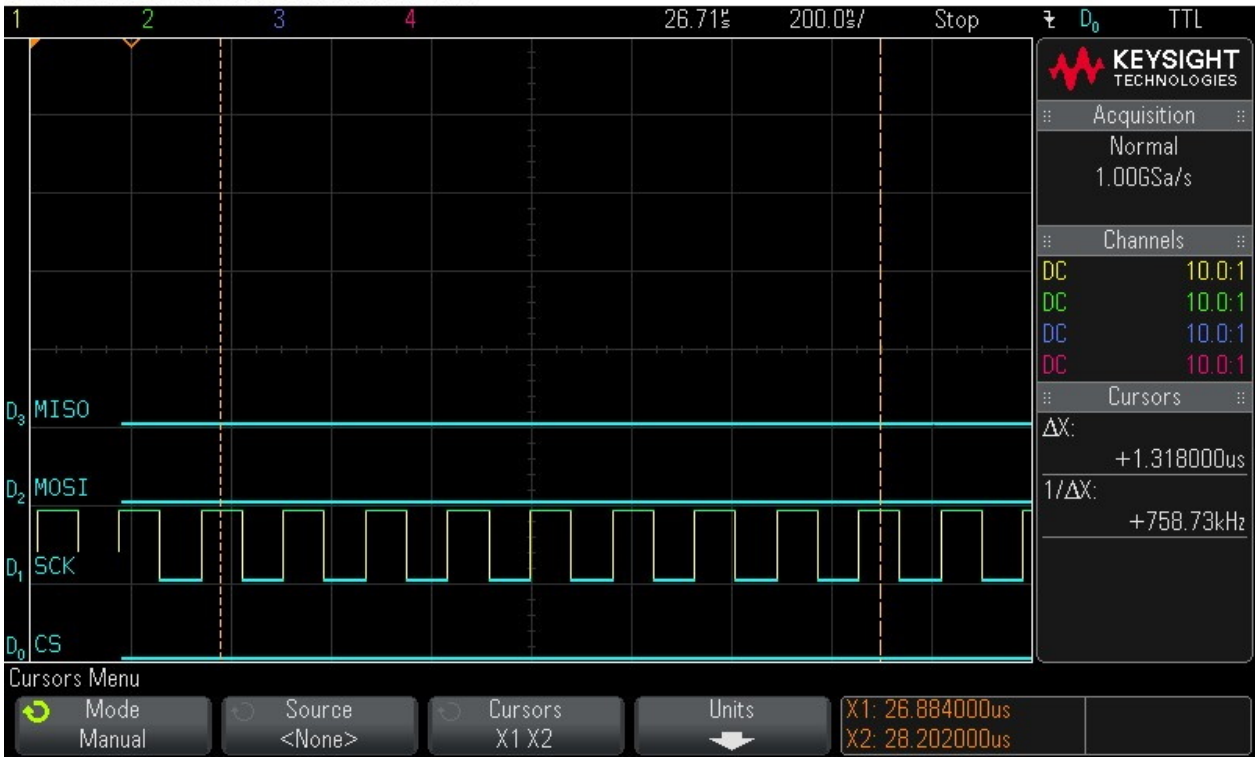
8. Clocking in 16 bytes of zero values - Byte 4.

Figure 9-47. Clock in 16 Bytes of Zero Values - Byte 4



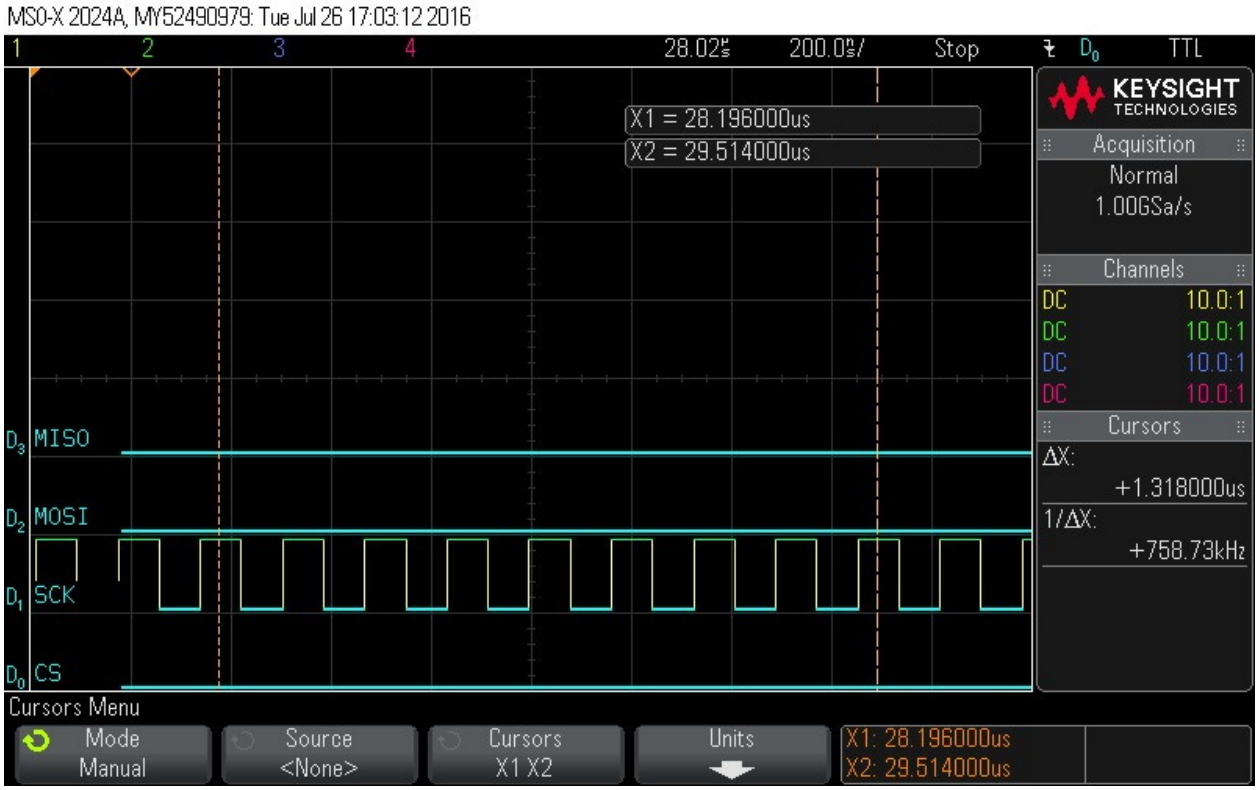
9. Clocking in 16 bytes of zero values - Byte 5.

Figure 9-48. Clock in 16 Bytes of Zero Values - Byte 5



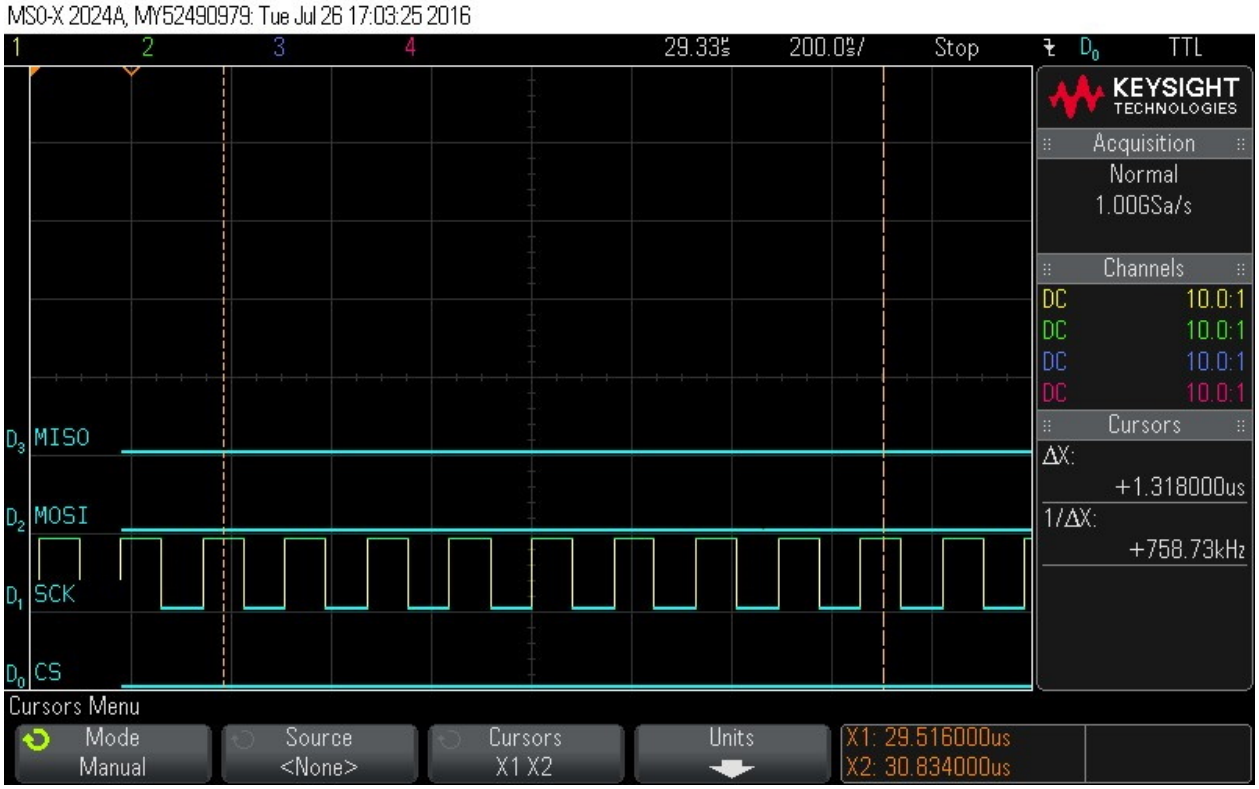
10. Clocking in 16 bytes of zero values - Byte 6.

Figure 9-49. Clock in 16 Bytes of Zero Values - Byte 6



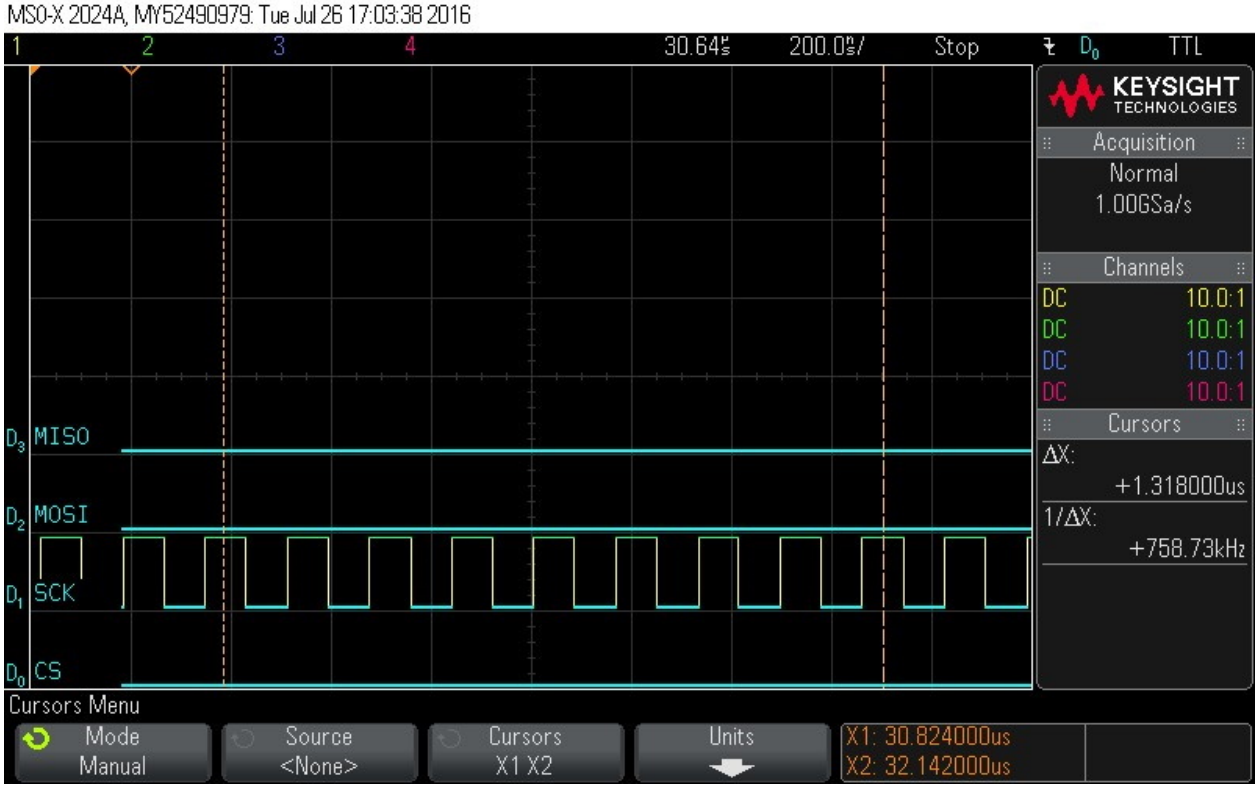
11. Clocking in 16 bytes of zero values - Byte 7.

Figure 9-50. Clock in 16 bytes of zero values - Byte 7



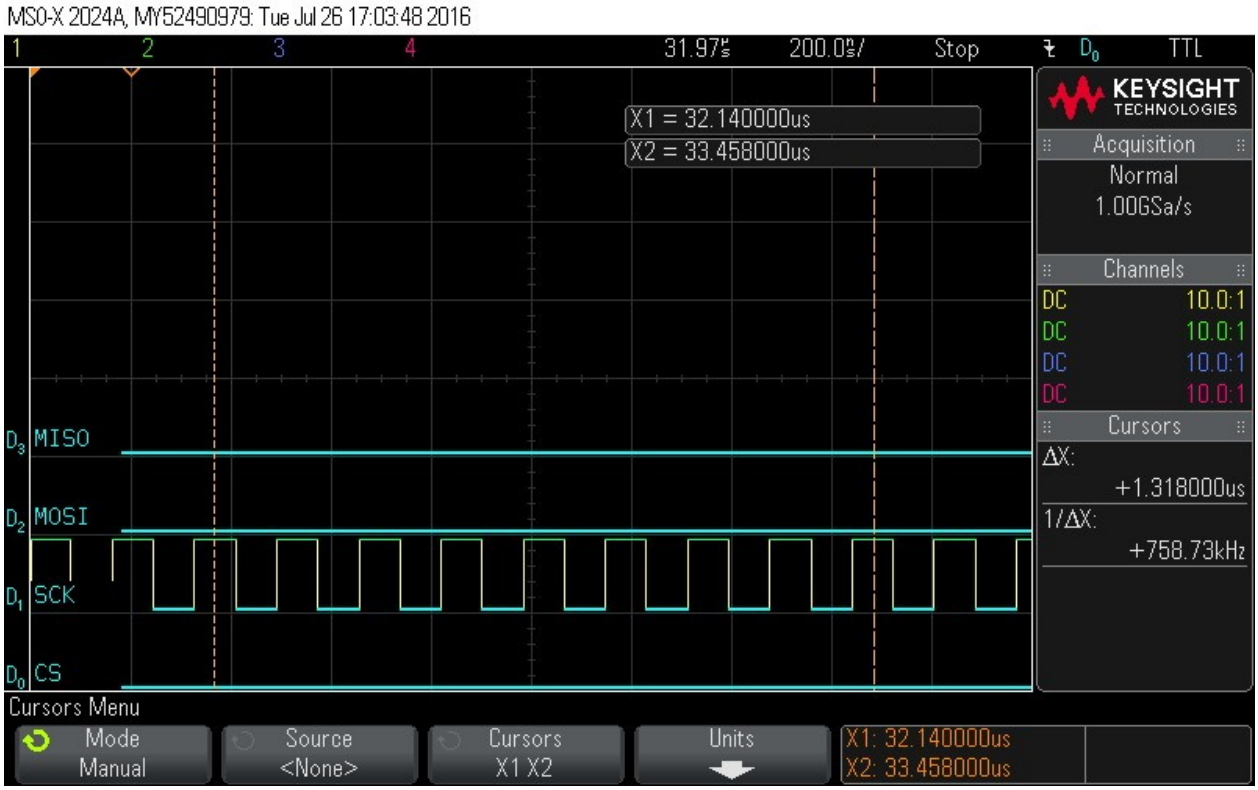
12. Clocking in 16 bytes of zero values - Byte 8.

Figure 9-51. Clock in 16 Bytes of Zero Values - Byte 8



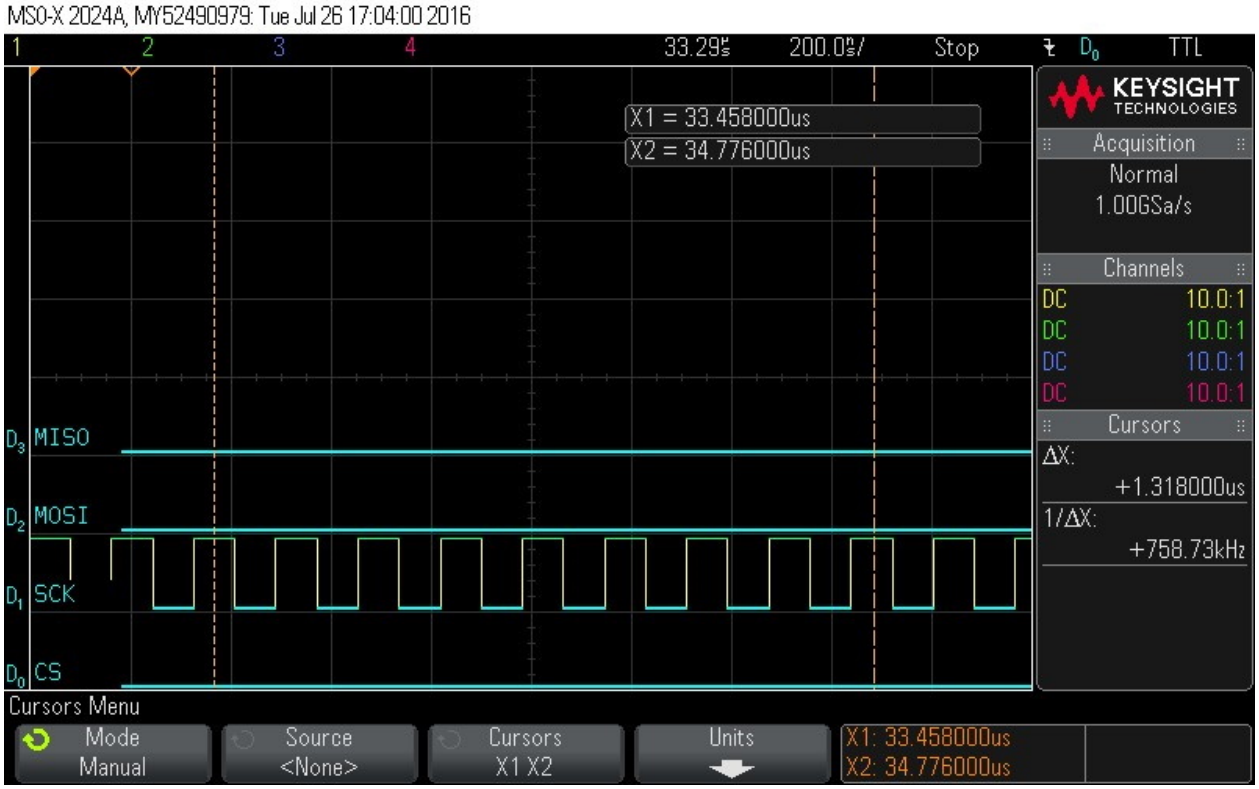
13. Clocking in 16 bytes of zero values - Byte 9.

Figure 9-52. Clock in 16 Bytes of Zero Values - Byte 9



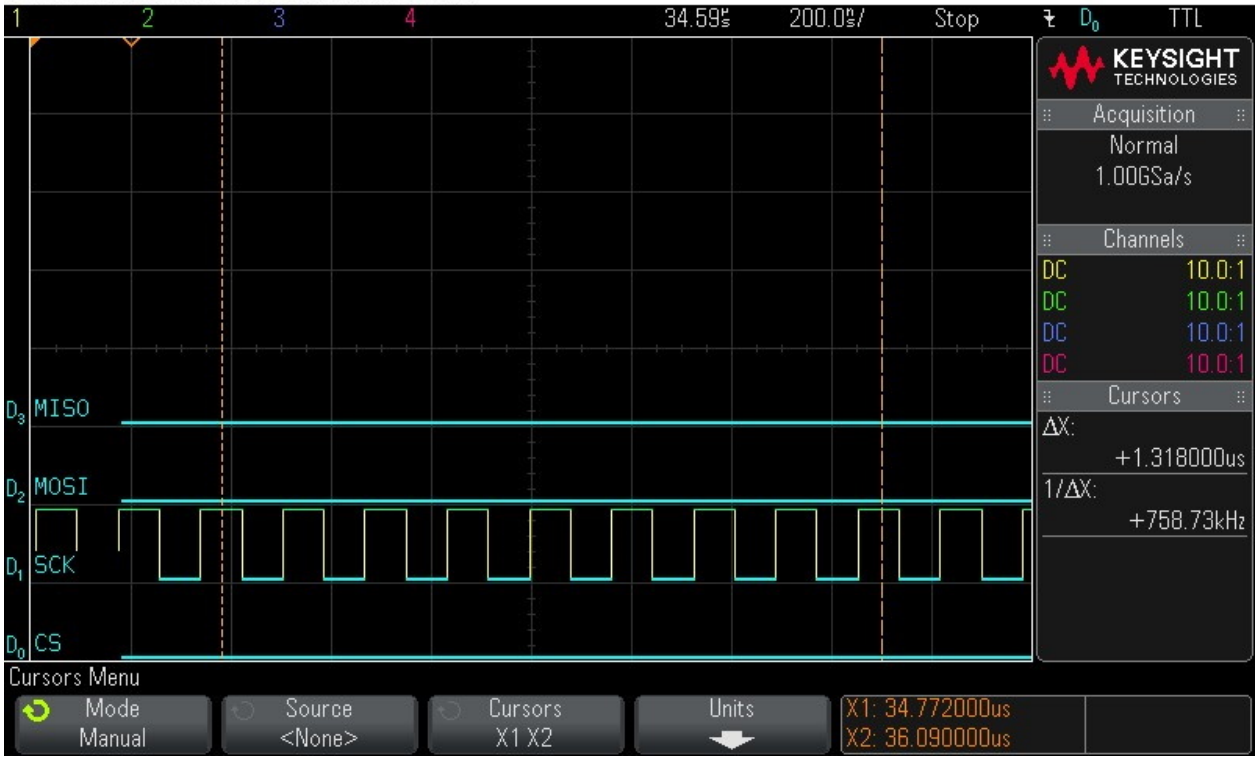
14. Clocking in 16 bytes of zero values - Byte 10.

Figure 9-53. Clock in 16 Bytes of Zero Values - Byte 10



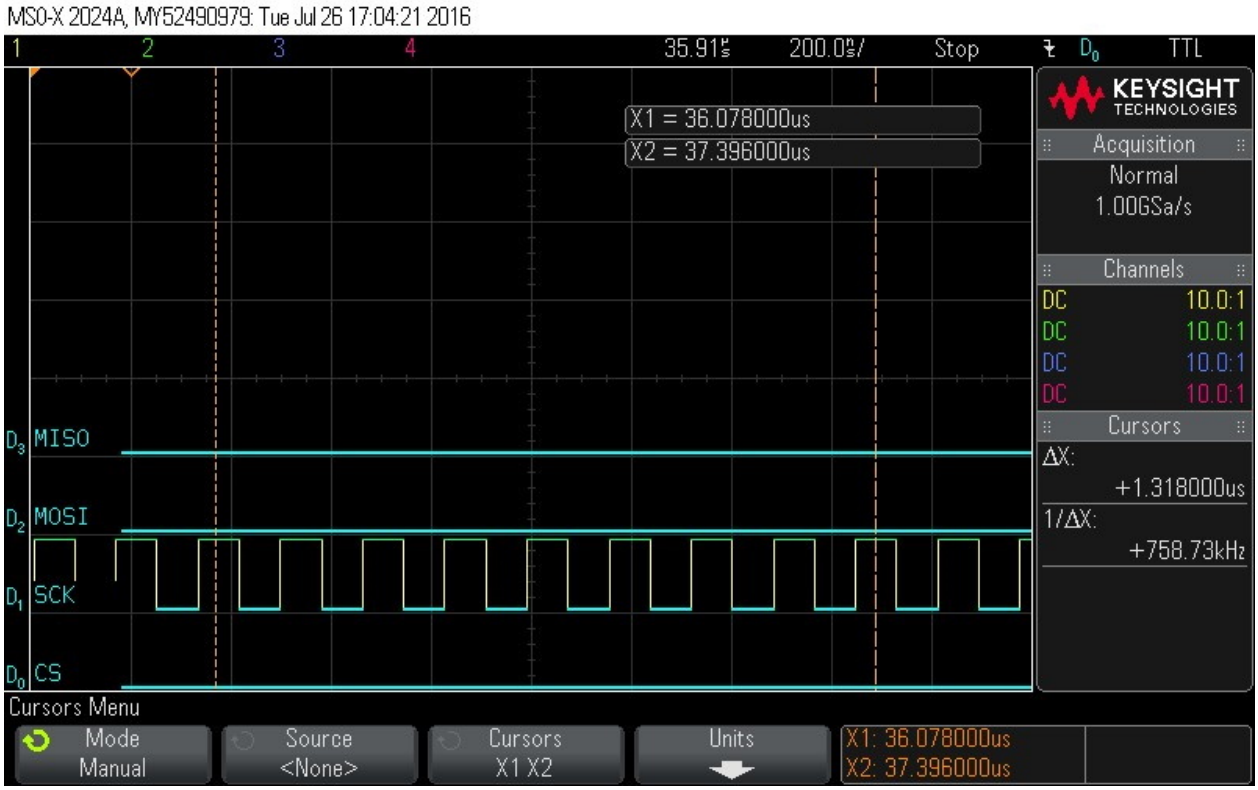
15. Clocking in 16 bytes of zero values - Byte 11.

Figure 9-54. Clock in 16 Bytes of Zero Values - Byte 11



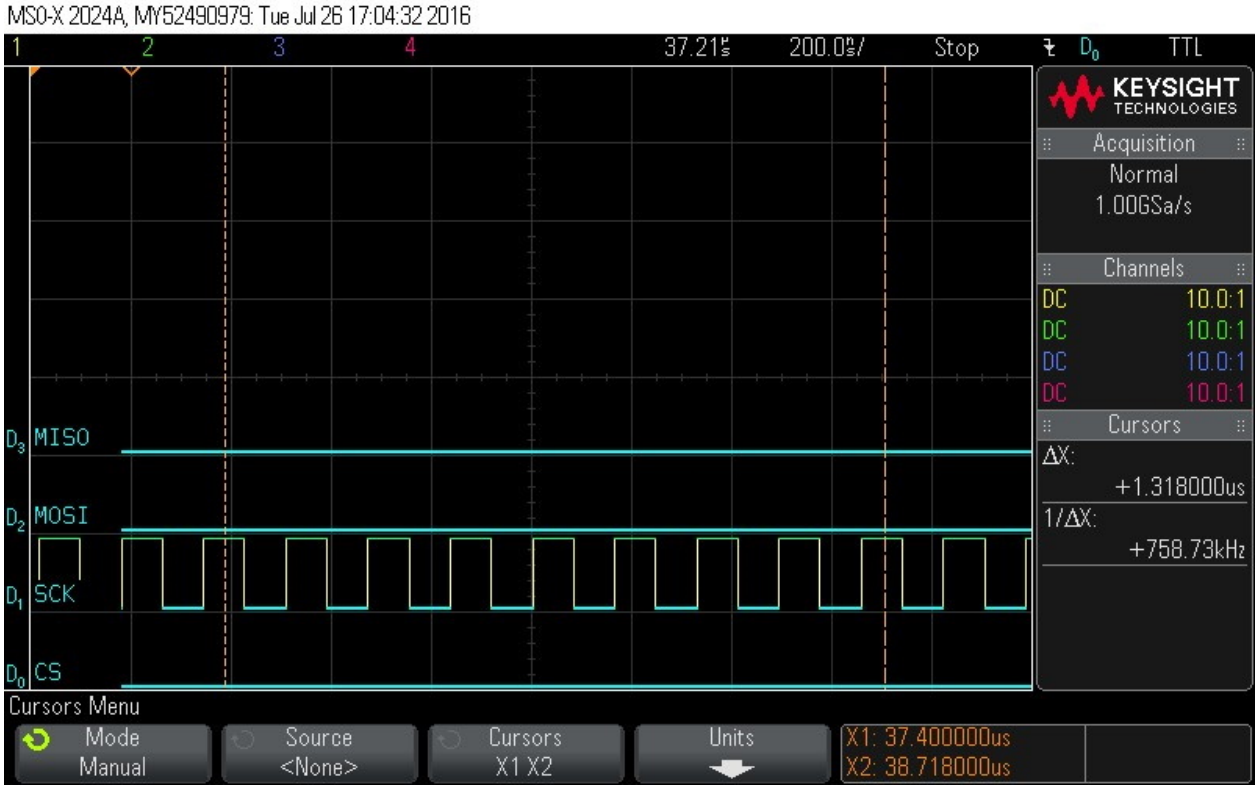
16. Clocking in 16 bytes of zero values - Byte 12.

Figure 9-55. Clock in 16 Bytes of Zero Values - Byte 12



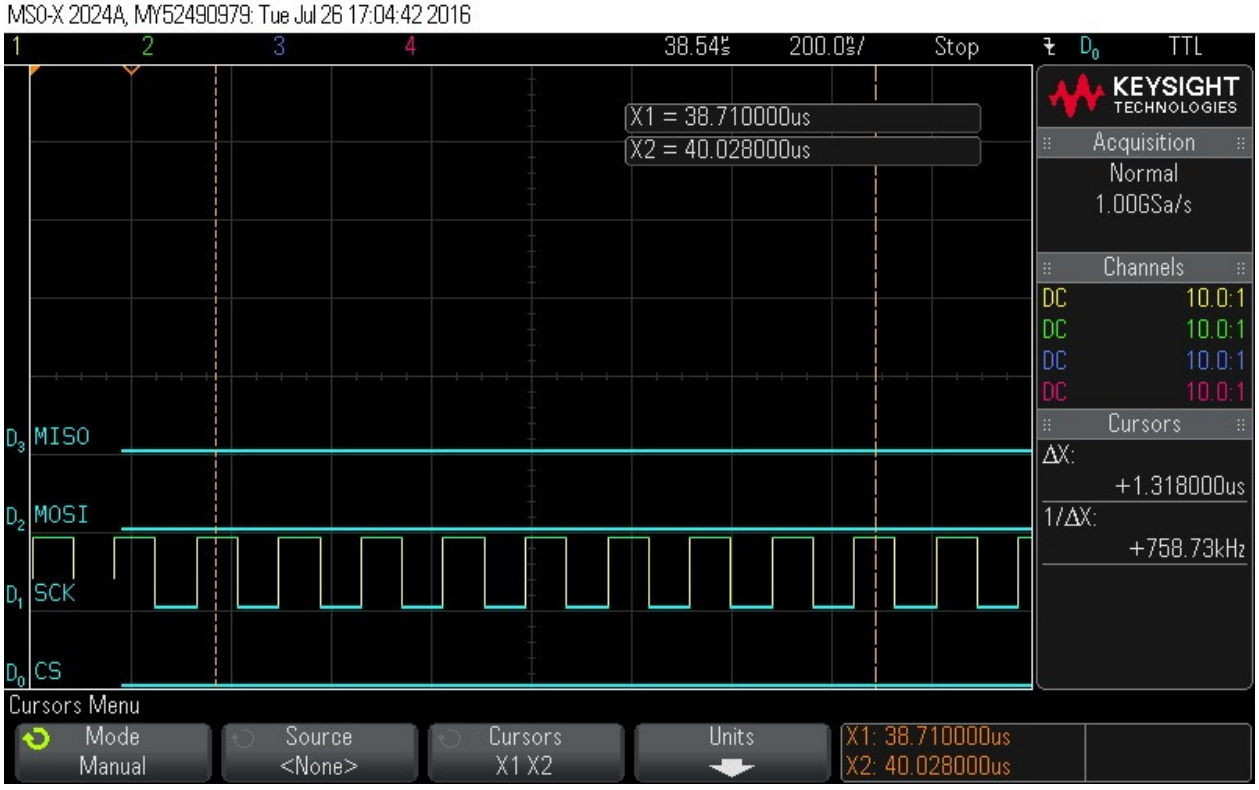
17. Clocking in 16 bytes of zero values - Byte 13.

Figure 9-56. Clock in 16 Bytes of Zero Values - Byte 13



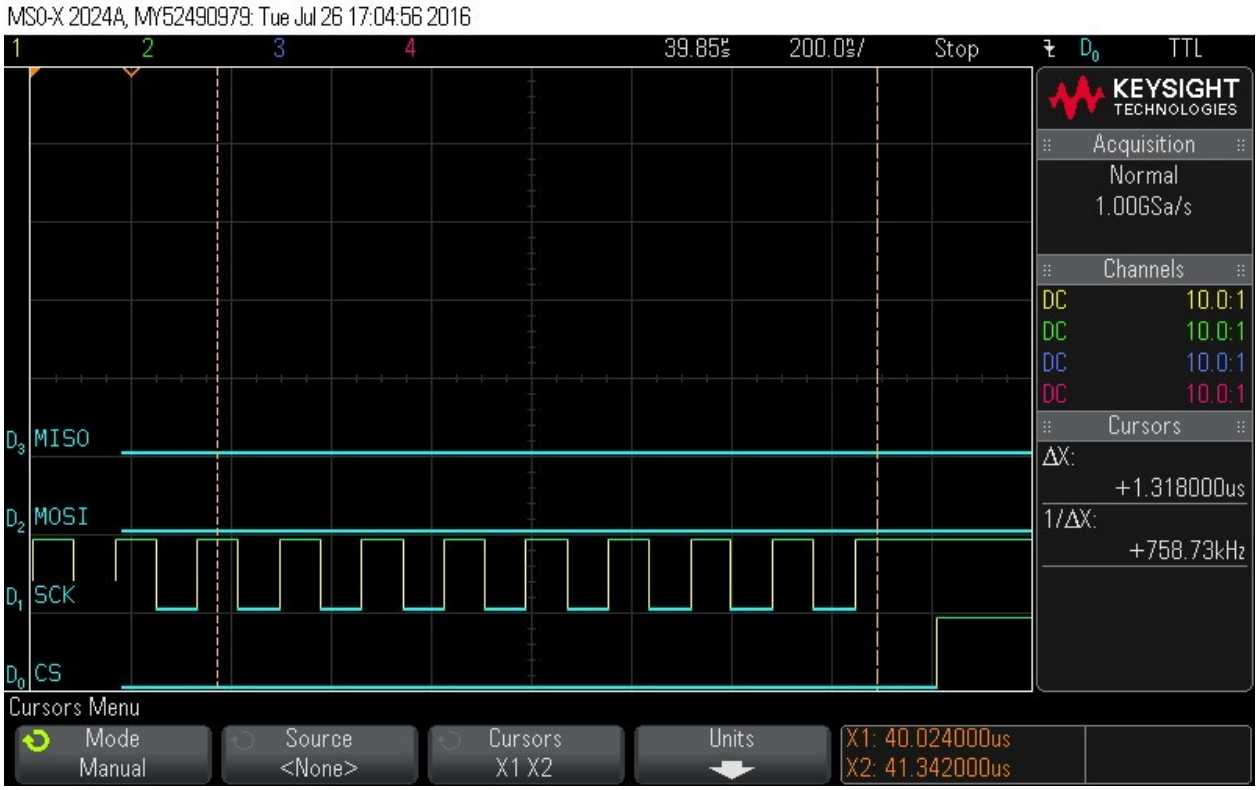
18. Clocking in 16 bytes of zero values - Byte 14.

Figure 9-57. Clock in 16 Bytes of Zero Values - Byte 14



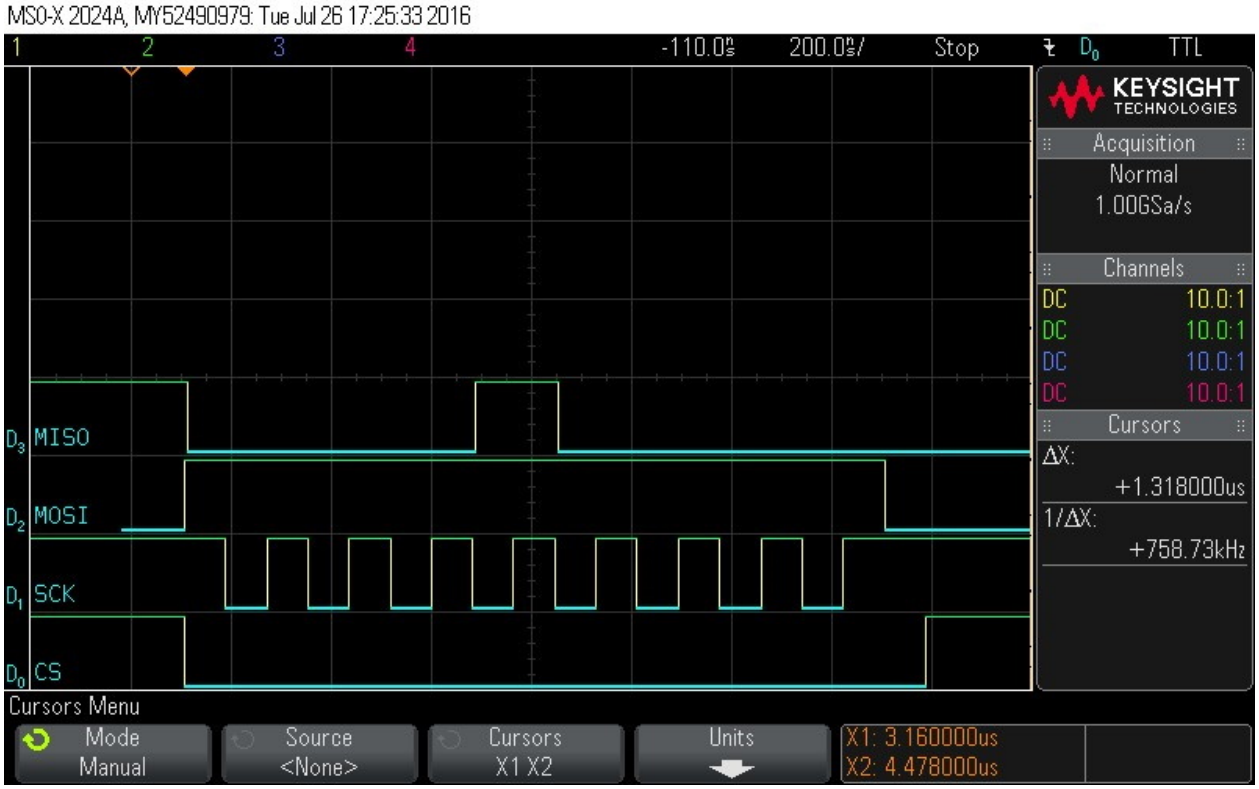
19. Clocking in 16 bytes of zero values - Byte 15.

Figure 9-58. Clock in 16 Bytes of Zero Values - Byte 15



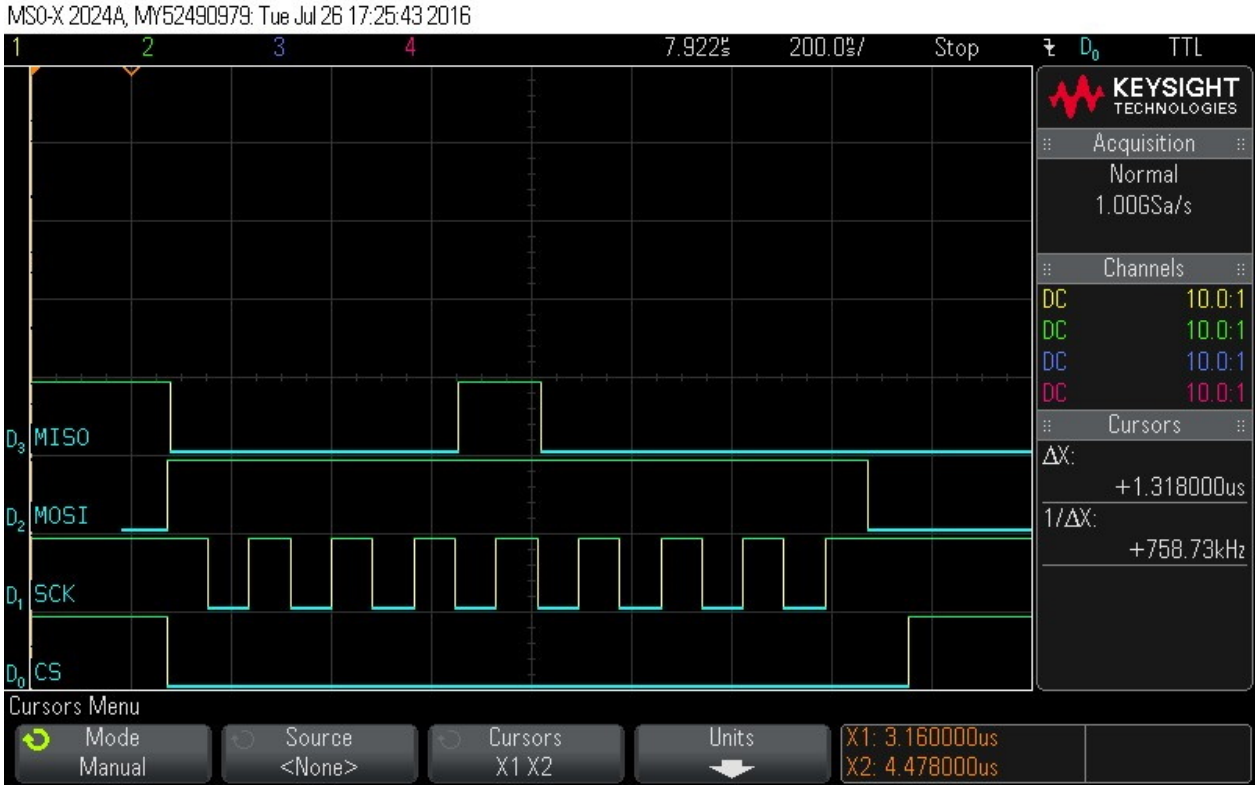
20. Checking hardware status.

Figure 9-59. Hardware Status Check



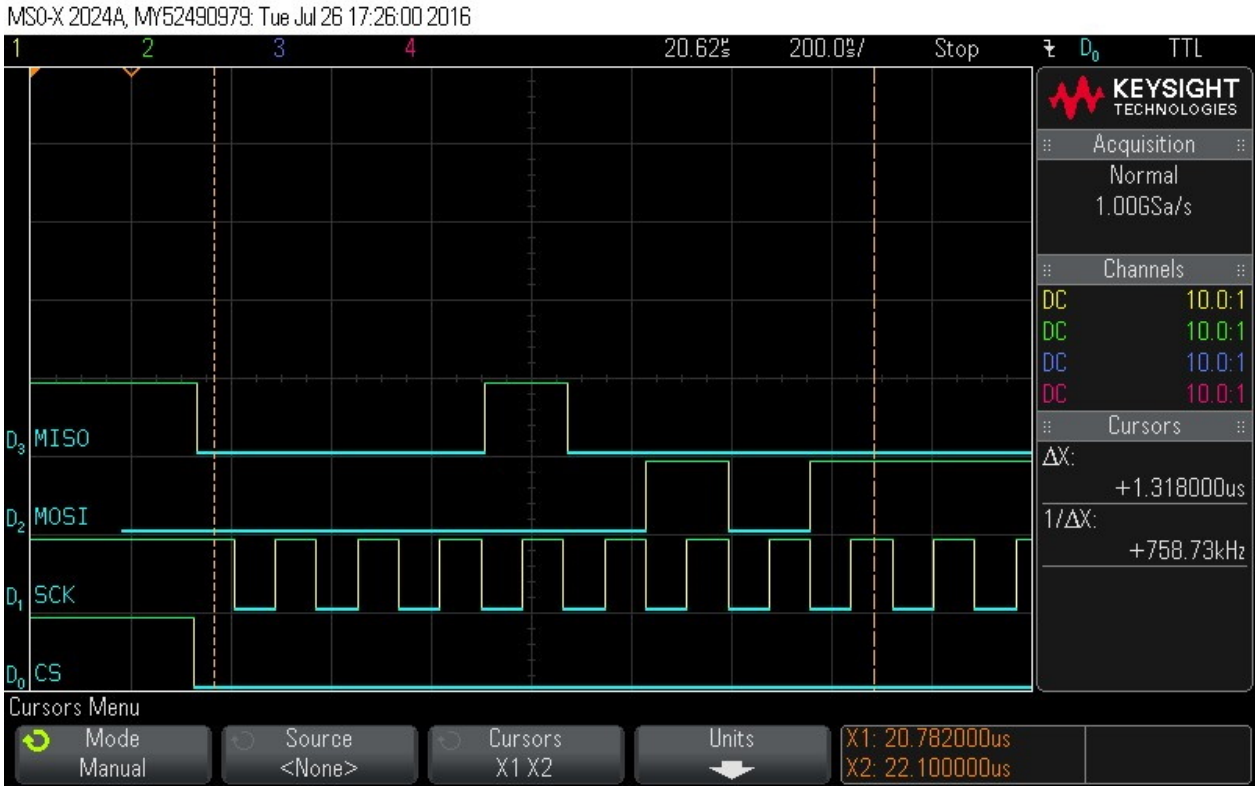
21. Checking hardware status.

Figure 9-60. Hardware Status Check



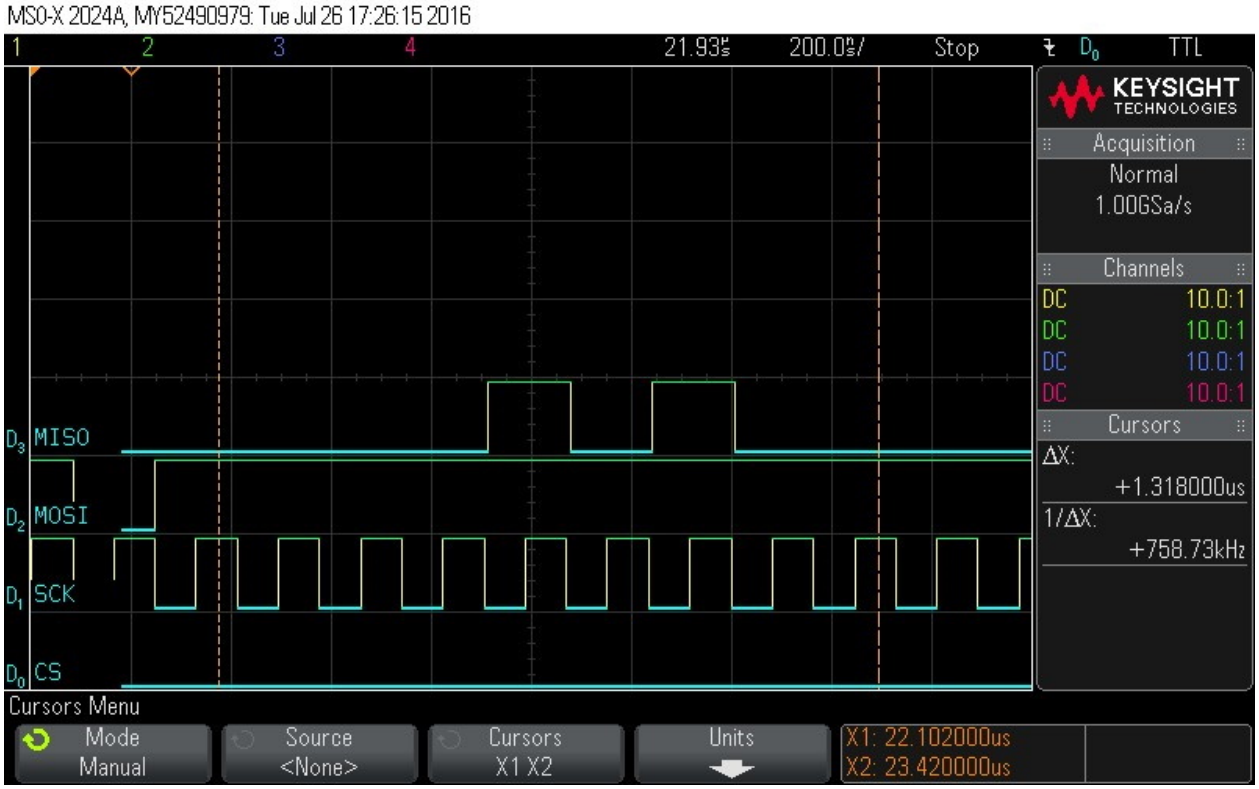
22. Clock in read command (0x5).

Figure 9-61. Clock in Read Command (0x5)



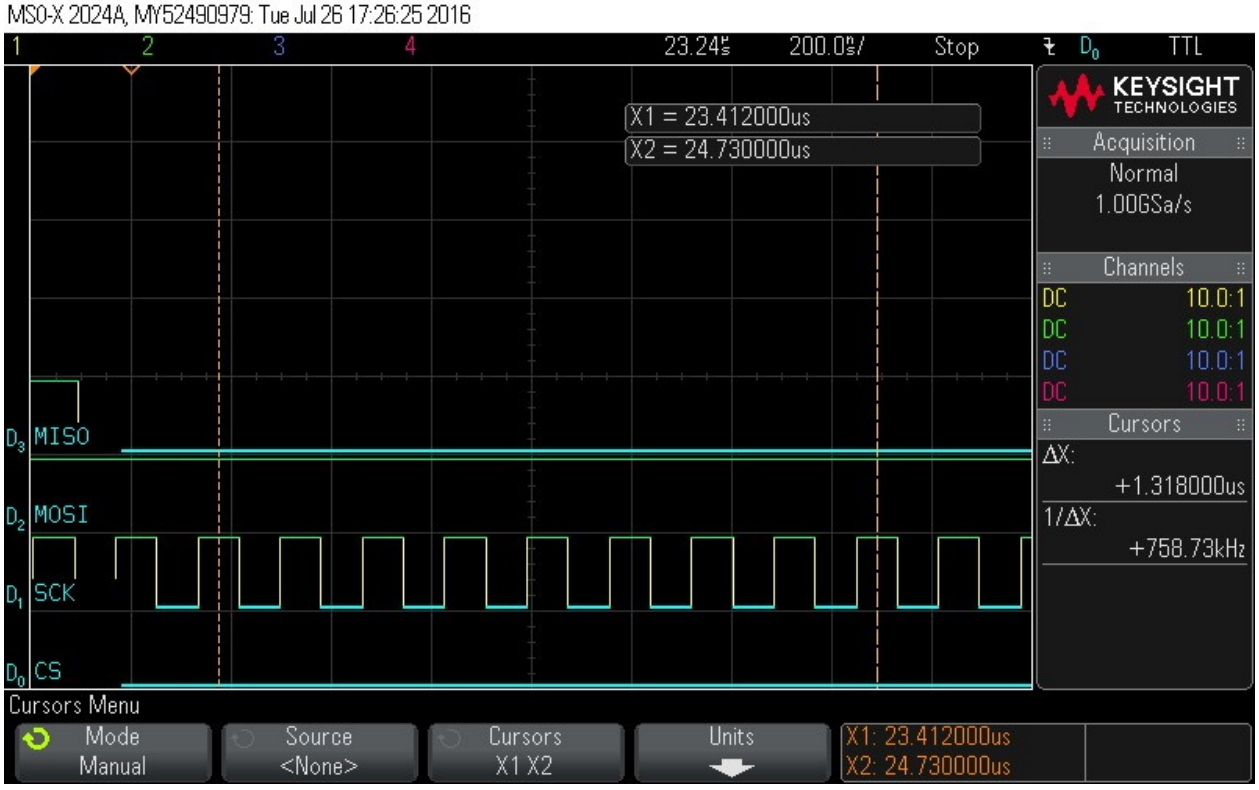
23. Reading out 16 Bytes of FSN data – Byte 0 = 0x14.

Figure 9-62. Reading out 16 Bytes of FSN Data – Byte 0 = 0x14



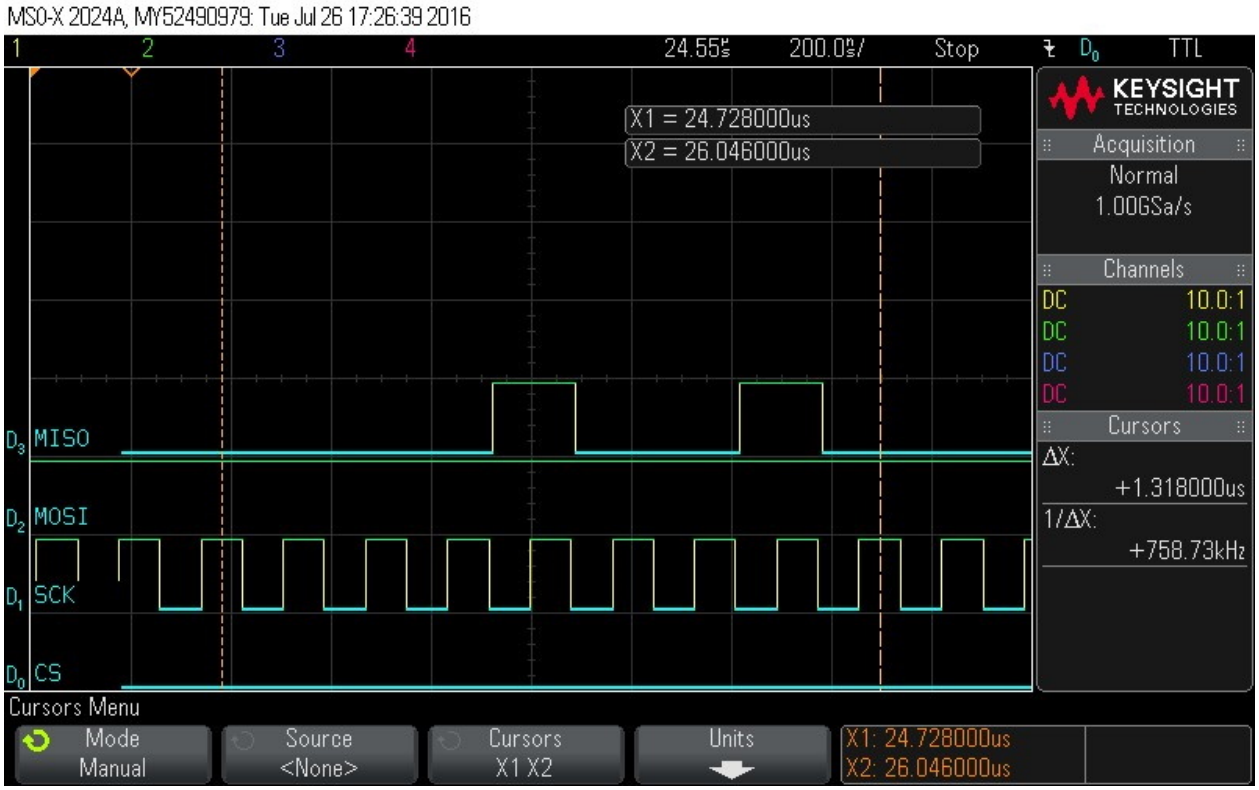
24. Reading out 16 Bytes of FSN data – Byte 1 = 0x0.

Figure 9-63. Reading out 16 Bytes of FSN Data – Byte 1 = 0x0



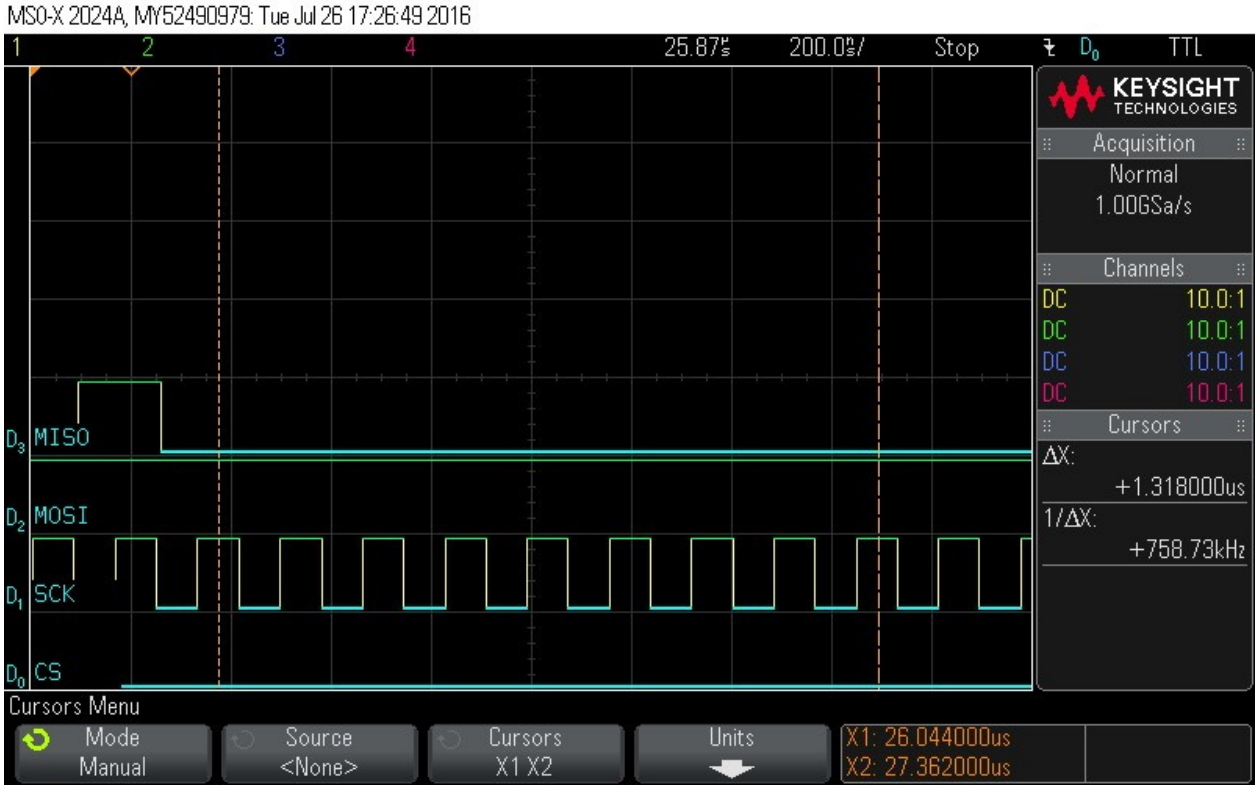
25. Reading out 16 Bytes of FSN data – Byte 2 = 0x12.

Figure 9-64. Reading out 16 Bytes of FSN Data – Byte 2 = 0x12



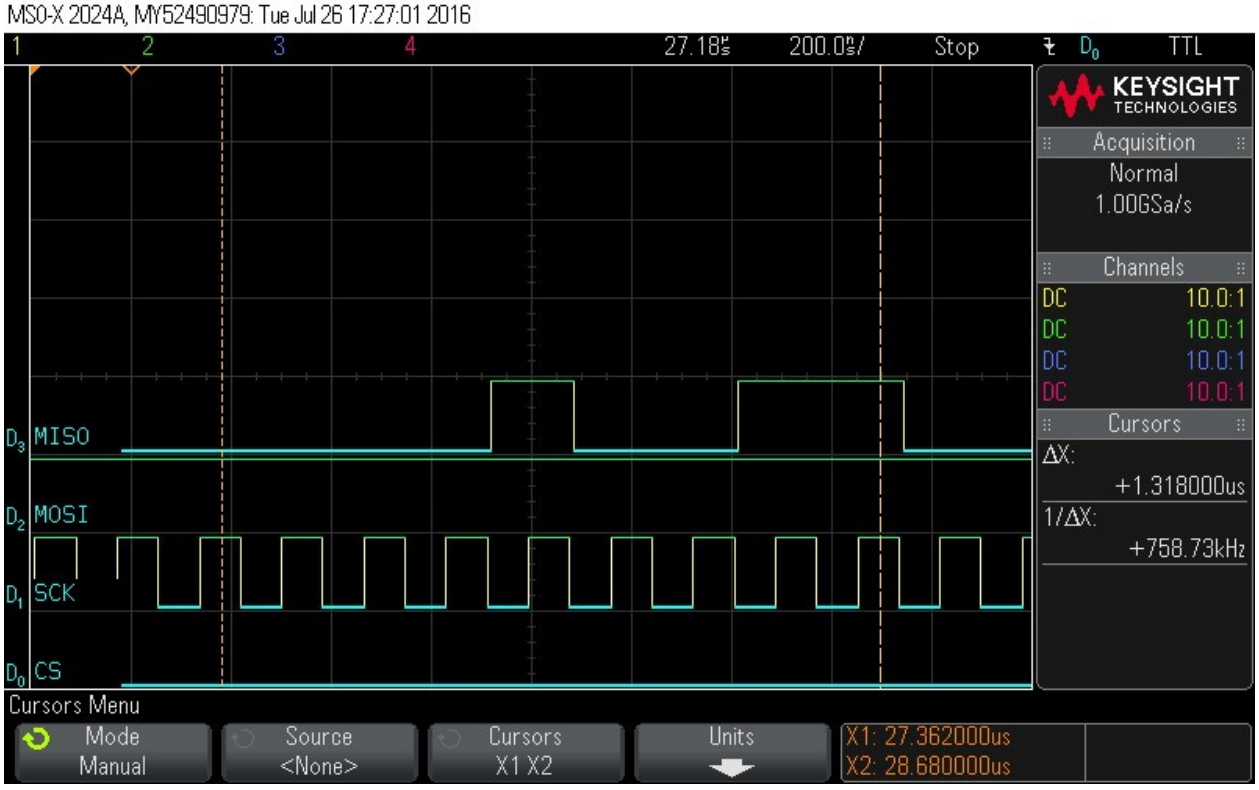
26. Reading out 16 Bytes of FSN data – Byte 3 = 0x.

Figure 9-65. Reading out 16 Bytes of FSN Data – Byte 3 = 0x



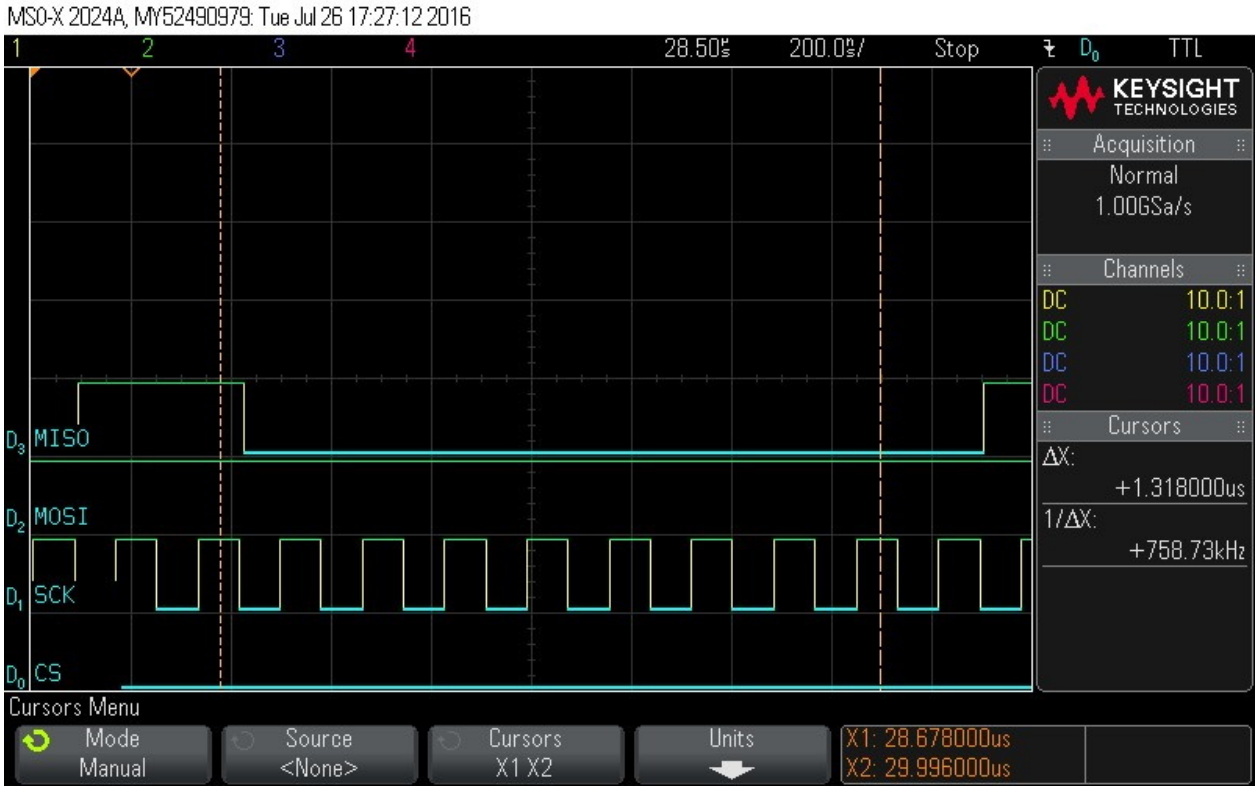
27. Reading out 16 Bytes of FSN data – Byte 4 = 0x13.

Figure 9-66. Reading out 16 Bytes of FSN Data – Byte 4 = 0x13



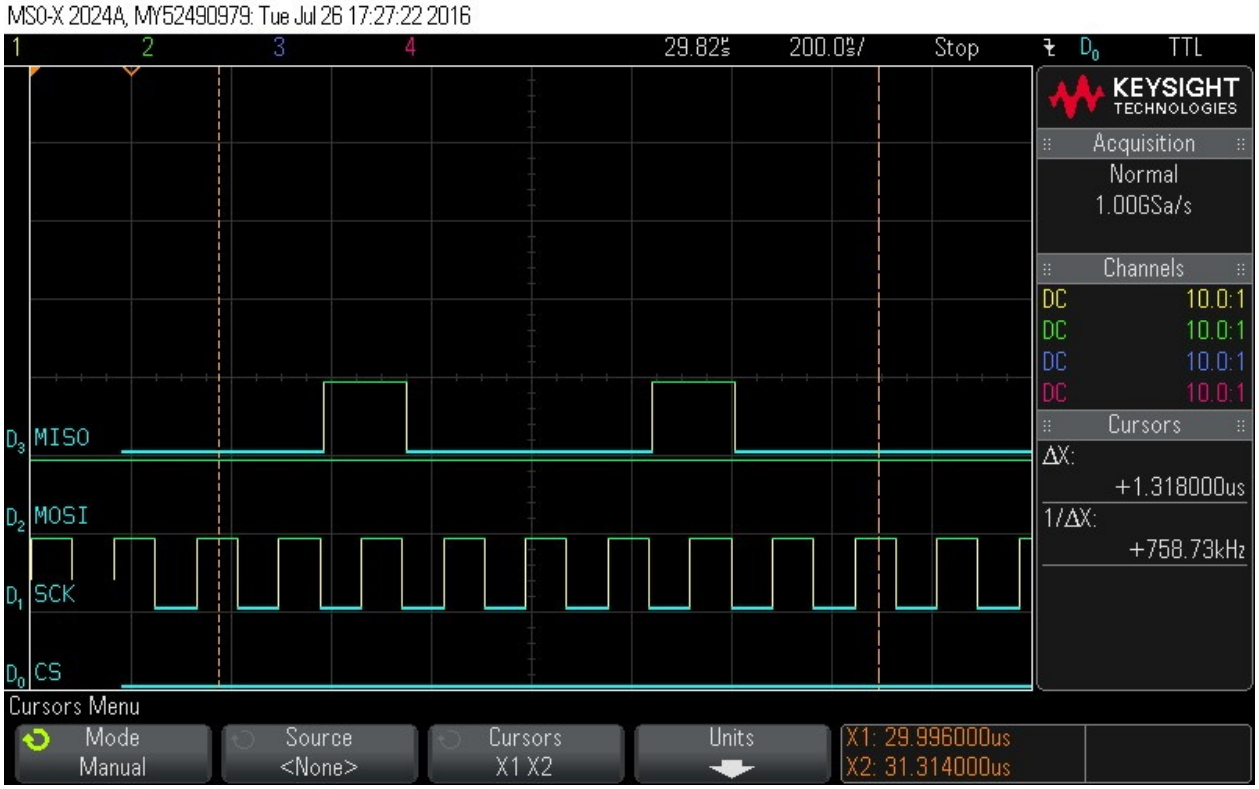
28. Reading out 16 Bytes of FSN data – Byte 5 = 0x0.

Figure 9-67. Reading out 16 Bytes of FSN Data – Byte 5 = 0x0



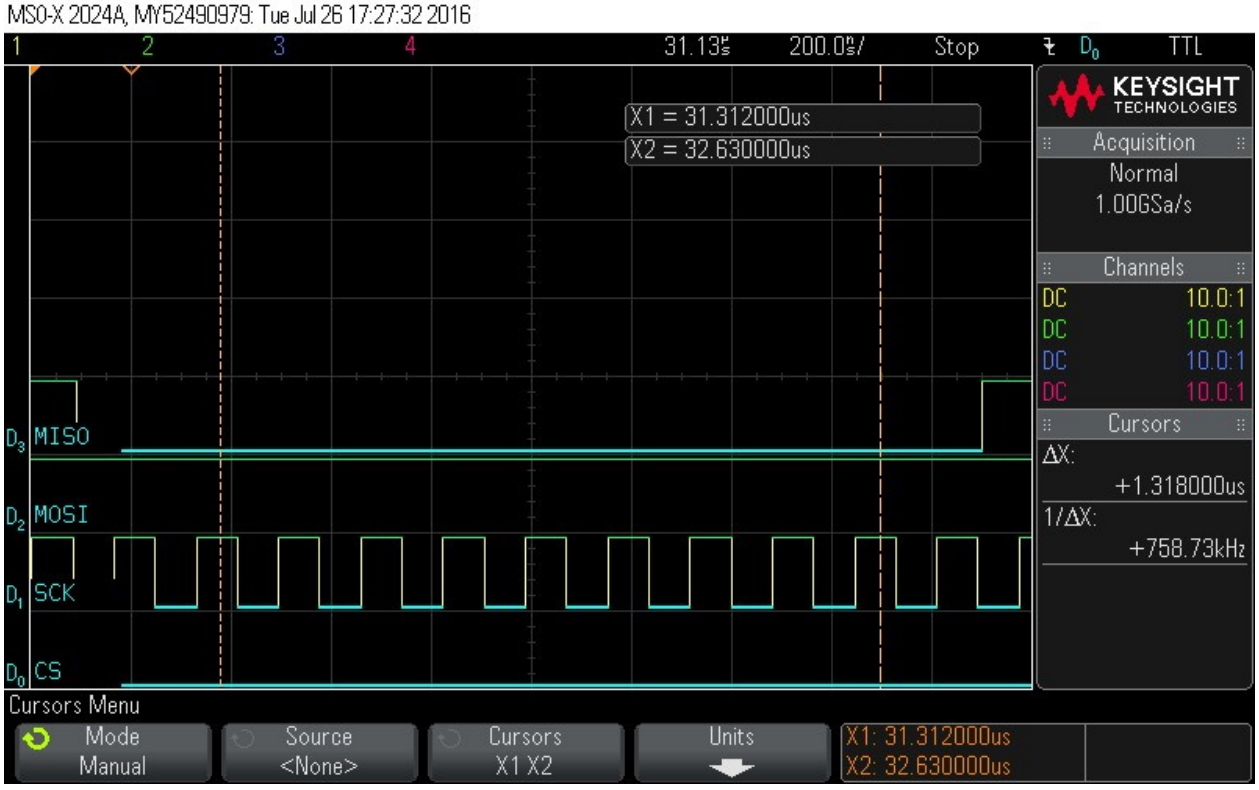
29. Reading out 16 Bytes of FSN data – Byte 6 = 0x44.

Figure 9-68. Reading out 16 Bytes of FSN Data – Byte 6 = 0x44



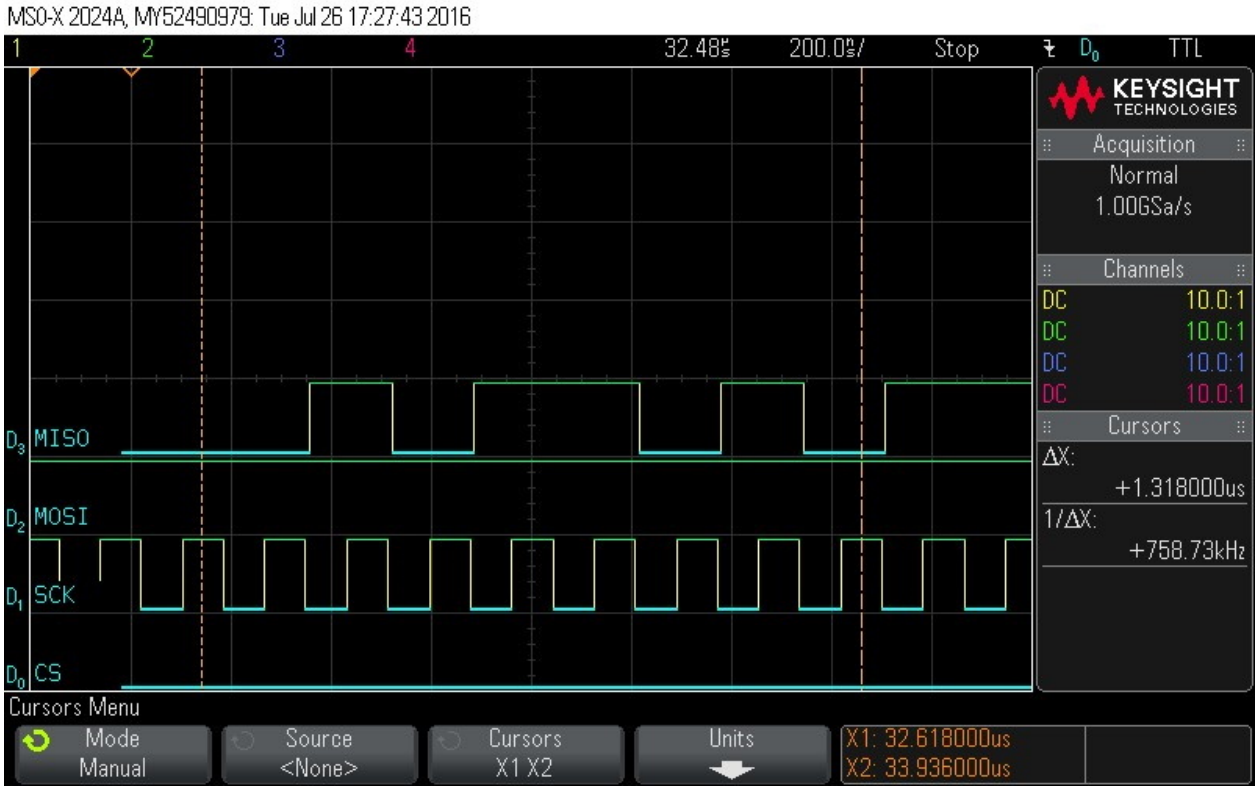
30. Reading out 16 Bytes of FSN data – Byte 7 = 0x0.

Figure 9-69. Reading out 16 Bytes of FSN Data – Byte 7 = 0x0



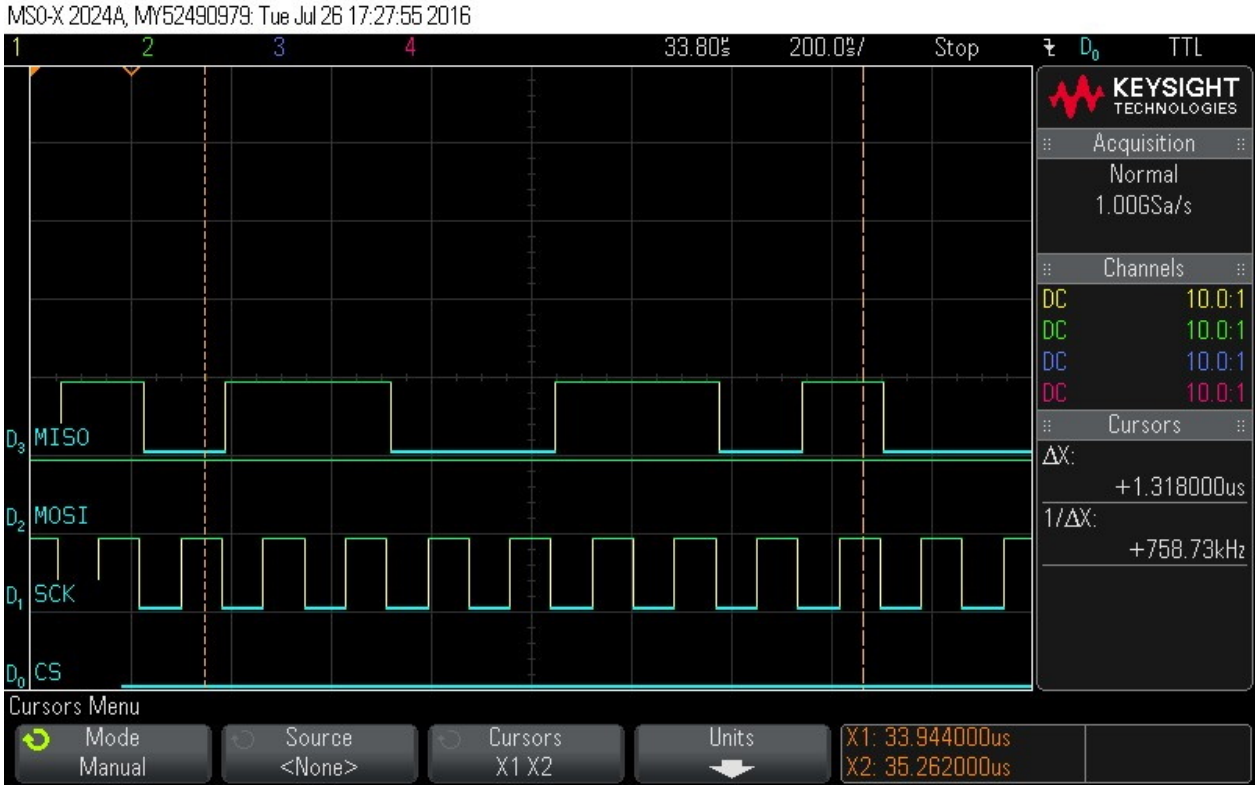
31. Reading out 16 Bytes of FSN data – Byte 8 = 0x5A

Figure 9-70. Reading out 16 Bytes of FSN Data – Byte 8 = 0x5A



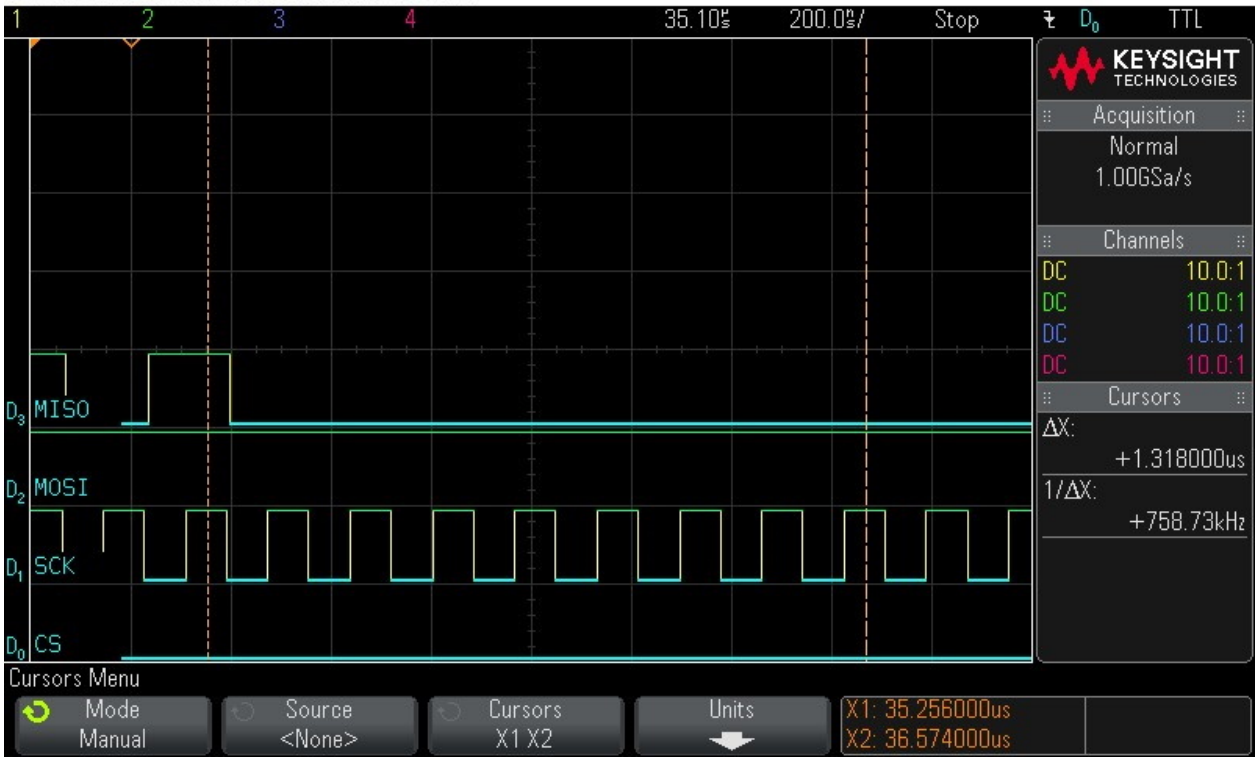
32. Reading out 16 Bytes of FSN data – Byte 9 = 0xCD.

Figure 9-71. Reading out 16 Bytes of FSN Data – Byte 9 = 0xCD



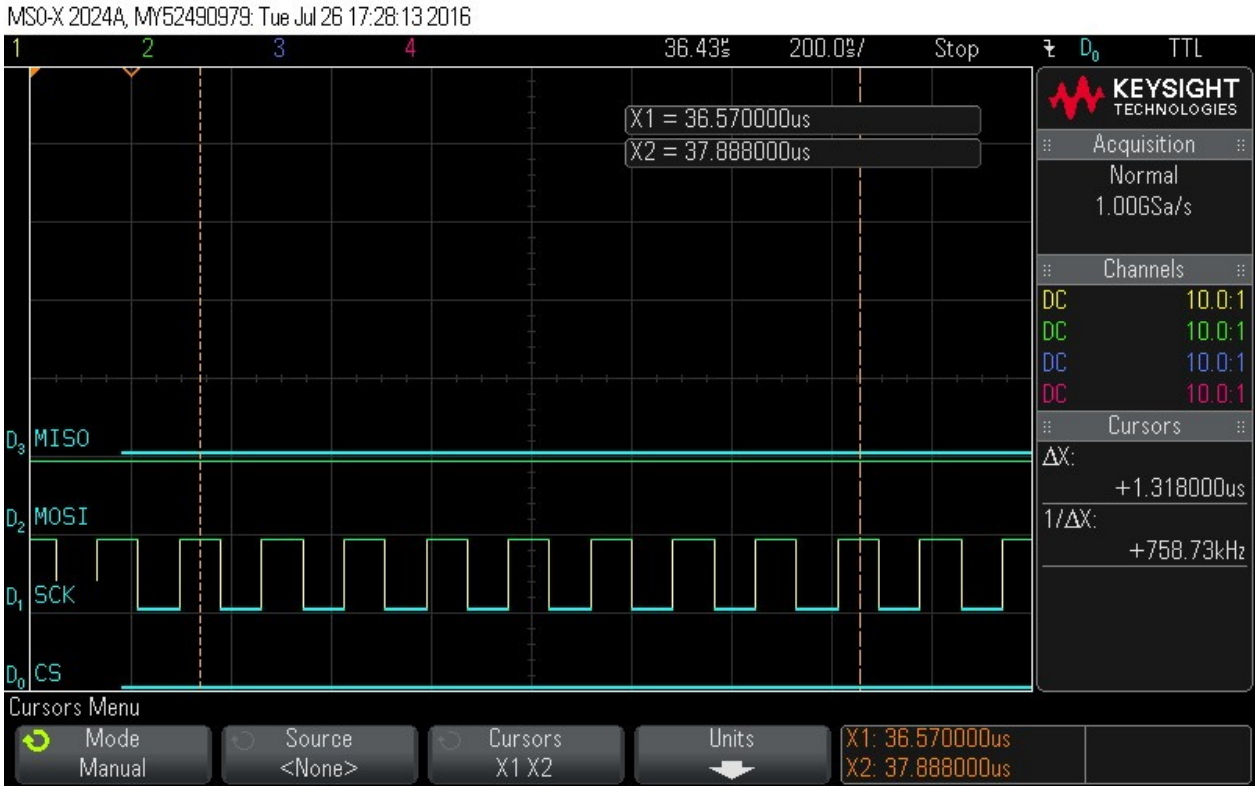
33. Reading out 16 Bytes of FSN data – Byte 10 = 0x0.

Figure 9-72. Reading out 16 Bytes of FSN Data – Byte 10 = 0x0



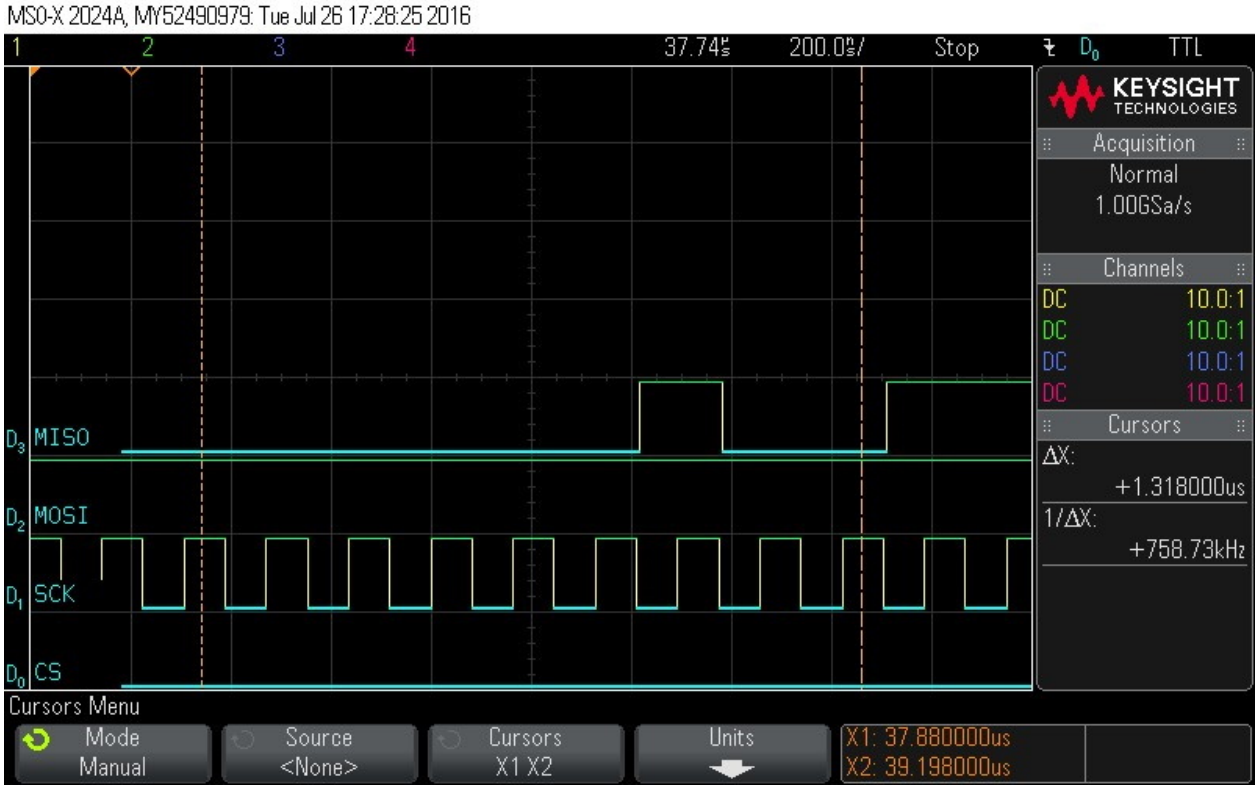
34. Reading out 16 Bytes of FSN data – Byte 11 = 0x0.

Figure 9-73. Reading out 16 Bytes of FSN Data – Byte 11 = 0x0



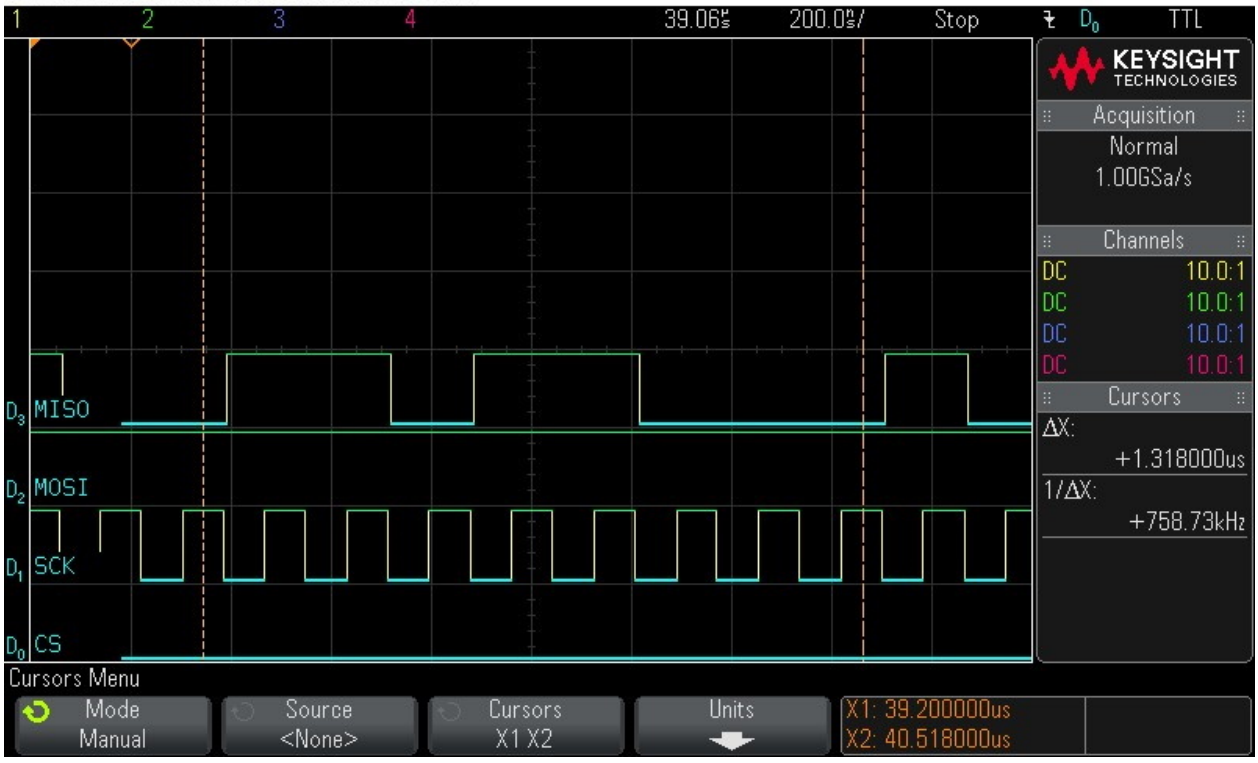
35. Reading out 16 Bytes of FSN data – Byte 12 = 0x4.

Figure 9-74. Reading out 16 Bytes of FSN Data – Byte 12 = 0x4



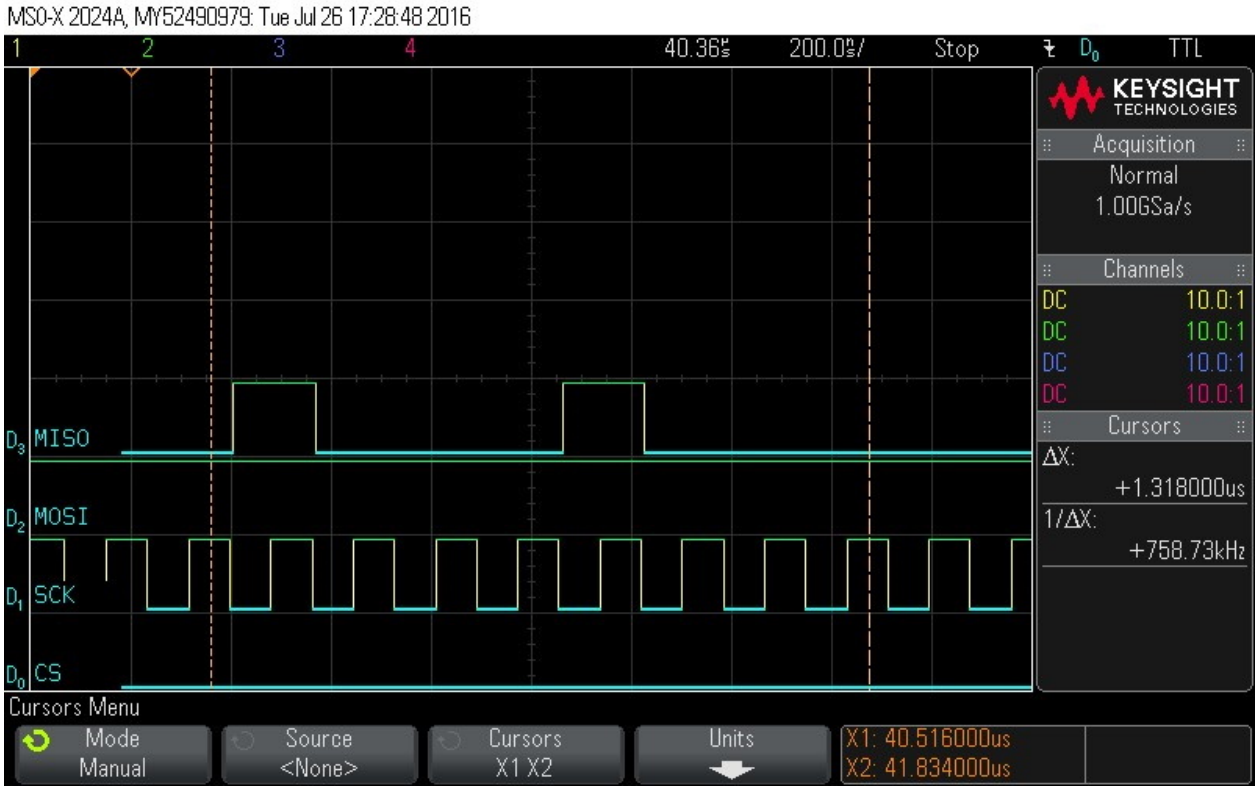
36. Reading out 16 Bytes of FSN data – Byte 13 = 0xD8.

Figure 9-75. Reading out 16 Bytes of FSN Data – Byte 13 = 0xD8



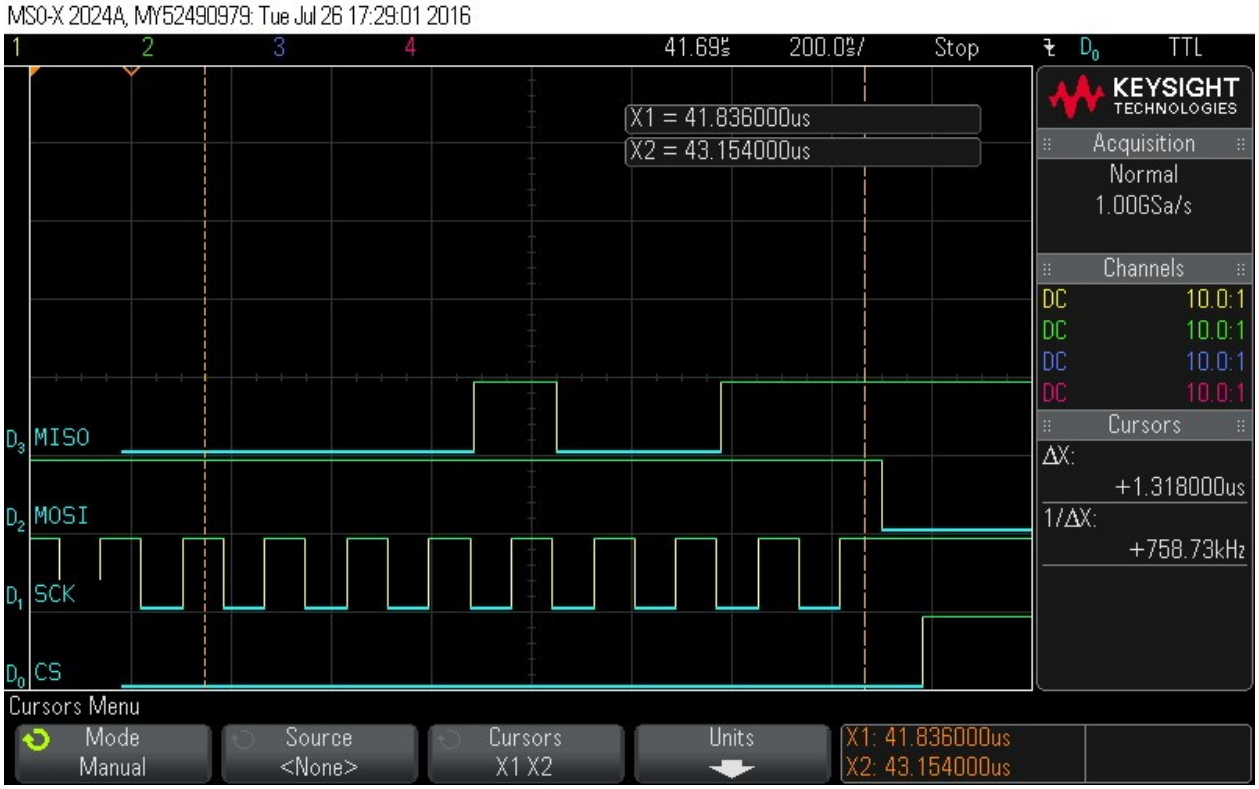
37. Reading out 16 Bytes of FSN data – Byte 14 = 0x88.

Figure 9-76. Reading out 16 Bytes of FSN Data – Byte 14 = 0x88



38. Reading out 16 Bytes of FSN data – Byte 15 = 0x13.

Figure 9-77. Reading out 16 Bytes of FSN Data – Byte 15 = 0x13



9.3 Program Frame Waveform

When performing Program, Verify or Authenticate actions, the data is clocked into the device starts at the beginning of the datastream block as shown in the following figure. This data is different depending on the device and the design, but in all cases, the data is clocked in 16 bytes at a time.

The following scope plots show how the first data frame is clocked.

Figure 9-78. Datastream Block

```
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000
numOfBlocks
2000407E93B005006400002000000
datastream
7C5D1C2B3D7519B3924AABEE4ED56C6286545B7B3266FDF5975C868564A76E06DC5E14F6C16E6DE0
C7F14DFC0DA0412A00030002000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000002F8FFFF030100C02FD94D3253303530000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
8C41036EF3EF6CC81E93FD29C4F2890B6FBA831F5F4A2BAB62092E6A9C604E7E9AC405860A7B097A
5B80D956D163A0012389AC4DB418B59558AB1A2F1CF5703EFB1915B8C20DA85FB053B7DA68E5490C
6B3E0239D719E17F9F2869F240AC43034122933EC8A0D2F58CBBD85B5AA8B2247496355 69CD51E95
CD18AC8359DCA64994043B10DA5B9A7596CC21727F01D3C8D20279283AD77D2726F1549822F1B52A
0103000200000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```

In this example, the following data is clocked.

Table 9-1. Clocked Data

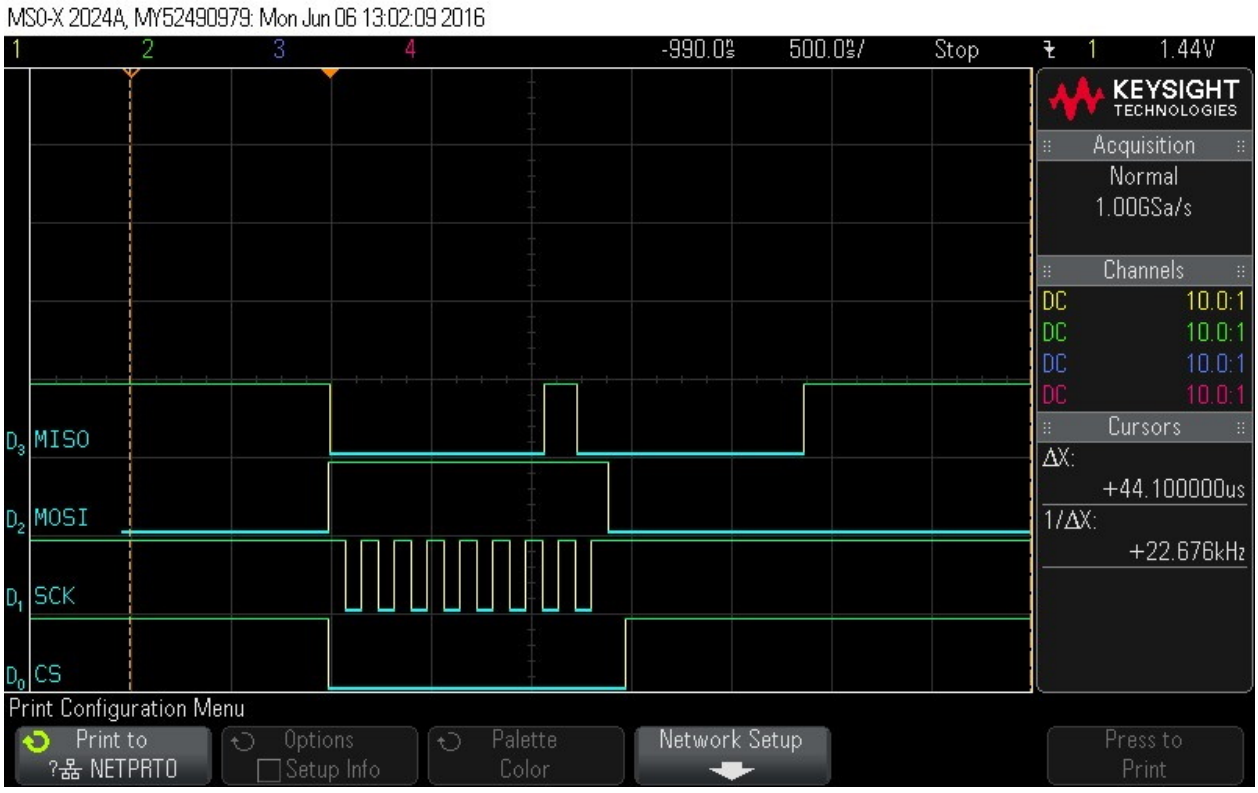
Bytes															
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
7C	5D	1C	2B	3D	75	19	B3	92	4A	AB	EE	4E	D5	6C	62

Mode 3 of the SPI mode is used and the data is clocked byte 0 MSB first. Note the following:

1. Before performing any data shift, the target device SPI buffer status is checked by shifting 0xff. This is the only instruction that is 8 bit long, and the data is read out at the same time as it is shifted in. The result of the first shift is ignored.
2. When shifting data, into the device, the first byte is the command followed by 16 bytes of data. 16 bytes of zero value must be shifted for commands that do not require data.
3. Shifting data out from the device is a two steps operation. The command is clocked into the device first and then the data is clocked out using a read command of 0x5.
4. All operations except for SPI hardware status check are made of one byte of command followed by 16 bytes of data. Chip Select (CS) line must be driven low before clocking the command and should remain low until the last bit of data is shifted in. Then, it must be driven high to execute the loaded instruction.
   Note: 1, 2, and 3 are taken care of by the programming algorithm.

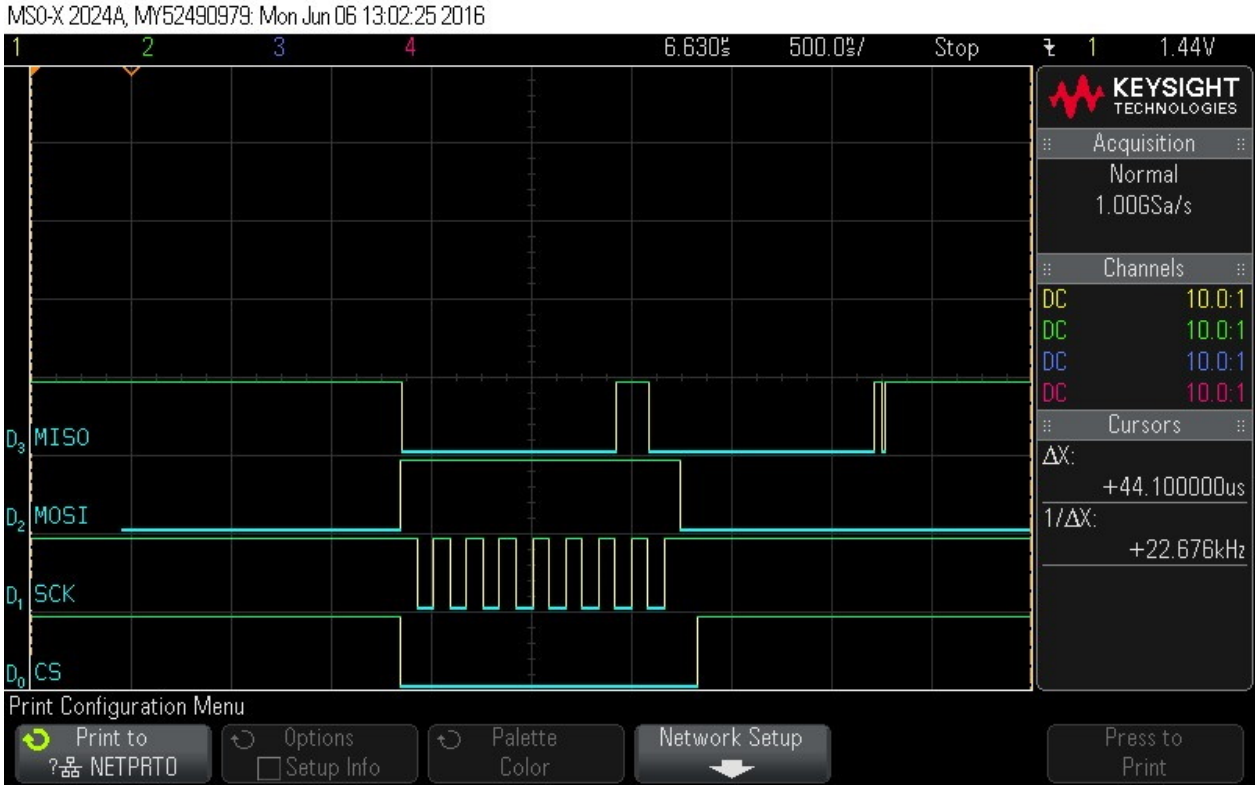
1. Checking hardware status.

   Figure 9-79. Hardware Status Check



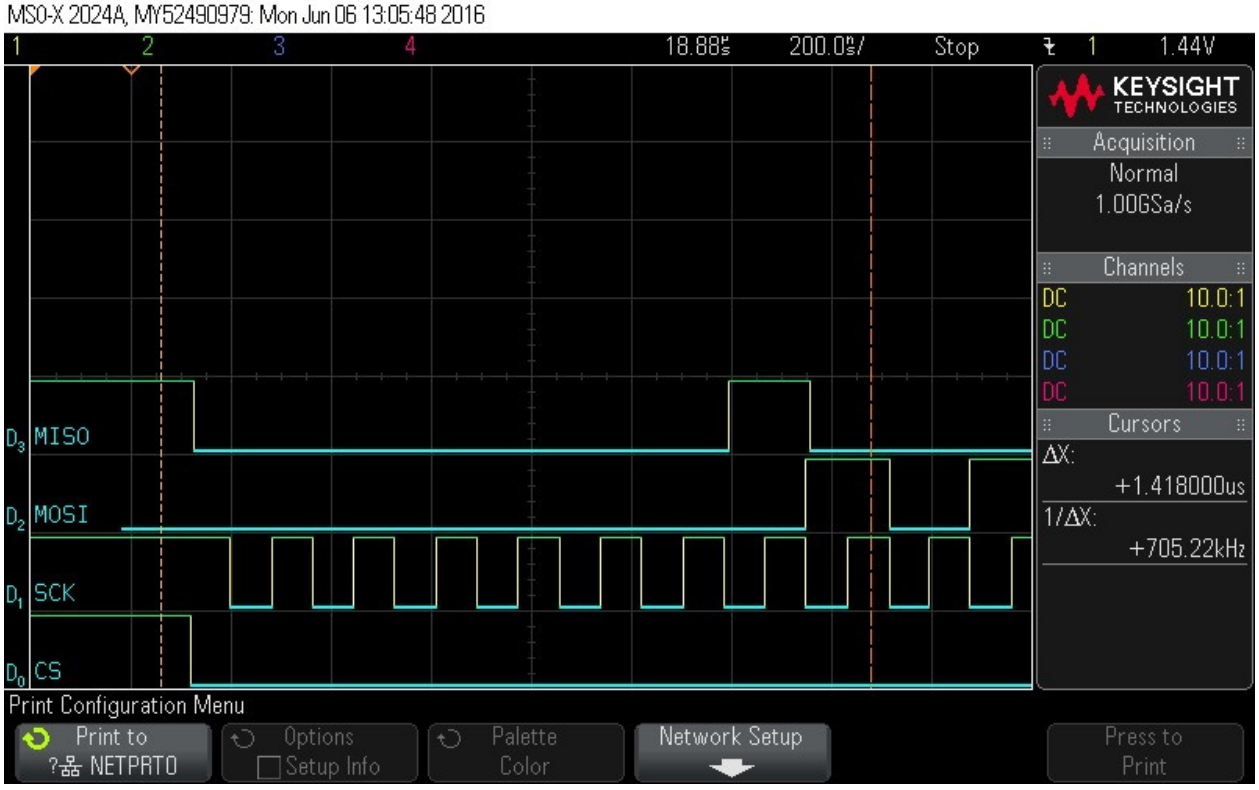
2. Checking hardware status.

Figure 9-80. Hardware Status Check



3. Shift in the first frame. Command = 0x1. Data to follow. Note CS signal.

Figure 9-81. Shift in the First Frame. Command = 0x1. Data to Follow. Note CS Signal

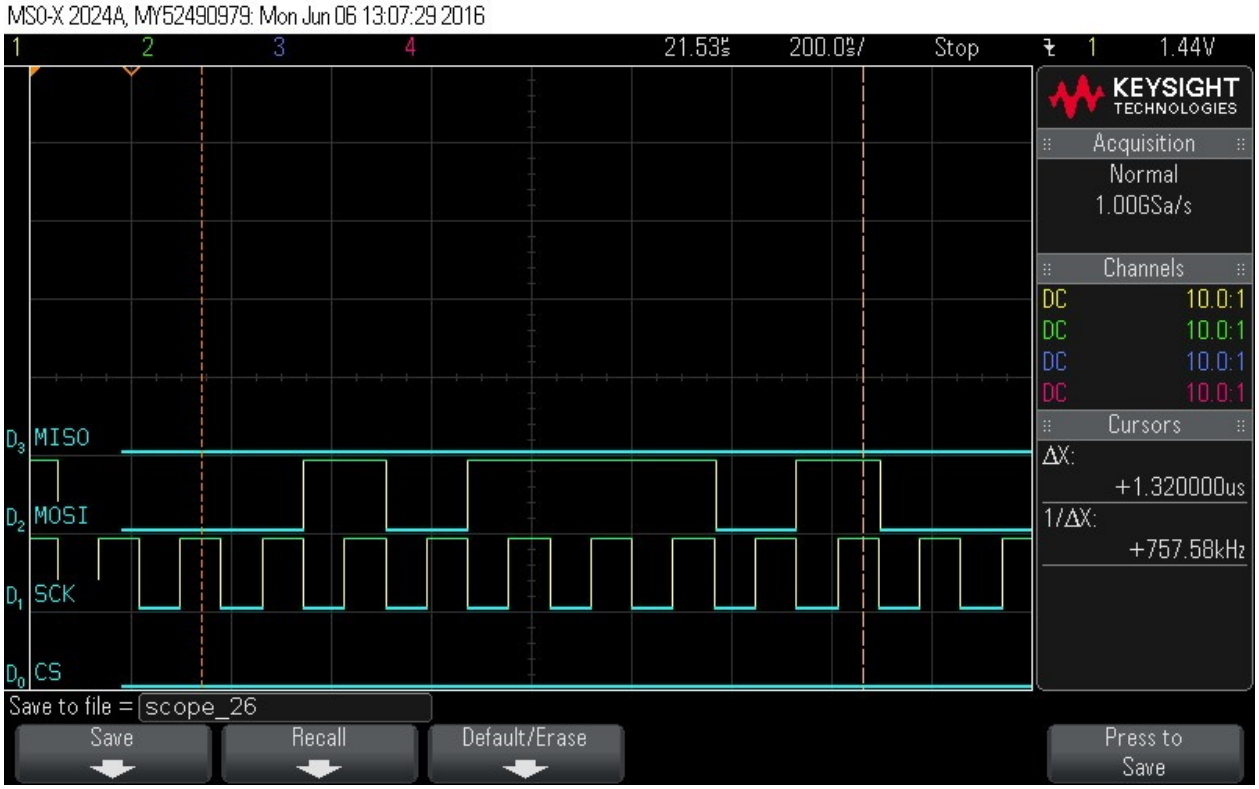


4. Data Byte 0 = 0x7C

Figure 9-82. Data Byte 0 = 0x7C

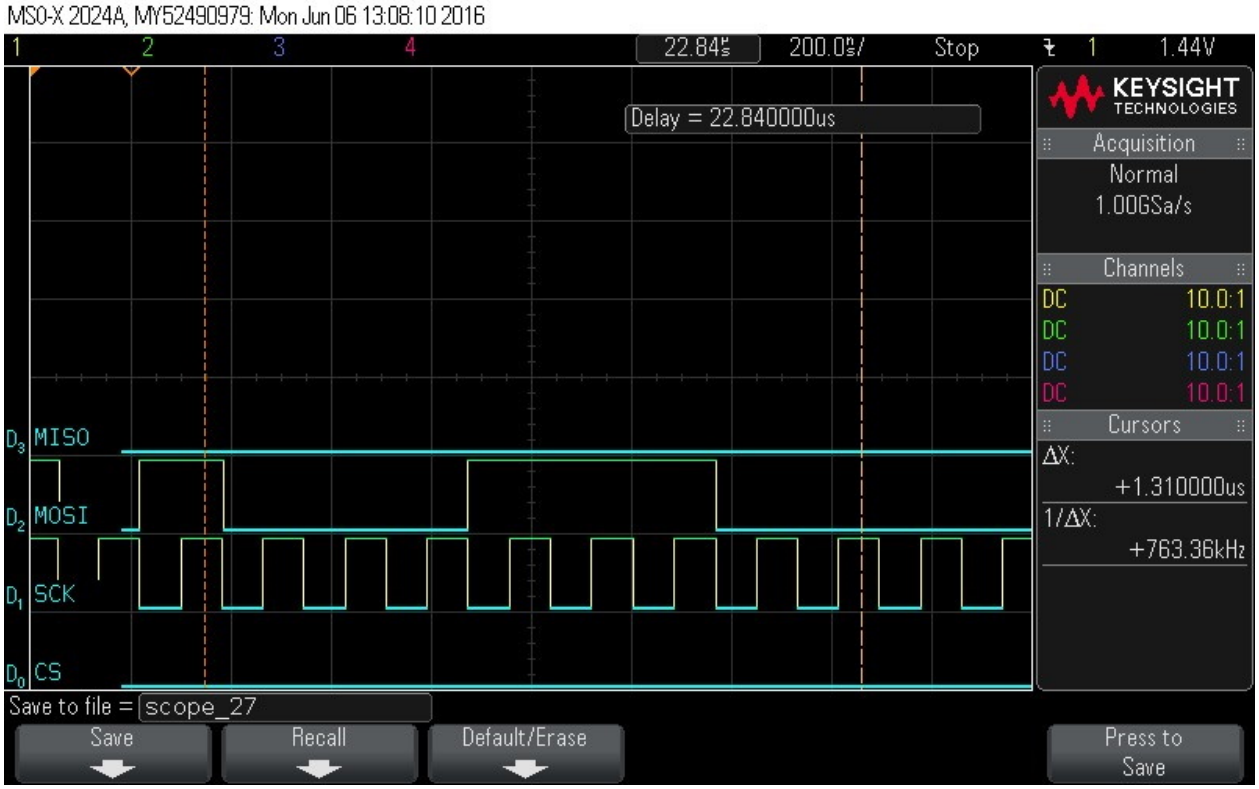
5. Data Byte 1 = 0x5D

Figure 9-83. Data Byte 1 = 0x5D



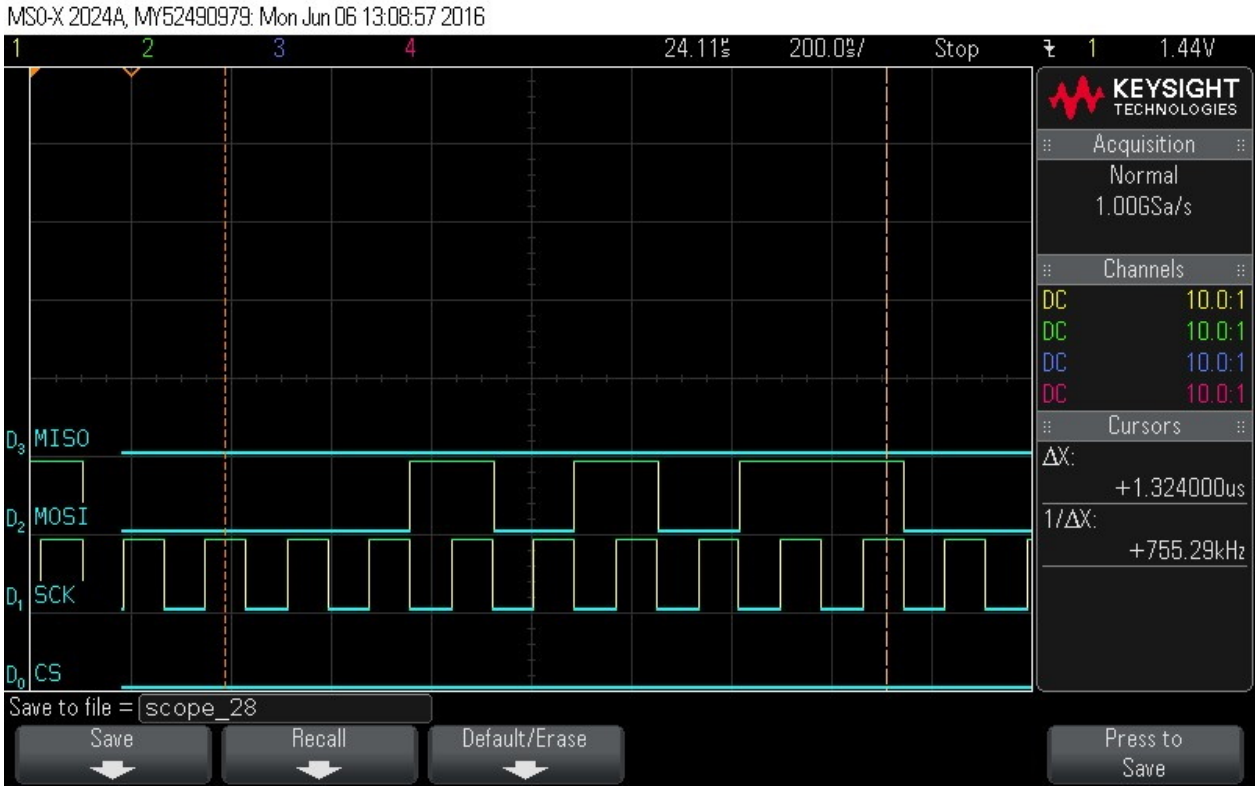
6. Data Byte 2 = 0x1C

Figure 9-84. Data Byte 2 = 0x1C



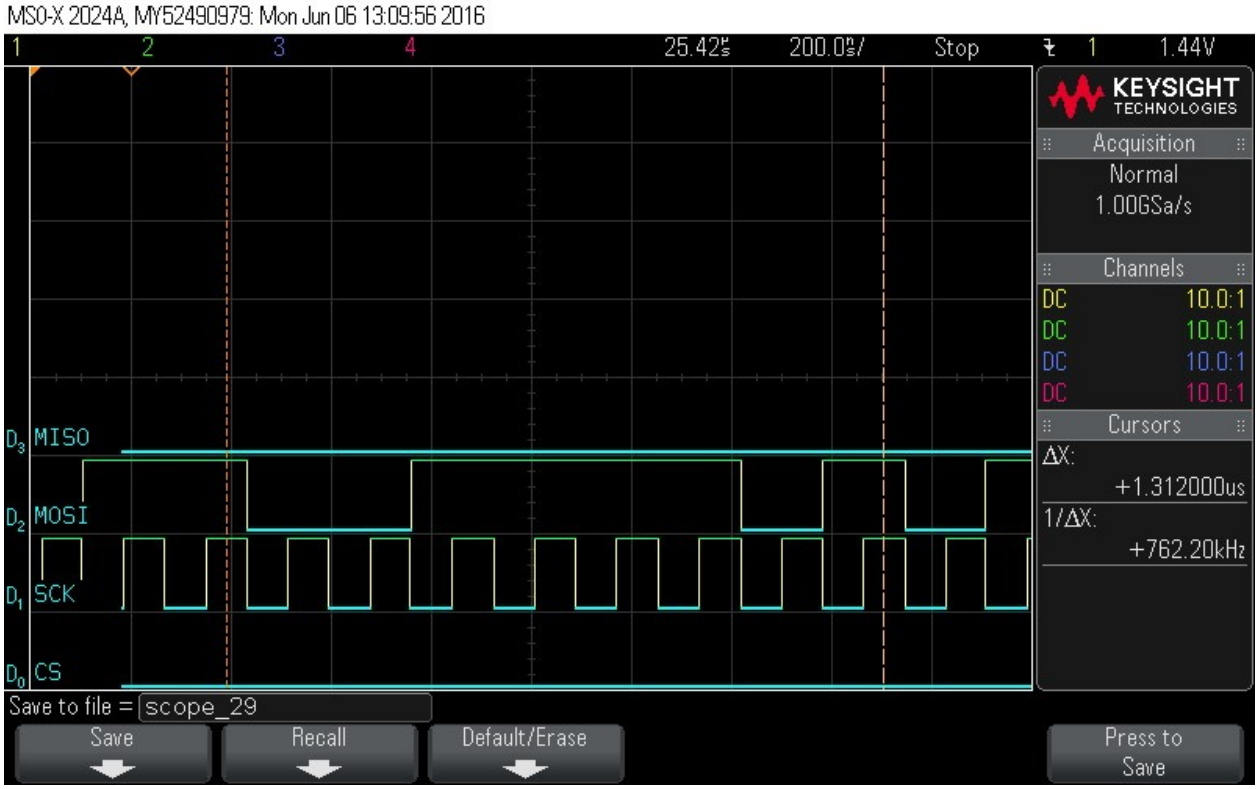
7. Data Byte 3 = 0x2B

Figure 9-85. Data Byte 3 = 0x2B



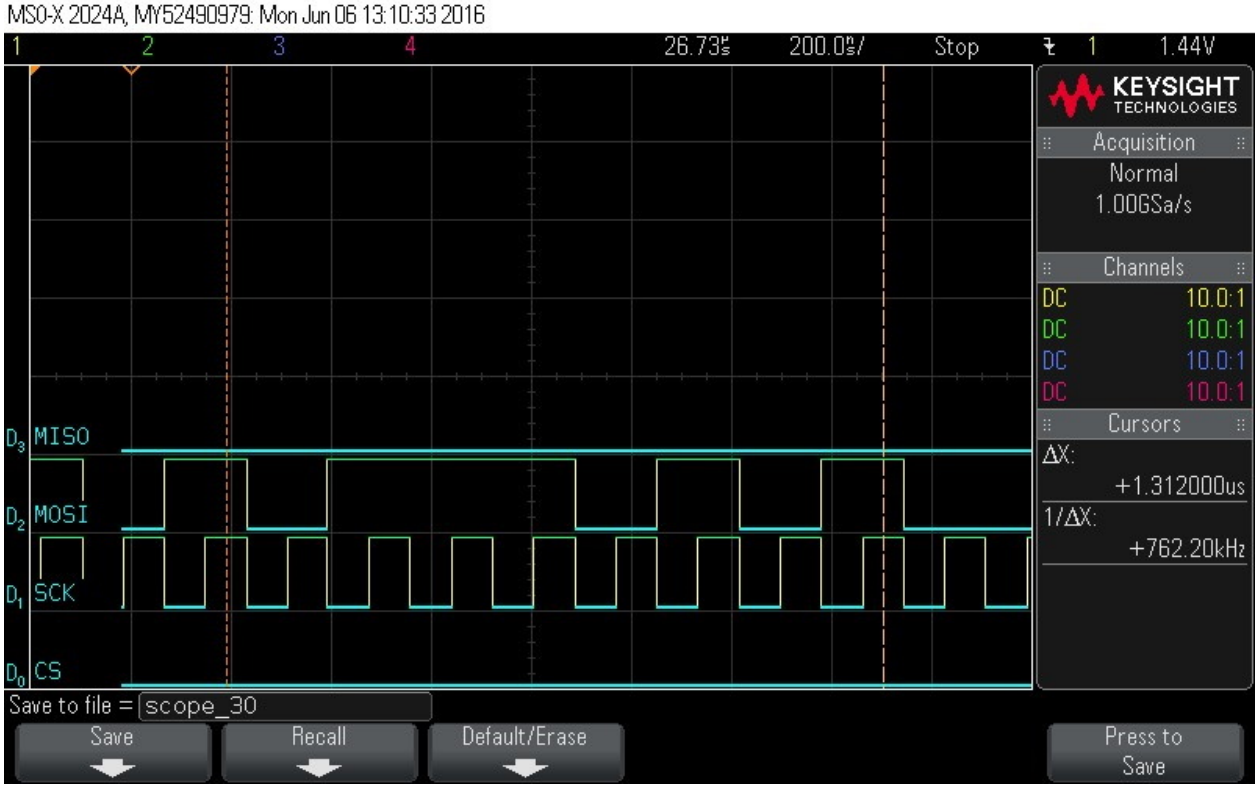
8. Data Byte 4 = 0x3D

Figure 9-86. Data Byte 4 = 0x3D



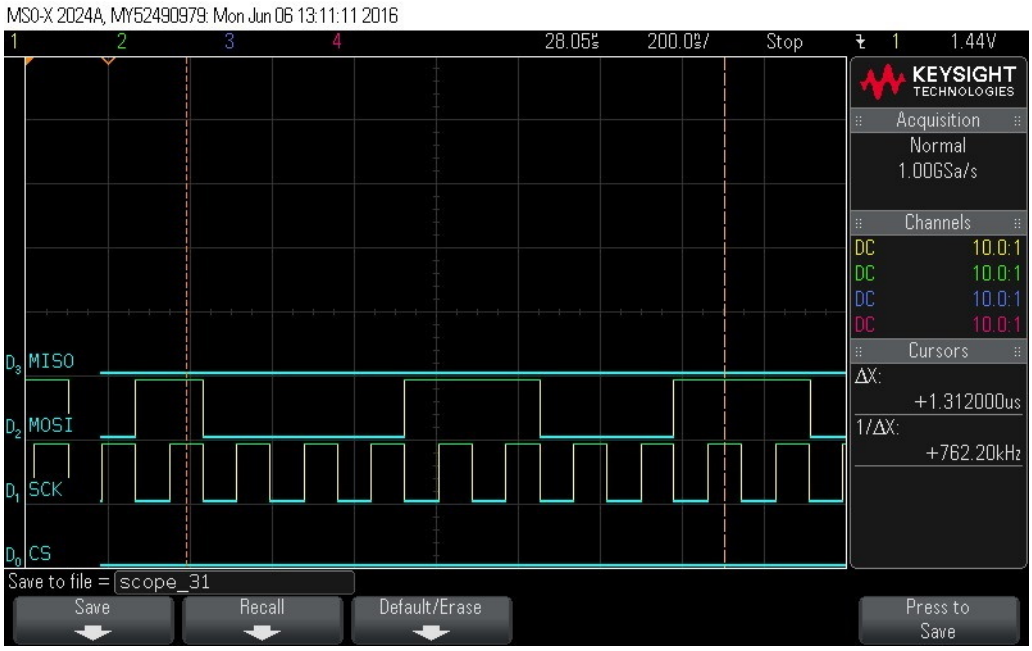
9. Data Byte 5 = 0x75

Figure 9-87. Data Byte 5 = 0x75



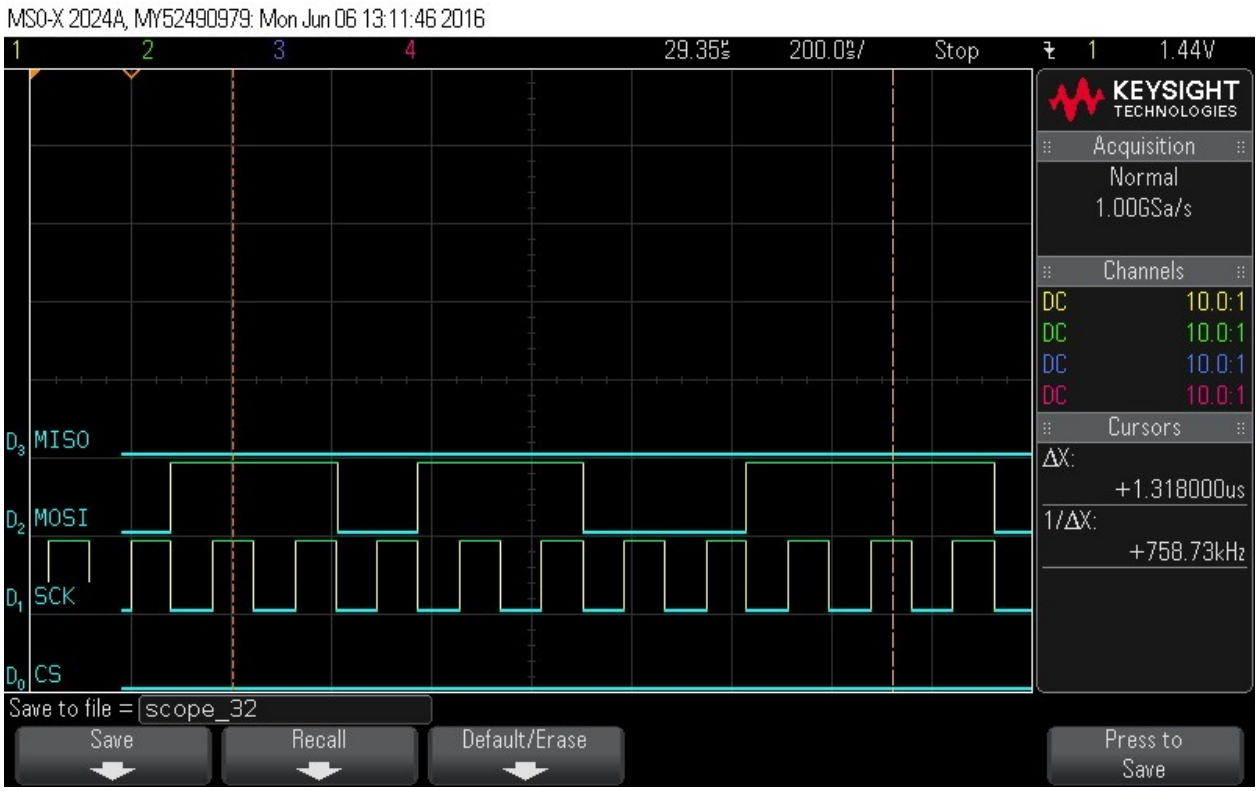
10. Data Byte 6 = 0x19

Figure 9-88. Data Byte 6 = 0x19



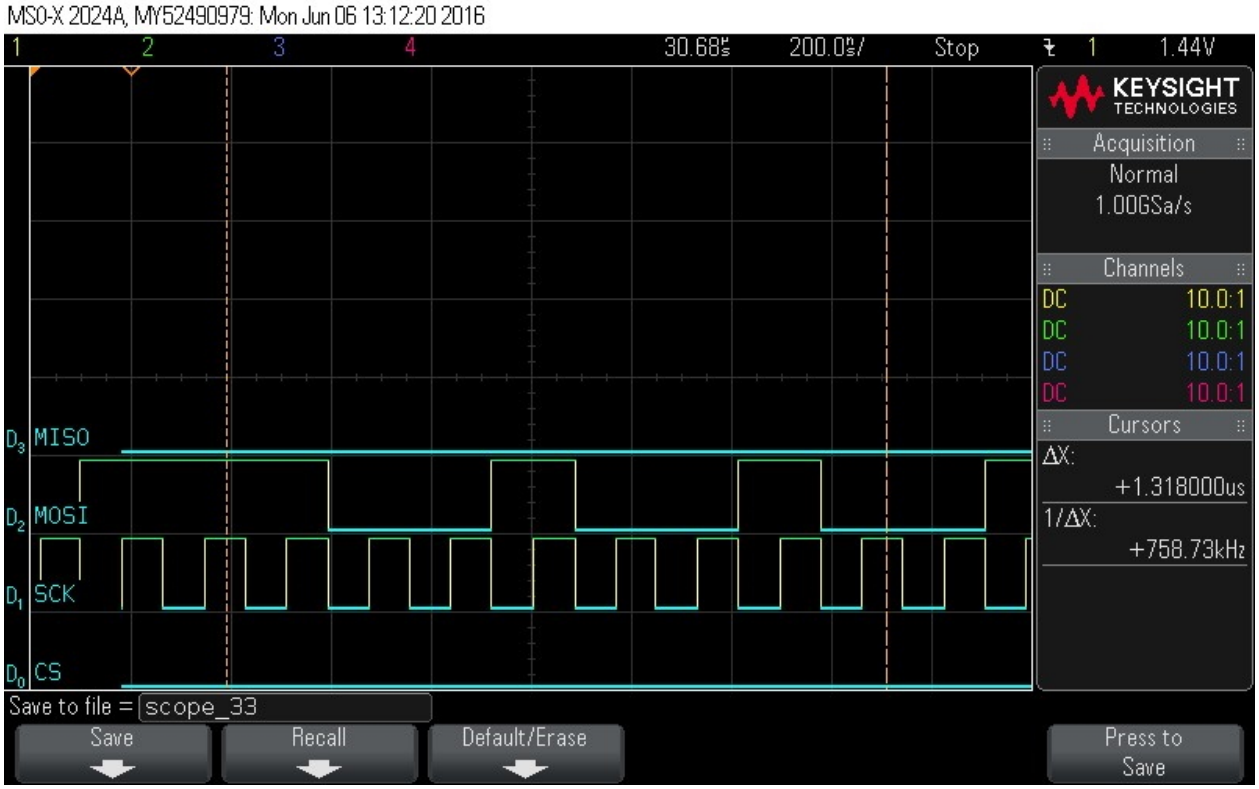
11. Data Byte 7 = 0xB3

Figure 9-89. Data Byte 7 = 0xB3



12. Data Byte 8 = 0x92

Figure 9-90. Data Byte 8 = 0x92

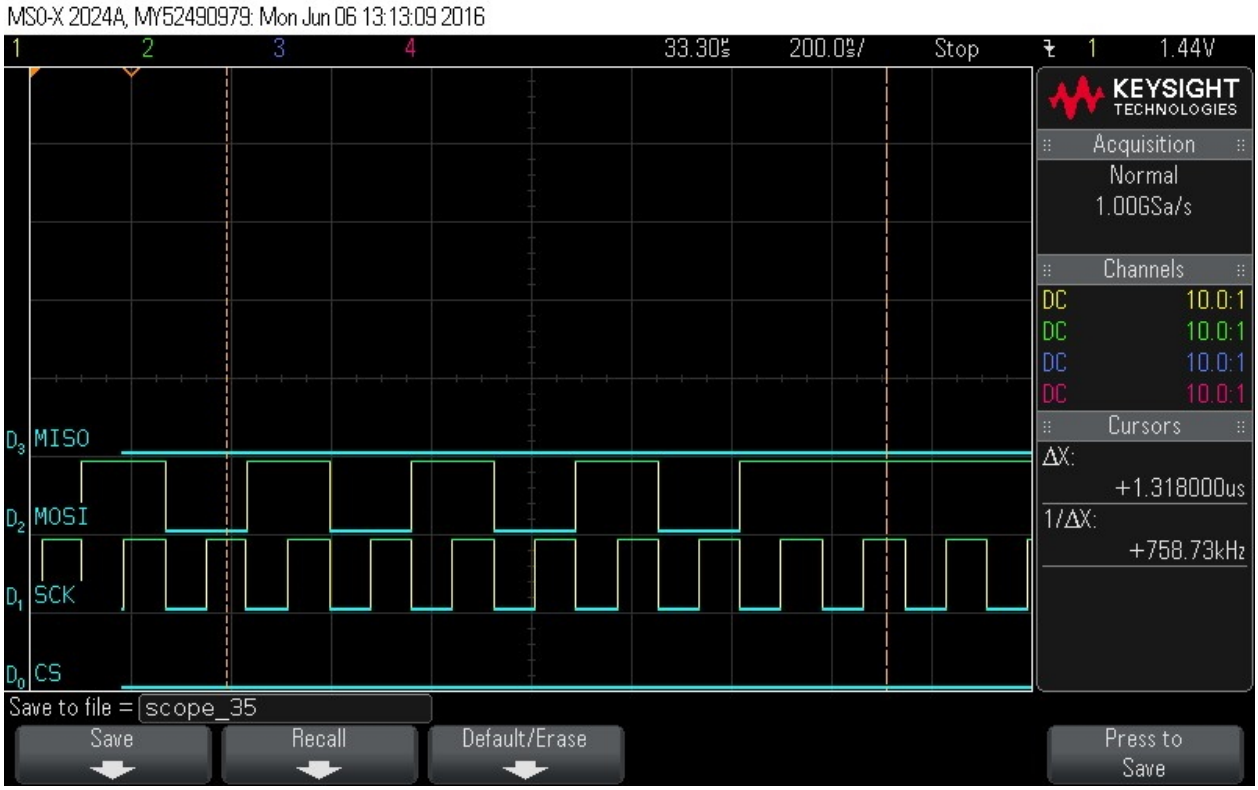


13. Data Byte 9 = 0x4A

Figure 9-91. Data Byte 9 = 0x4A

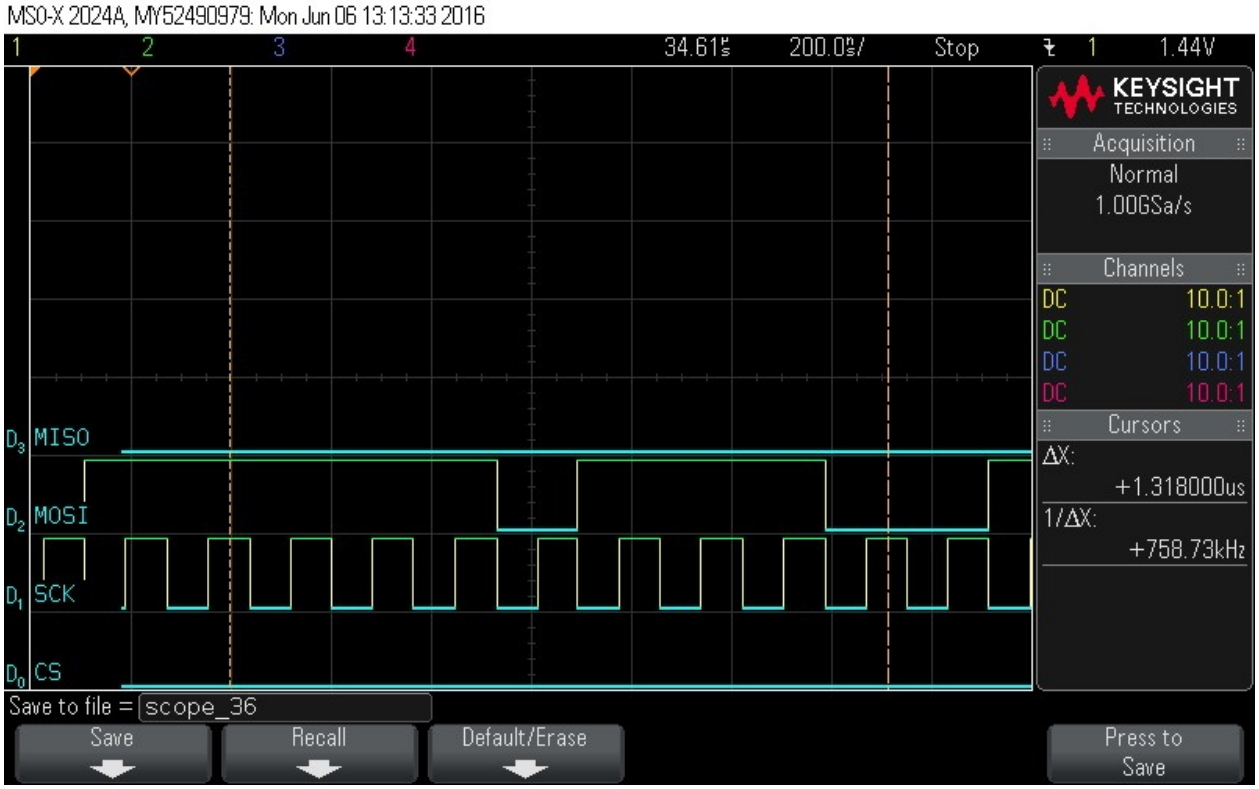
14. Data Byte 10 = 0xAB

Figure 9-92. Data Byte 10 = 0xAB



15. Data Byte 11 = 0xEE

Figure 9-93. Data Byte 11 = 0xEE

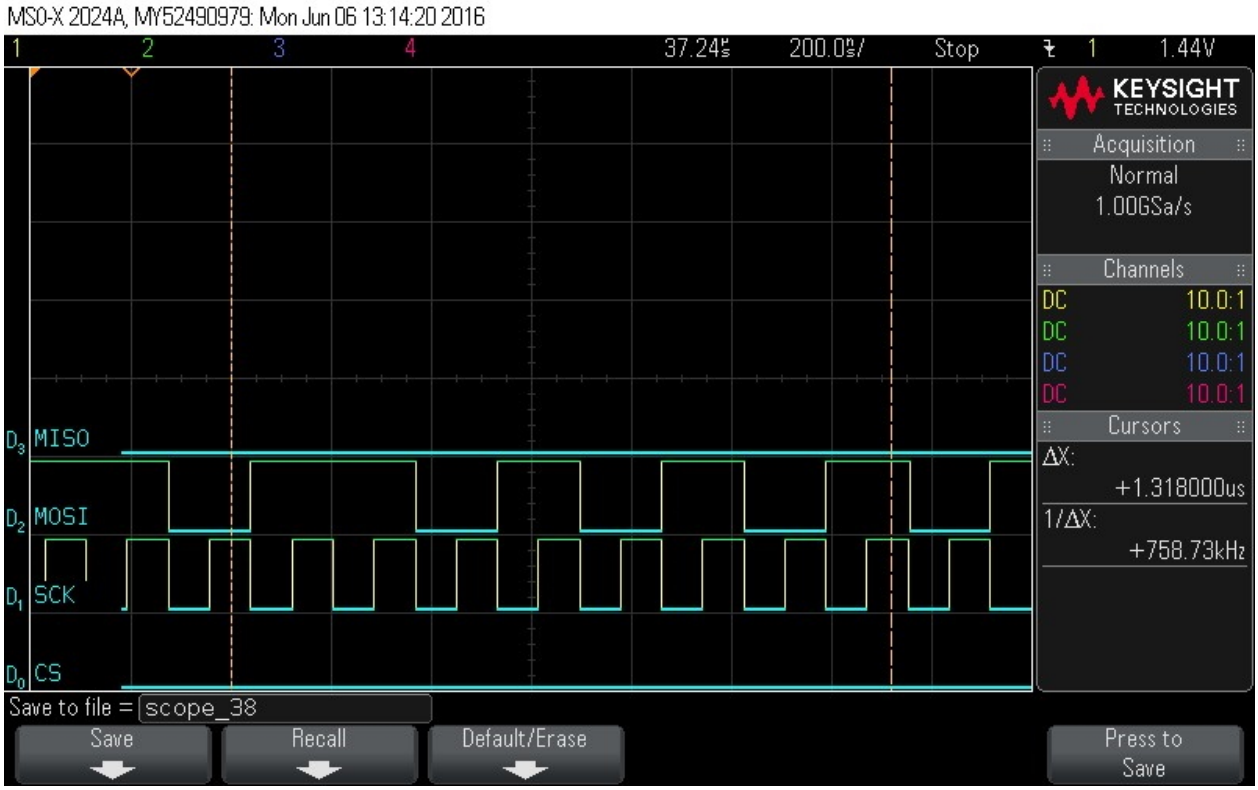


16. Data Byte 12 = 0x4E

Figure 9-94. Data Byte 12 = 0x4E

17. Data Byte 13 = 0xD5

Figure 9-95. Data Byte 13 = 0xD5

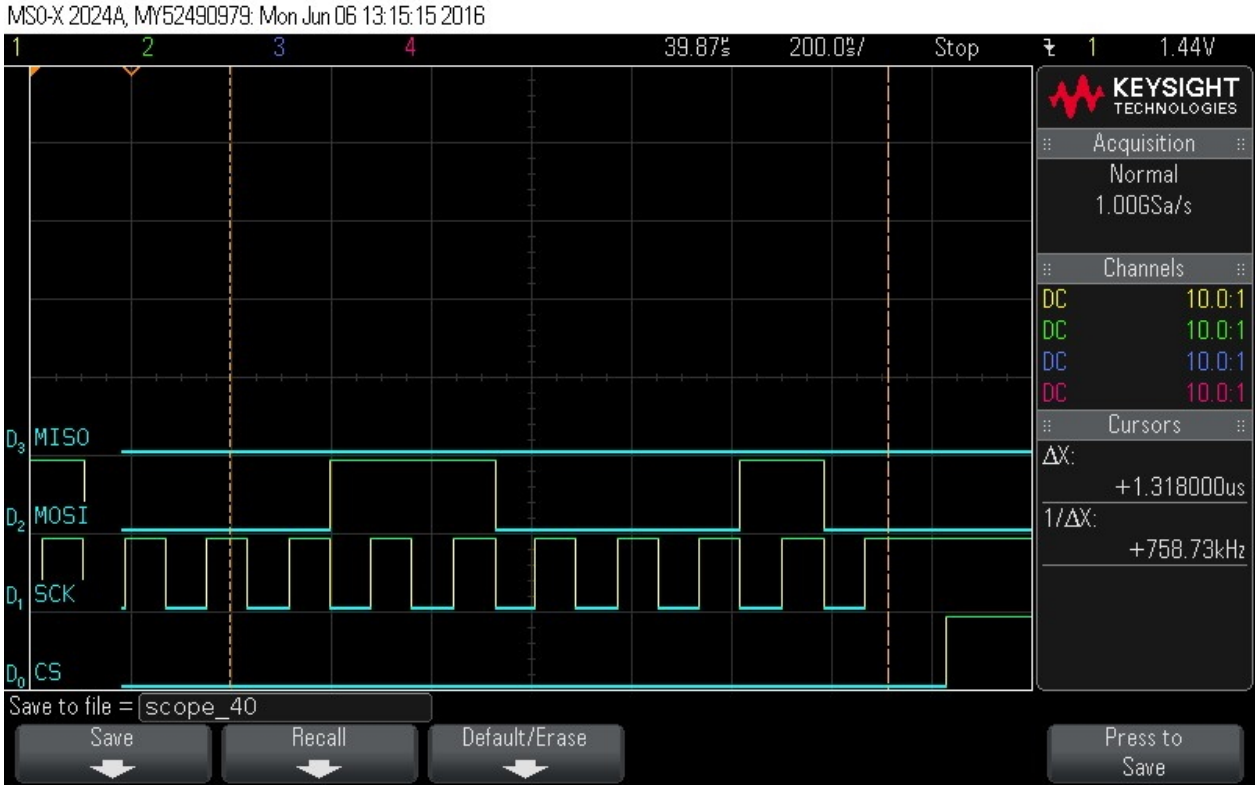


18. Data Byte 14 = 0x6C

Figure 9-96. Data Byte 14 = 0x6C

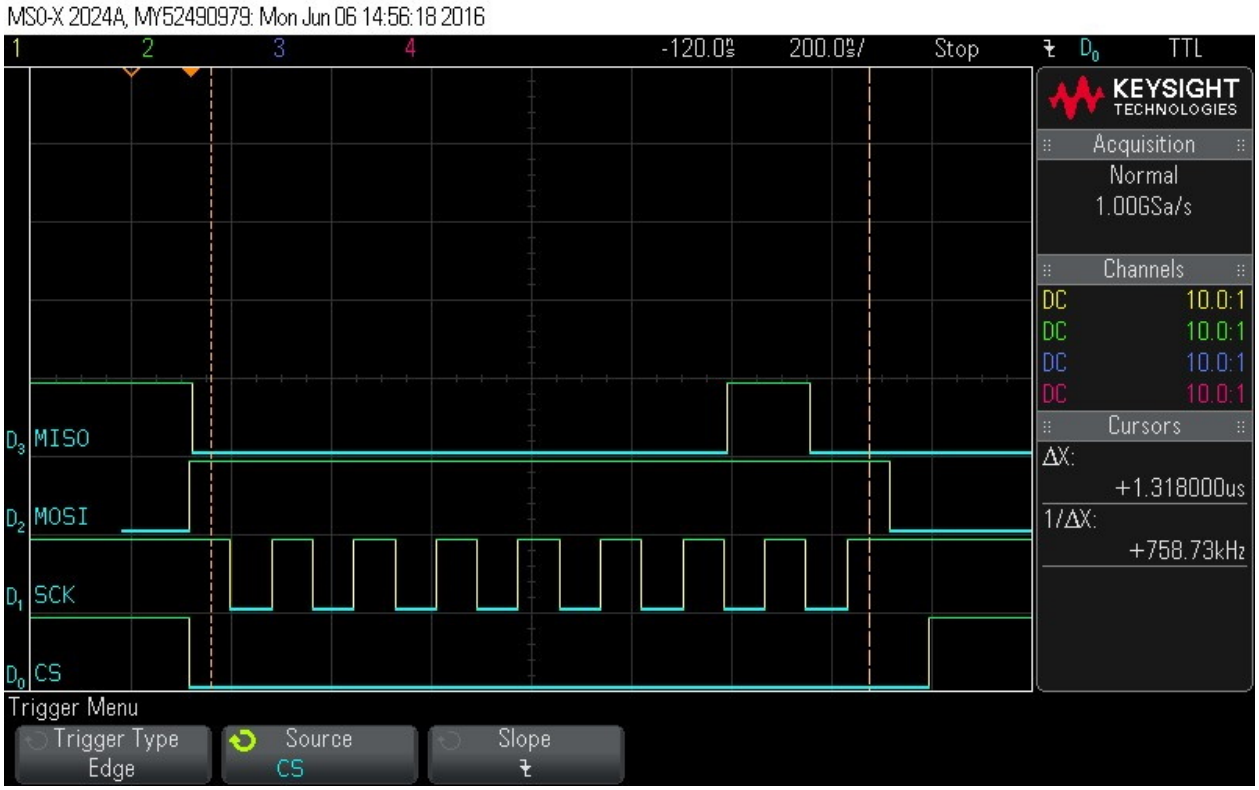
19. At this point, the first frame of data is clocked in. The next operation is to check the status.

Figure 9-97. Data Byte 15 = 0x62. Note CS signal



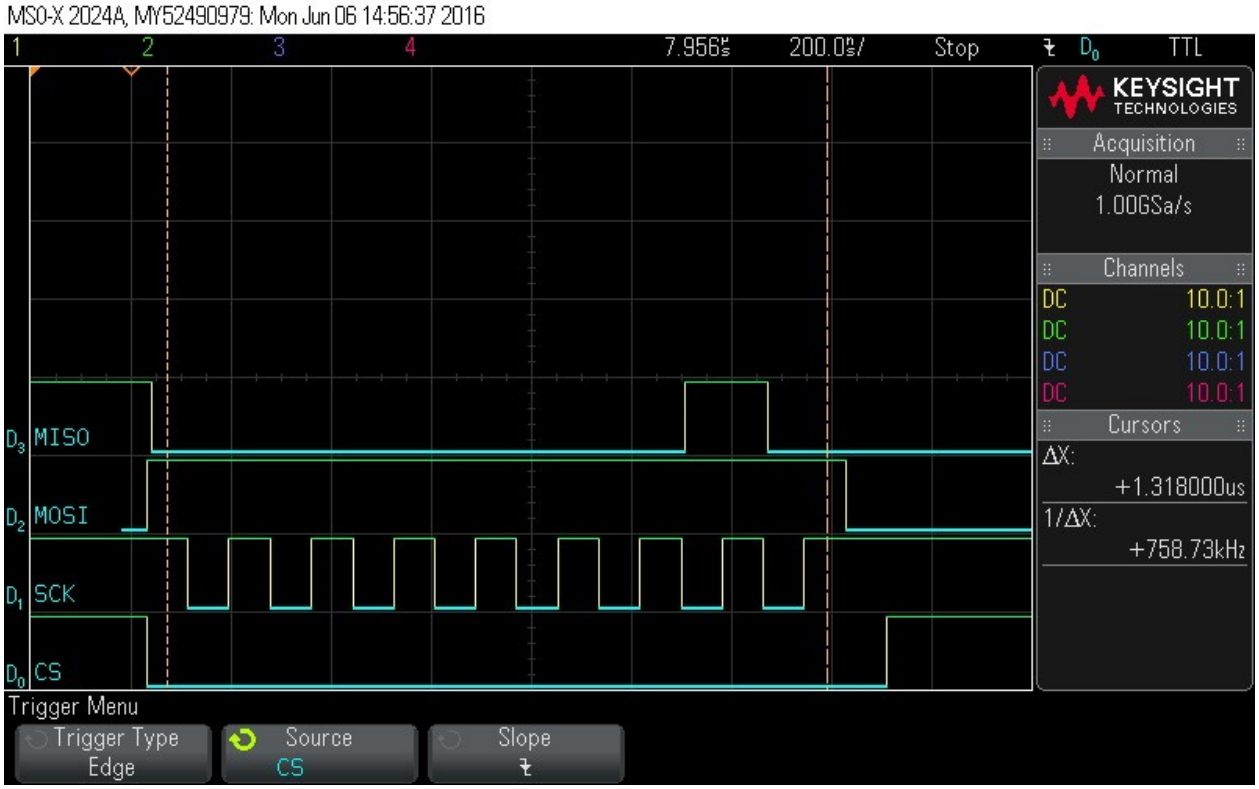
20. Checking hardware status.

Figure 9-98. Hardware Status Check



21. Framing status command. Command = 0x4. Note CS signal.

Figure 9-99. Frame Status command. Command = 0x4. Note CS signal

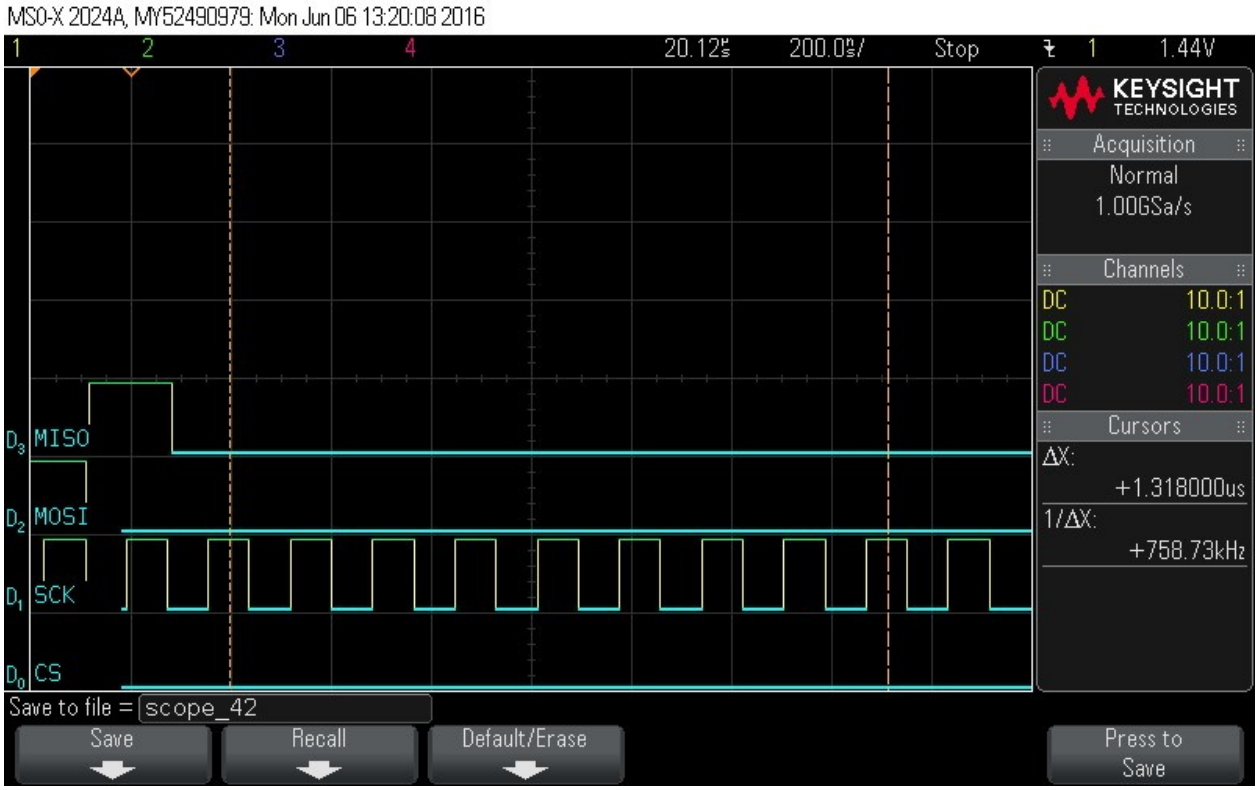


22. Framing status command. Command = 0x4. Note CS signal.

Figure 9-100. Frame Status Command. Command = 0x4. Note CS signal

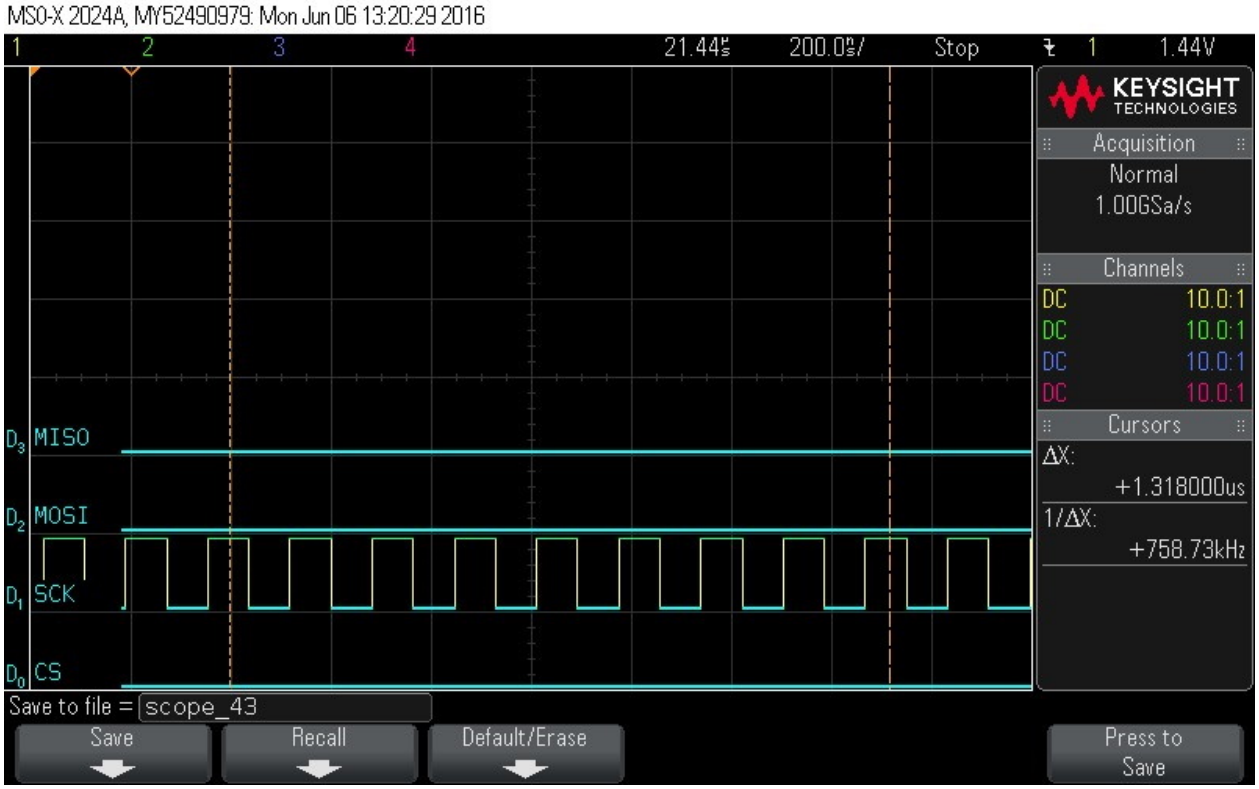
23. Data Byte 0 = 0x0

Figure 9-101. Data Byte 0 = 0x0



24. Data Byte 1 = 0x0

Figure 9-102. Data Byte 1 = 0x0

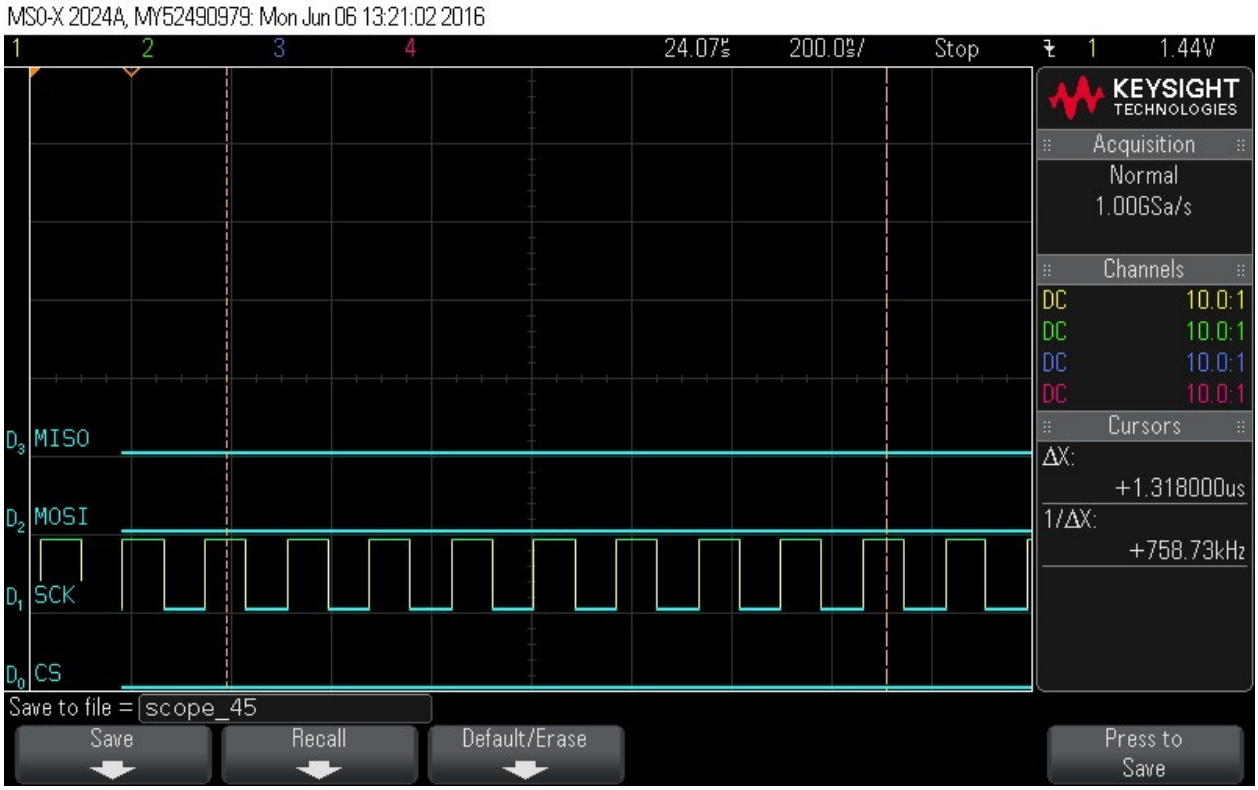


25. Data Byte 2 = 0x0

Figure 9-103. Data Byte 2 = 0x0

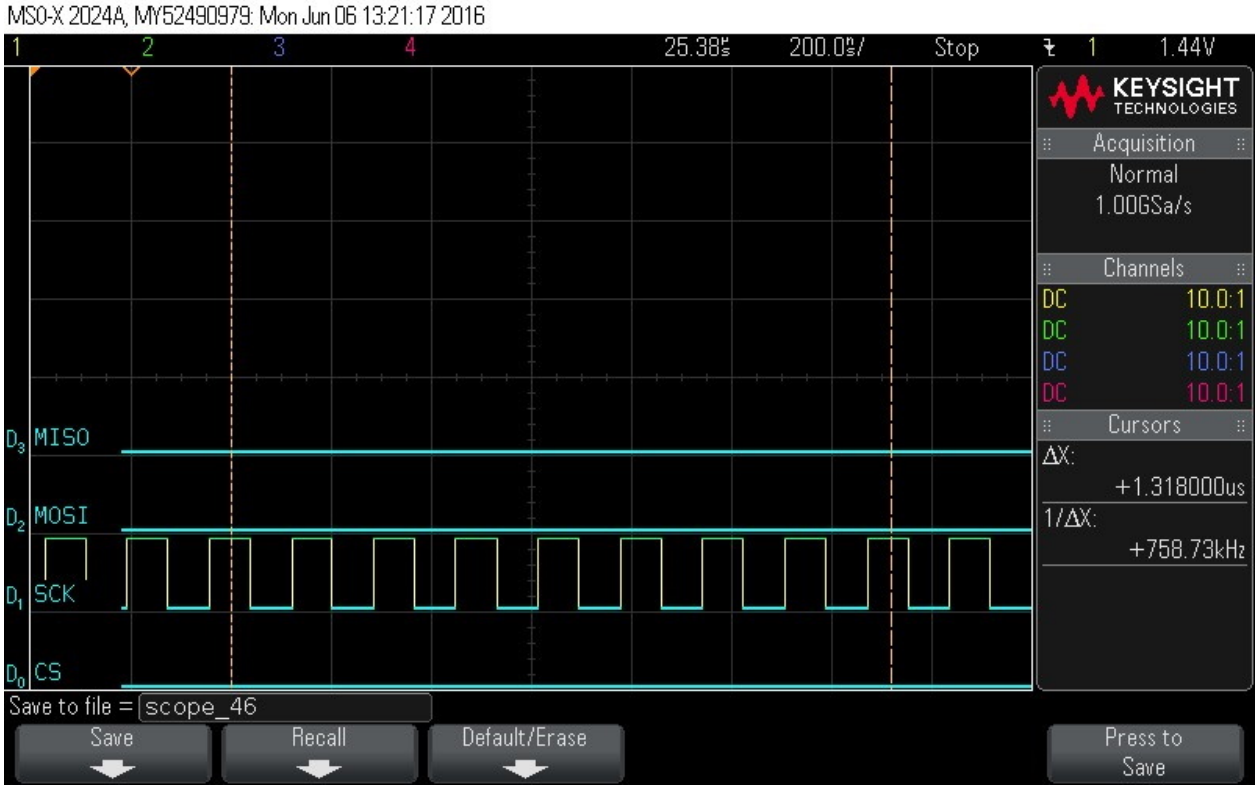
26. Data Byte 3 = 0x0

Figure 9-104. Data Byte 3 = 0x0



27. Data Byte 4 = 0x0

Figure 9-105. Data Byte 4 = 0x0

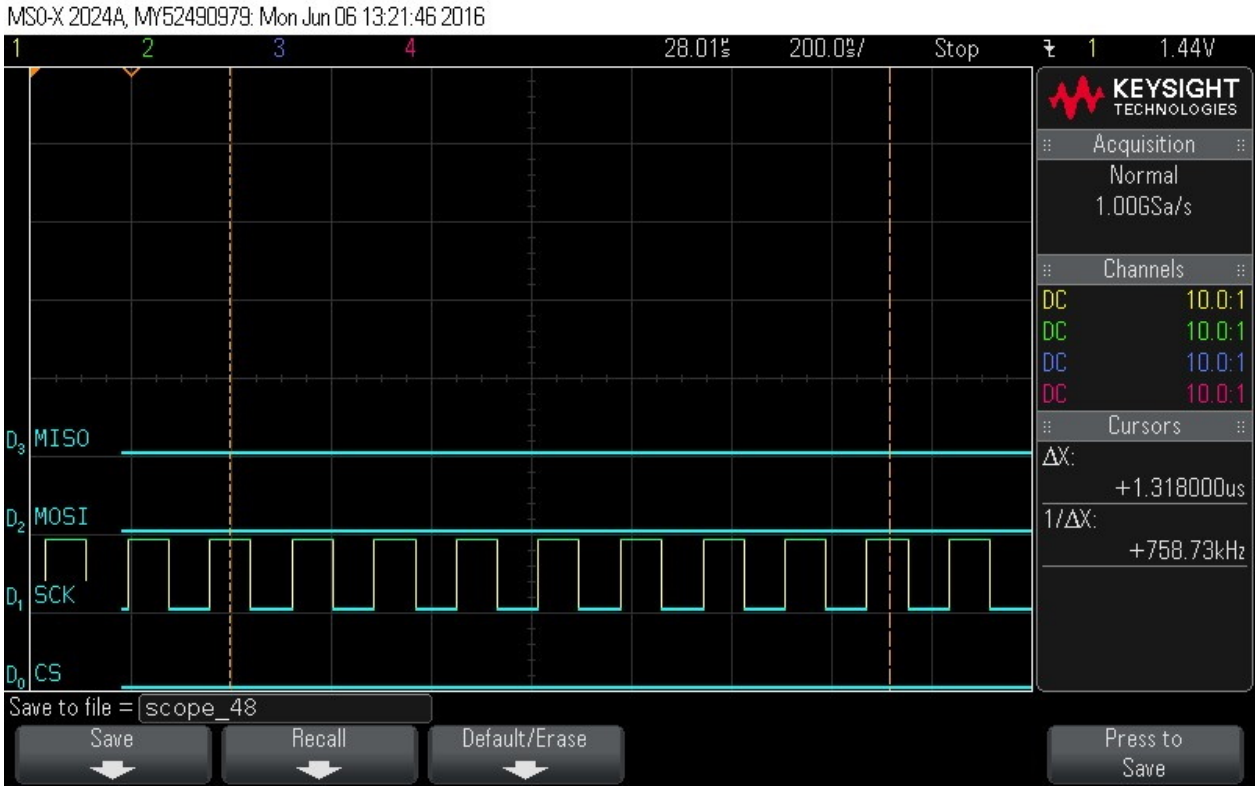


28. Data Byte 5 = 0x0

Figure 9-106. Data Byte 5 = 0x0

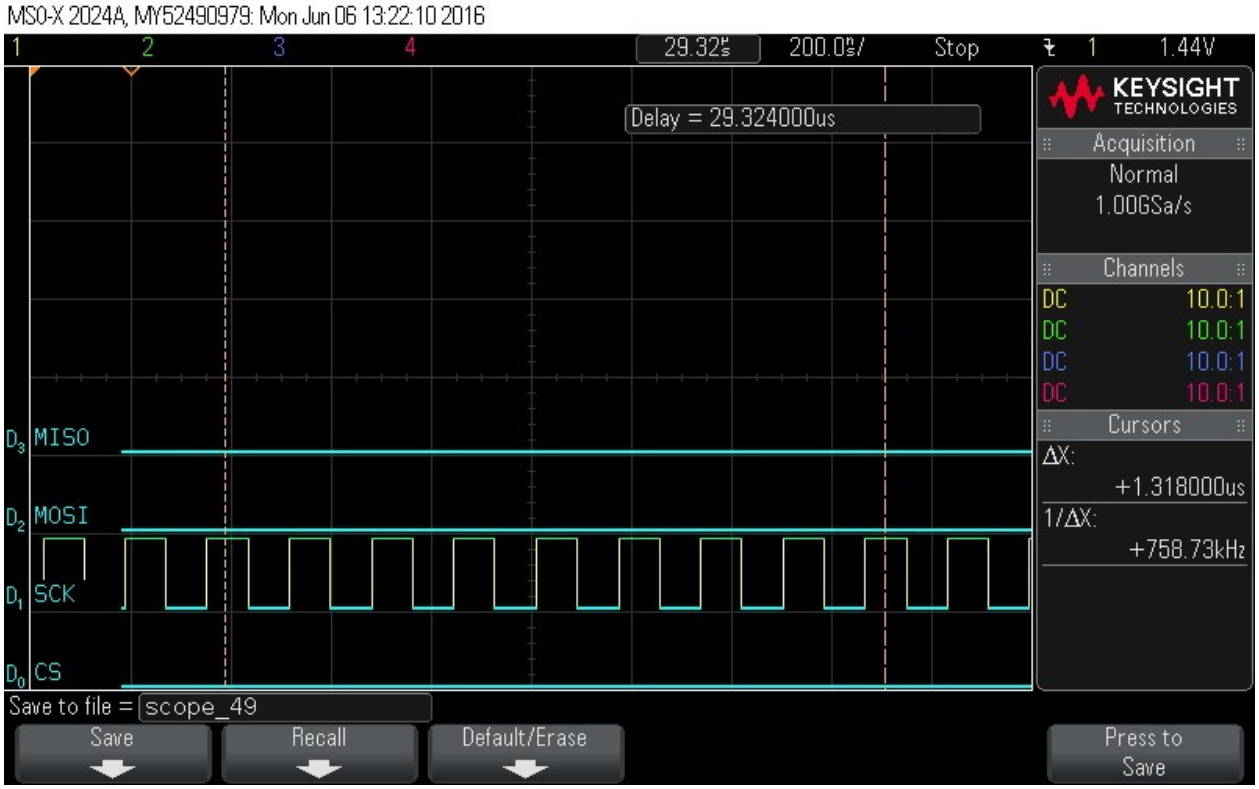
29. Data Byte 6 = 0x0

Figure 9-107. Data Byte 6 = 0x0



30. Data Byte 7 = 0x0

Figure 9-108. Data Byte 7 = 0x0

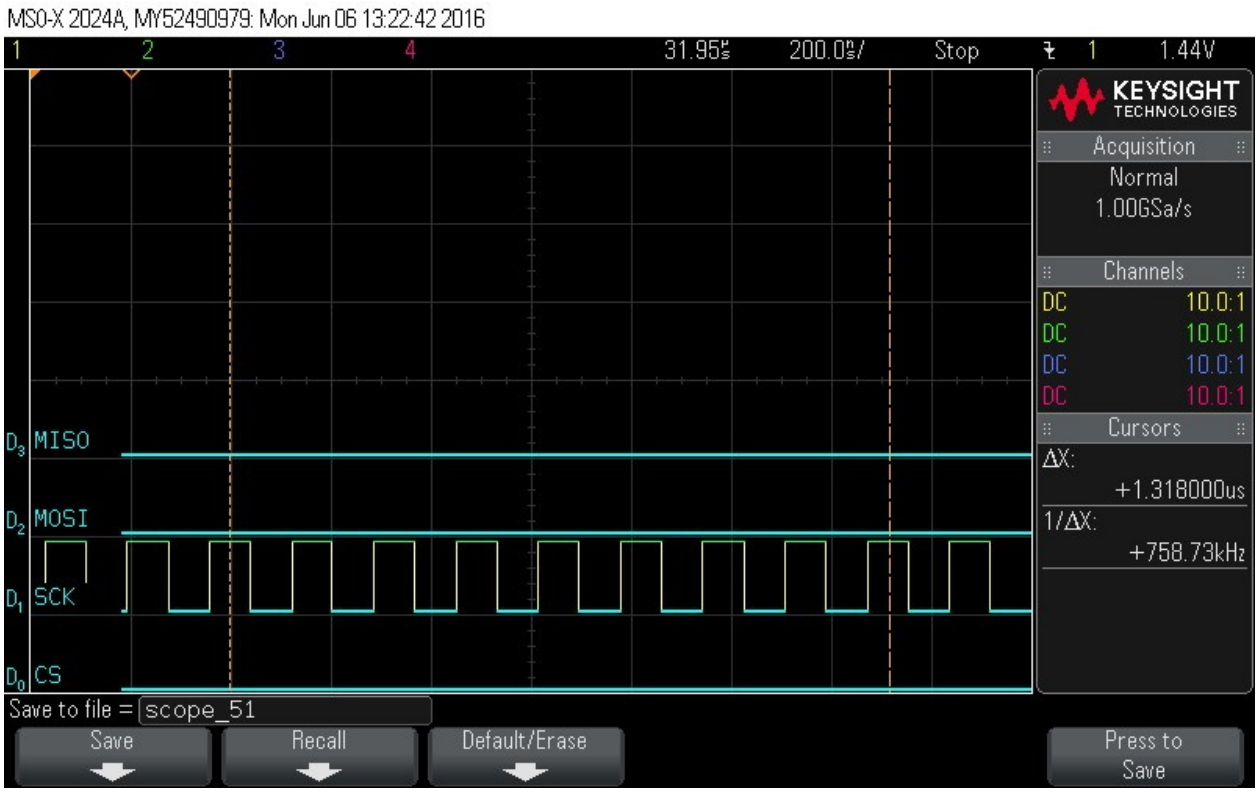


31. Data Byte 8 = 0x0

Figure 9-109. Data Byte 8 = 0x0

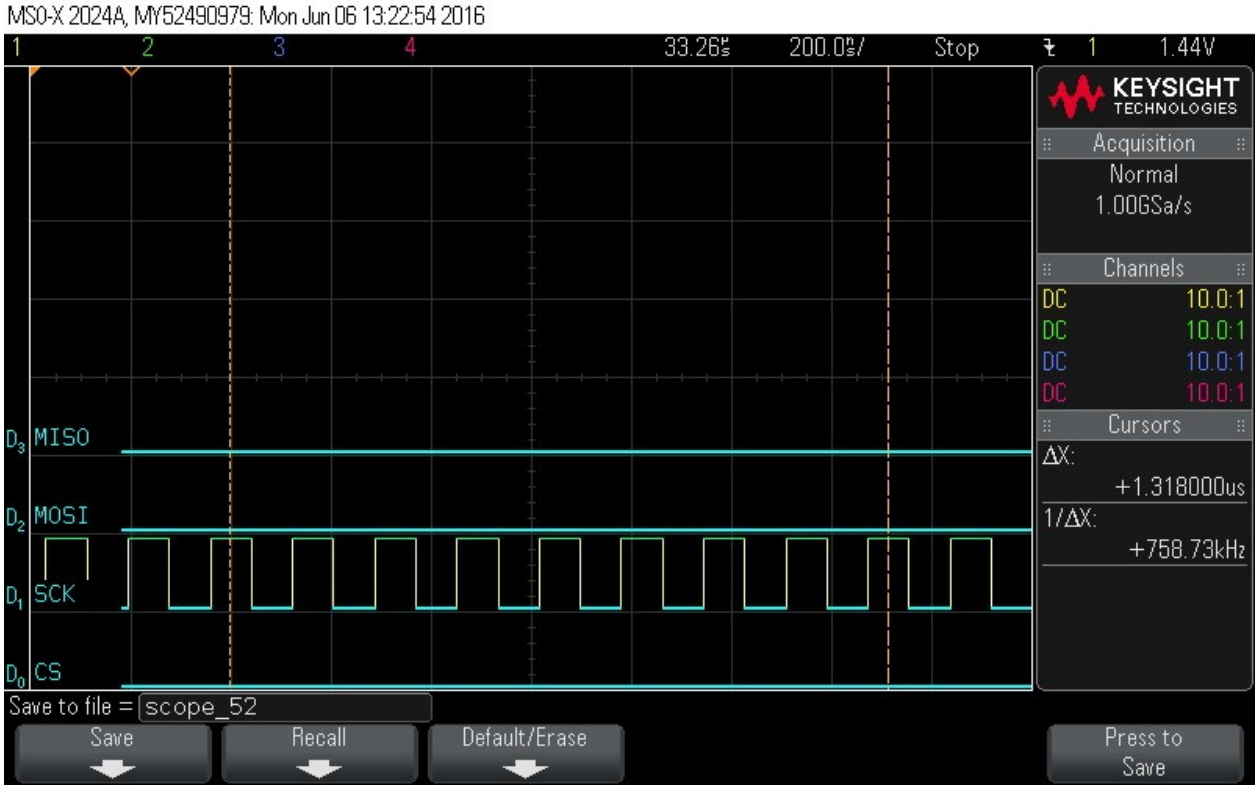
32. Data Byte 9 = 0x0

Figure 9-110. Data Byte 9 = 0x0



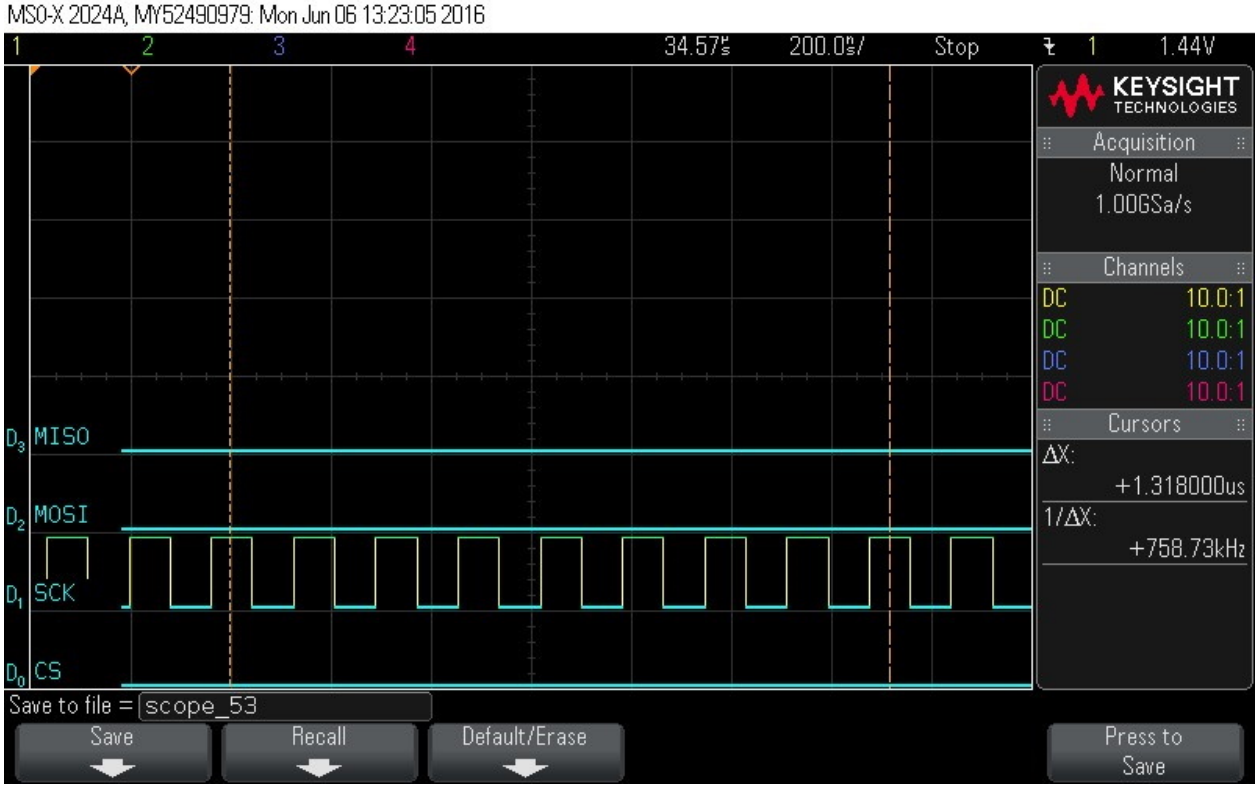
33. Data Byte 10 = 0x0

Figure 9-111. Data Byte 10 = 0x0



34. Data Byte 11 = 0x0

Figure 9-112. Data Byte 11 = 0x0

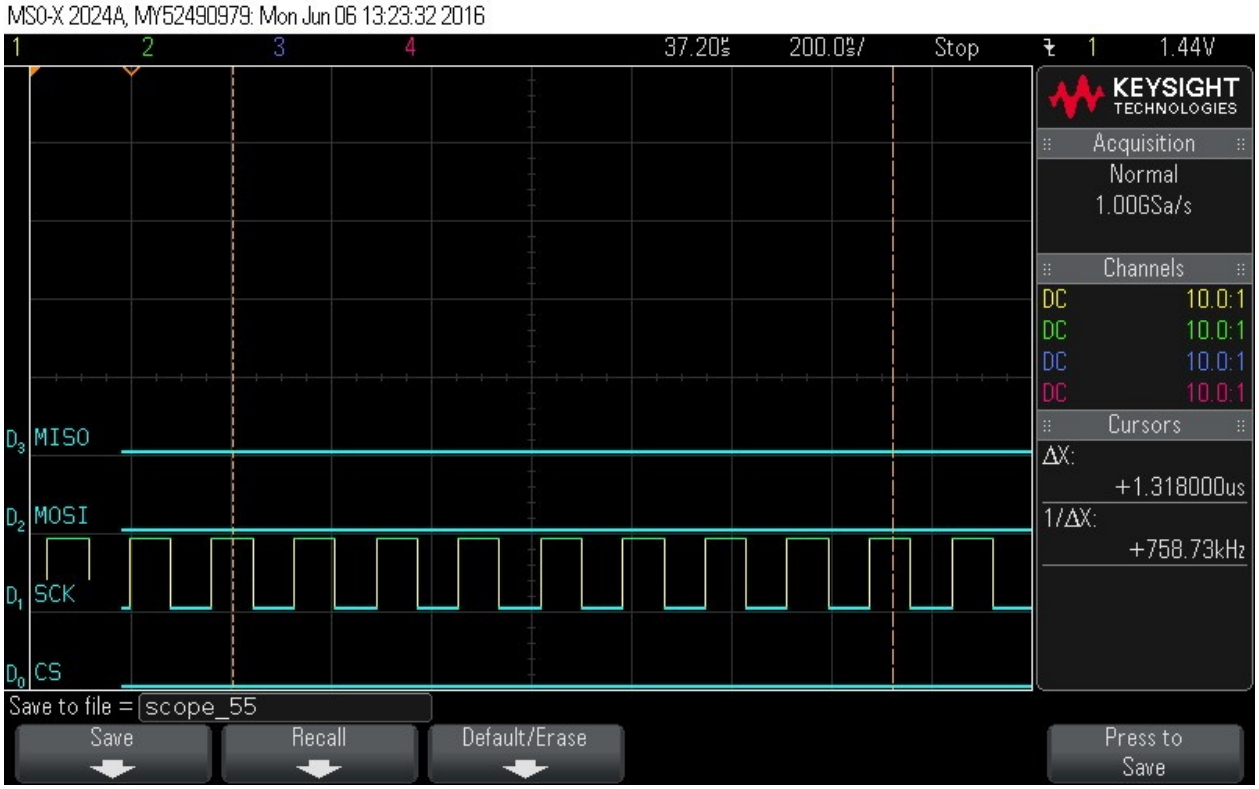


35. Data Byte 12 = 0x0

Figure 9-113. Data Byte 12 = 0x0

36. Data Byte 13 = 0x0

Figure 9-114. Data Byte 13 = 0x0

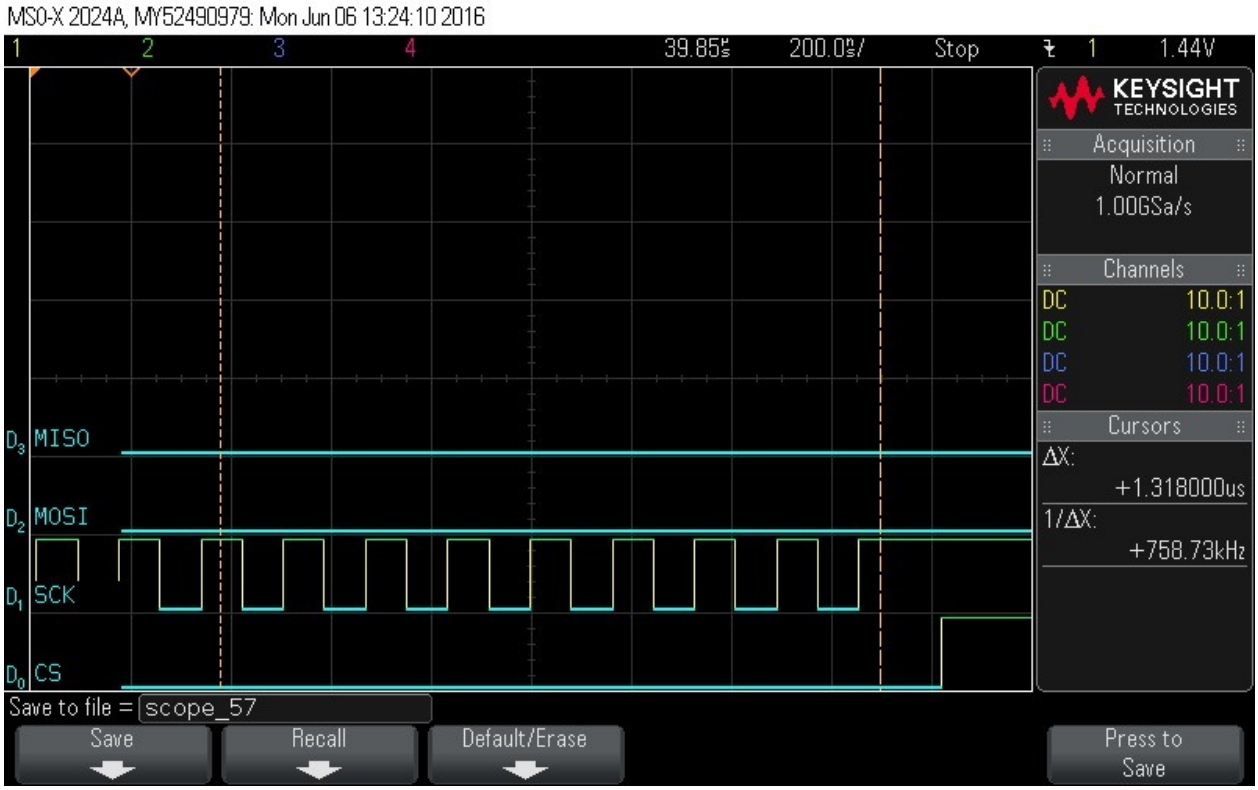


37. Data Byte 14 = 0x0

Figure 9-115. Data Byte 14 = 0x0

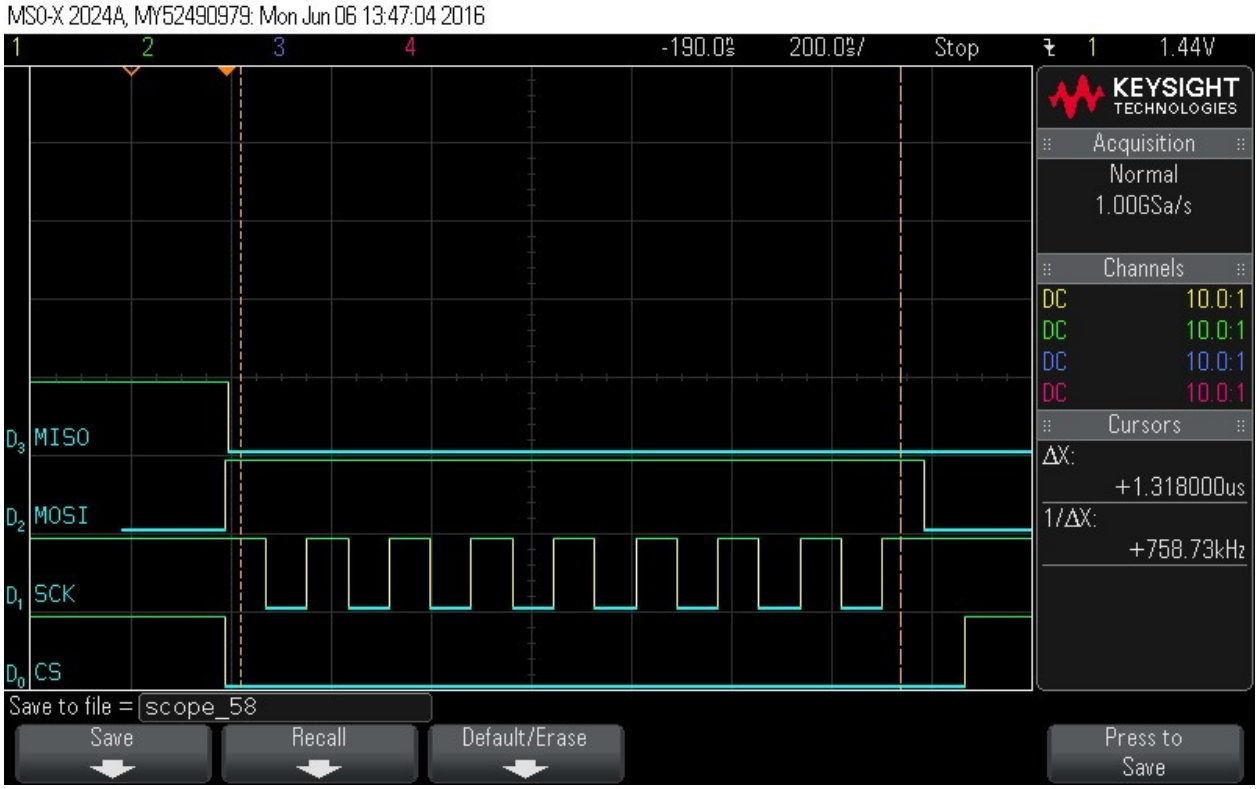
38. Data Byte 15 = 0x0

Figure 9-116. Data Byte 15 = 0x0



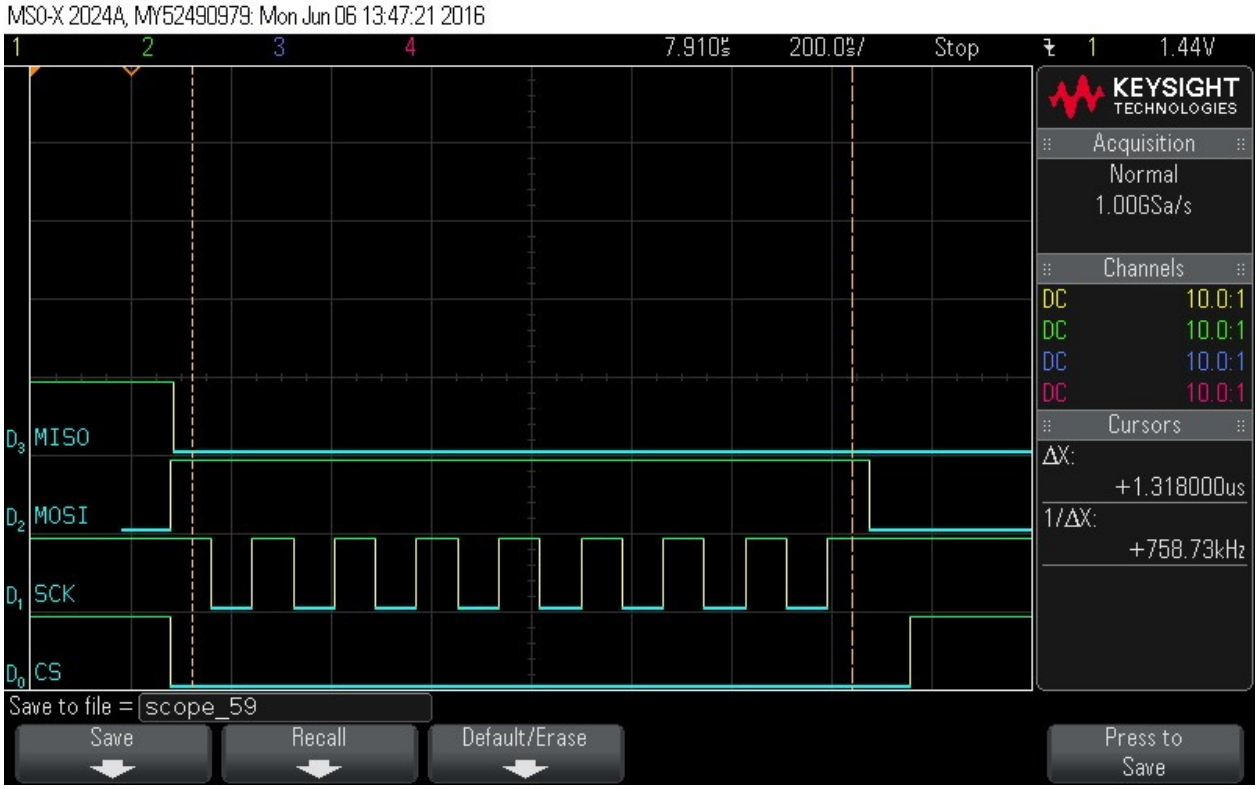
39. Instruction is loaded. Issue read instruction using 0x5 command.

Figure 9-117. Hardware Status Check



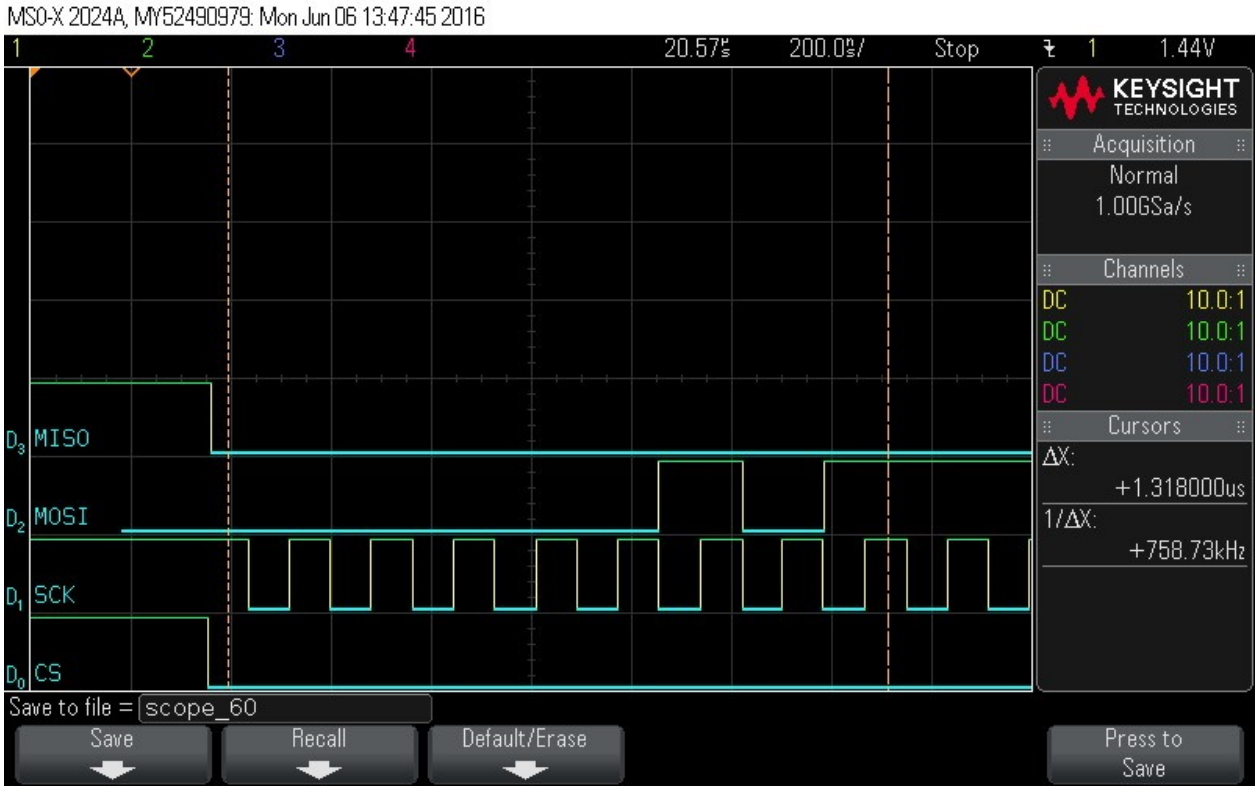
40. Checking hardware status.

Figure 9-118. Hardware Status Check



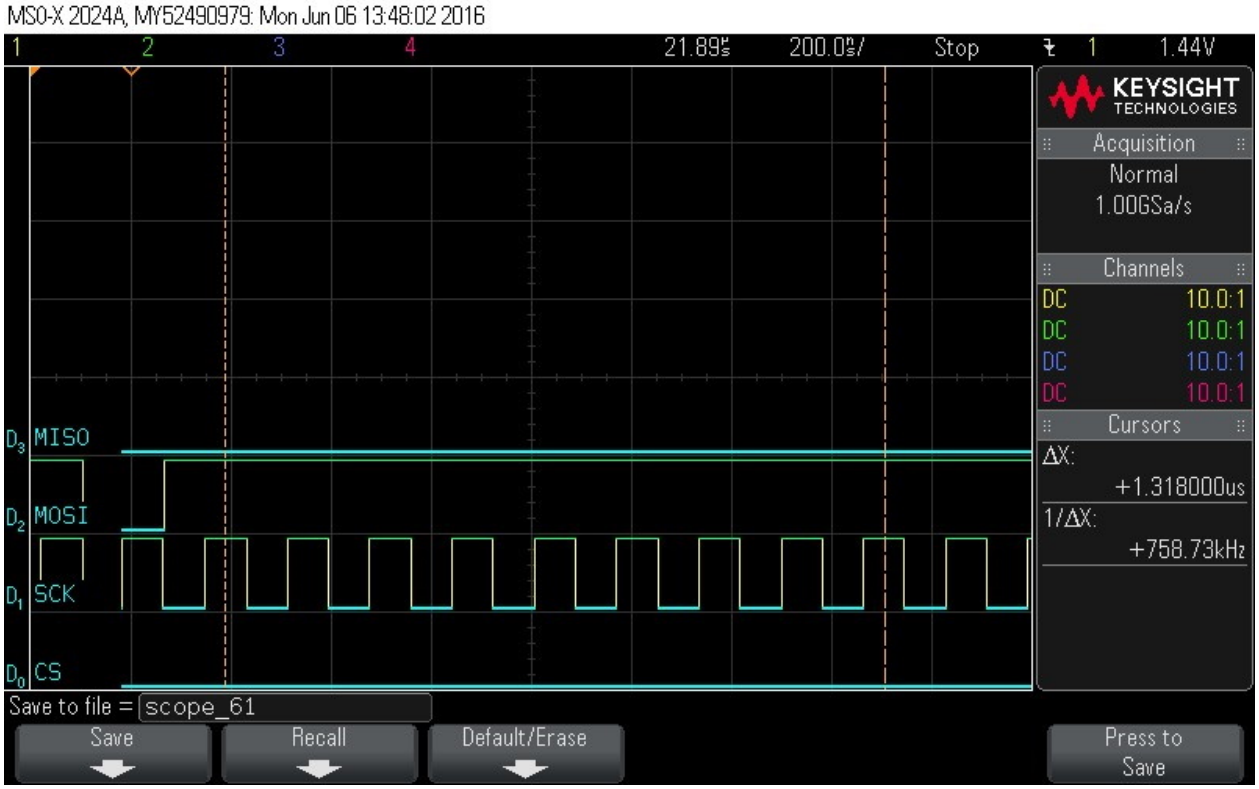
41. Reading command. Command = 0x5. Note CS signal.

Figure 9-119. Read Command. Command = 0x5. Note CS signal



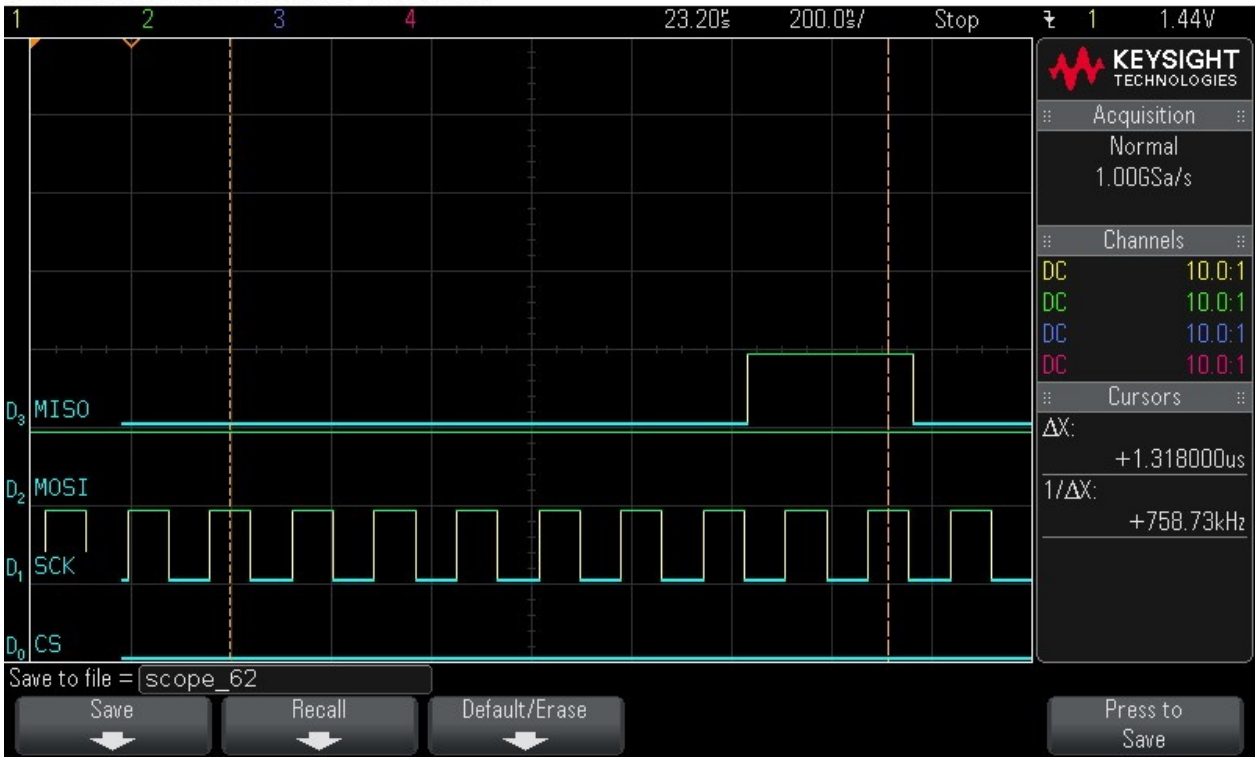
42. Reading Data Byte 0.

Figure 9-120. Data Byte 0 read



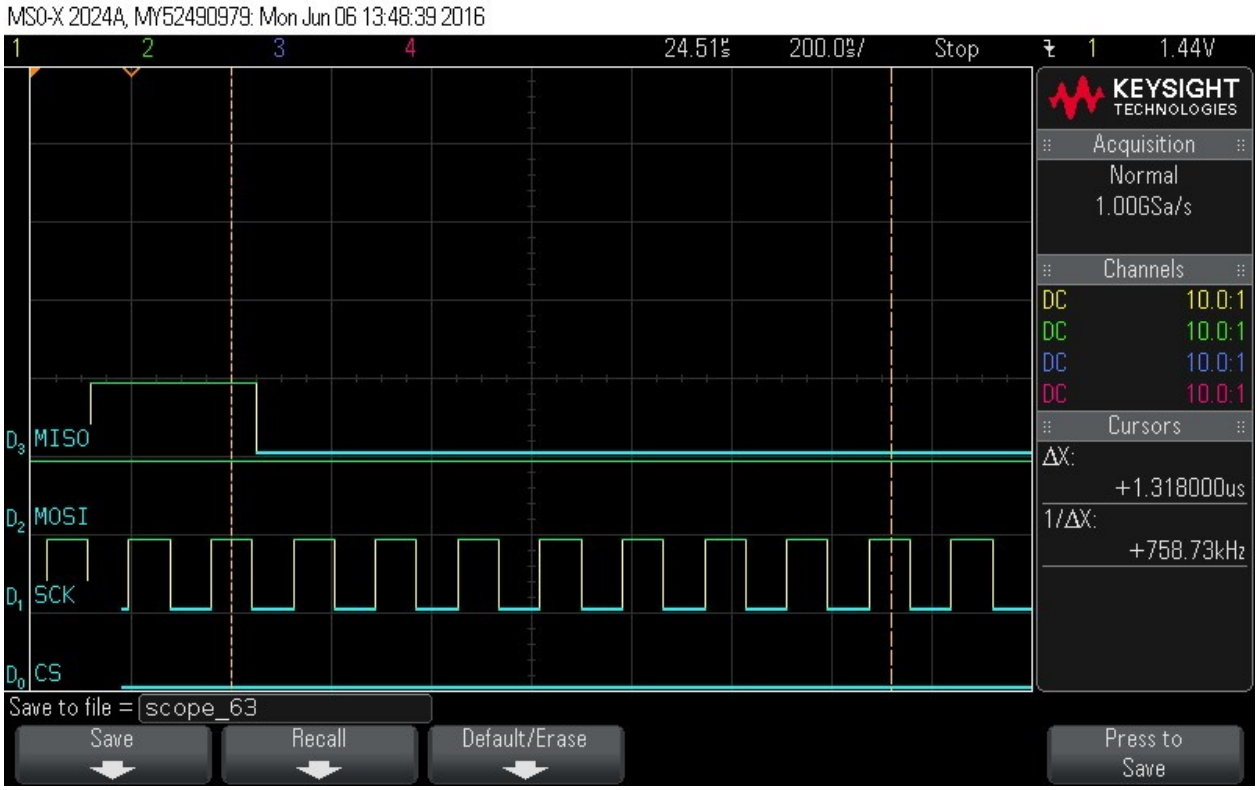
43. Reading Data Byte 1.

Figure 9-121. Data Byte 1 read



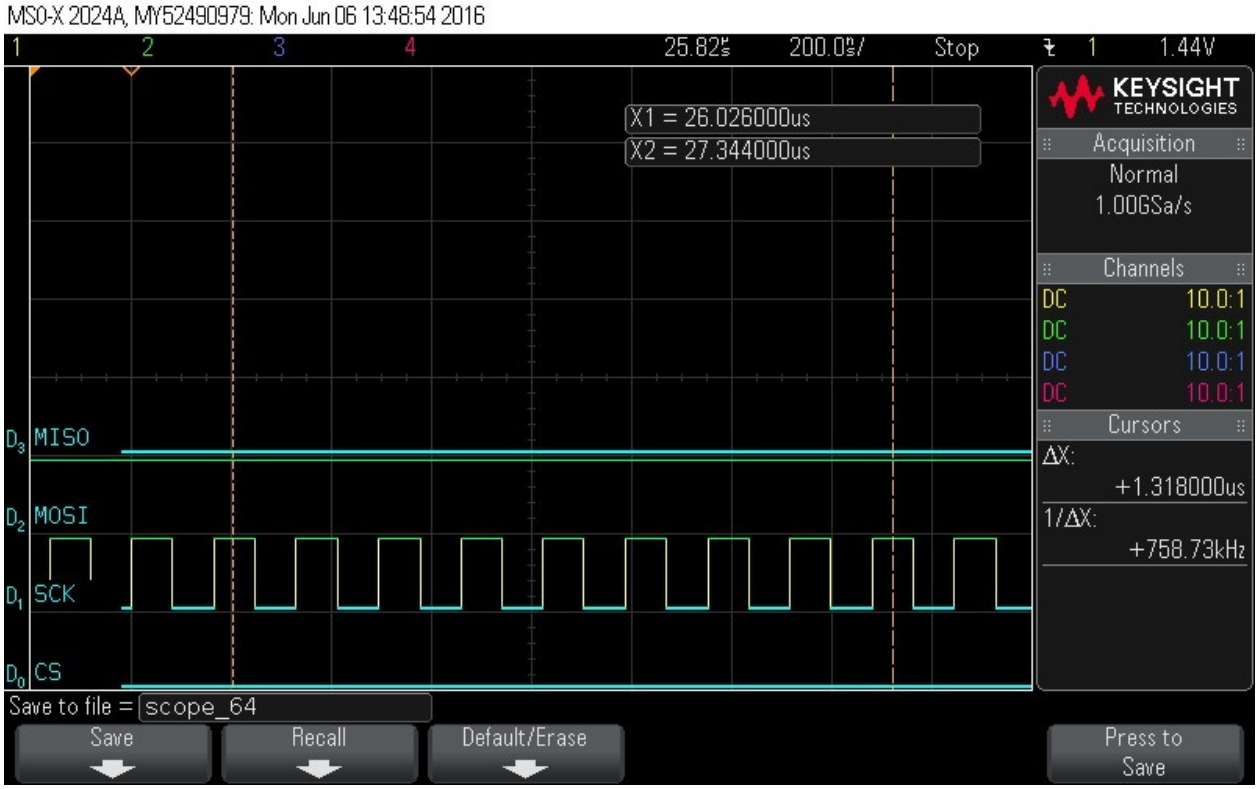
44. Reading Data Byte 2.

Figure 9-122. Data Byte 2 read



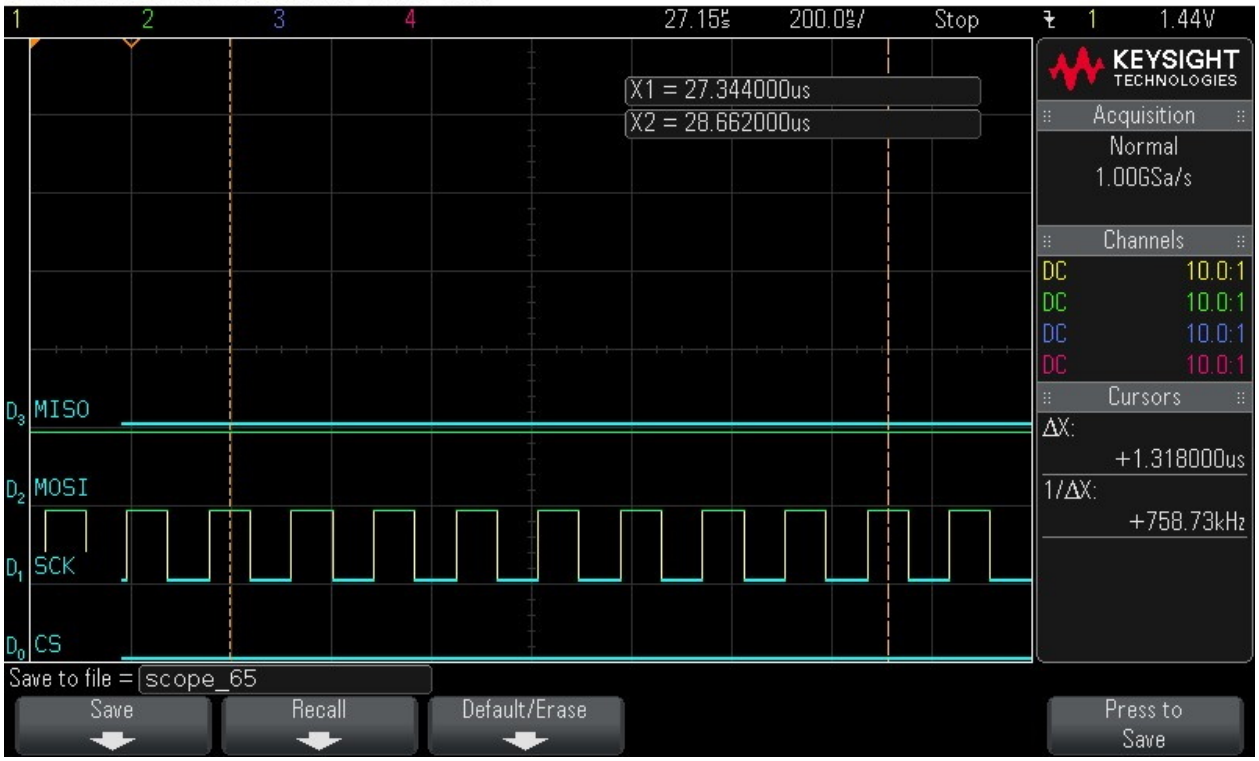
45. Reading Data Byte 3.

Figure 9-123. Data Byte 3 Read



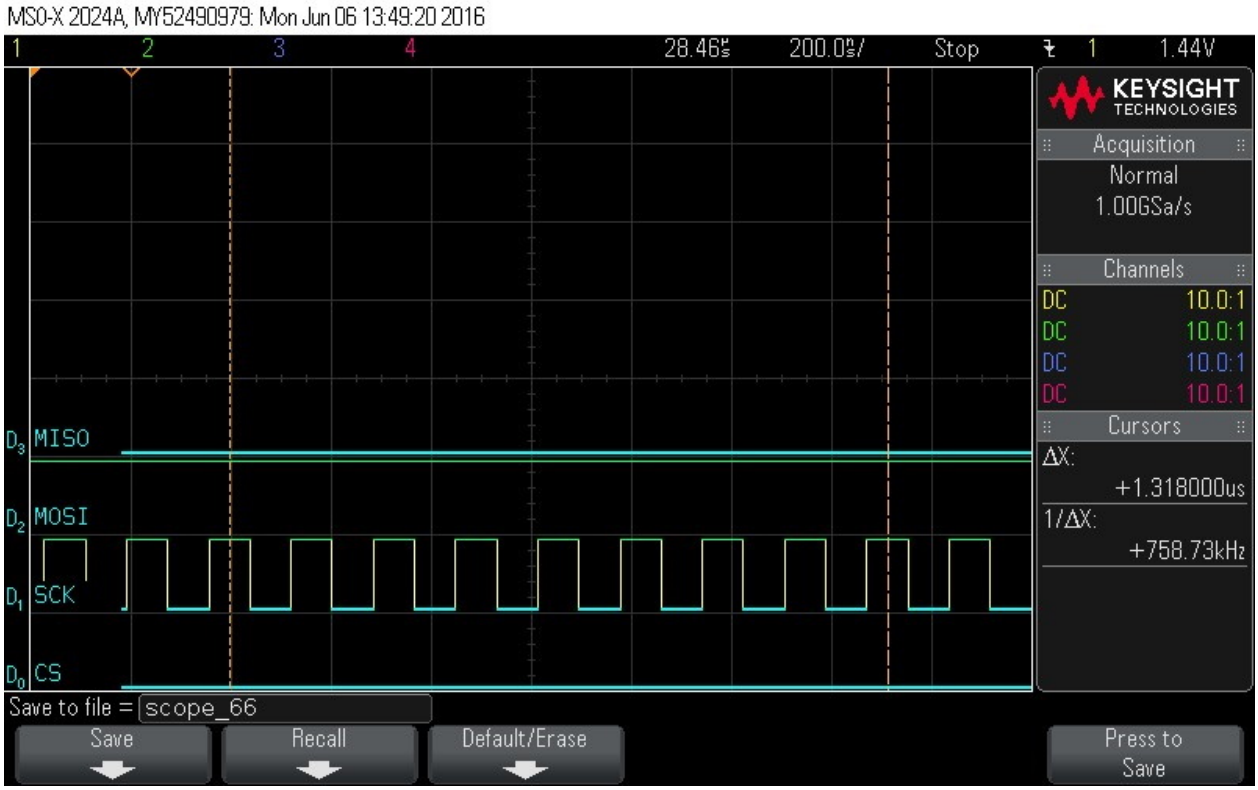
46. Reading Data Byte 4.

Figure 9-124. Data Byte 4 Read



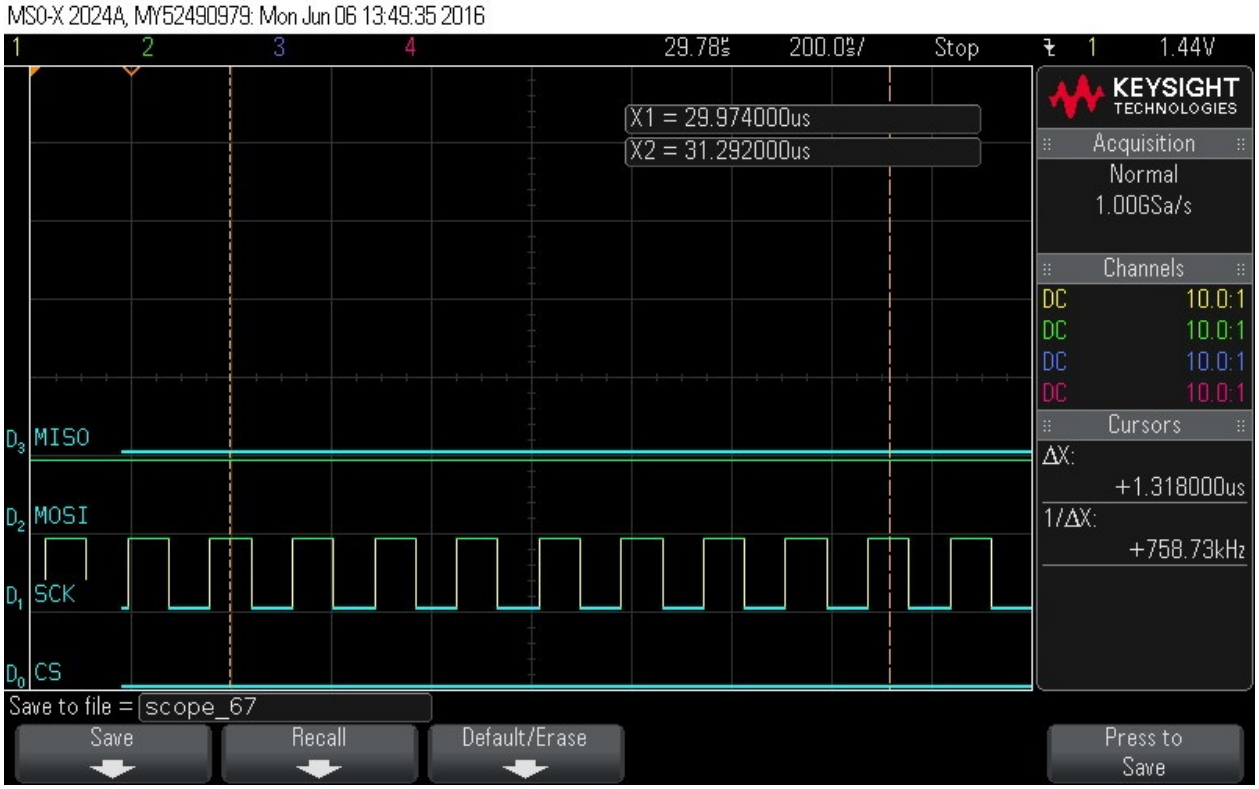
47. Reading Data Byte 5.

Figure 9-125. Data Byte 5 Read



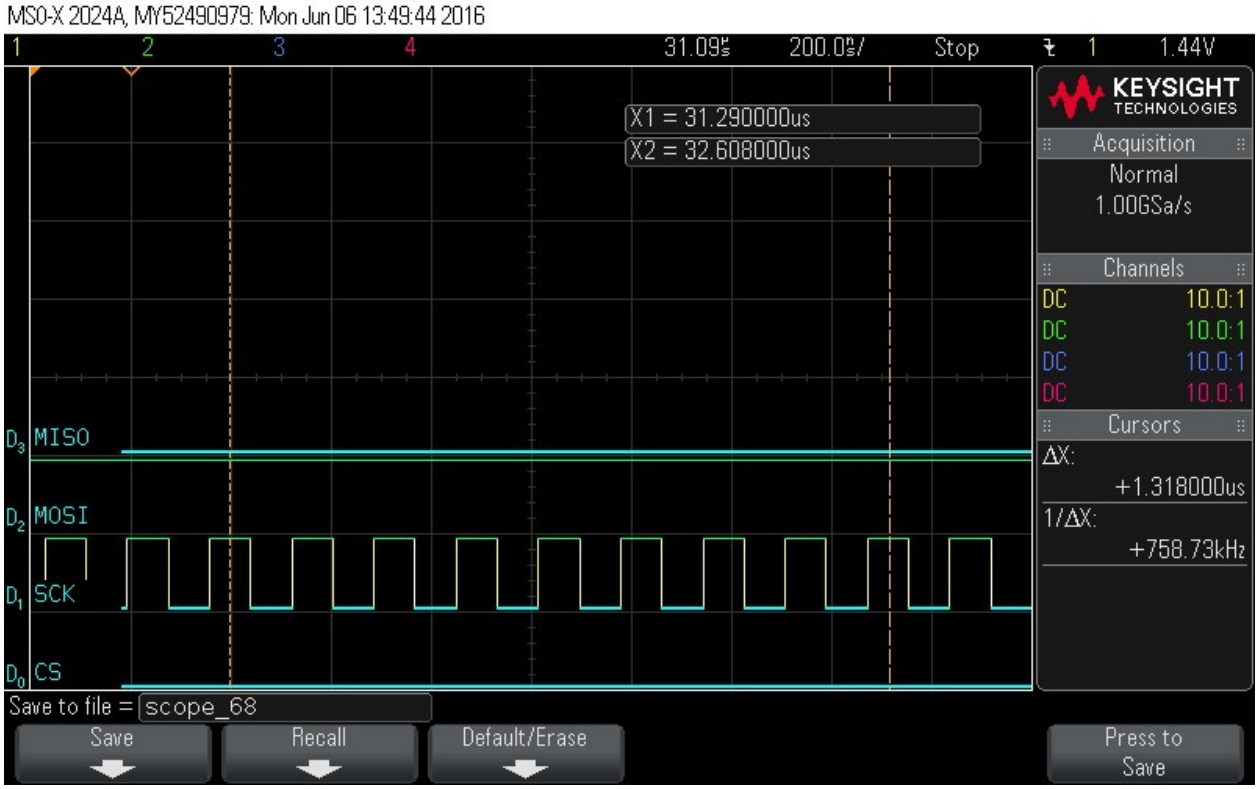
48. Reading Data Byte 6.

Figure 9-126. Data Byte 6 Read



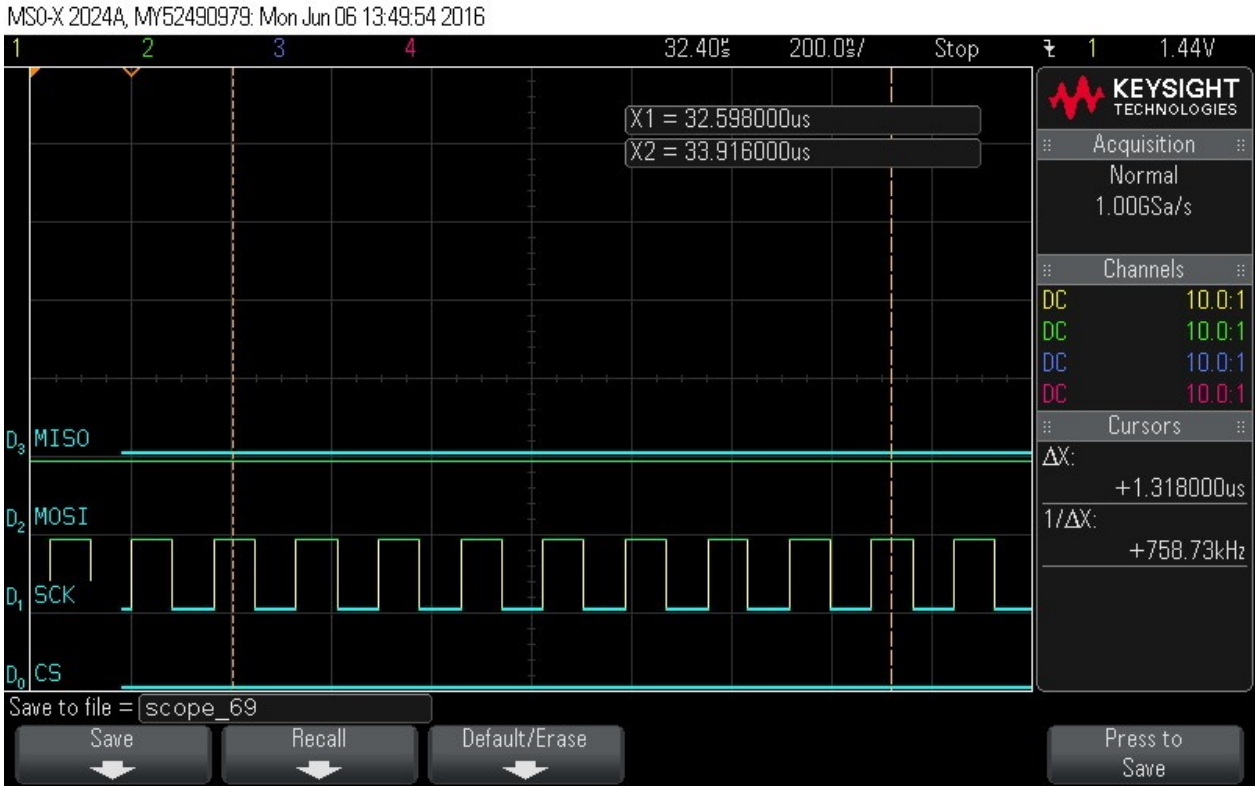
49. Reading Data Byte 7.

Figure 9-127. Data Byte7 Read



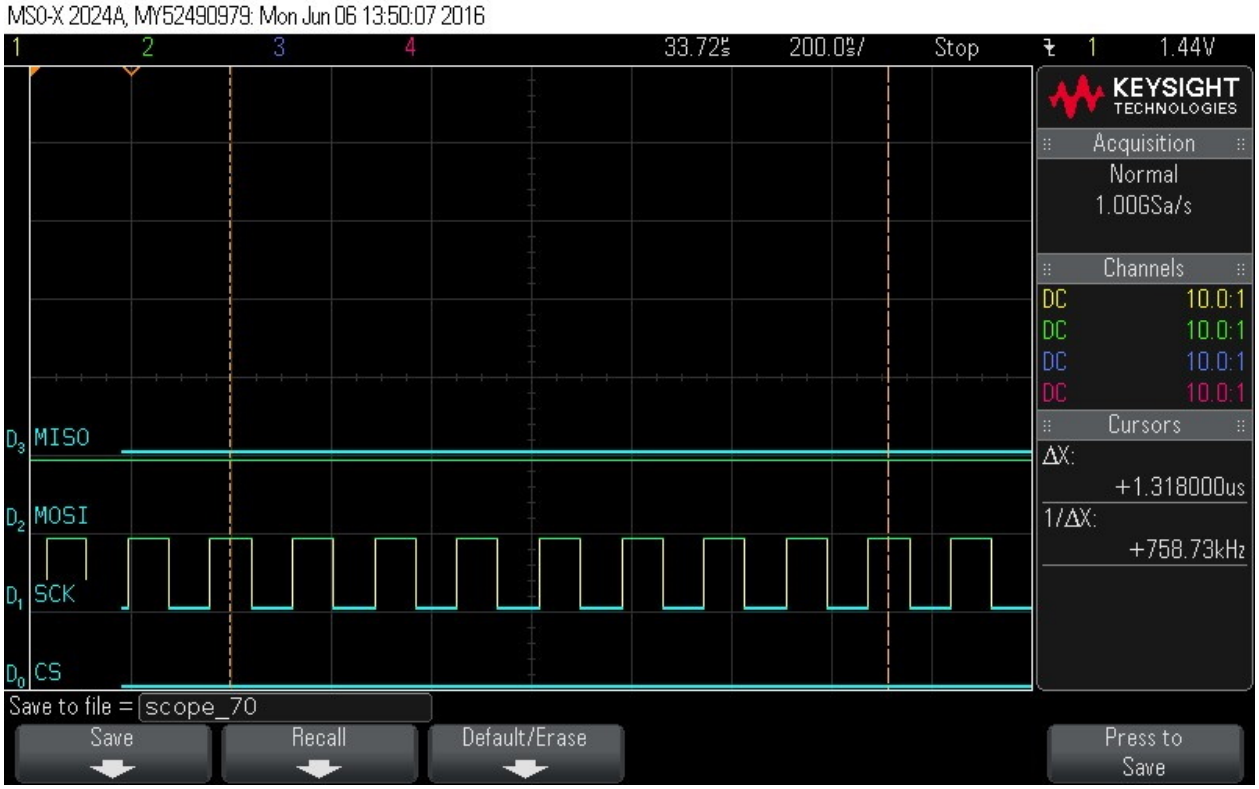
50. Reading Data Byte 8.

Figure 9-128. Data Byte 8 Read



51. Reading Data Byte 9.

Figure 9-129. Data Byte 9 Read

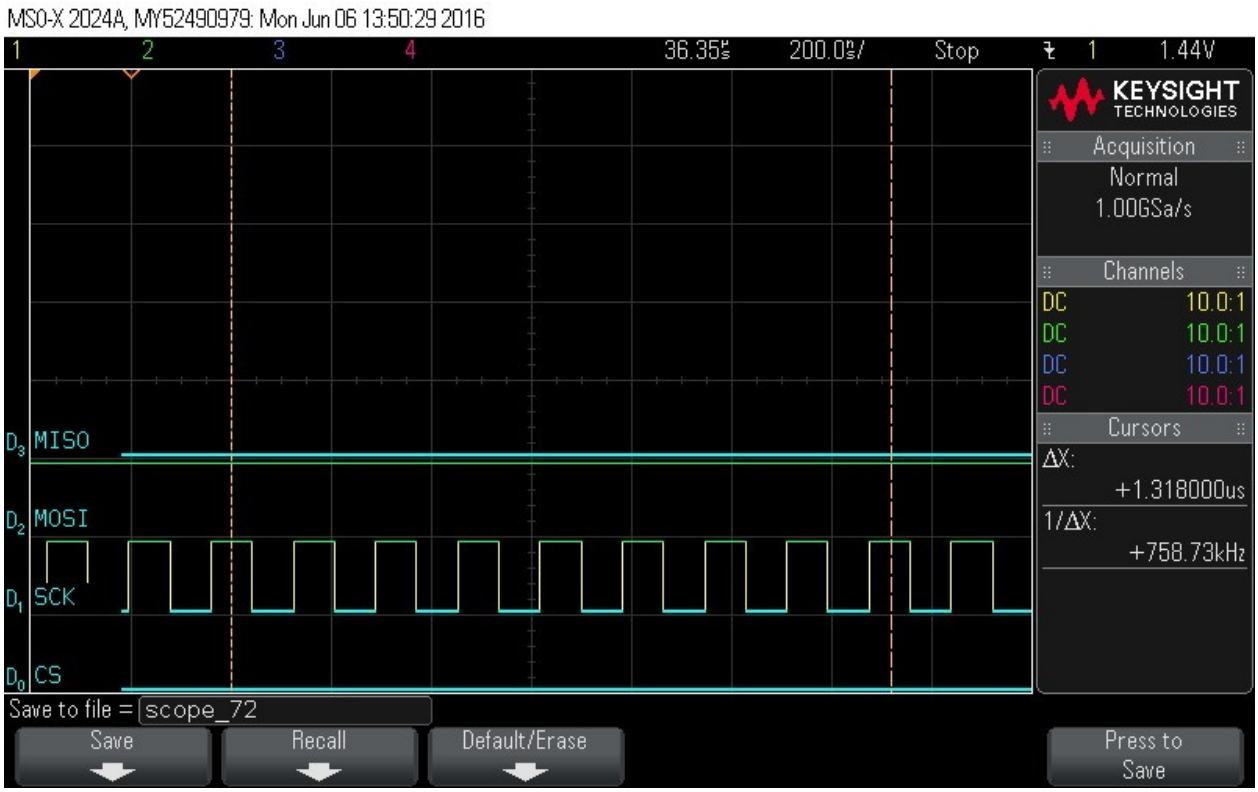


52. Reading Data Byte 10.

Figure 9-130. Data Byte 10 Read

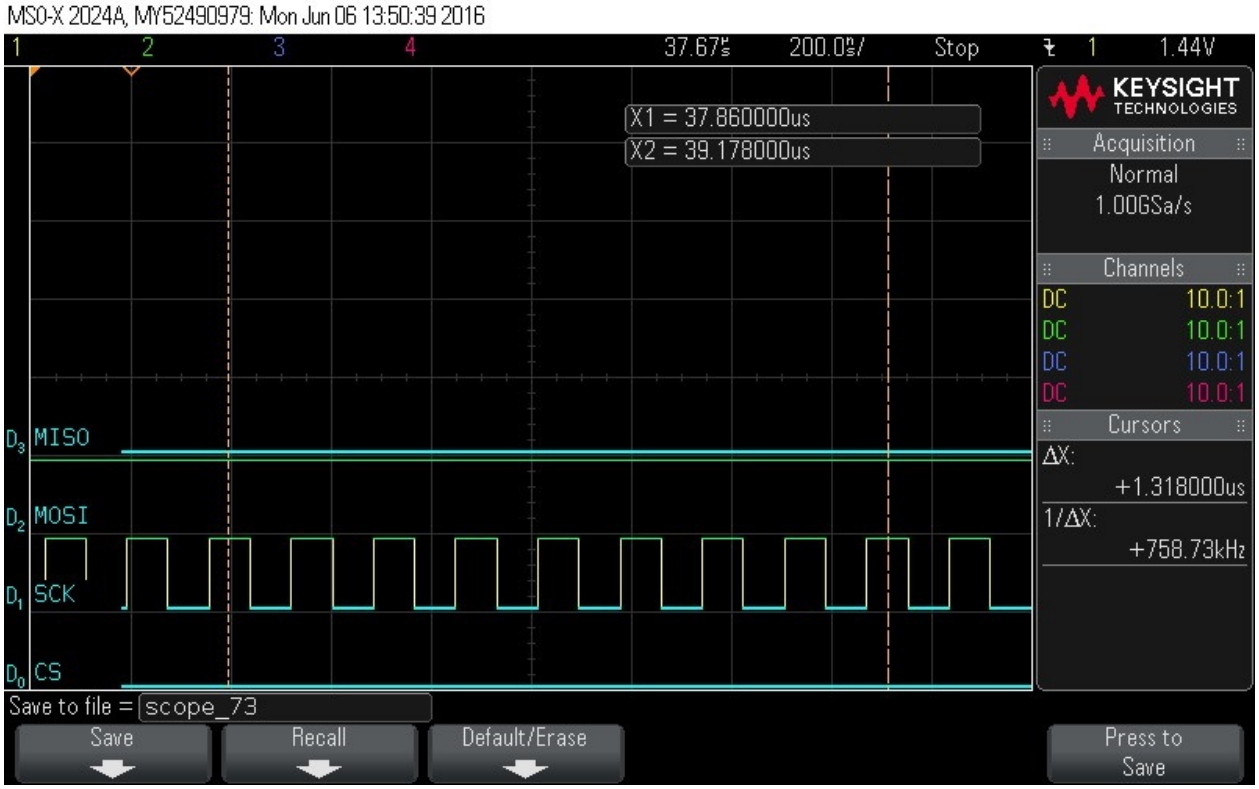
53. Reading Data Byte 11 read

Figure 9-131. Data Byte 11 Read



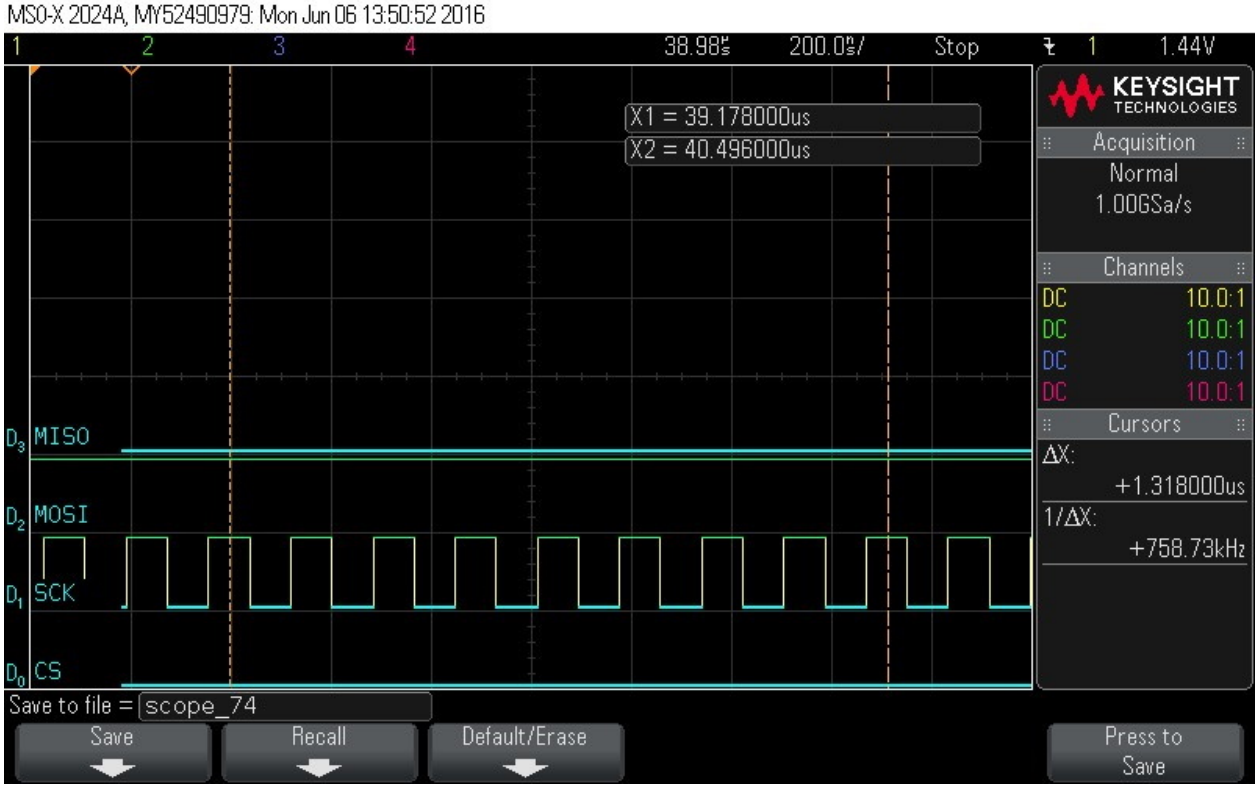
54. Reading Data Byte 12.

Figure 9-132. Data Byte 12 Read



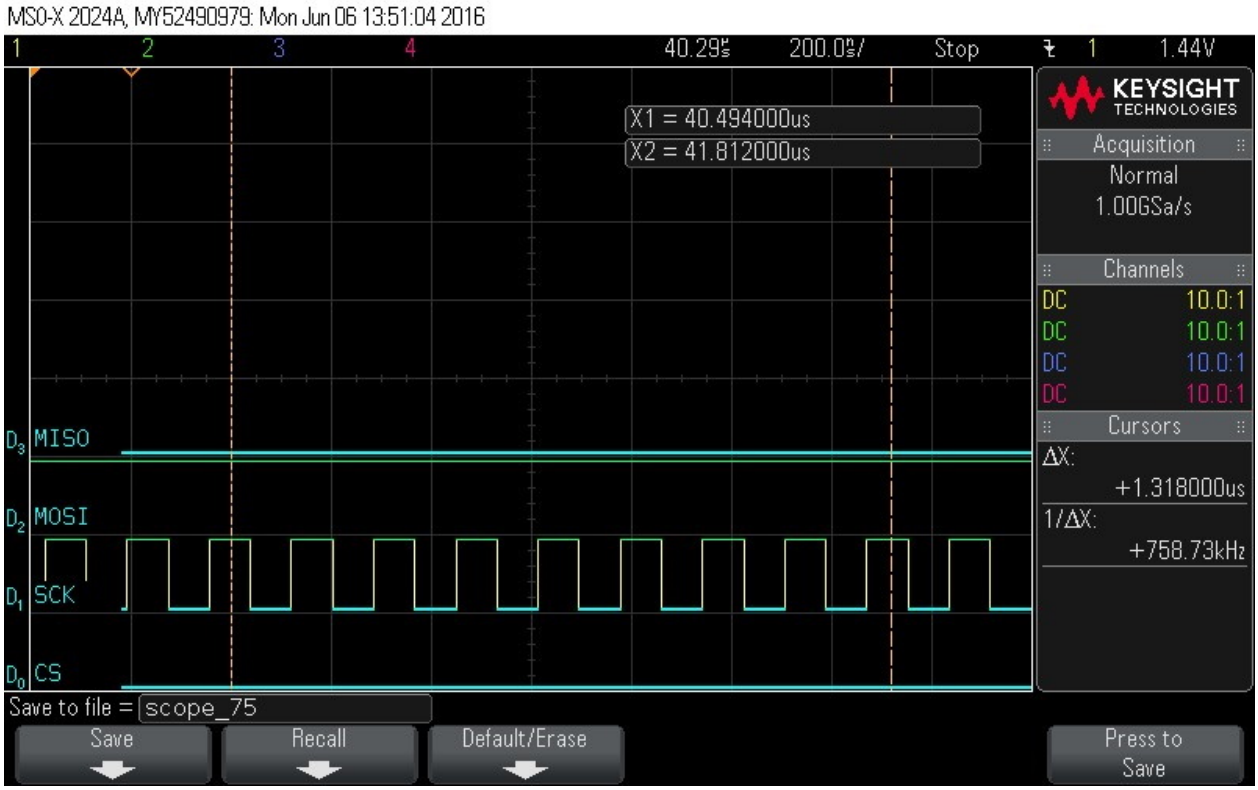
55. Reading Data Byte 13.

Figure 9-133. Reading Data Byte 13



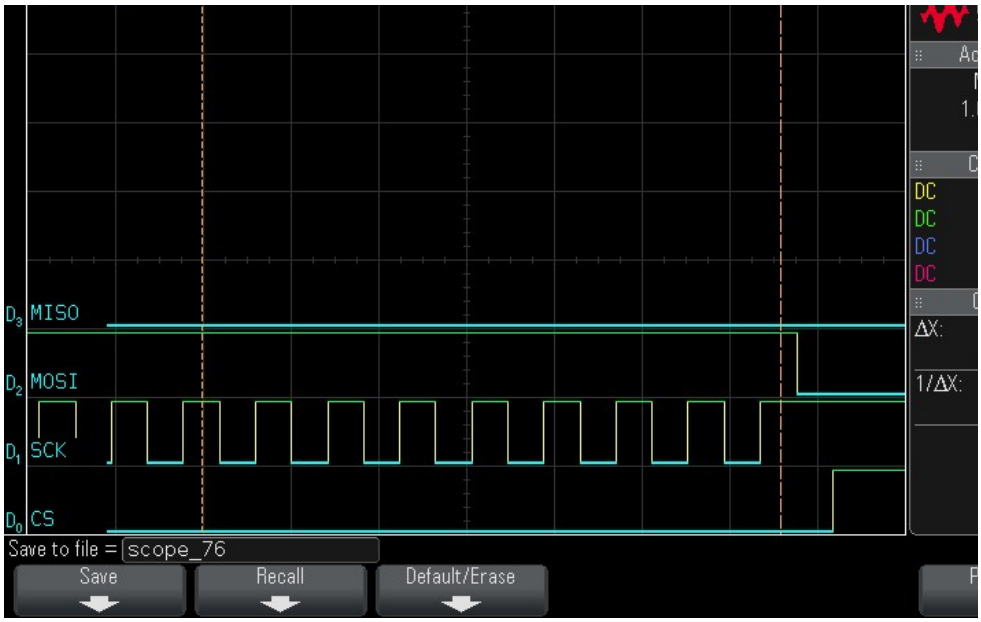
56. Reading Data Byte 14.

Figure 9-134. Data Byte 14 Read



57. Reading Data Byte 15.

Figure 9-135. Data Byte 15 Read



10. Revision History

Revision	Date	Description
A	09/2021	• Migrated to the Microchip standard template format. • Updated for the new version numbering schema (v202x.x) for DirectC solutions.

Microchip FPGA Support

Microchip FPGA products group backs its products with various support services, including Customer Service, Customer Technical Support Center, a website, and worldwide sales offices. Customers are suggested to visit Microchip online resources prior to contacting support as it is very likely that their queries have been already answered.

Contact Technical Support Center through the website at www.microchip.com/support. Mention the FPGA Device Part number, select appropriate case category, and upload design files while creating a technical support case.

Contact Customer Service for non-technical product support, such as product pricing, product upgrades, update information, order status, and authorization.

- From North America, call **800.262.1060**
- From the rest of the world, call **650.318.4460**
- Fax, from anywhere in the world, **650.318.8044**

The Microchip Website

Microchip provides online support via our website at www.microchip.com/. This website is used to make files and information easily available to customers. Some of the content available includes:

- **Product Support** – Data sheets and errata, application notes and sample programs, design resources, user's guides and hardware support documents, latest software releases and archived software
- **General Technical Support** – Frequently Asked Questions (FAQs), technical support requests, online discussion groups, Microchip design partner program member listing
- **Business of Microchip** – Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives

Product Change Notification Service

Microchip's product change notification service helps keep customers current on Microchip products. Subscribers will receive email notification whenever there are changes, updates, revisions or errata related to a specified product family or development tool of interest.

To register, go to www.microchip.com/pcn and follow the registration instructions.

Customer Support

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Embedded Solutions Engineer (ESE)
- Technical Support

Customers should contact their distributor, representative or ESE for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in this document.

Technical support is available through the website at: www.microchip.com/support

Microchip Devices Code Protection Feature

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specifications contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is secure when used in the intended manner and under normal conditions.

- There are dishonest and possibly illegal methods being used in attempts to breach the code protection features of the Microchip devices. We believe that these methods require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Attempts to breach these code protection features, most likely, cannot be accomplished without violating Microchip's intellectual property rights.
- Microchip is willing to work with any customer who is concerned about the integrity of its code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of its code. Code protection does not mean that we are guaranteeing the product is "unbreakable." Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Legal Notice

Trademarks

ISBN: 978-1-5224-8876-7

Quality Management System

For information regarding Microchip's Quality Management Systems, please visit www.microchip.com/quality.

Worldwide Sales and Service

AMERICAS	ASIA/PACIFIC	ASIA/PACIFIC	EUROPE
Corporate Office 2355 West Chandler Blvd. Chandler, AZ 85224-6199 Tel: 480-792-7200 Fax: 480-792-7277 Technical Support: www.microchip.com/support Web Address: www.microchip.com **Atlanta** Duluth, GA Tel: 678-957-9614 Fax: 678-957-1455 **Austin, TX** Tel: 512-257-3370 **Boston** Westborough, MA Tel: 774-760-0087 Fax: 774-760-0088 **Chicago** Itasca, IL Tel: 630-285-0071 Fax: 630-285-0075 **Dallas** Addison, TX Tel: 972-818-7423 Fax: 972-818-2924 **Detroit** Novi, MI Tel: 248-848-4000 **Houston, TX** Tel: 281-894-5983 **Indianapolis** Noblesville, IN Tel: 317-773-8323 Fax: 317-773-5453 Tel: 317-536-2380 **Los Angeles** Mission Viejo, CA Tel: 949-462-9523 Fax: 949-462-9608 Tel: 951-273-7800 **Raleigh, NC** Tel: 919-844-7510 **New York, NY** Tel: 631-435-6000 **San Jose, CA** Tel: 408-735-9110 Tel: 408-436-4270 **Canada - Toronto** Tel: 905-695-1980 Fax: 905-695-2078	**Australia - Sydney** Tel: 61-2-9868-6733 **China - Beijing** Tel: 86-10-8569-7000 **China - Chengdu** Tel: 86-28-8665-5511 **China - Chongqing** Tel: 86-23-8980-9588 **China - Dongguan** Tel: 86-769-8702-9880 **China - Guangzhou** Tel: 86-20-8755-8029 **China - Hangzhou** Tel: 86-571-8792-8115 **China - Hong Kong SAR** Tel: 852-2943-5100 **China - Nanjing** Tel: 86-25-8473-2460 **China - Qingdao** Tel: 86-532-8502-7355 **China - Shanghai** Tel: 86-21-3326-8000 **China - Shenyang** Tel: 86-24-2334-2829 **China - Shenzhen** Tel: 86-755-8864-2200 **China - Suzhou** Tel: 86-186-6233-1526 **China - Wuhan** Tel: 86-27-5980-5300 **China - Xian** Tel: 86-29-8833-7252 **China - Xiamen** Tel: 86-592-2388138 **China - Zhuhai** Tel: 86-756-3210040	**India - Bangalore** Tel: 91-80-3090-4444 **India - New Delhi** Tel: 91-11-4160-8631 **India - Pune** Tel: 91-20-4121-0141 **Japan - Osaka** Tel: 81-6-6152-7160 **Japan - Tokyo** Tel: 81-3-6880- 3770 **Korea - Daegu** Tel: 82-53-744-4301 **Korea - Seoul** Tel: 82-2-554-7200 **Malaysia - Kuala Lumpur** Tel: 60-3-7651-7906 **Malaysia - Penang** Tel: 60-4-227-8870 **Philippines - Manila** Tel: 63-2-634-9065 **Singapore** Tel: 65-6334-8870 **Taiwan - Hsin Chu** Tel: 886-3-577-8366 **Taiwan - Kaohsiung** Tel: 886-7-213-7830 **Taiwan - Taipei** Tel: 886-2-2508-8600 **Thailand - Bangkok** Tel: 66-2-694-1351 **Vietnam - Ho Chi Minh** Tel: 84-28-5448-2100	**Austria - Wels** Tel: 43-7242-2244-39 Fax: 43-7242-2244-393 **Denmark - Copenhagen** Tel: 45-4485-5910 Fax: 45-4485-2829 **Finland - Espoo** Tel: 358-9-4520-820 **France - Paris** Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79 **Germany - Garching** Tel: 49-8931-9700 **Germany - Haan** Tel: 49-2129-3766400 **Germany - Heilbronn** Tel: 49-7131-72400 **Germany - Karlsruhe** Tel: 49-721-625370 **Germany - Munich** Tel: 49-89-627-144-0 Fax: 49-89-627-144-44 **Germany - Rosenheim** Tel: 49-8031-354-560 **Israel - Ra'anana** Tel: 972-9-744-7705 **Italy - Milan** Tel: 39-0331-742611 Fax: 39-0331-466781 **Italy - Padova** Tel: 39-049-7625286 **Netherlands - Drunen** Tel: 31-416-690399 Fax: 31-416-690340 **Norway - Trondheim** Tel: 47-72884388 **Poland - Warsaw** Tel: 48-22-3325737 **Romania - Bucharest** Tel: 40-21-407-87-50 **Spain - Madrid** Tel: 34-91-708-08-90 Fax: 34-91-708-08-91 **Sweden - Gothenberg** Tel: 46-31-704-60-40 **Sweden - Stockholm** Tel: 46-8-5090-4654 **UK - Wokingham** Tel: 44-118-921-5800 Fax: 44-118-921-5820