SPI-DirectC v2.1

User Guide

Microsemi
Power Matters.™

Table of Contents

Introduction

This document describes how to enable processor-based embedded ISP (In-System Programming) on Microsemi IGLOO2™, SmartFusion2™, and PolarFire™ devices using the SPI Slave programming method. In-System Programming refers to an external processor on board programming one of the IGLOO2, SmartFusion2, or PolarFire devices via SPI peripheral interface.

The document assumes that the target system contains a processor or a soft-core microprocessor with a minimum 1200 bytes of RAM, a SPI interface to the target device from the processor, and access to the programming data to be used for programming the device. Access to programming data can be provided by a telecommunications link for most remote systems.

SPI-DirectC is a set of C code designed to support embedded In-System Programming for the M2S, M2GL, and MPF families of devices. To use SPI-DirectC v2.1, 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 version 11.2 or later. The detailed specification of the programming file is included in "Data File Format" on page 16.

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.

1 – System Overview

To perform In-System Programming (ISP) for the SmartFusion2, IGLOO2, or PolarFire target device, the system must contain the following parameters:

- A microprocessor with at least 1200 bytes of RAM or a softcore processor implemented in another FPGA
- SPI IP to interface to 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

Note: See your device datasheet for information on power requirements for V_{pump} and other power supplies.

Table 1-1 shows the memory requirements.

Text - This is the compiled code size memory requirements.

Data - This is the run time memory requirement, i.e. the free data memory space required to execute the code.

BSS - This is the Block Started by Symbol allocation for variables that do not yet have values, i.e. uninitialized data. It is part of the overall Data size.

***Table 1-1* • Code Memory Requirements- SPI-DirectC Code Size on M3 16-Bit Mode**

Compile Options Enabled	Units are in Bytes		
	Text	Data	BSS
ENABLE_G4M_SUPPORT	13558	4860	1196
ENABLE_G5M_SUPPORT	12482	4904	762
All the above	18630	4960	1196

Systems with Direct Access to Memory

Figure 1-1 shows the overview of a typical system with direct access to the memory space holding the data file. See Table 1-2 for data storage memory requirements.

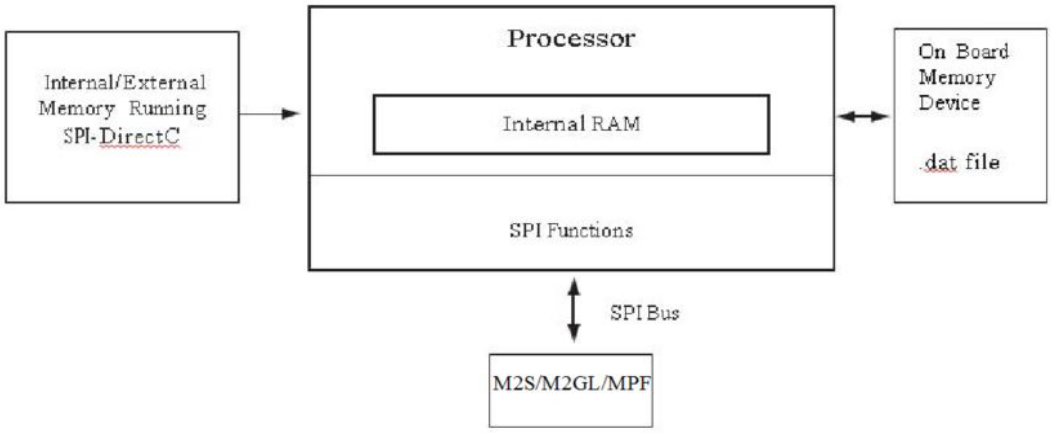


***Figure 1-1* • System with Direct Access to Memory**

***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 & 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
MPF300	9472	N/A	N/A
The total image size is the sum of all the corresponding enabled blocks for the specific target device.			

Systems with Indirect Access to Memory

Figure 1-2 is an overview of a system with no direct access to the memory space holding the data file. For example, the programming data may 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.

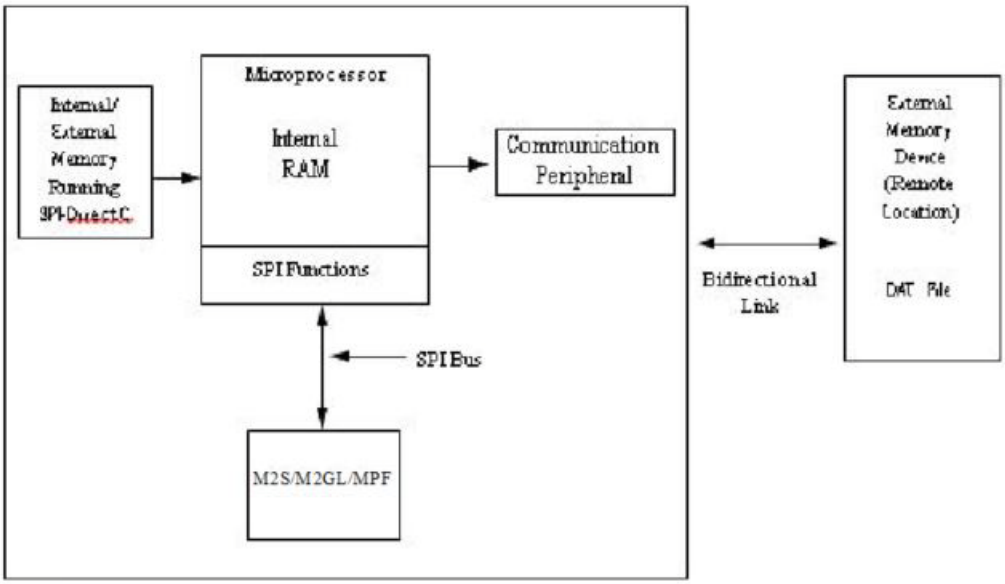


***Figure 1-2* • System With Indirect Access to Memory**

Motorola SPI Protocol

Motorla SPI Mode 3 is required to communicate with M2S, M2GL, and MPF devices using dedicated system controller SPI port. Please refer 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 shown in Table 1-3.

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

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

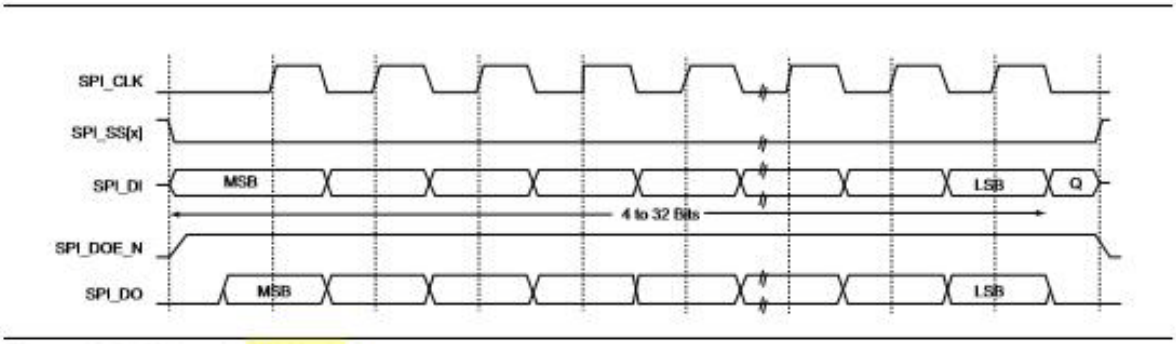


***Figure 1-3* • Motorola SPI Mode 0**

Multiple Frame Transfer - Mode 0: SPO = 0, SPH = 0

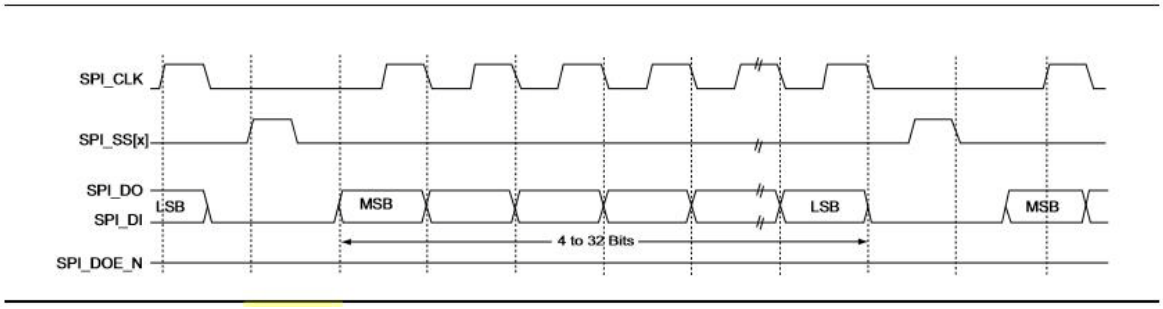


***Figure 1-4* • Motorola SPI Mode 0 Multiple Frame Transfer**

Notes:

- Between frames, the slave select (SPI_SS[x]) signal 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 enable (SPI_DOE_N) signal 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

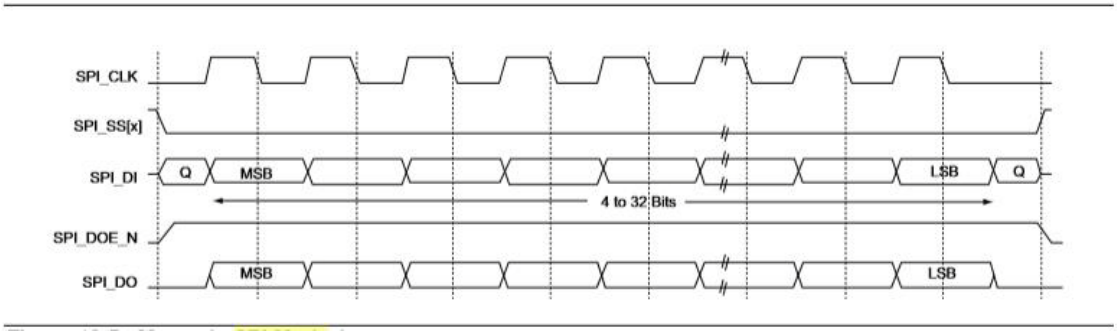


***Figure 1-5* • Motorola SPI Mode 1**

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

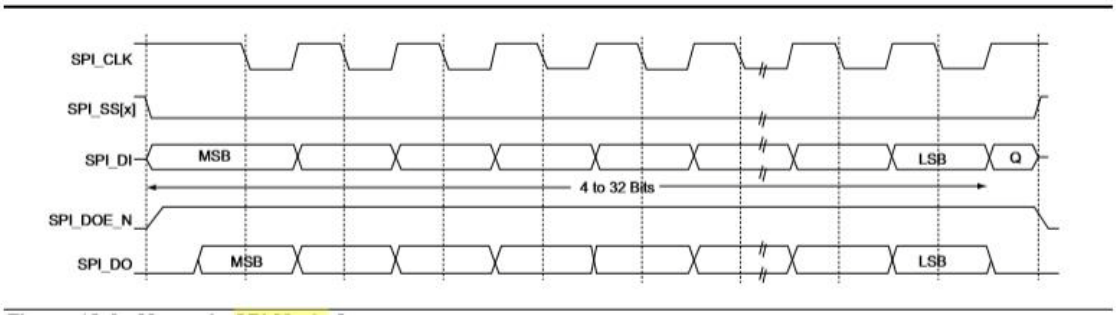


***Figure 1-6* • Motorola SPI Mode 2**

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

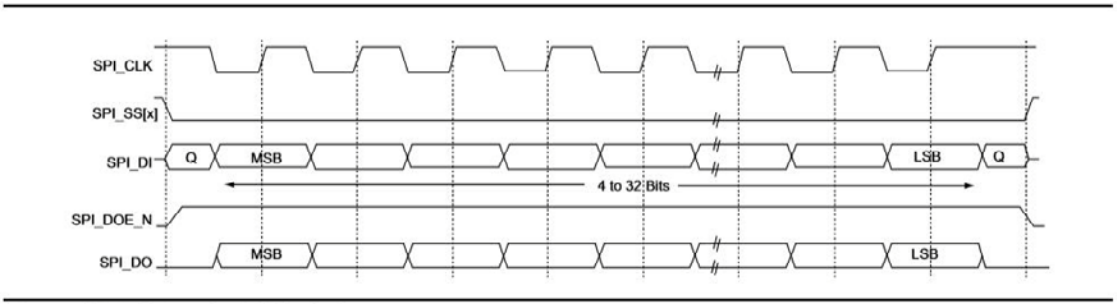


***Figure 1-7* • Motorola SPI Mode 3**

2 – Generating Data Files and Integrating DirectC

This chapter describes the flows for data file generation and SPI-DirectC code integration.

To generate your data file:

1. Generate the DAT file using Libero SoC v11.2 or later. If programming security is required, use Libero SoC v11.4 or later to generate the DAT file. See the latest Libero SoC online help for information on generating a DAT file.
2. Program the DAT file into the storage memory.

SPI-DirectC v2.1 Code Integration

Figure 2-1 shows the SPI-DirectC integration use flow.

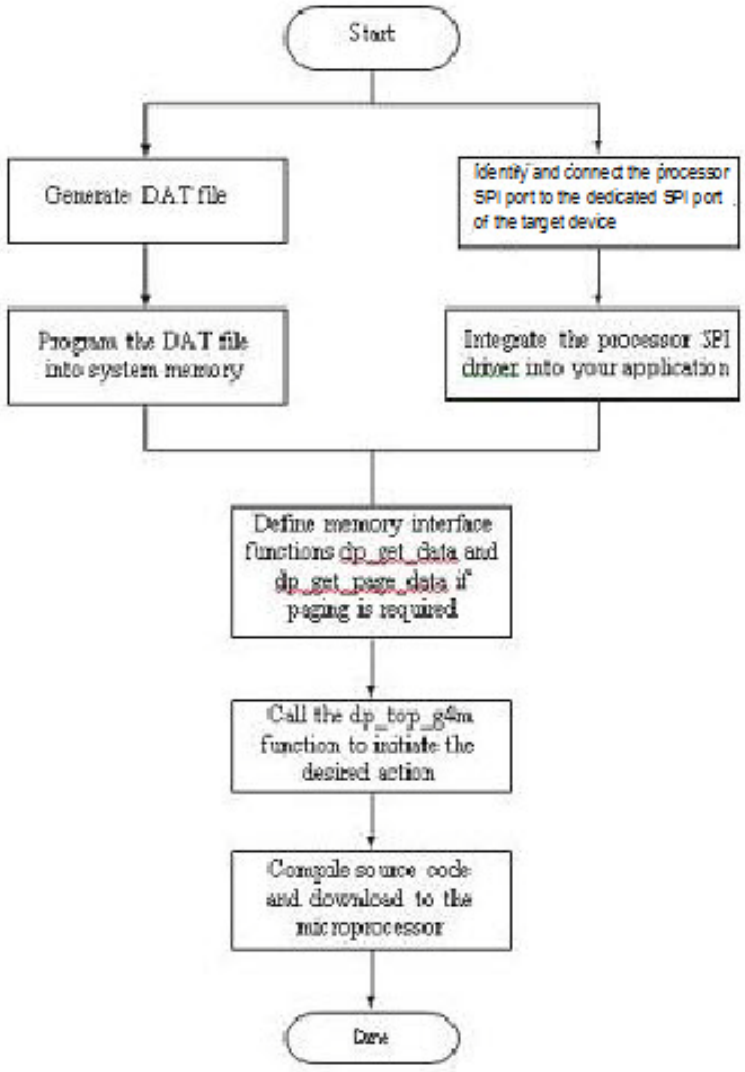


***Figure 2-1* • Importing SPI-DirectC Files**

To use SPI-DirectC code integration:

1. Import the SPI-DirectC files shown in Figure 2-2 into your development environment.

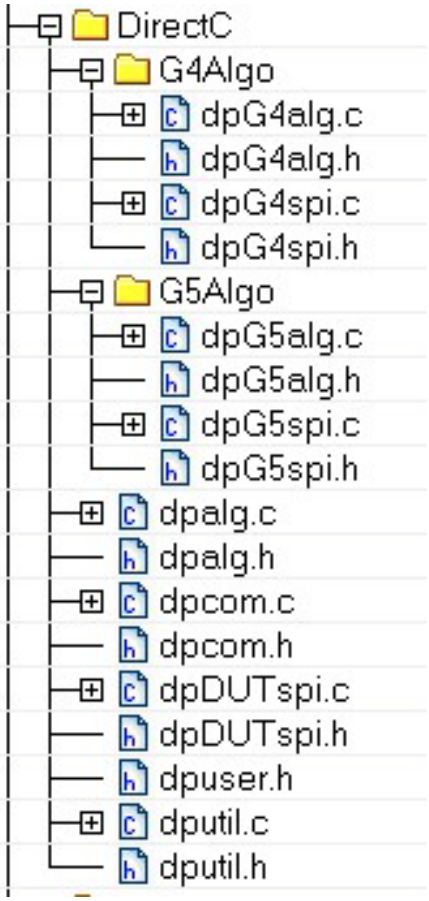


***Figure 2-2* • SPI-DirectC Files to import into your Development Environment**

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 "Data File Bit Orientation" on page 19.
   - 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. contains a short description of SPI-DirectC source code and their functions. Functions that must be modified are listed in Table 3-1.

***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

Compiler Switches

The compiler switches are shown in Table 3-2

***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 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.

Hardware Interface Components

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. These functions should 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 should 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 target device.

Display Functions

Three functions, *dp_display_array*, *dp_display_text*, and *dp_display_value*, are available to display text as well as numeric values. You must modify these functions for proper operation.

Memory Interface Functions

All access to the memory blocks within the data file is 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* will indicate the total number of valid bytes available for the client of the function.

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

Systems with Direct Access to the Memory Containing the Data File

Since 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. See "Compiler Switches" on page 11.
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.

An example of the *dp_get_data* function implementation is:

```
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;
}
```

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. See "Compiler Switches" on page 11.
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 "Data File Bit Orientation" on page 19 for additional information. For correct operation:
   - Fill the entire page unless the end of the image is reached. See "Data File Format" on page 16.
   - Update *return_bytes* to reflect the number of valid bytes in the page.

SPI-DirectC programming functions call the *dp_get_data* function every time access to a data block within the image data file is needed. 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.

Example of 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. Since 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.

```
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;
}
```

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 below.

```
#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
```

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.

Data Type Definitions

Microsemi uses DPUCHAR, 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 */
/*             i.e. 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;
```

Supported Actions

Table 3-3 lists supported actions and devices.

Table 3-3 • **Supported Actions**

Action	Supported Devices	Description
DP_DEVICE_INFO_ACTION	SmartFusion2, IGLOO2, PolarFire	Displays device security settings.
DP_READ_IDCODE_ACTION	SmartFusion2, IGLOO2, PolarFire	Reads and displays the content of the IDCODE register.

***Table 3-3* • Supported Actions (continued)**

DP_ERASE_ACTION	SmartFusion2, IGLOO2, PolarFire	Erases all supported blocks in the data file.
DP_PROGRAM_ACTION	SmartFusion2, IGLOO2, PolarFire	Performs erase, program and verify operations for all the supported blocks in the data file.
DP_VERIFY_ACTION	SmartFusion2, IGLOO2, PolarFire	Performs verify operation for all the supported blocks in the data file.
DP_ENC_DATA_AUTHENTICATION_ACTION	SmartFusion2, IGLOO2, PolarFire	It performs data authentication of the bitstream within the data file
DP_VERIFY_DIGEST_ACTION_CODE	SmartFusion2, IGLOO2, PolarFire	This action checks the digest of a programmed target device.

4 – Data File Format

DAT File Description for M2GL, M2S, and MPF Devices

The M2GL and M2S data file contains the following sections:

- **Header Block** - Contains information identifying the type of the binary file and data size blocks.
- **Constant Data Block** - Includes device ID, silicon signature and other information needed for programming.
- **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 format is described in Table 4-1.
- **Data Block** - Contains the raw data for all the different variables specified in the lookup table.

***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	
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

***Table 4-1* • DAT Image Description (continued)**

Header Section of DAT File	
ENVM Verify Data Size	2
UEK1_EXISTS	1
UEK2_EXISTS	1
SEC_ERASE	1
UEK3_EXISTS (M2S, M2GL only)	1
Number of Records	1
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	
Information	# of Bytes
Binary Data	Variable
CRC of the entire image	2

5 – Source File Description

DPUSER.H

File contains definitions of all Action codes as well as possible error codes that could be reported within SPI-DirectC code.

DPCOM.C and DPCOM.H

These files contain memory interface functions.

DPALG.C and DPALG.H

These files contain the main entry function *dp_top* and device ID check function.

DPG4ALG.C and DPG4ALG.H

These files contain the main entry function *dp_top_g4* and all other functions common to M2S and MGL families.

DPG5ALG.C and DPG5ALG.H

These files contain the main entry function *dp_top_g5* and all other functions common to the MPF family of devices.

DPDUTSPI.C and DPDUTSPI.H

These files contain the SPI interface function declaration and definition to the target device. SPI Mode 3 must be used to program M2S/M2GL devices. Refer to SPI IP block used for proper initialization.

DPG4SPI.C and DPG4SPI.H

These files contain the SPI interface function declaration and definition to the target device specific to M2S and M2GL device families.

DPG5SPI.C and DPG5SPI.H

These files contain the SPI interface function declaration and definition to the target device specific to MPF device families.

DPUTIL.C and DPUTIL.H

These files 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 software. The SPI-DirectC implementation must be in sync with the specified data orientation. Table 6-1 illustrates how the data is stored in the binary data file. See "Data File Format" on page 16" for additional information on the data file..

***Table 6-1* • Binary Data File Example**

Byte O	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 bits (msb) in the last byte are filled with zeros. An example below 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). A binary representation of the same data follows.

20E60A9AB06FAC78A6	tdi
10000011100110 00001010100110101011000001101111101011000111100010100110 tdi	
Bit 69	Bit 0

This data is stored in the data block section. Table 6-2 shows how the data is stored in the data block.

***Table 6-2* • Data Block Section Example**

Byte O	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 6.40. It is designed to work on M2GL_M2S-EVAL-KIT with SmartFusion2 M2S025-FGG484 device.

Project Requirements

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

Hardware:

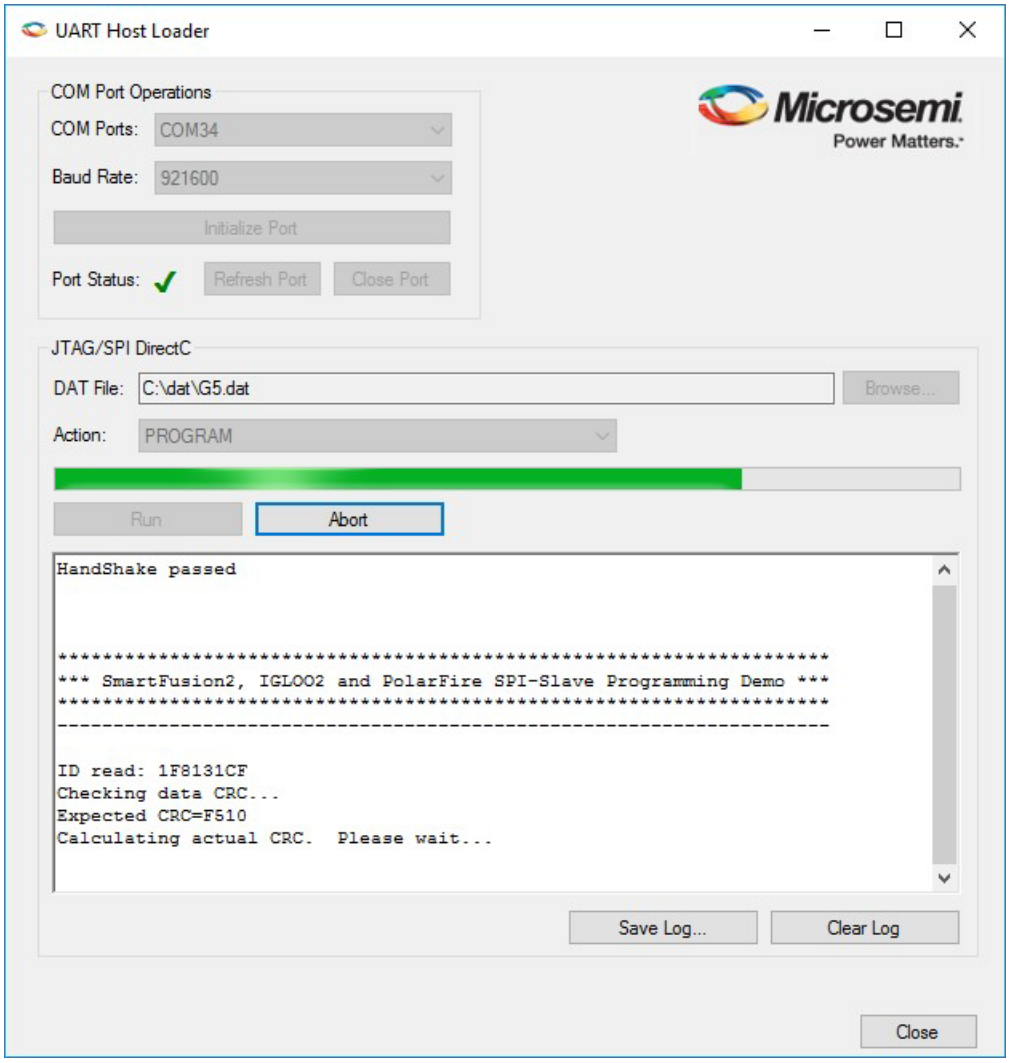
- SmartFusion2 Security Evaluation Kit with SmartFusion2 M2S090-FGG484 device.
- jLink from IAR.
- Target board with Microsemi device to be programmed.

Software:

- IAR Embedded Workbench version 6.4.
- UART Host Loader available with this release package.

Procedure

1. Program the evaluation kit with SPI_DC_top.stp STAPL 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 Host Loader available with this release package.
6. There should be 4 com ports available in the serial port setup window. Select the 4th one from the list and configure the Baud Rate as shown below. If more than 4 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 4th 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 UART Host Loader application waits for data from the SmartFusion2 evaluation kit.
10. The STAPL file programmed into the evaluation kit has a SPI-DirectC sample project that supports SmartFusion2, IGLOO2, and PolarFire 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 UART Host Loader and click **Run** again.



11. To make changes to the embedded project, run IAR workbench and modify the compile options as desired. 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 on download and run from IAR.

8 – Error Messages & Troubleshooting Tips

The information in this chapter may help you solve or identify a problem when using SPI-DirectC code. If you have a problem that you cannot solve, visit the Microsemi website at http://www.microsemi.com/products/fpga-soc/design-support/fpga-soc-support or contact Microsemi Customer Technical Support at tech@microsemi.com or call our hotline 1-800-262-1060.

See Table 7-1 for a description of exit codes and their solutions.

***Table 8-1* • Exit Codes**

Exit Code	Error Message	Action/Solution
0	This code does not indicate an error.	This message indicates success
2	Data processing failed.	- Check the Vpump 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 was 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.	- Check the Vpump 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.	- Check Vpump level. - Try with new device. - Measure SPI pins and noise or reflection.

***Table 8-1* • Exit Codes (continued)**

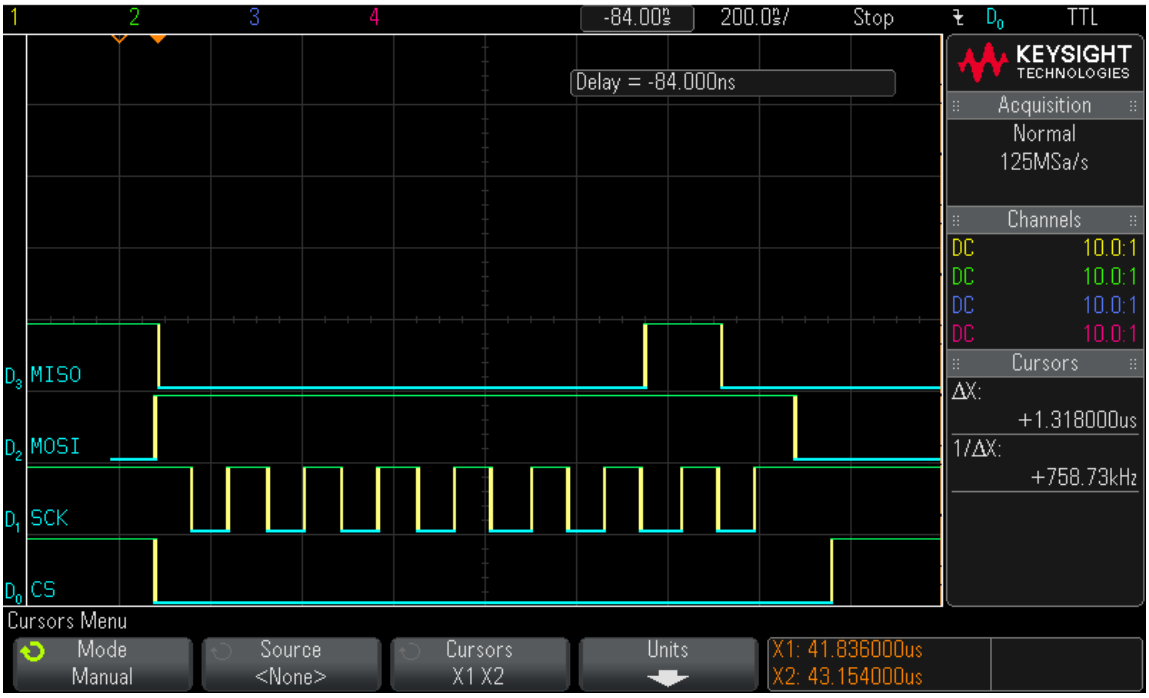
11	FPGA failed verify.	Possible Cause: - The device is secured, and the corresponding DAT file is not loaded. - The device is programmed with an incorrect design. Solution: - Load the correct DAT file. - Check Vpump level. - Measure SPI pins and noise or reflection.
18	Failed to authenticate the encrypted data.	- Make sure the AES key used to encrypt the data matches the AES key programmed in the device.
25	Device initialization failure.	- Check Vpump level. - Try with new device. - Measure SPI pins and noise or reflection.
100	CRC data error. Data file is corrupted or programming on system board is not successful.	- 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.

A – SmartFusion2 and IGLOO2 SPI-Slave Programming Waveform Analysis

Read ID code waveform:

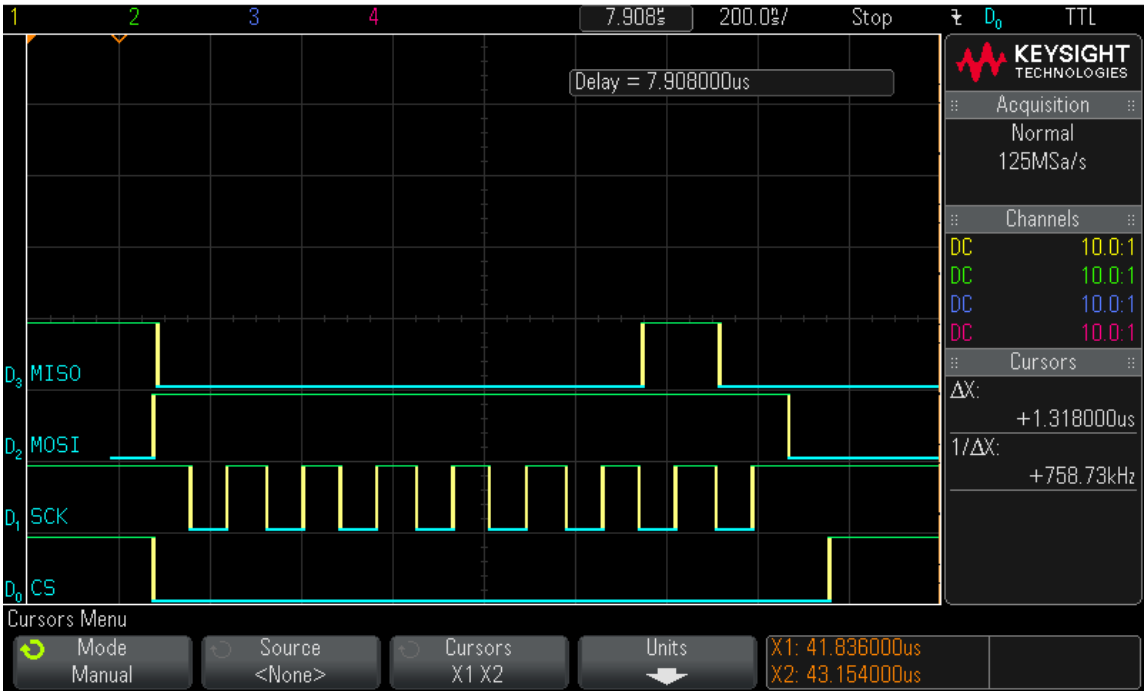
Step 1: Hardware Status Check

MSO-X 2024A, MY52490979: Tue Jul 26 17:49:32 2016

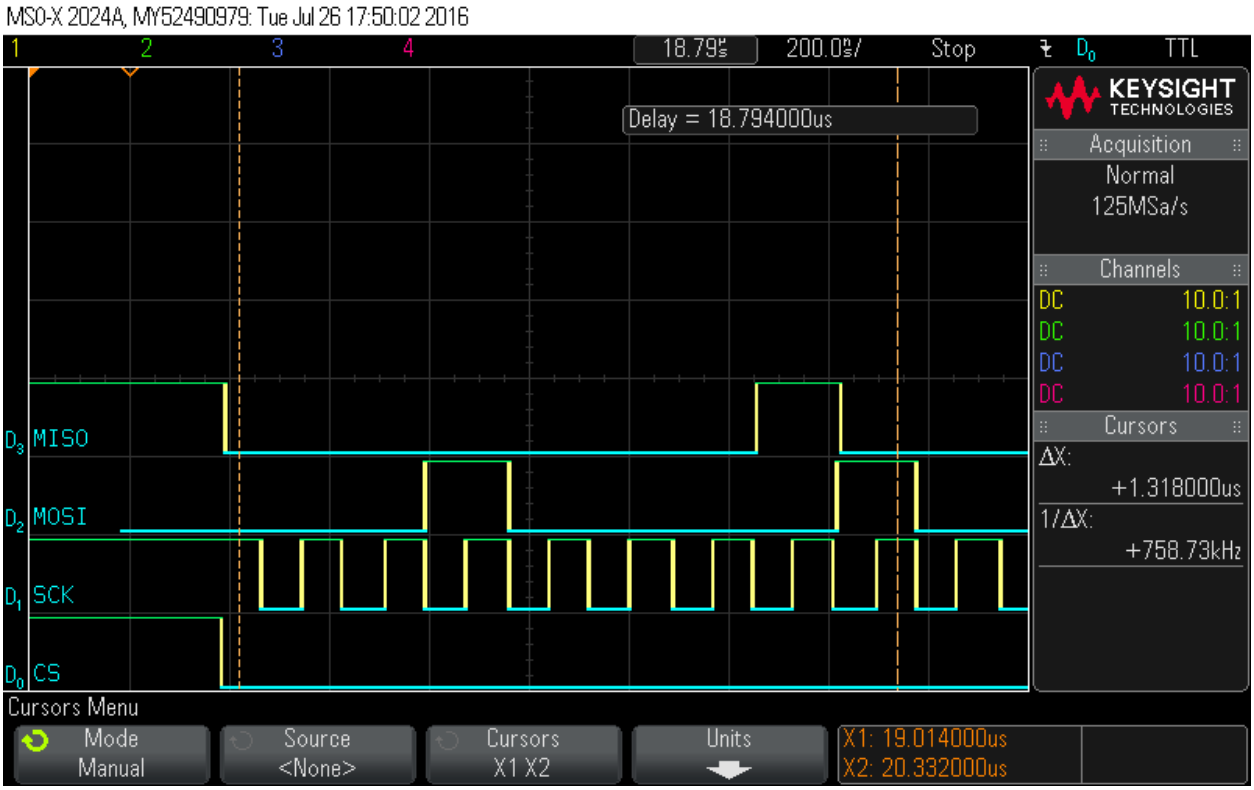


Step 2: Hardware Status Check

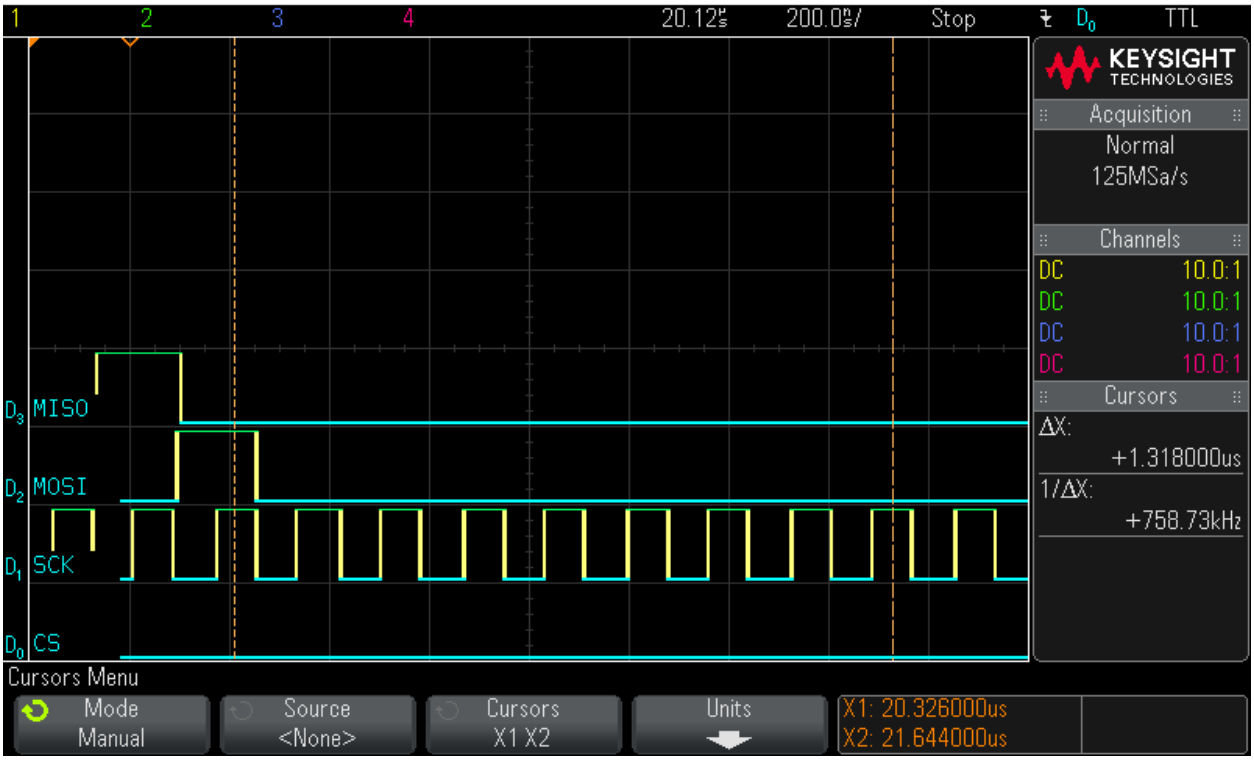
MSO-X 2024A, MY52490979: Tue Jul 26 17:49:43 2016



Step 3: Clock in read_id command (0x21)

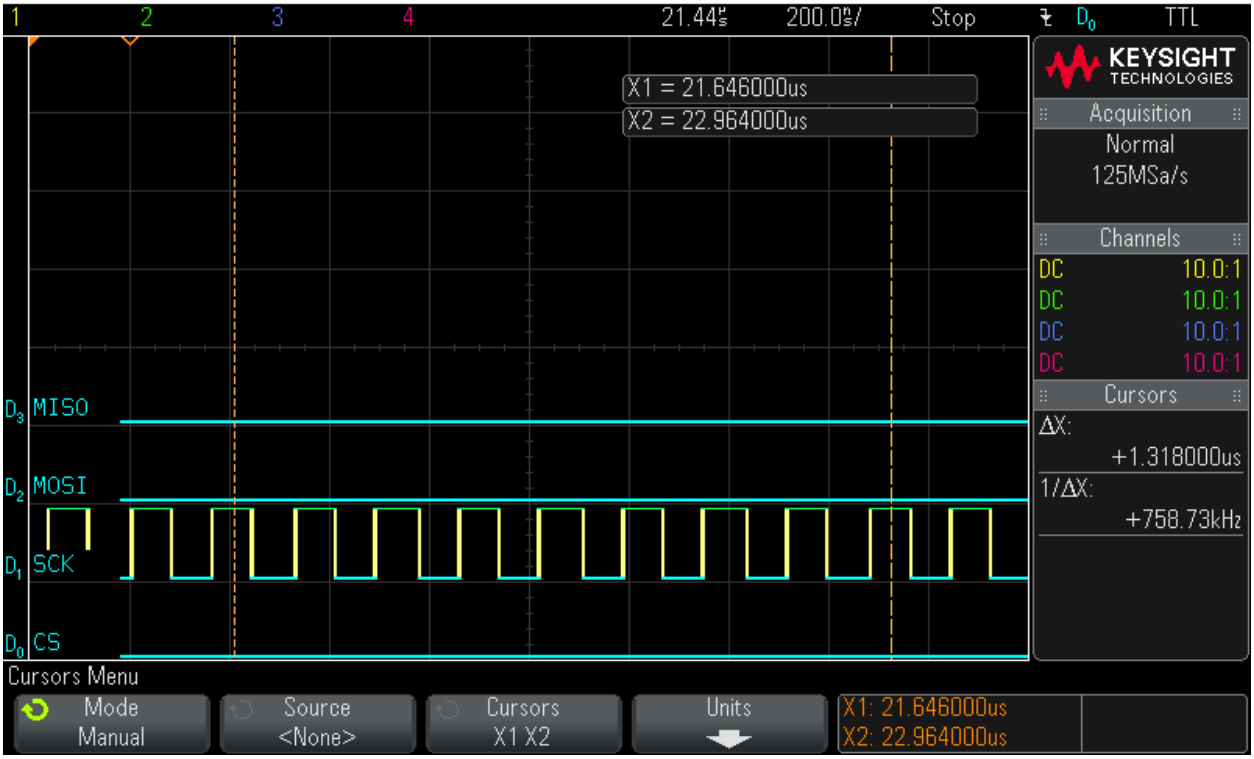


Step 4: Clock in 16 bytes of zero values - Byte 0



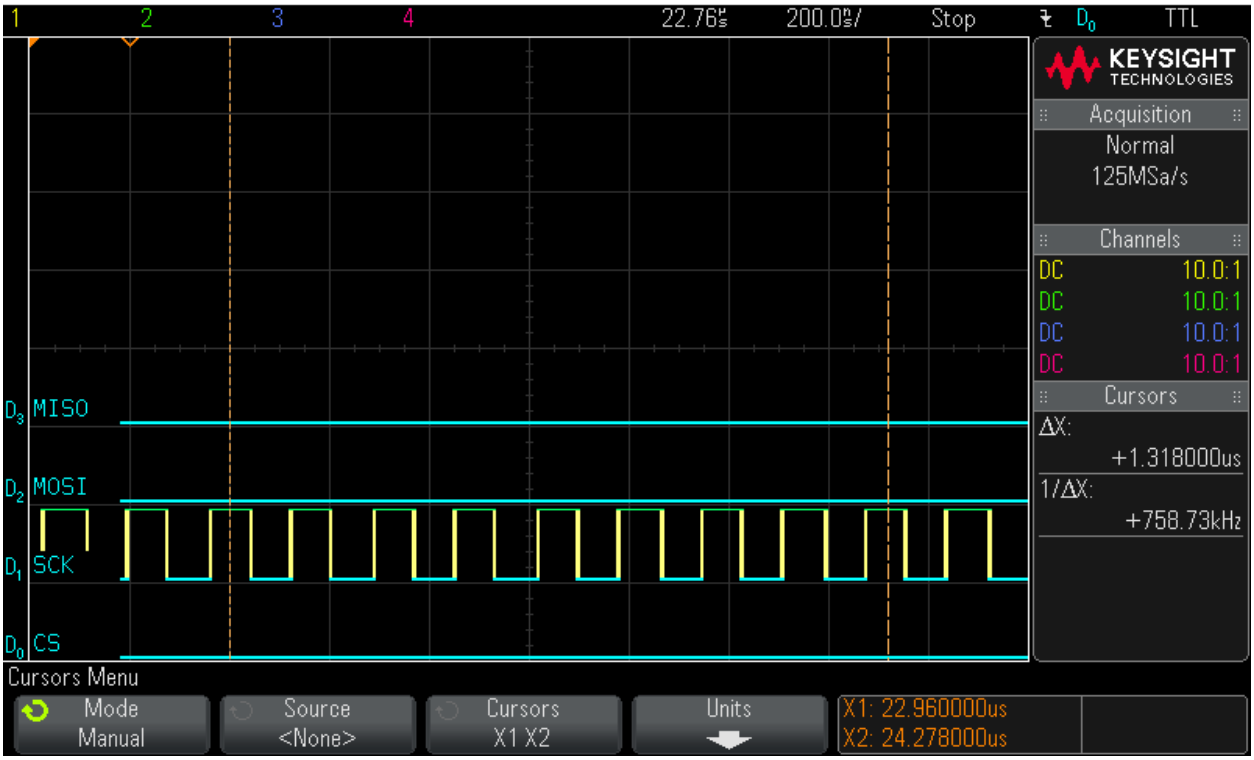
Step 5: Clock in 16 bytes of zero values - Byte 1

MSO-X 2024A, MY52490979: Tue Jul 26 17:50:24 2016



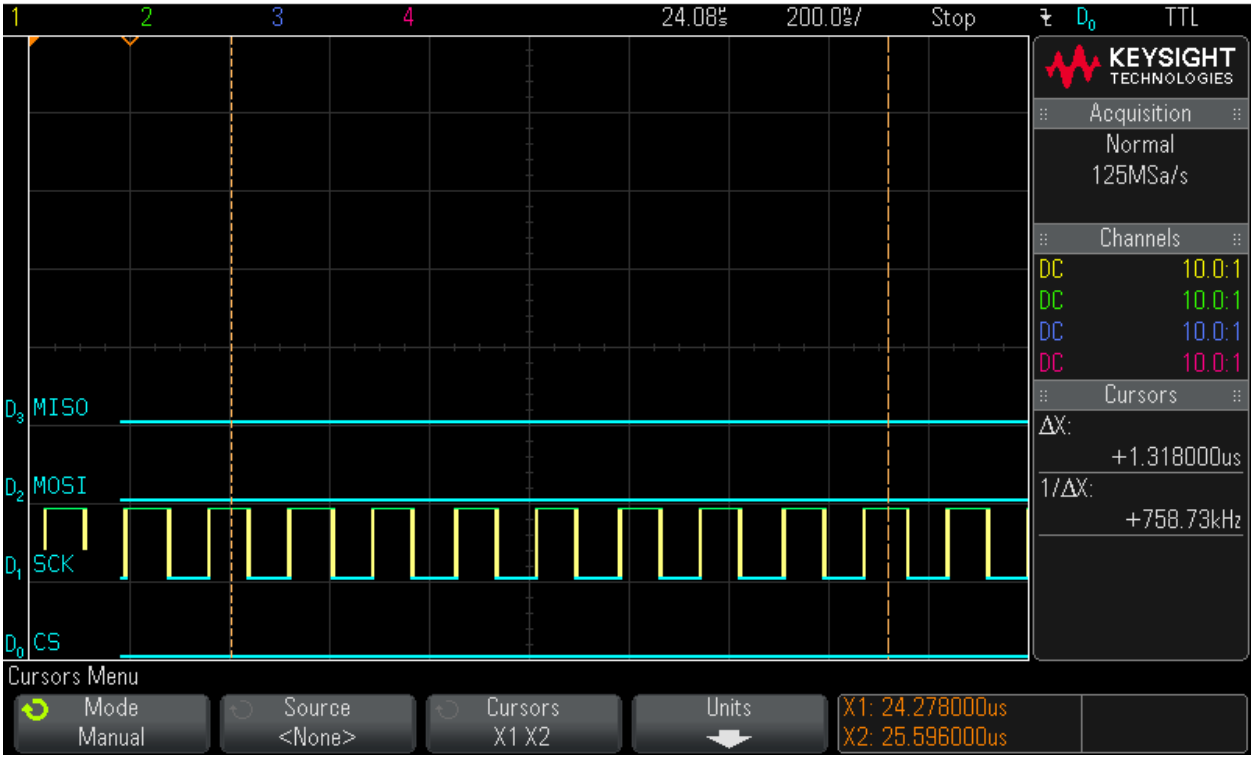
Step 6: Clock in 16 bytes of zero values - Byte 2

MSO-X 2024A, MY52490979: Tue Jul 26 17:50:34 2016



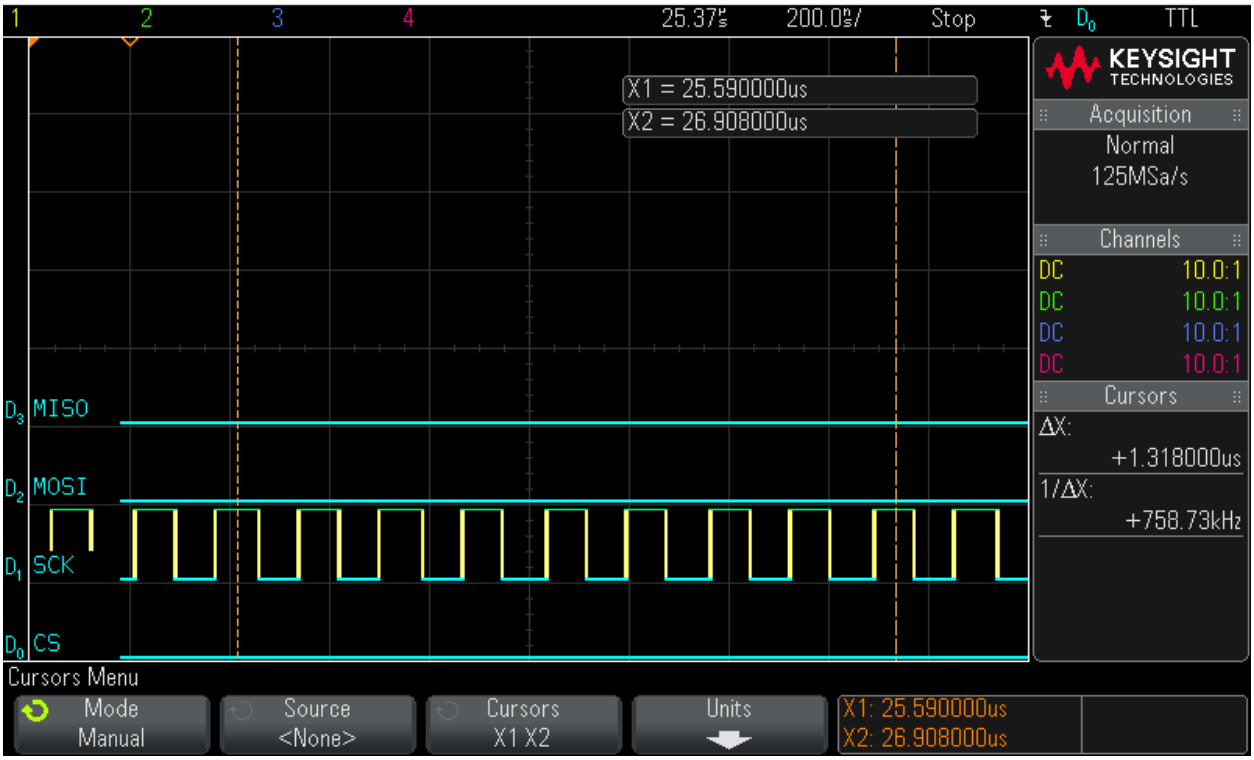
Step 7: Clock in 16 bytes of zero values - Byte 3

MSO-X 2024A, MY52490979: Tue Jul 26 17:50:45 2016

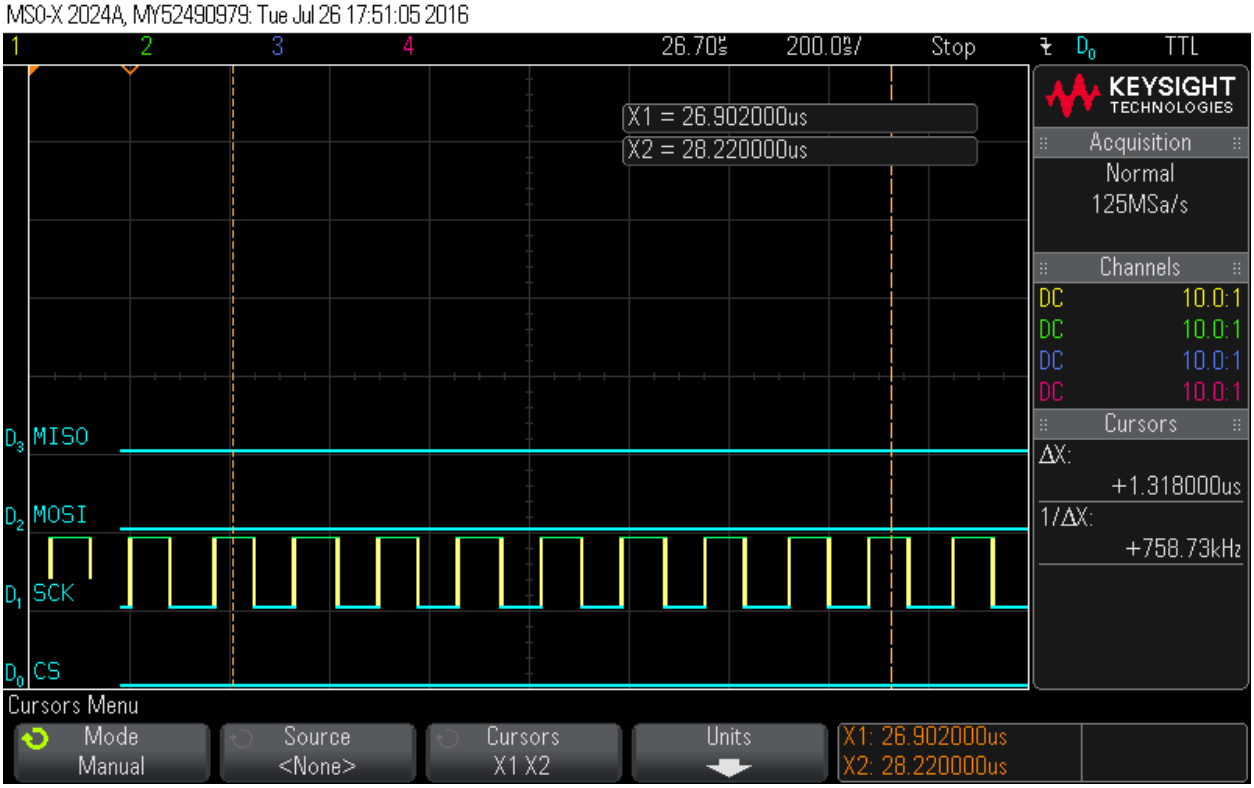


Step 8: Clock in 16 bytes of zero values - Byte 4

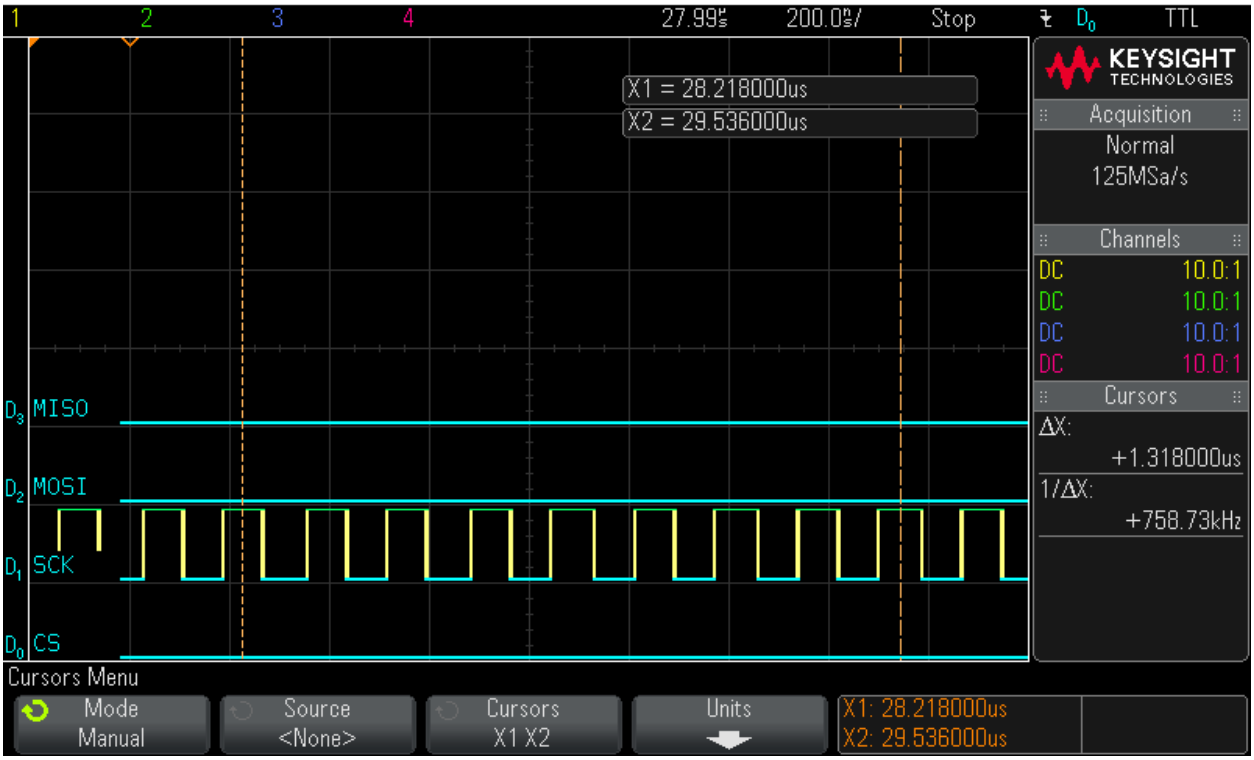
MSO-X 2024A, MY52490979: Tue Jul 26 17:50:54 2016



Step 9: Clock in 16 bytes of zero values - Byte 5

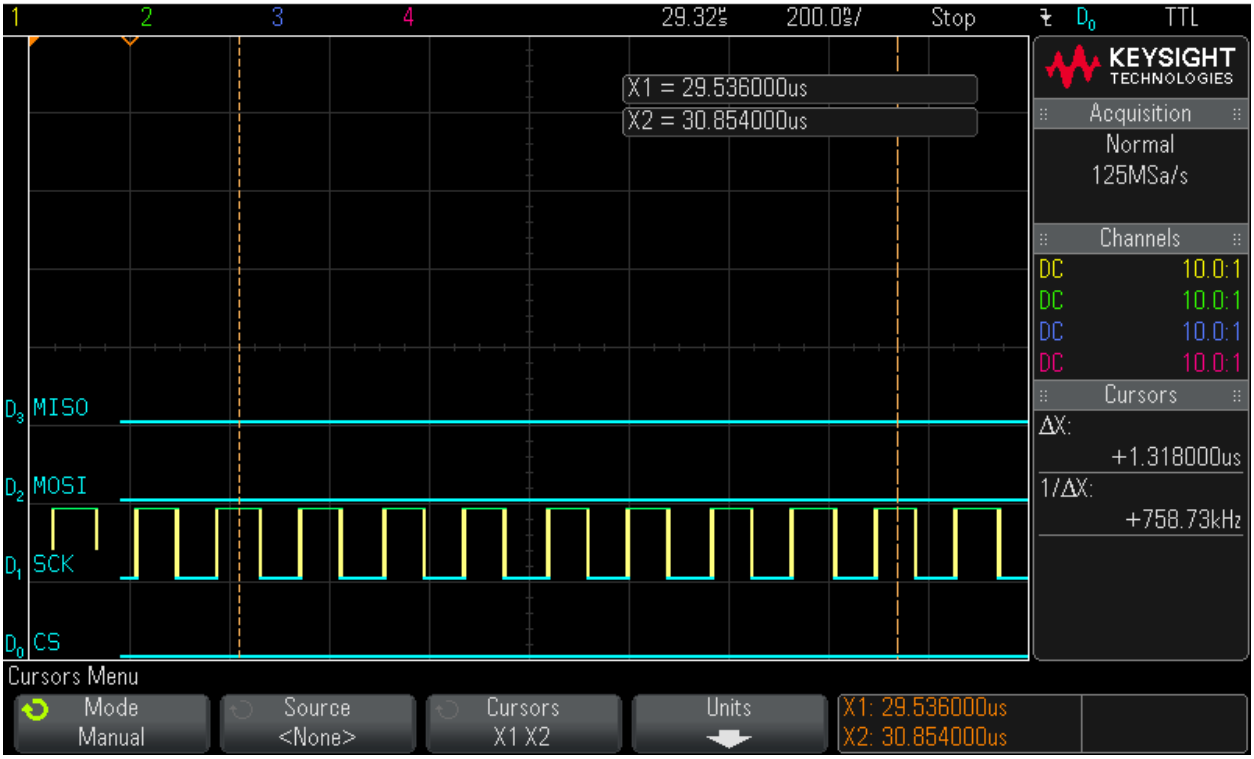


Step 10: Clock in 16 bytes of zero values - Byte 6



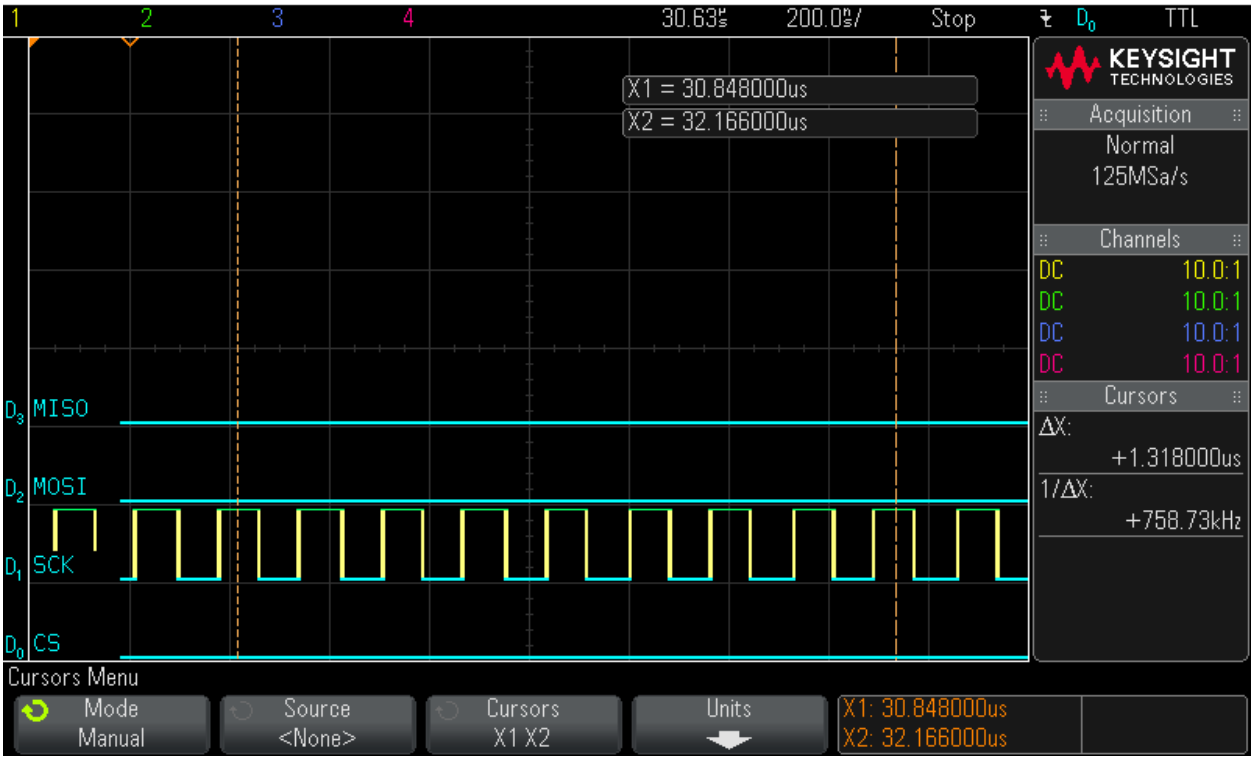
Step 11: Clock in 16 bytes of zero values - Byte 7

MSO-X 2024A, MY52490979: Tue Jul 26 17:51:25 2016



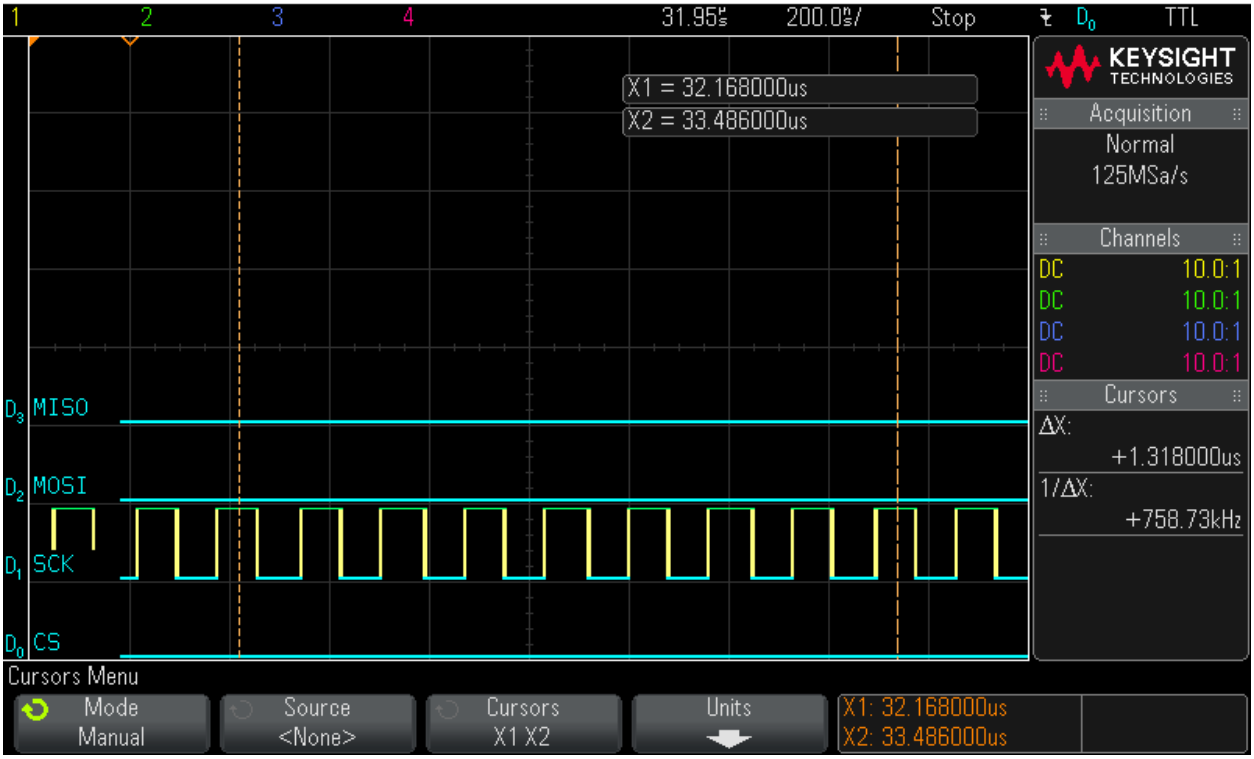
Step 12: Clock in 16 bytes of zero values - Byte 8

MSO-X 2024A, MY52490979: Tue Jul 26 17:51:43 2016



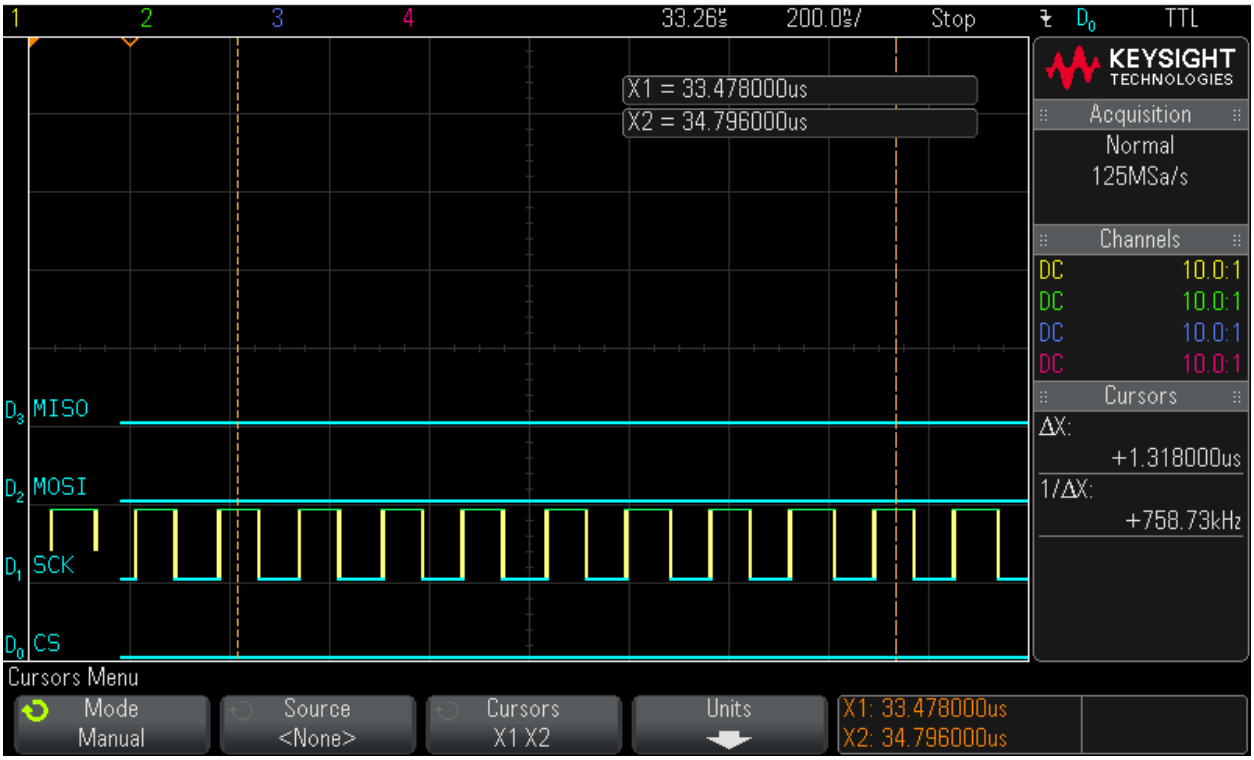
Step 13: Clock in 16 bytes of zero values - Byte 9

MSO-X 2024A, MY52490979: Tue Jul 26 17:51:55 2016



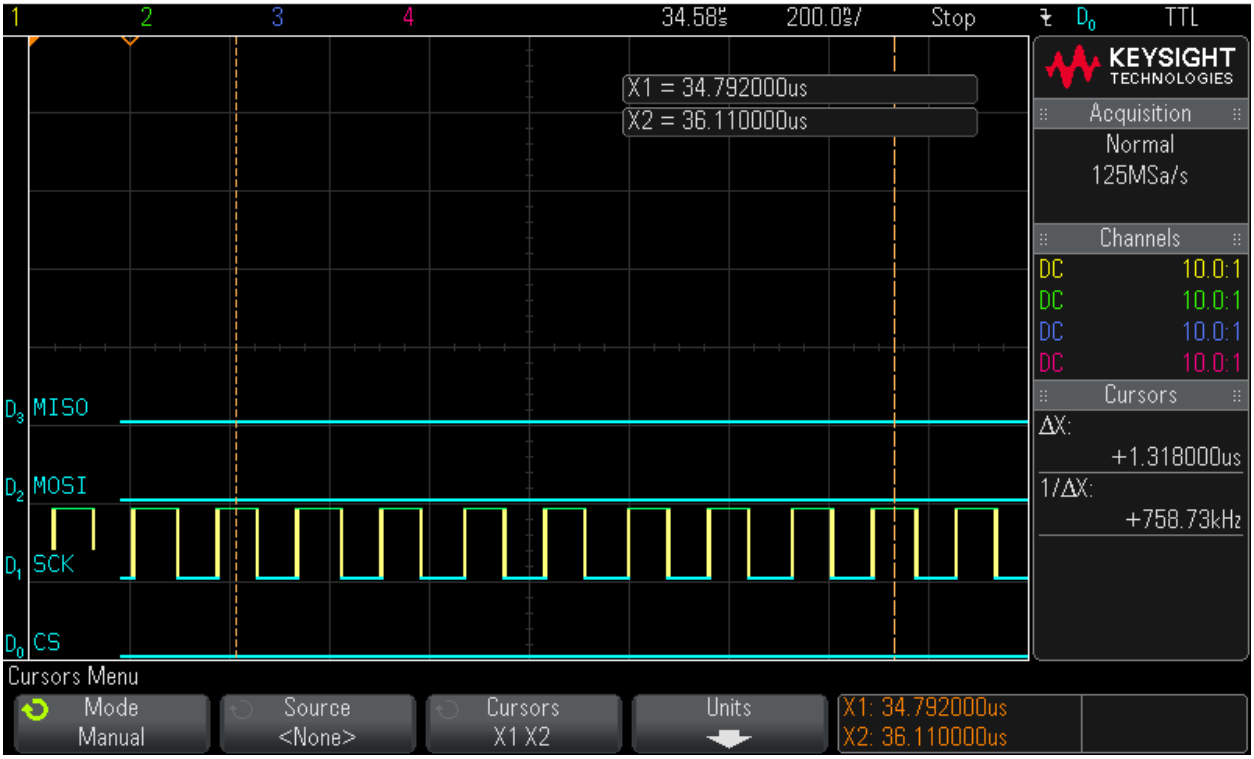
Step 14: Clock in 16 bytes of zero values - Byte 10

MSO-X 2024A, MY52490979: Tue Jul 26 17:52:04 2016



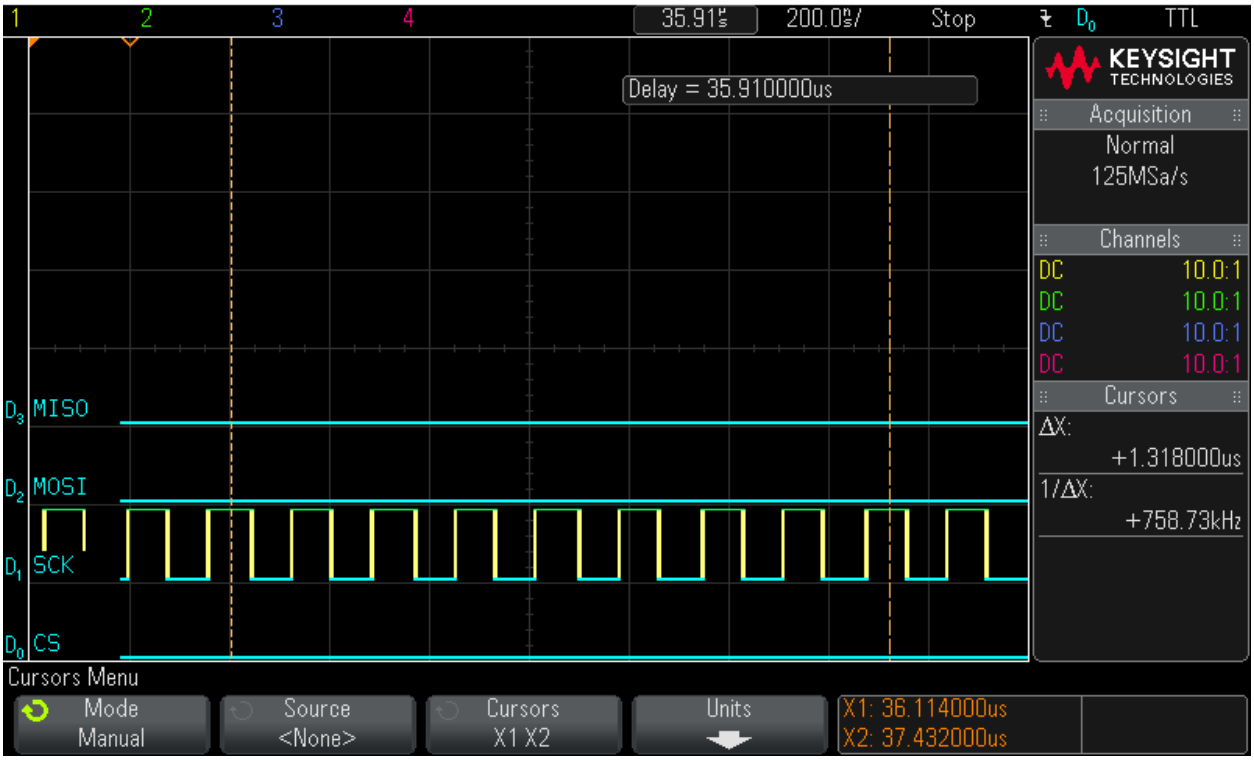
Step 15: Clock in 16 bytes of zero values - Byte 11

MSO-X 2024A, MY52490979: Tue Jul 26 17:52:15 2016



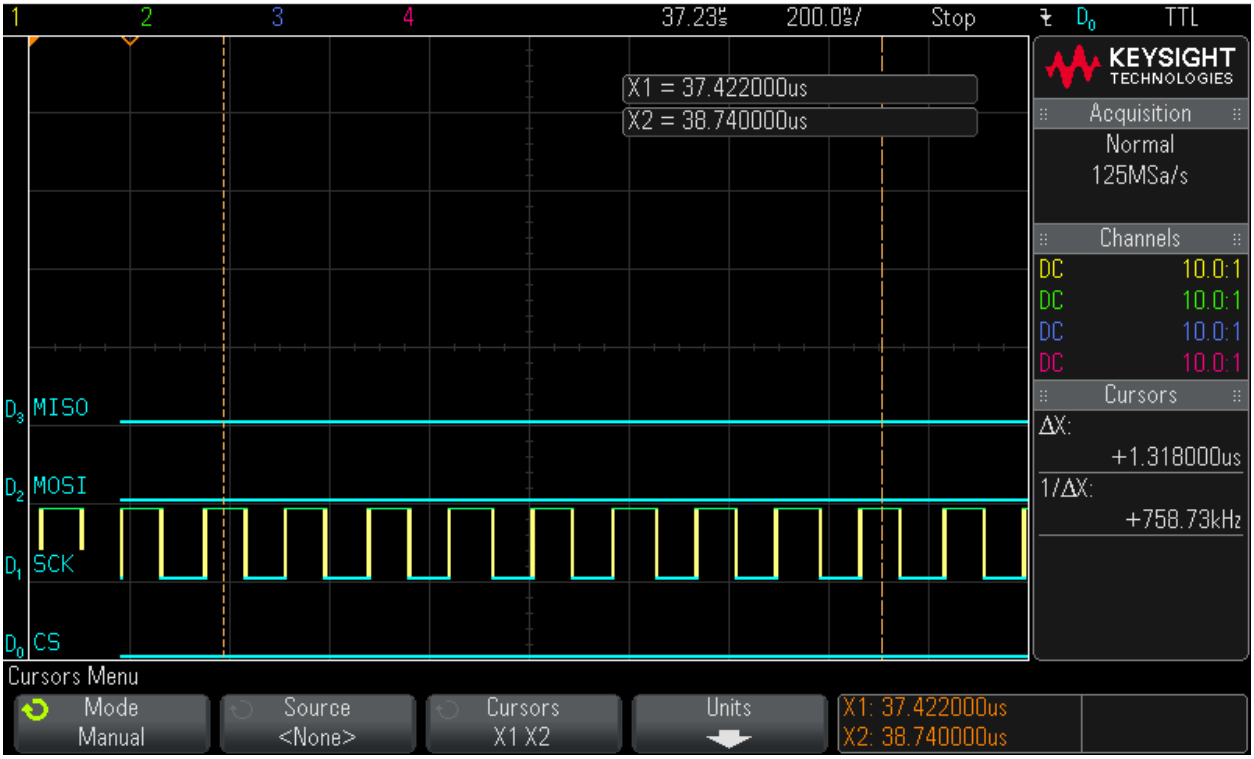
Step 16: Clock in 16 bytes of zero values - Byte 12

MSO-X 2024A, MY52490979: Tue Jul 26 17:52:26 2016



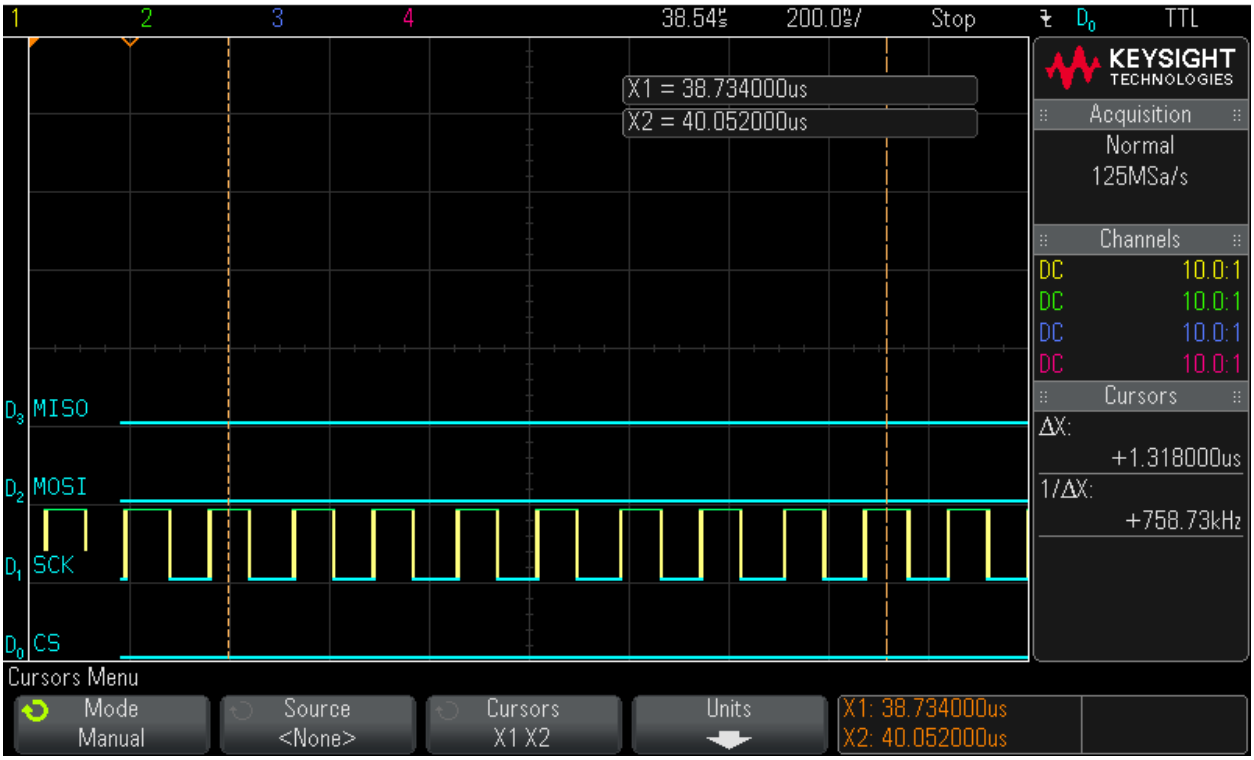
Step 17: Clock in 16 bytes of zero values - Byte 13

MSO-X 2024A, MY52490979: Tue Jul 26 17:52:37 2016



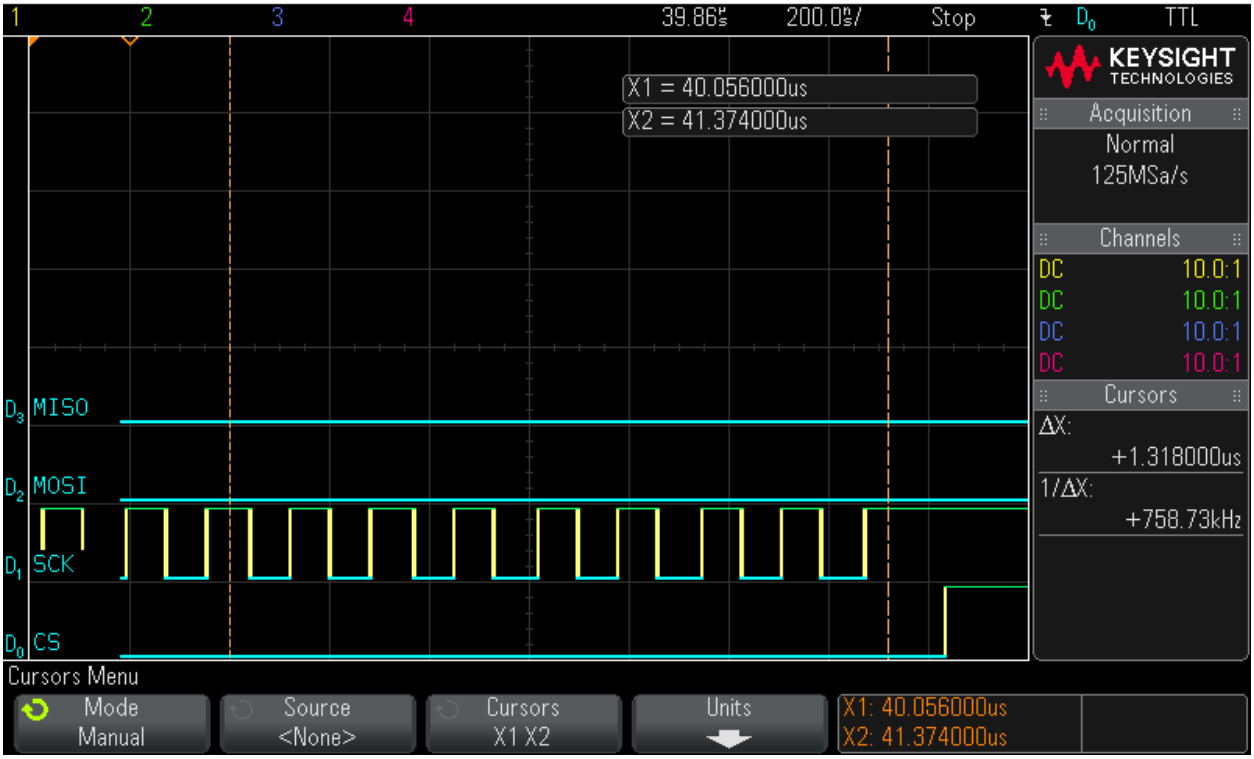
Step 18: Clock in 16 bytes of zero values - Byte 14

MSO-X 2024A, MY52490979: Tue Jul 26 17:52:46 2016



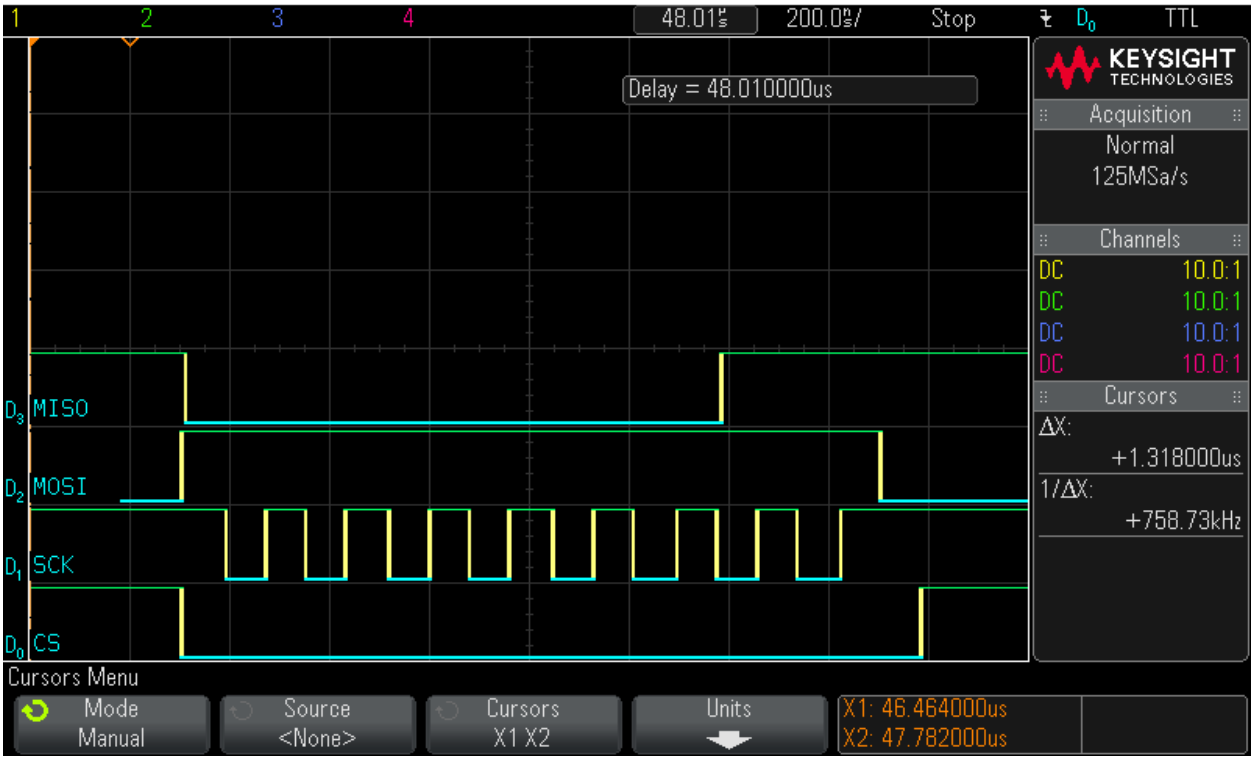
Step 19: Clock in 16 bytes of zero values - Byte 15

MSO-X 2024A, MY52490979: Tue Jul 26 17:52:58 2016



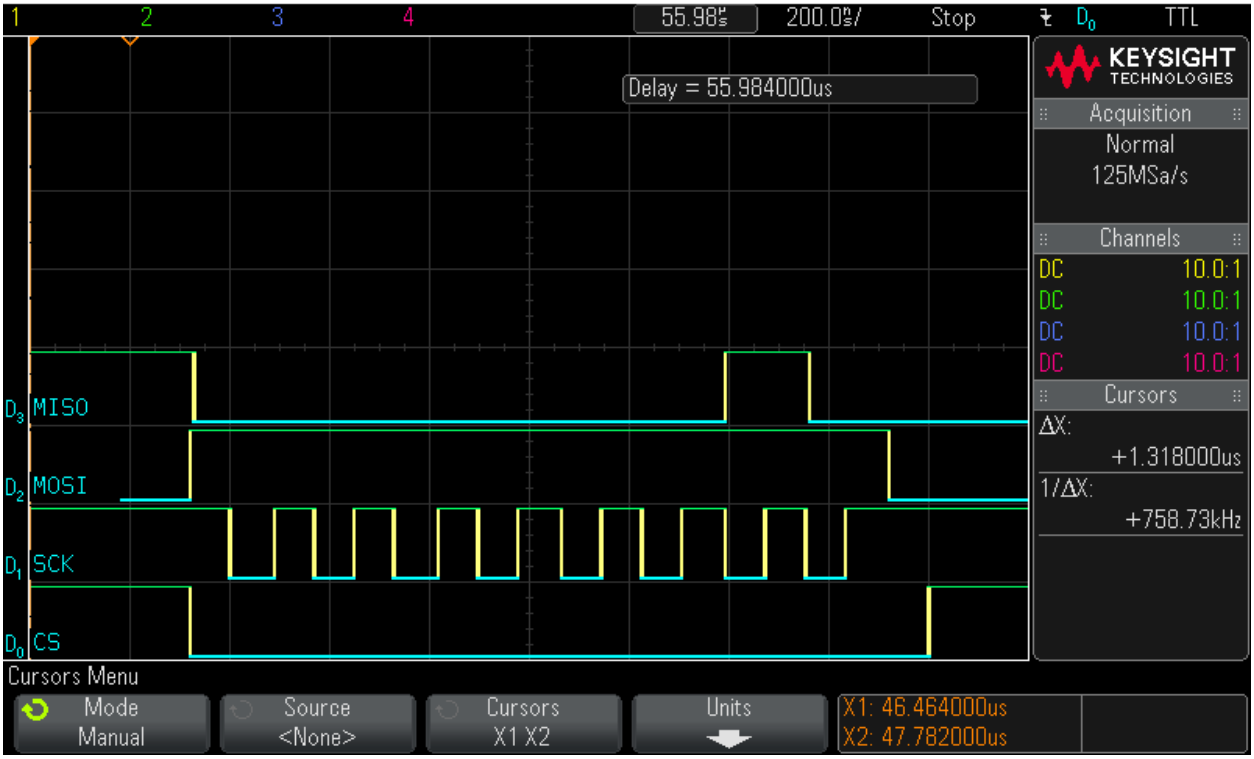
Step 20: Hardware Status Check

MSO-X 2024A, MY52490979: Tue Jul 26 17:53:22 2016



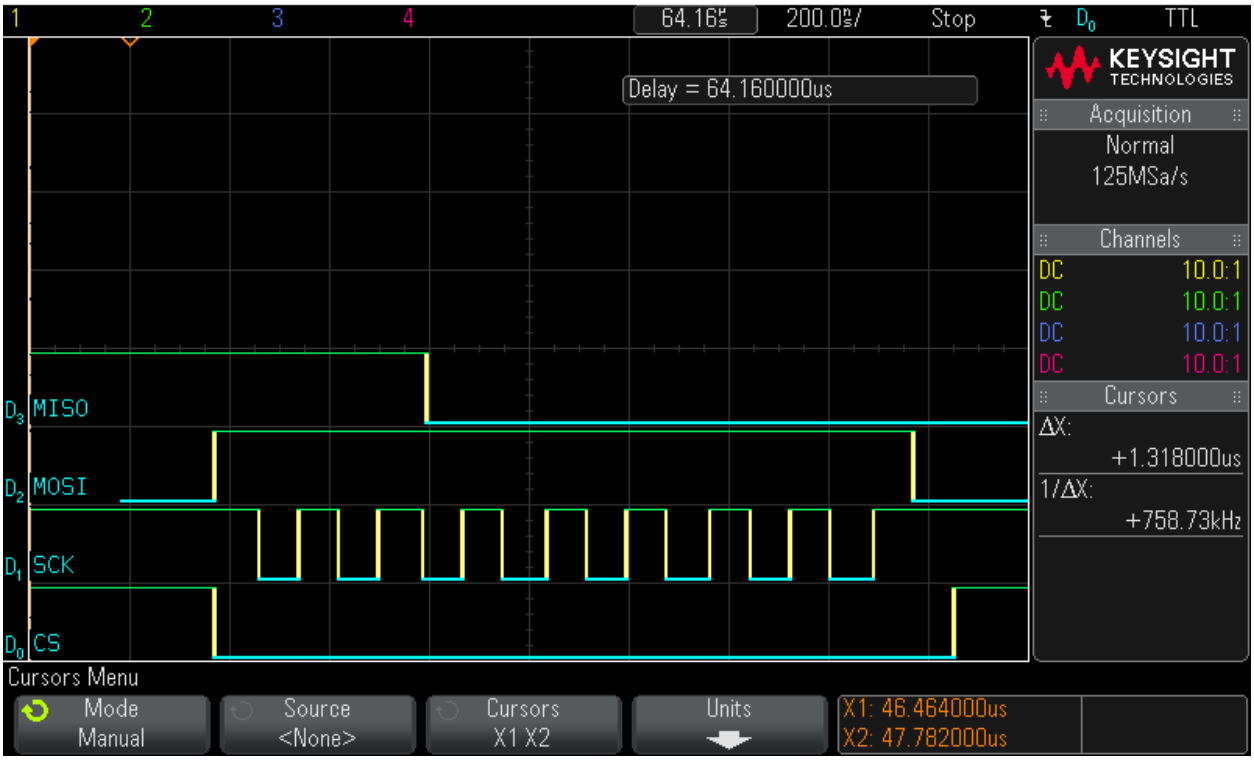
Step 21: Hardware Status Check

MSO-X 2024A, MY52490979: Tue Jul 26 17:53:30 2016

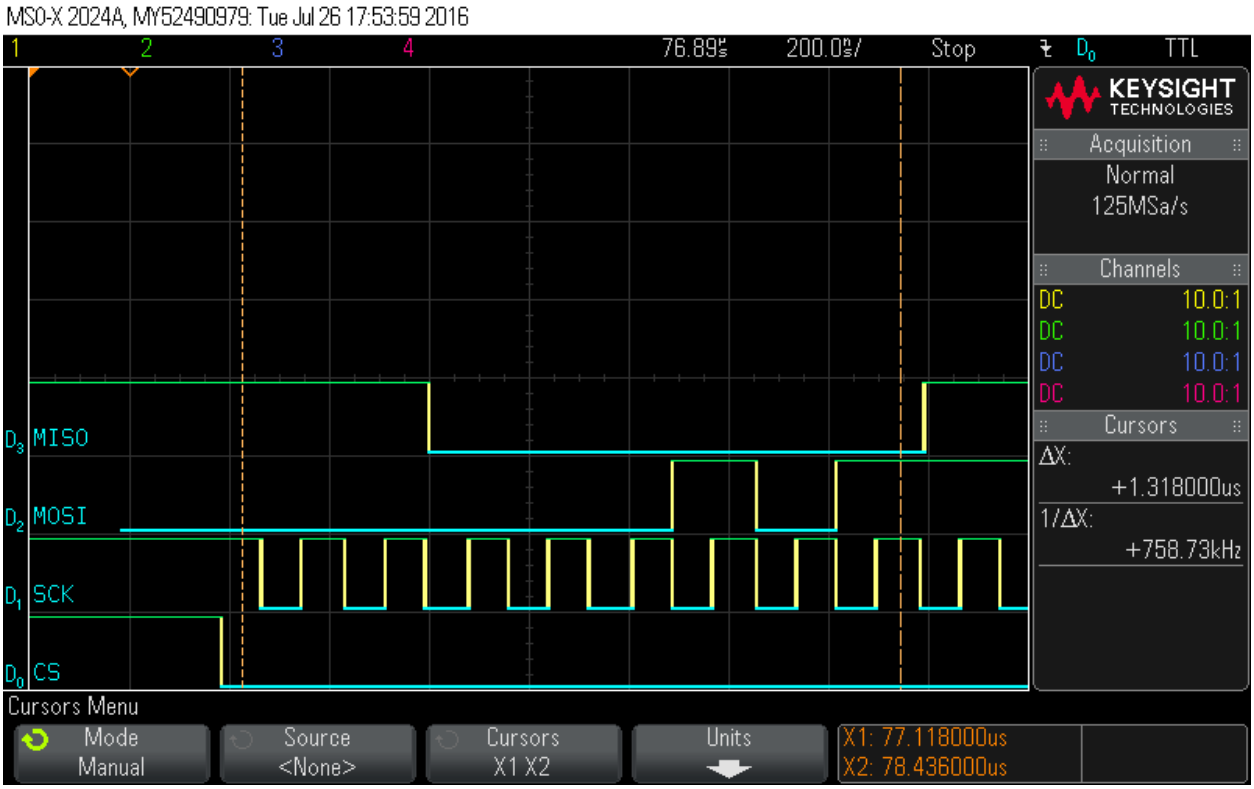


Step 22: Hardware Status Check

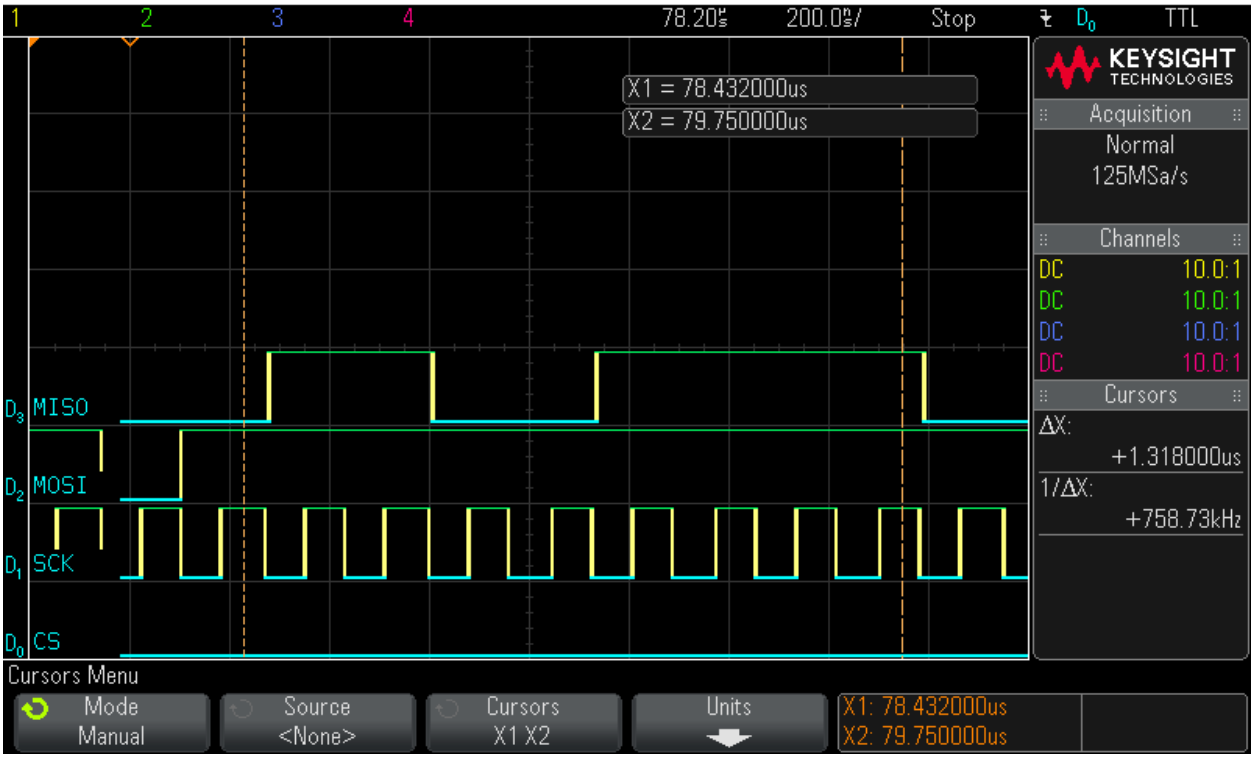
MSO-X 2024A, MY52490979: Tue Jul 26 17:53:38 2016



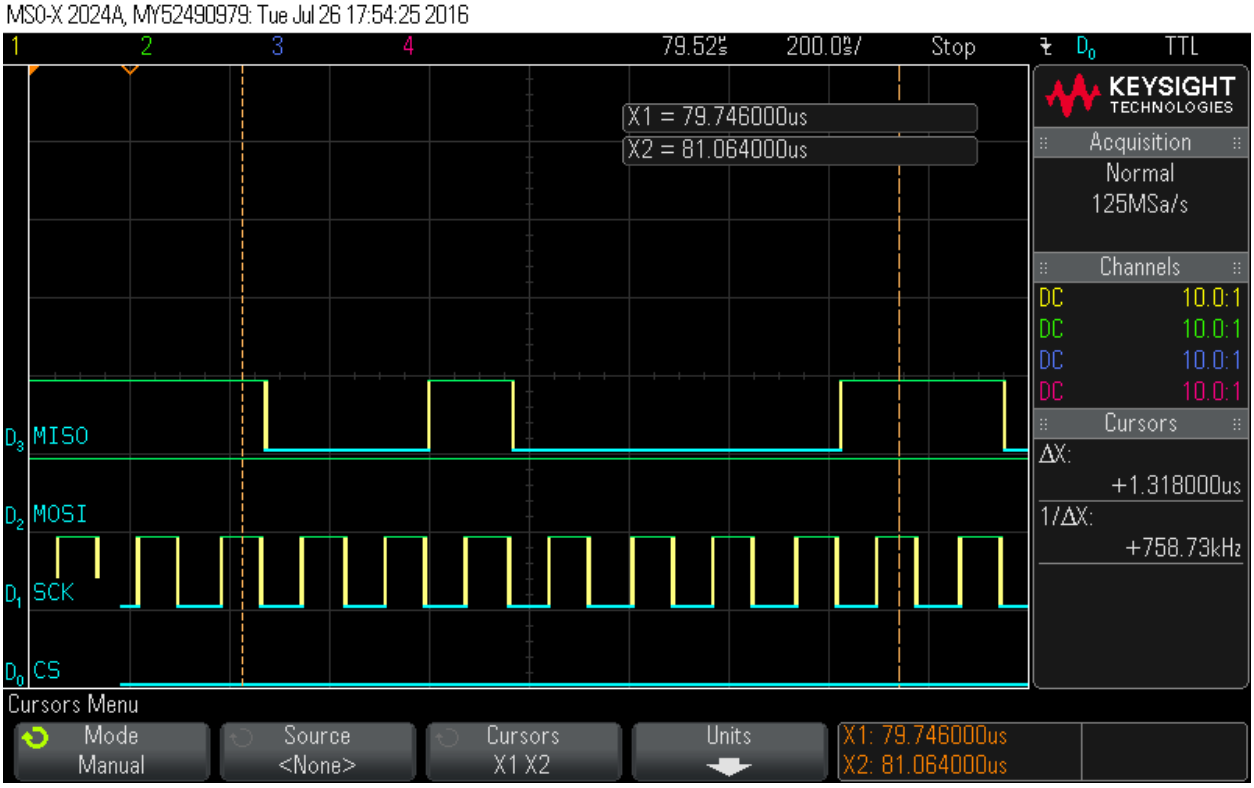
Step 23: Clock in read command (0x5)



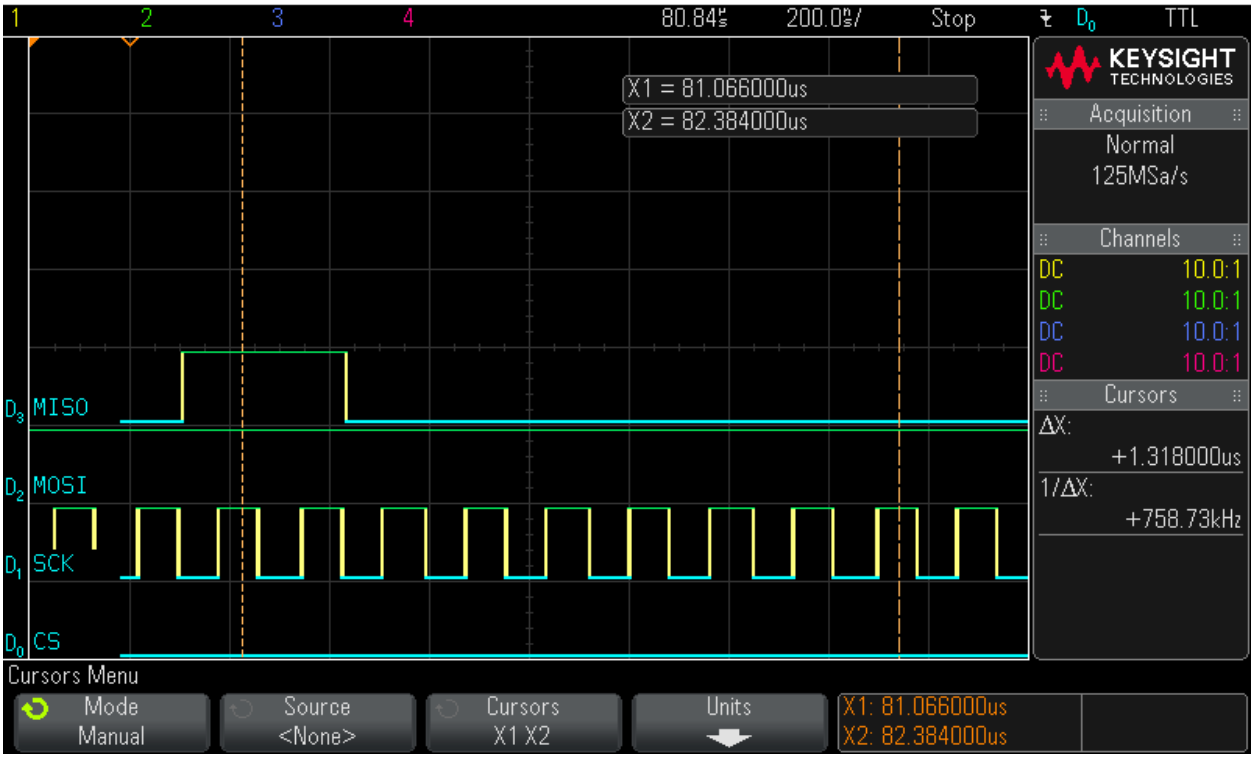
Step 24: Reading out 16 Bytes of data – Byte 0 = 0xCF



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

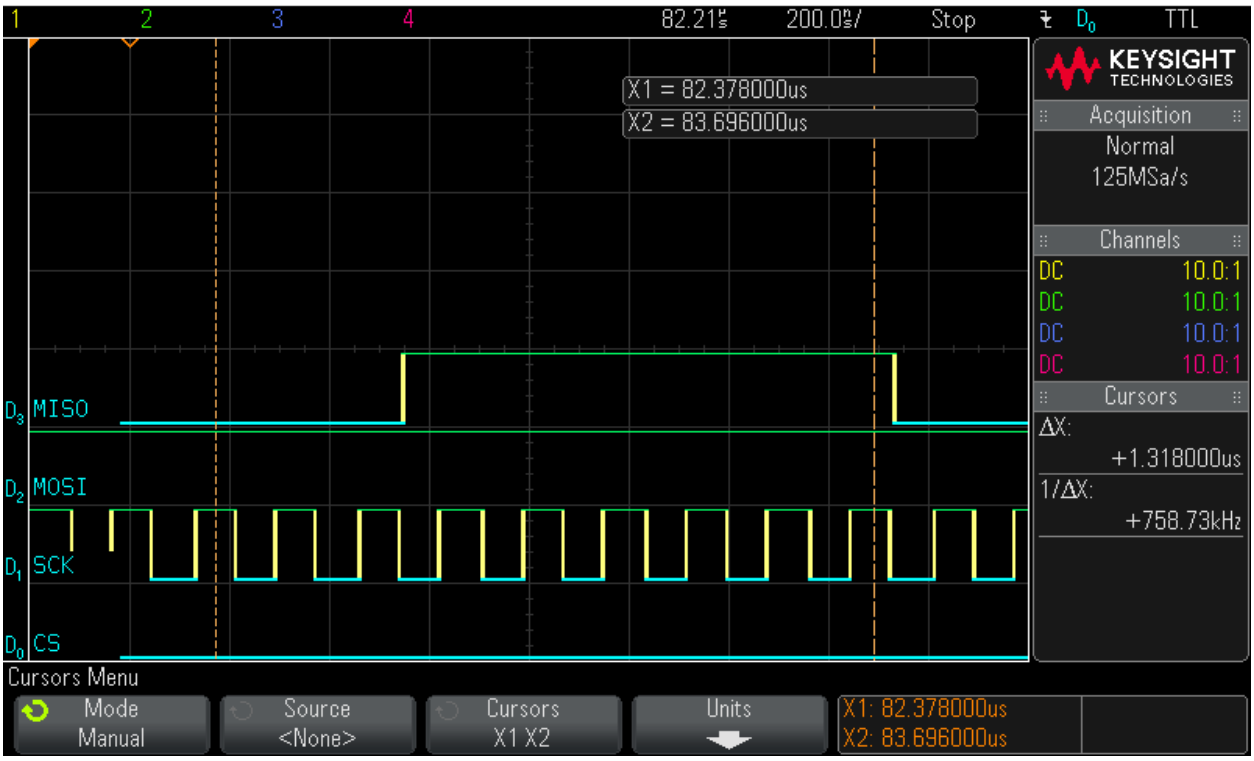


Step 26: Reading out 16 Bytes of data – Byte 2 = 0x80



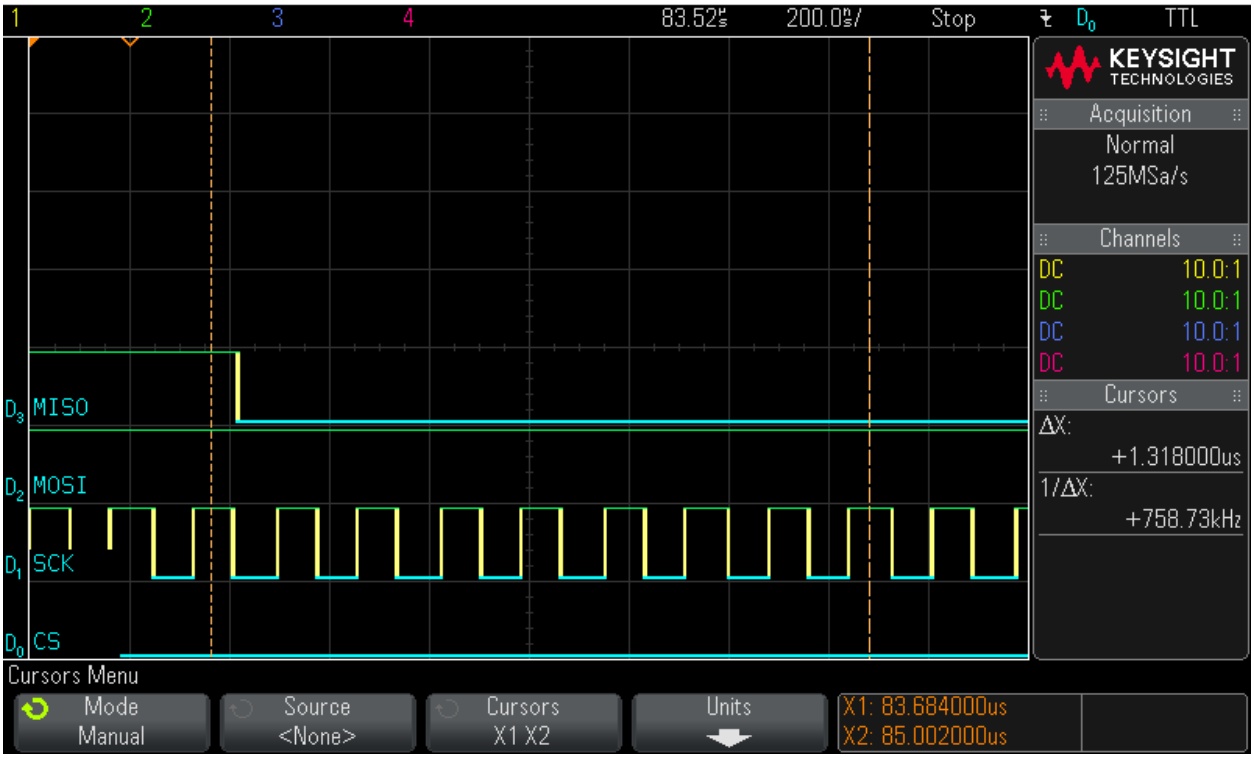
Step 27: Reading out 16 Bytes of data – Byte 3 = 0x3F

MSO-X 2024A, MY52490979: Tue Jul 26 17:54:48 2016

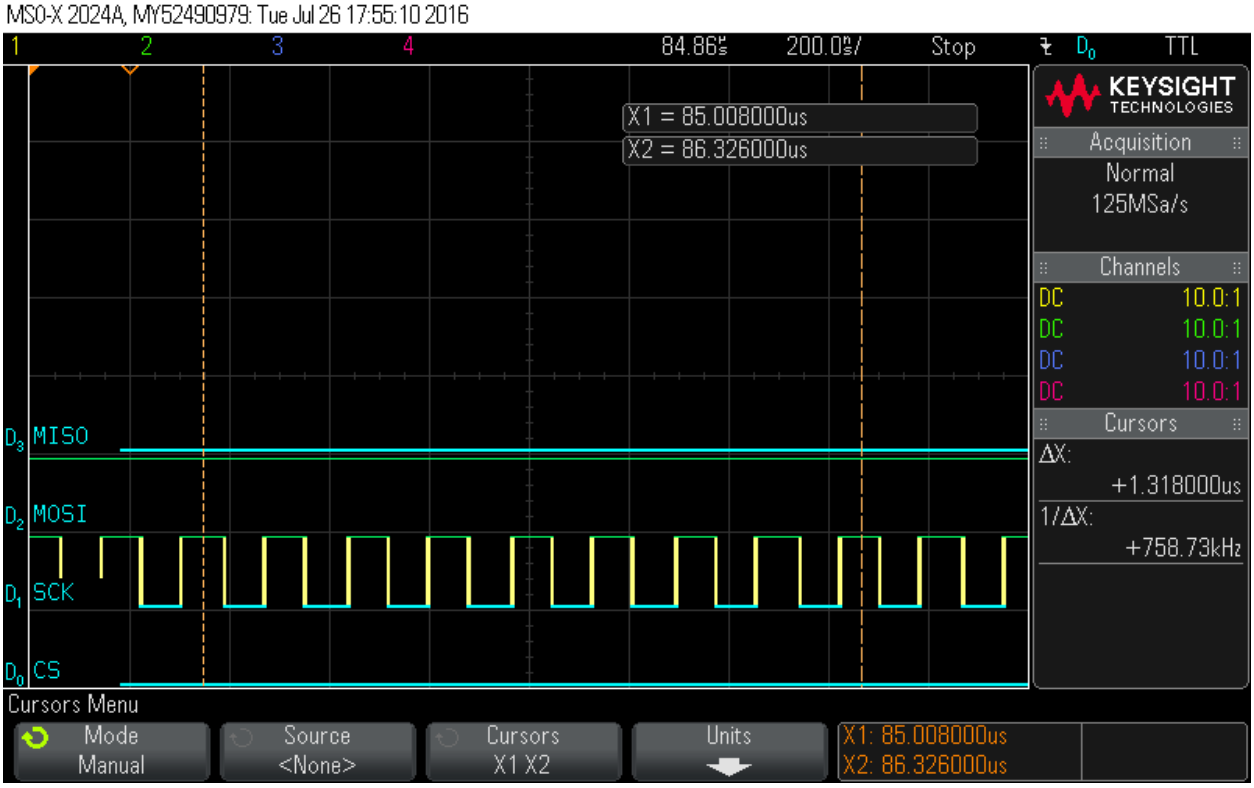


Step 28: Reading out 16 Bytes of data – Byte 4 = 0x0

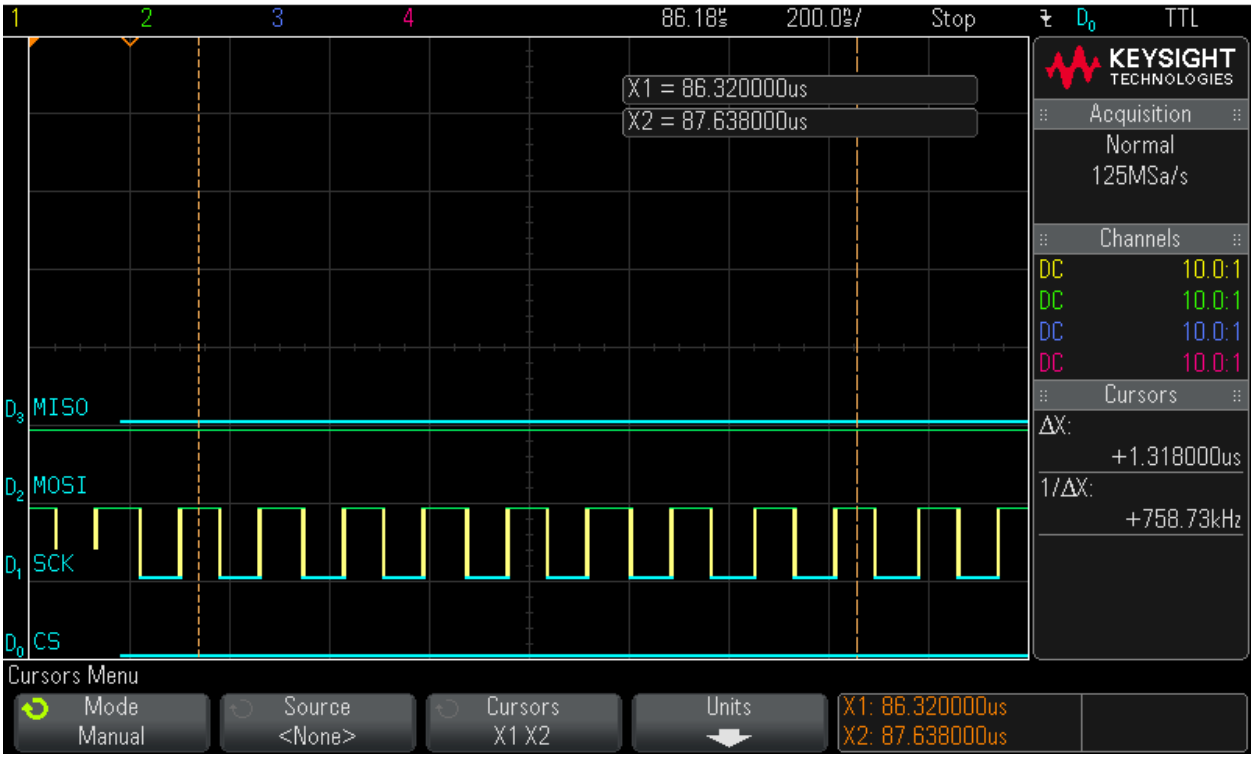
MSO-X 2024A, MY52490979: Tue Jul 26 17:54:57 2016



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

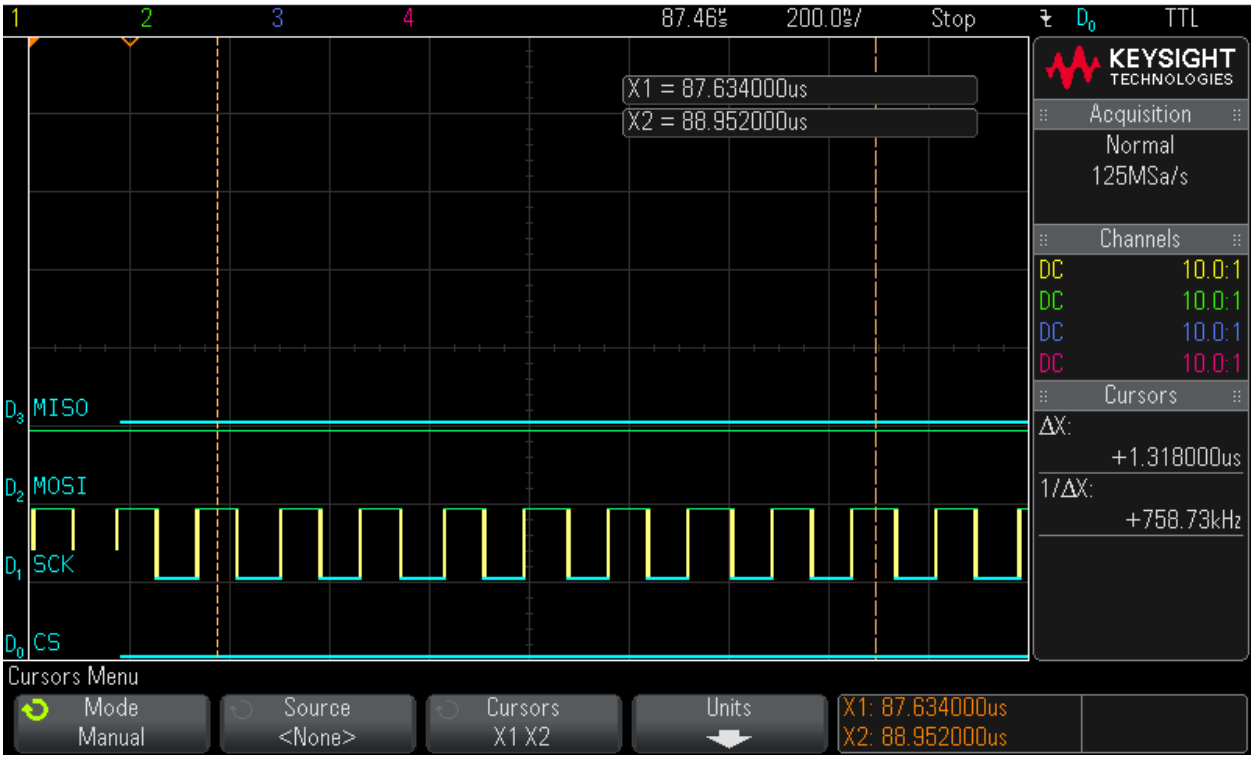


Step 30: Reading out 16 Bytes of data – Byte 6 = 0x0



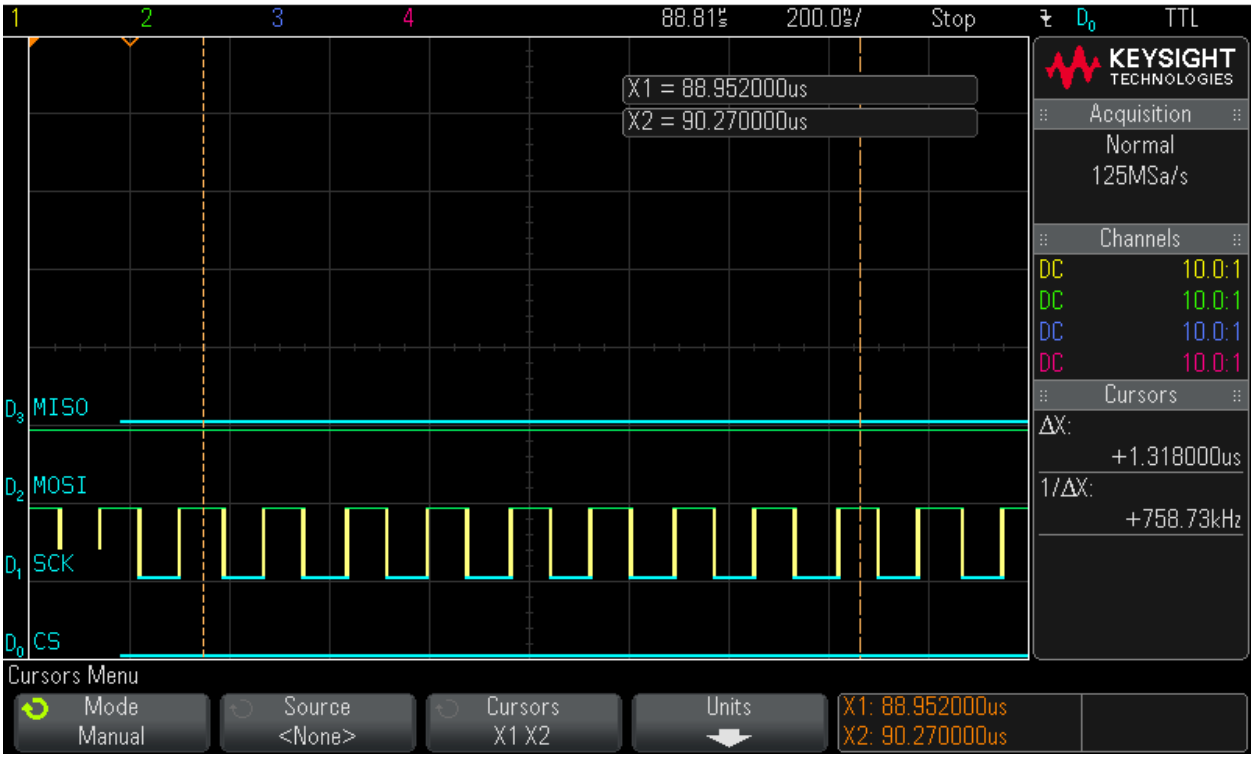
Step 31: Reading out 16 Bytes of data – Byte 7 = 0x0

MSO-X 2024A, MY52490979: Tue Jul 26 17:55:32 2016

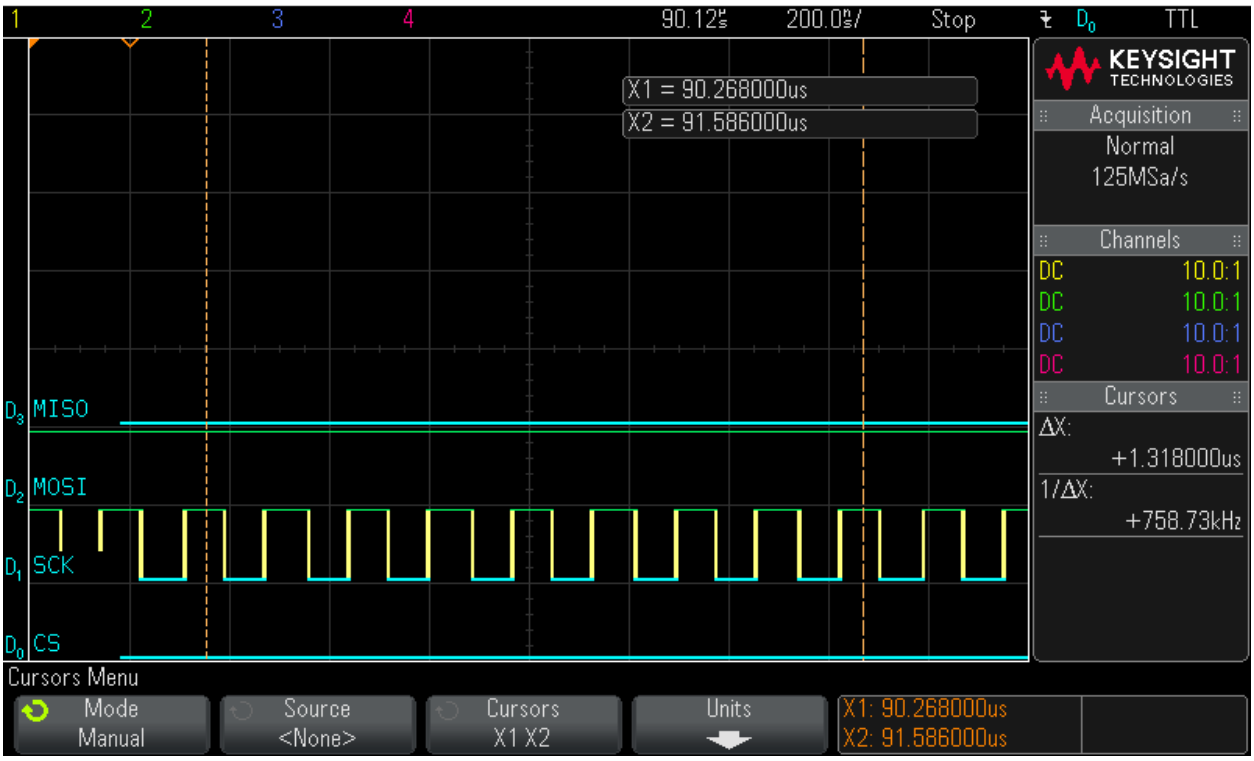


Step 32: Reading out 16 Bytes of data – Byte 8 = 0x0

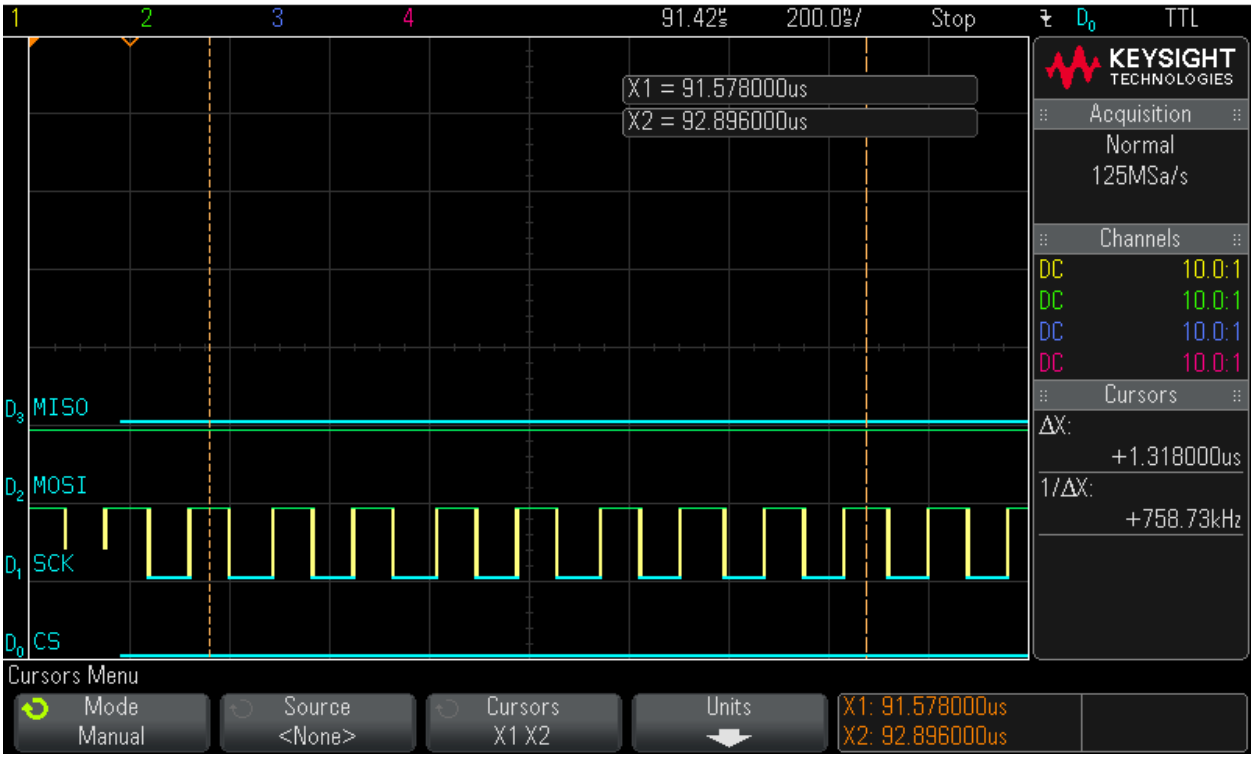
MSO-X 2024A, MY52490979: Tue Jul 26 17:55:41 2016



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

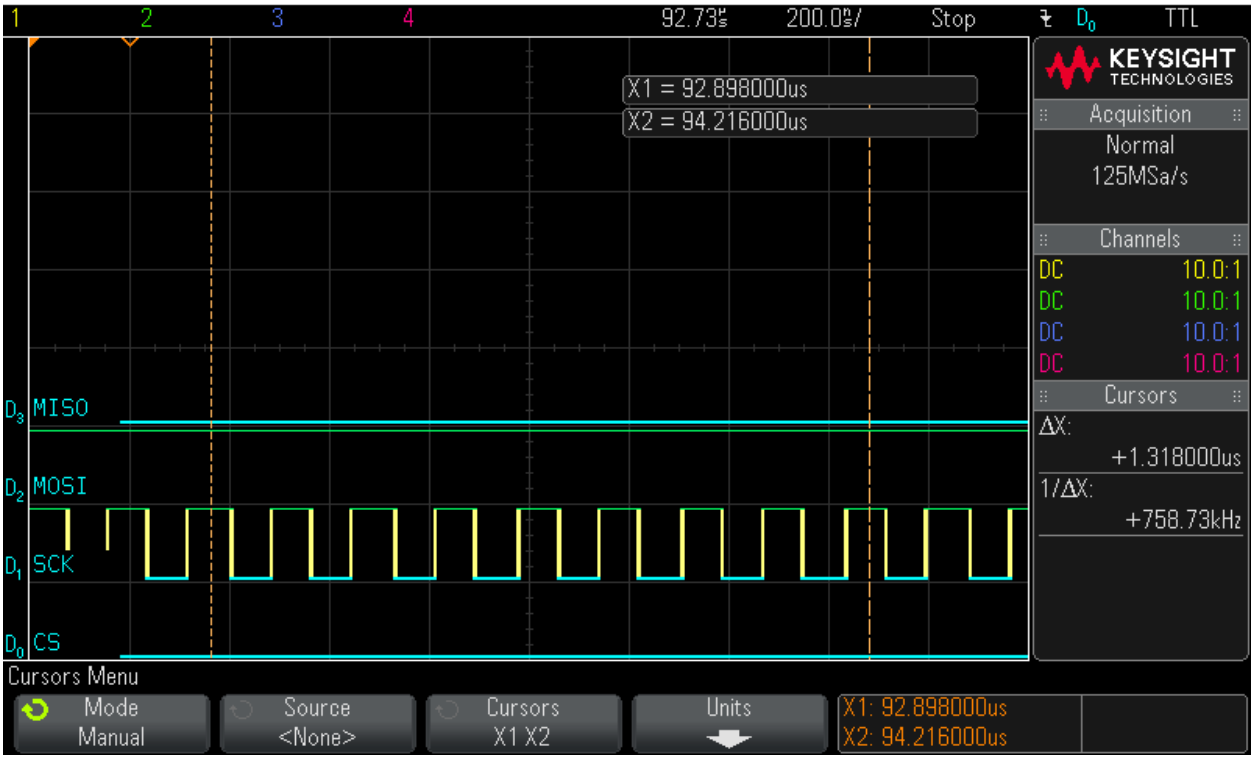


Step 34: Reading out 16 Bytes of data – Byte 10 = 0x0



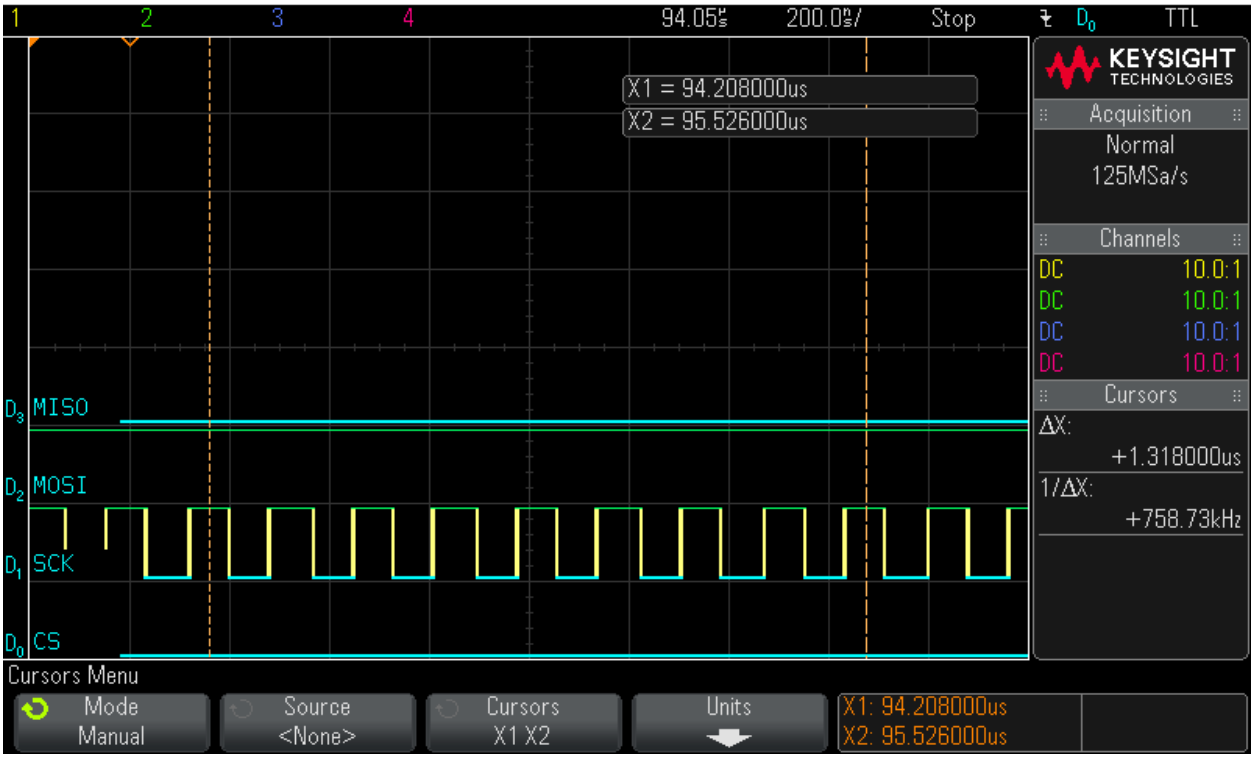
Step 35: Reading out 16 Bytes of data – Byte 11 = 0x0

MSO-X 2024A, MY52490979: Tue Jul 26 17:56:10 2016

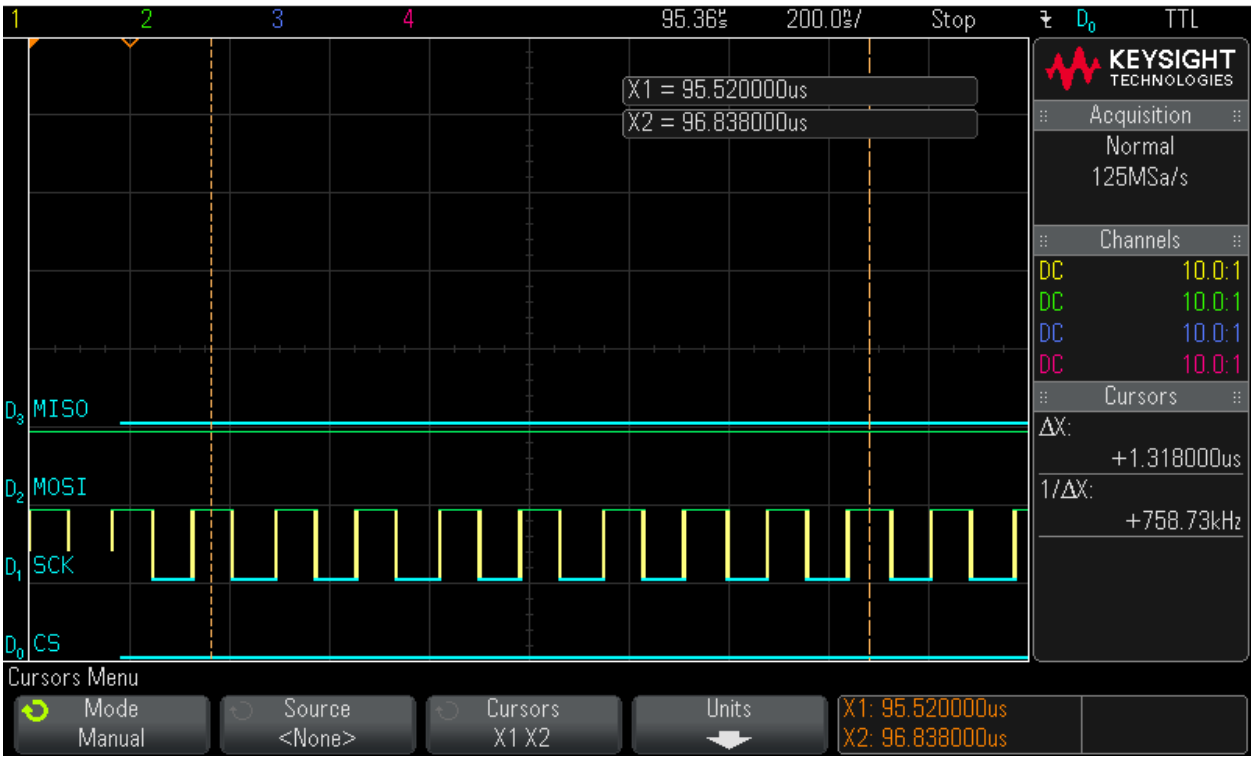


Step 36: Reading out 16 Bytes of data – Byte 12 = 0x0

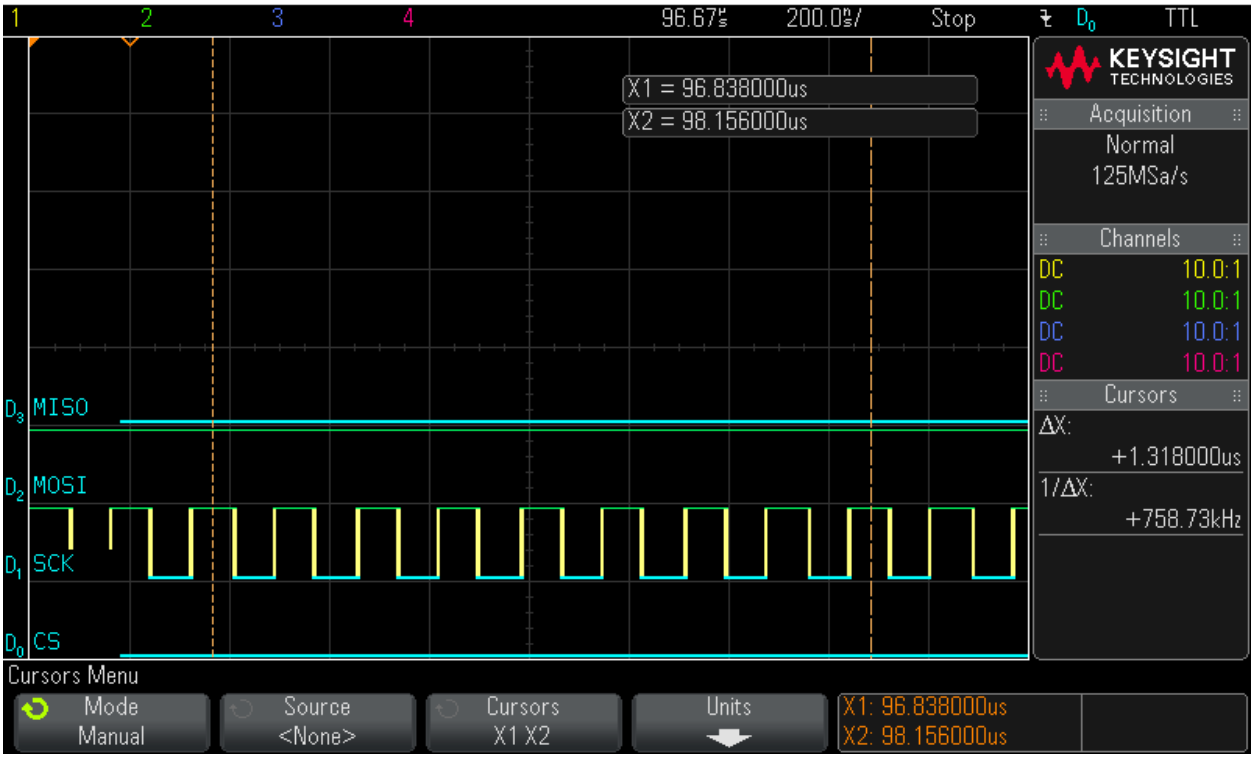
MSO-X 2024A, MY52490979: Tue Jul 26 17:56:20 2016



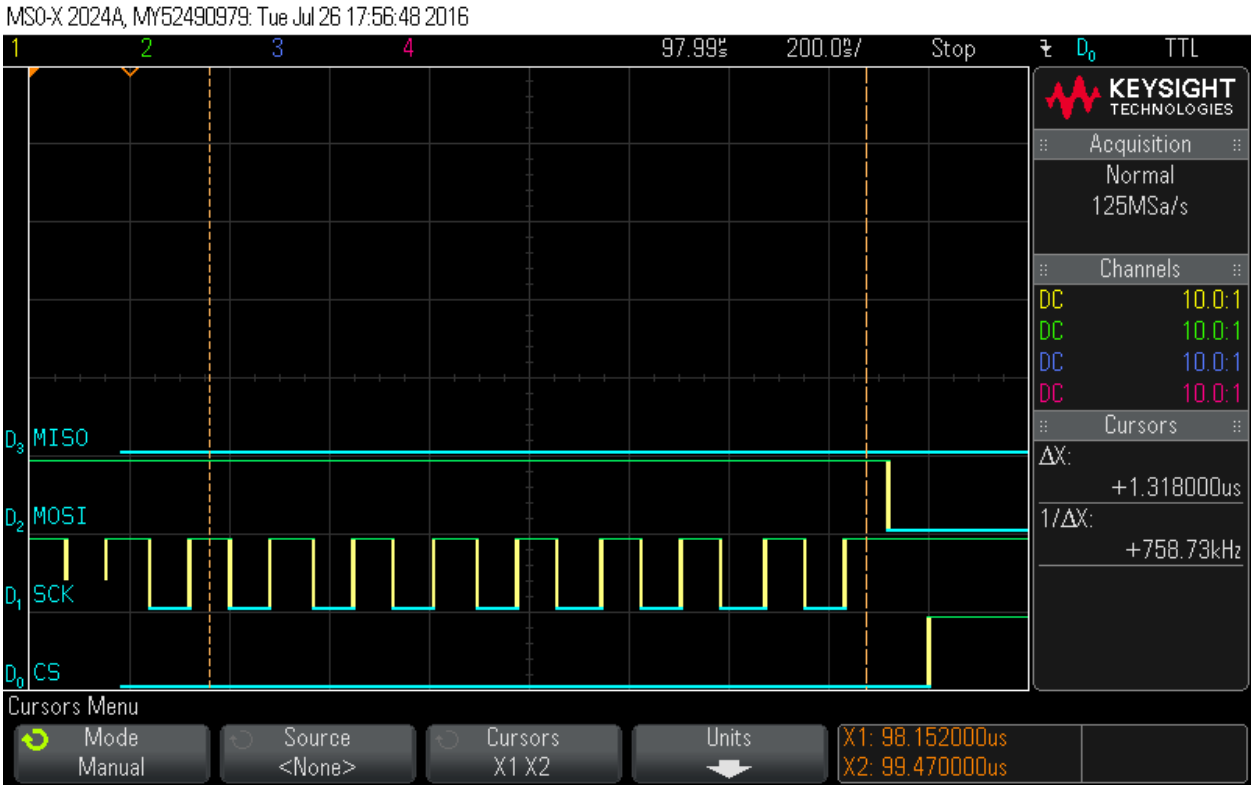
Step 37: Reading out 16 Bytes of data – Byte 13 = 0x0



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



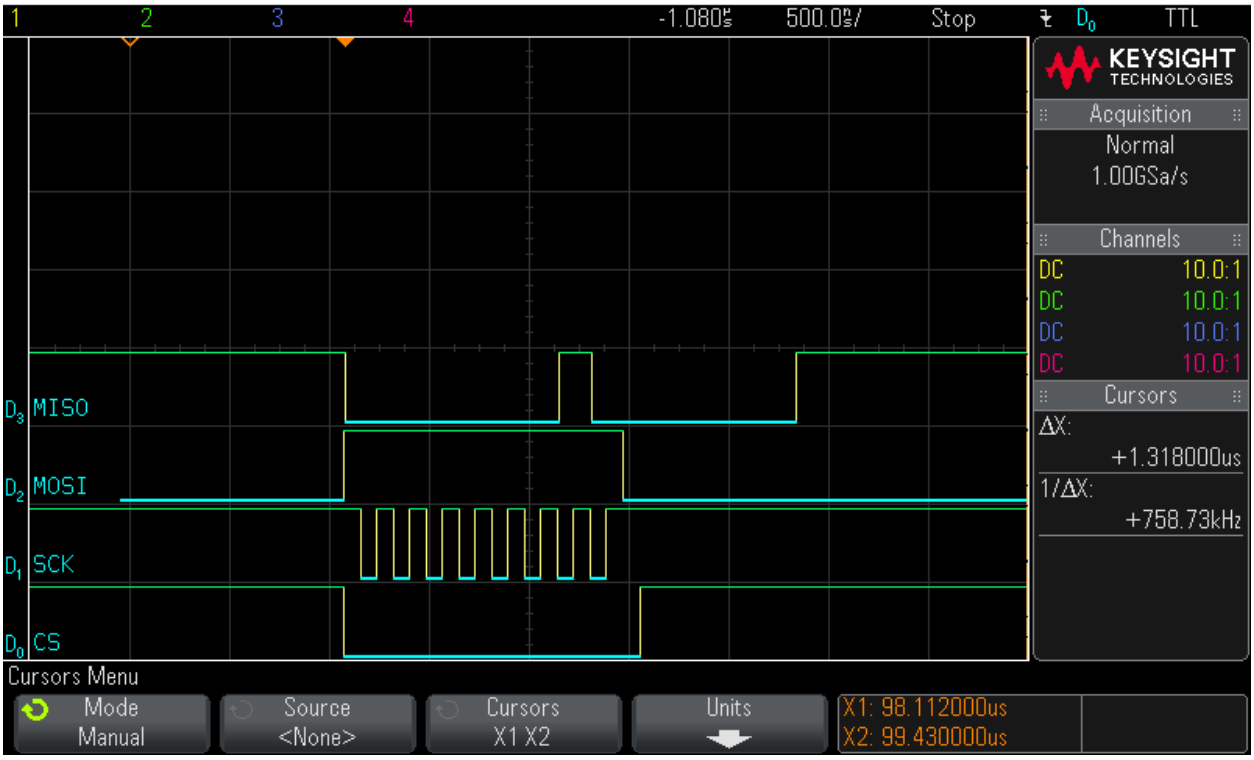
Step 39: Reading out 16 Bytes of data – Byte 15 = 0x0



Read FSN waveform:

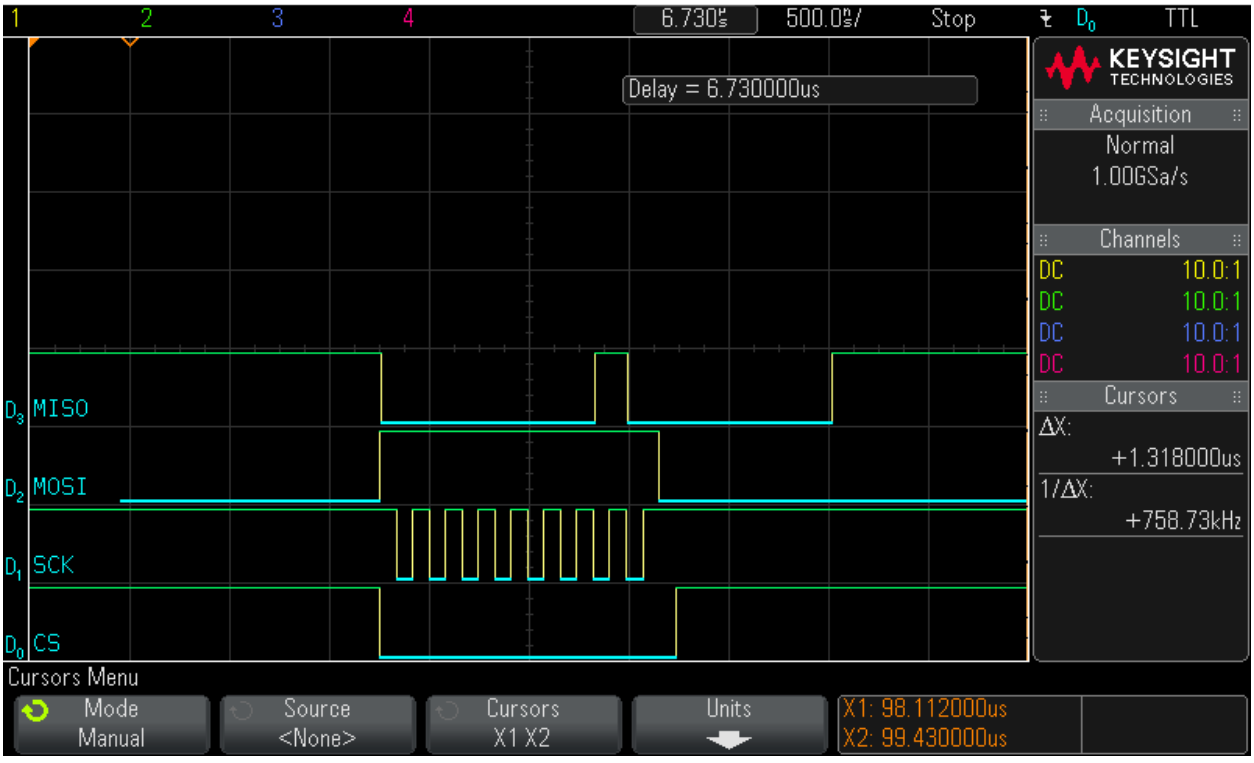
Step 1: Hardware Status Check

MSO-X 2024A, MY52490979: Tue Jul 26 16:57:30 2016

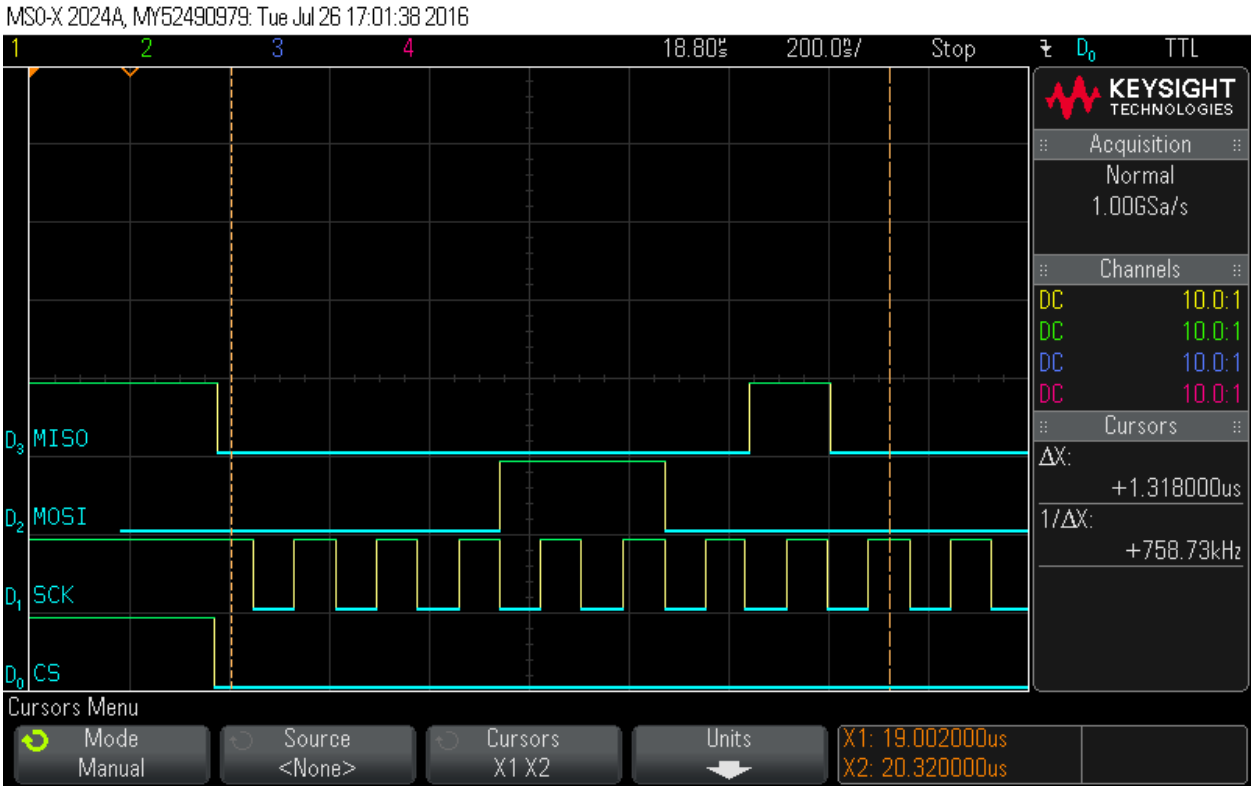


Step 2: Hardware Status Check

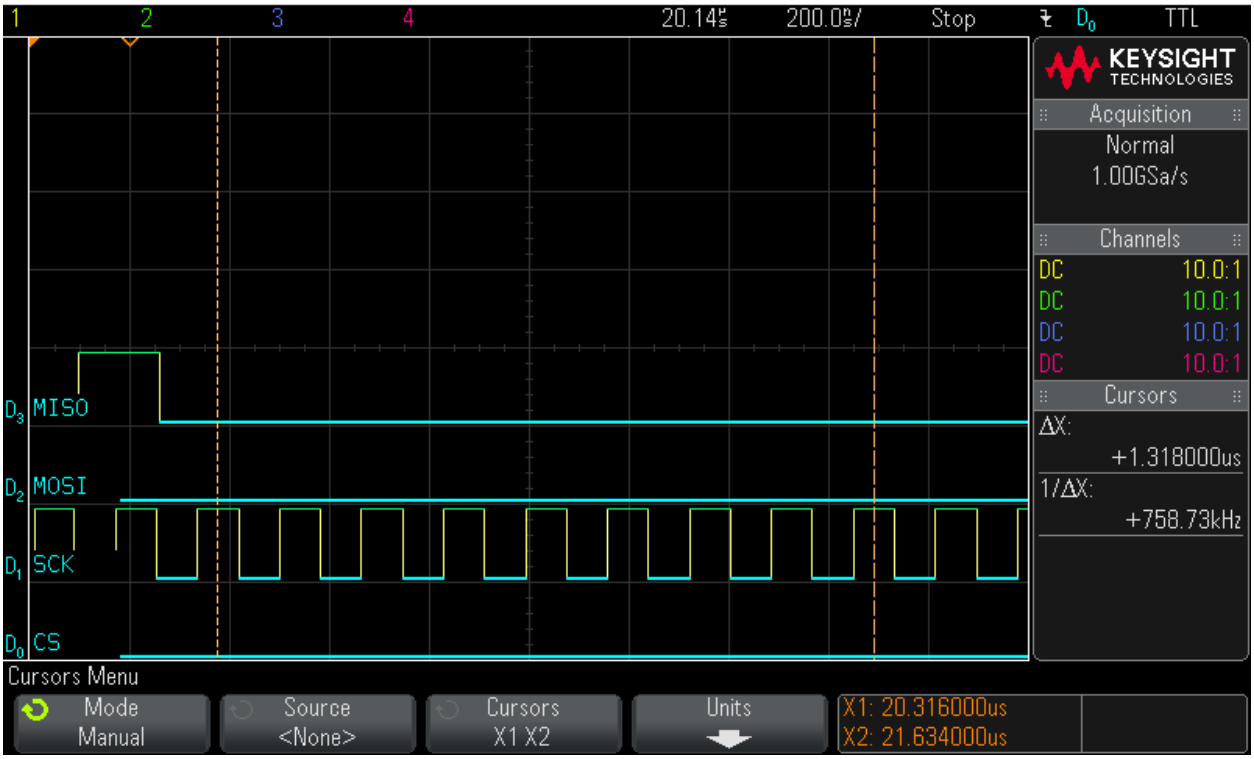
MSO-X 2024A, MY52490979: Tue Jul 26 16:57:38 2016



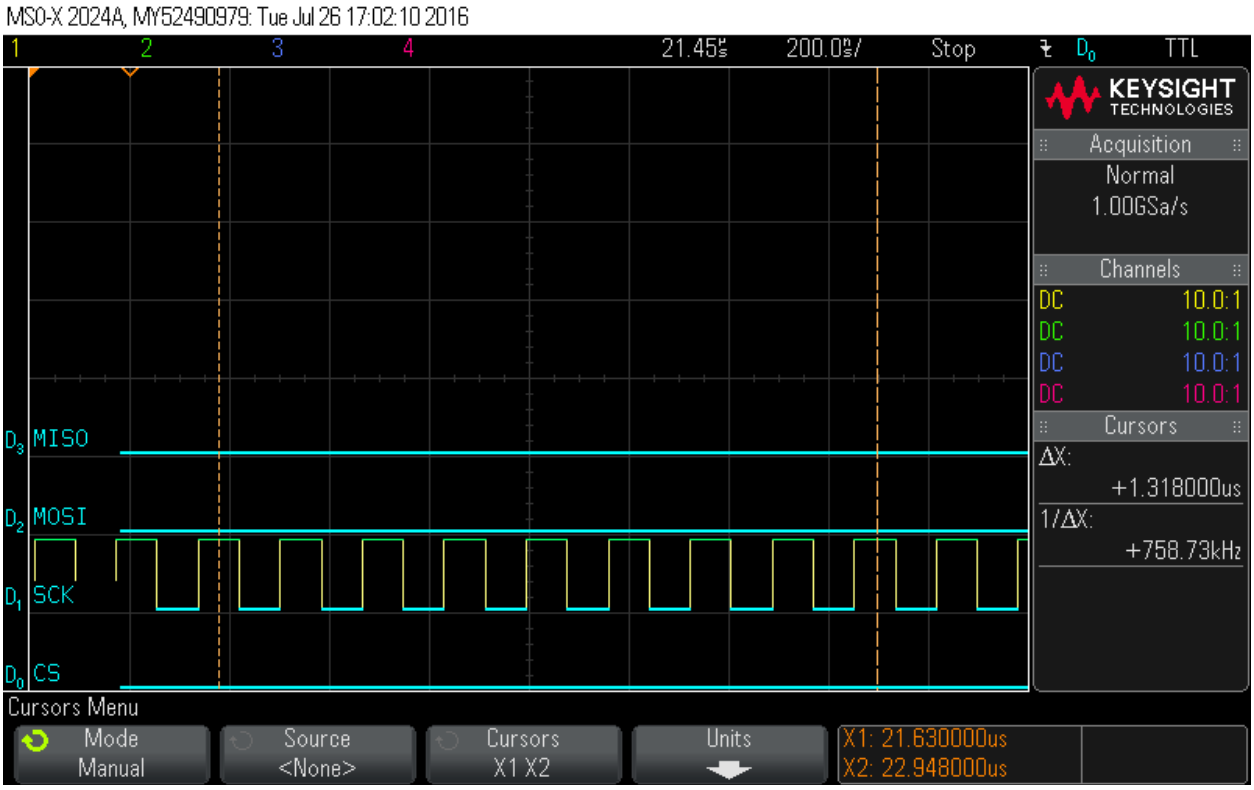
Step 3: Clock in read_FSN command (0x18)



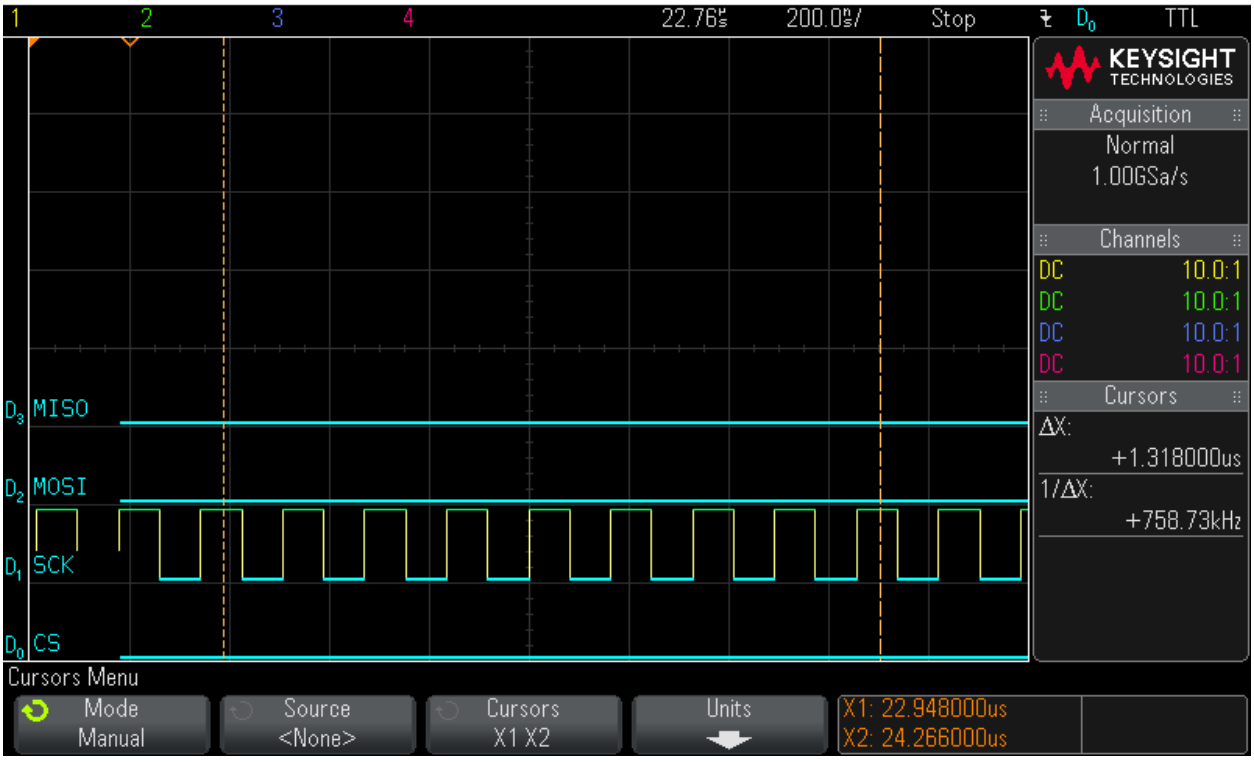
Step 4: Clock in 16 bytes of zero values - Byte 0



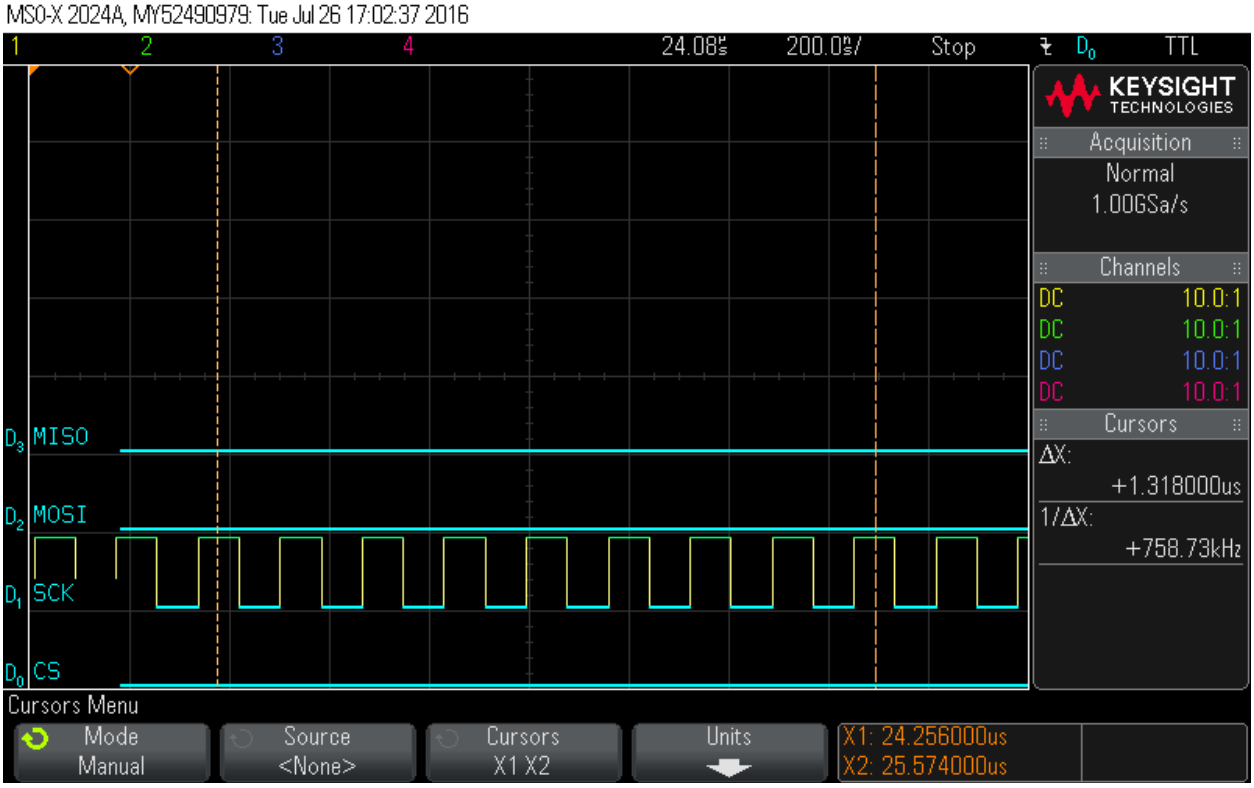
Step 5: Clock in 16 bytes of zero values - Byte 1



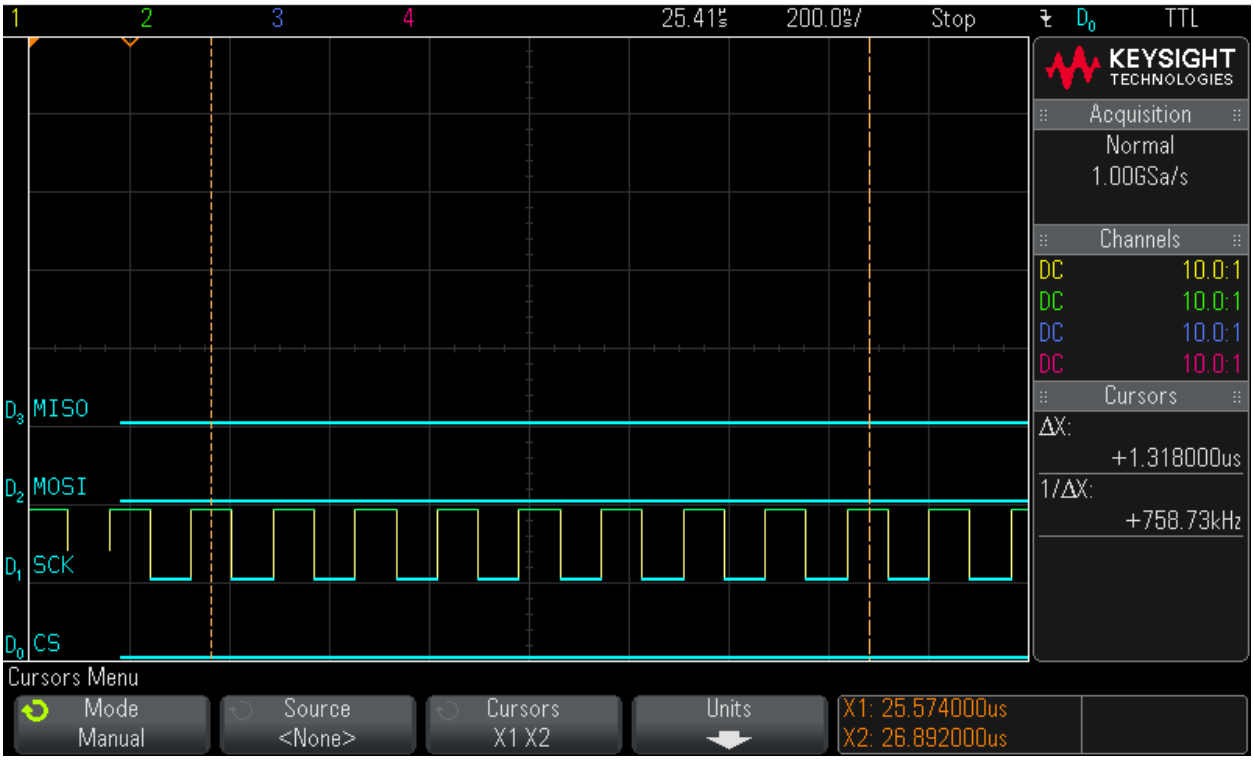
Step 6: Clock in 16 bytes of zero values - Byte 2



Step 7: Clock in 16 bytes of zero values - Byte 3

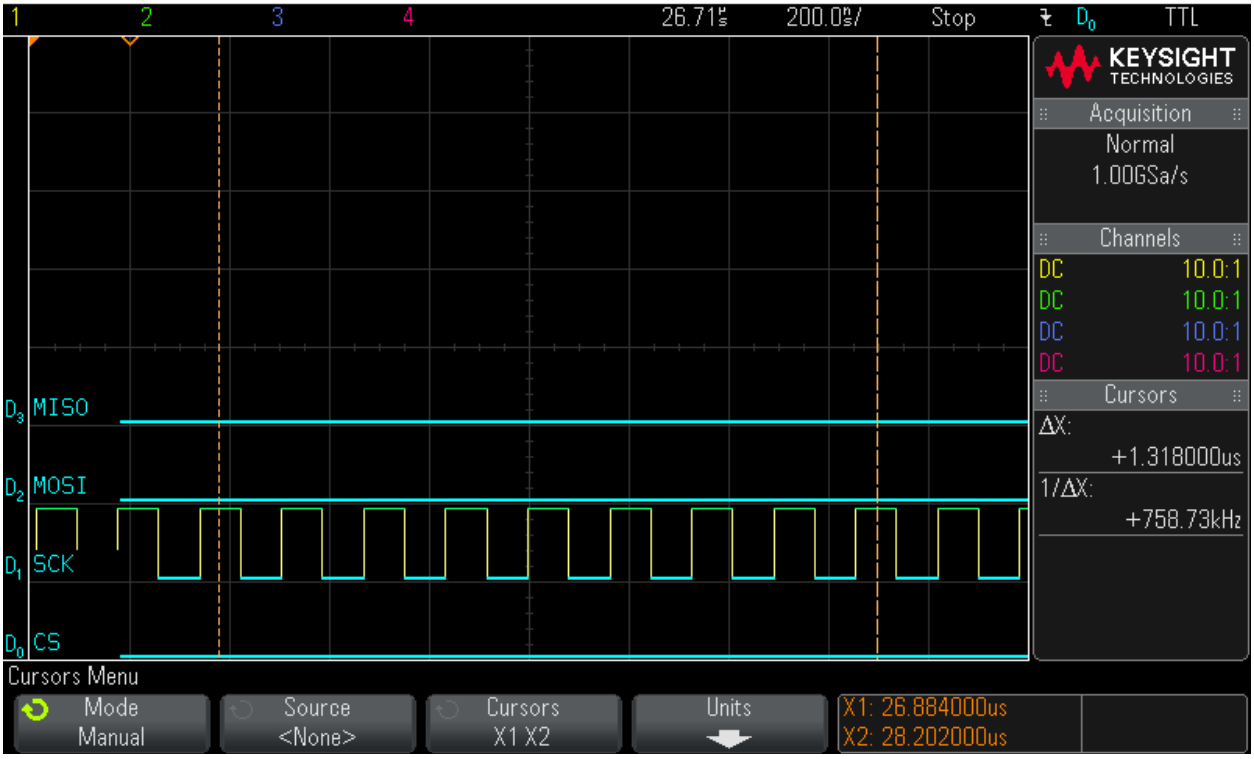


Step 8: Clock in 16 bytes of zero values - Byte 4



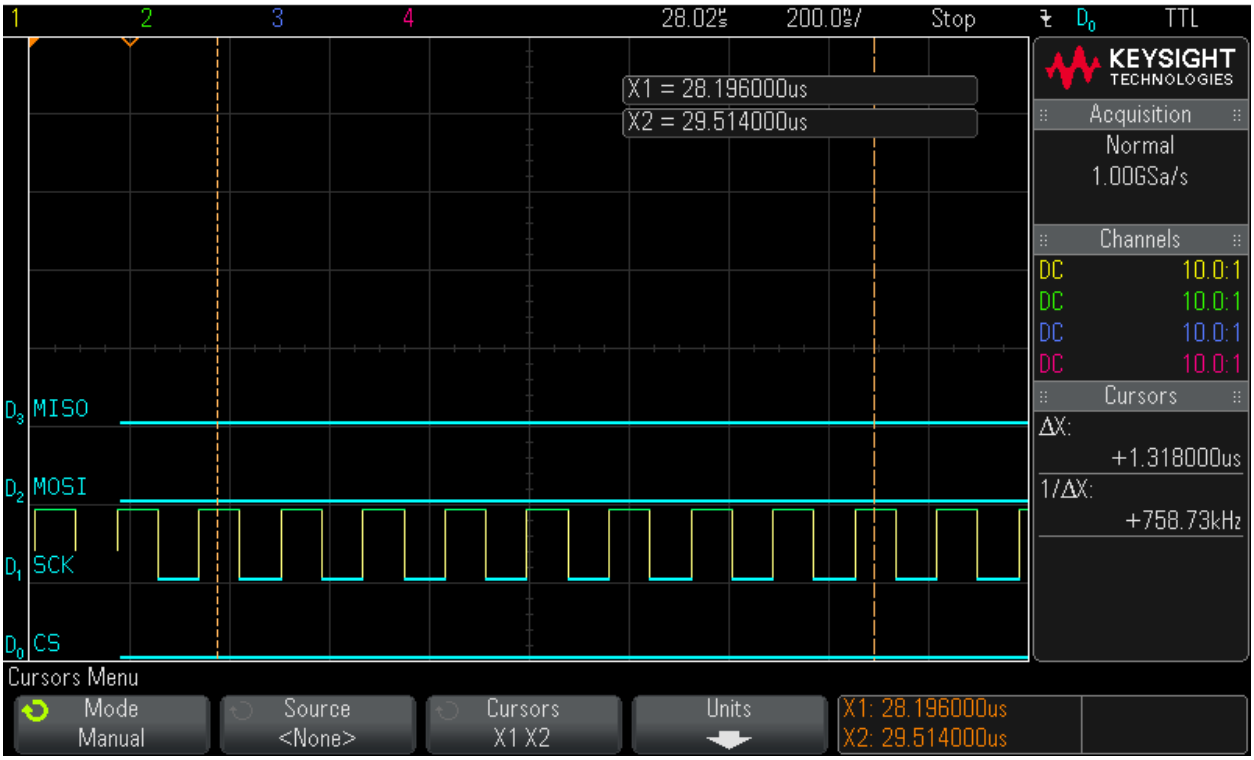
Step 9: Clock in 16 bytes of zero values - Byte 5

MSO-X 2024A, MY52490979: Tue Jul 26 17:03:01 2016

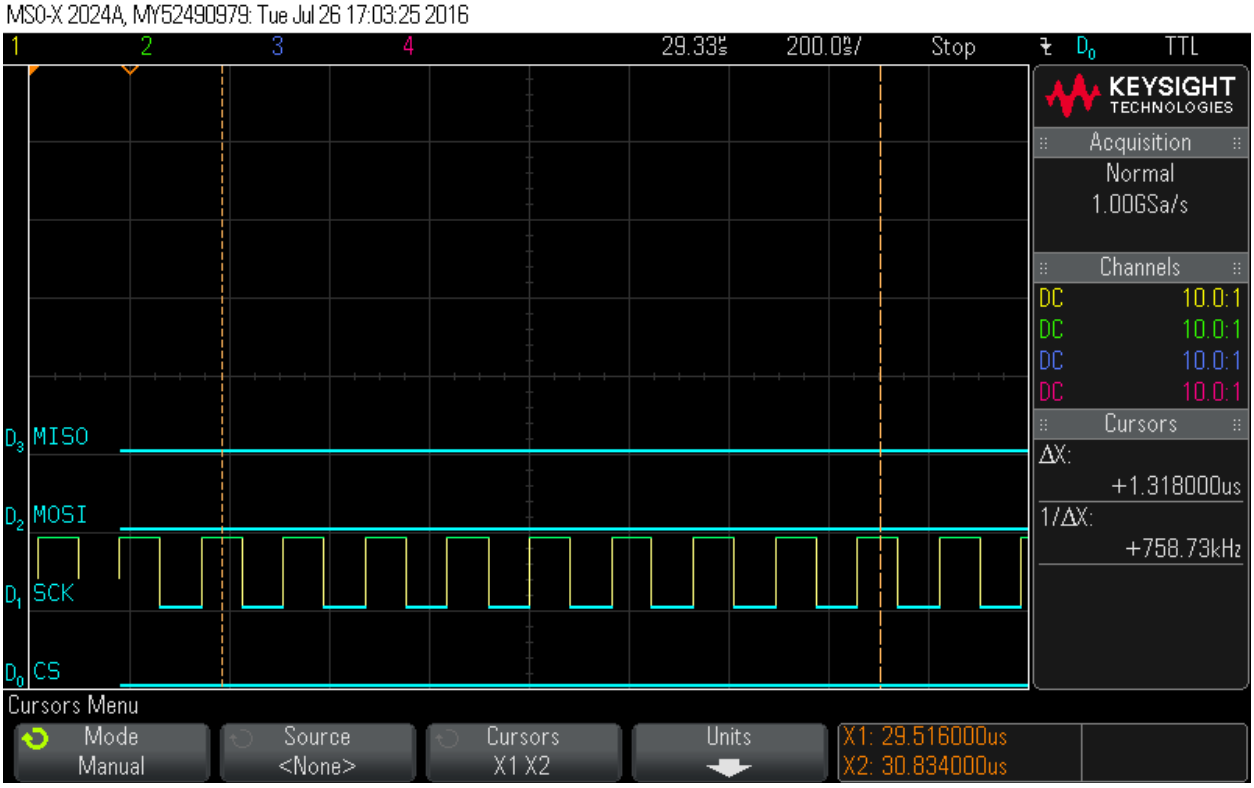


Step 10: Clock in 16 bytes of zero values - Byte 6

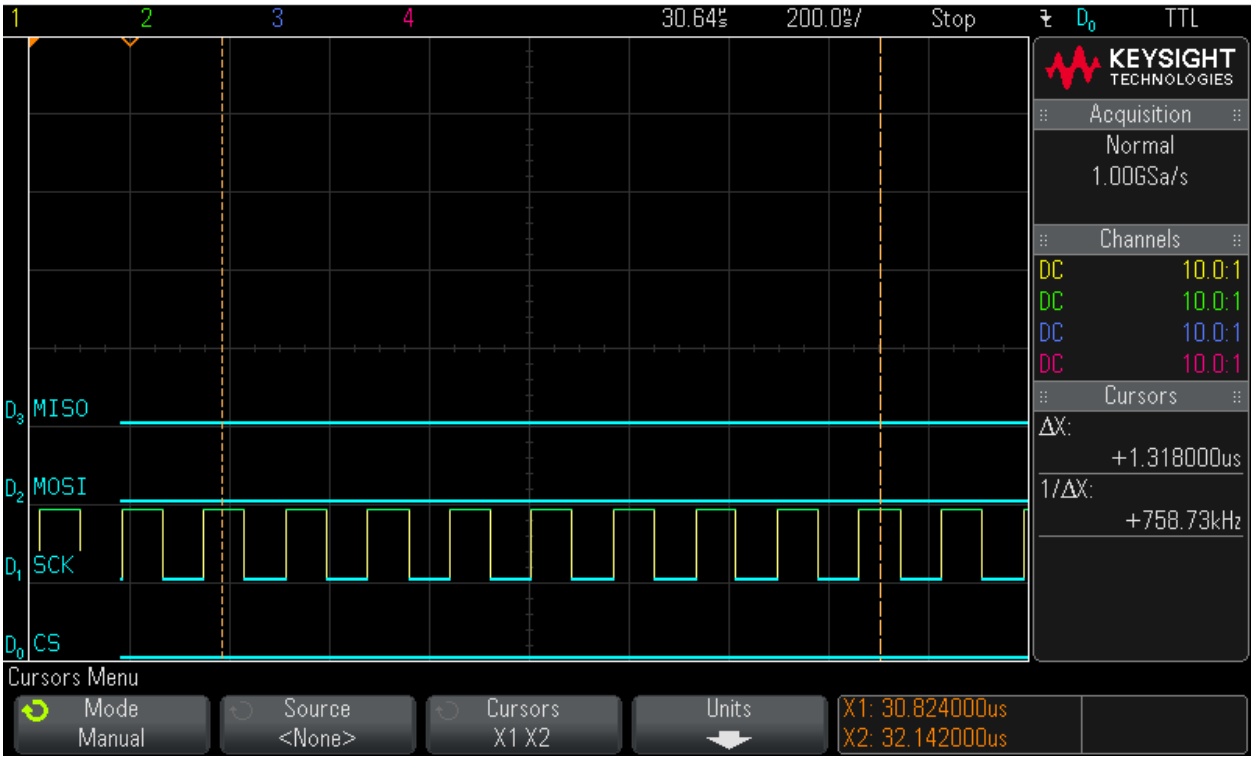
MSO-X 2024A, MY52490979: Tue Jul 26 17:03:12 2016



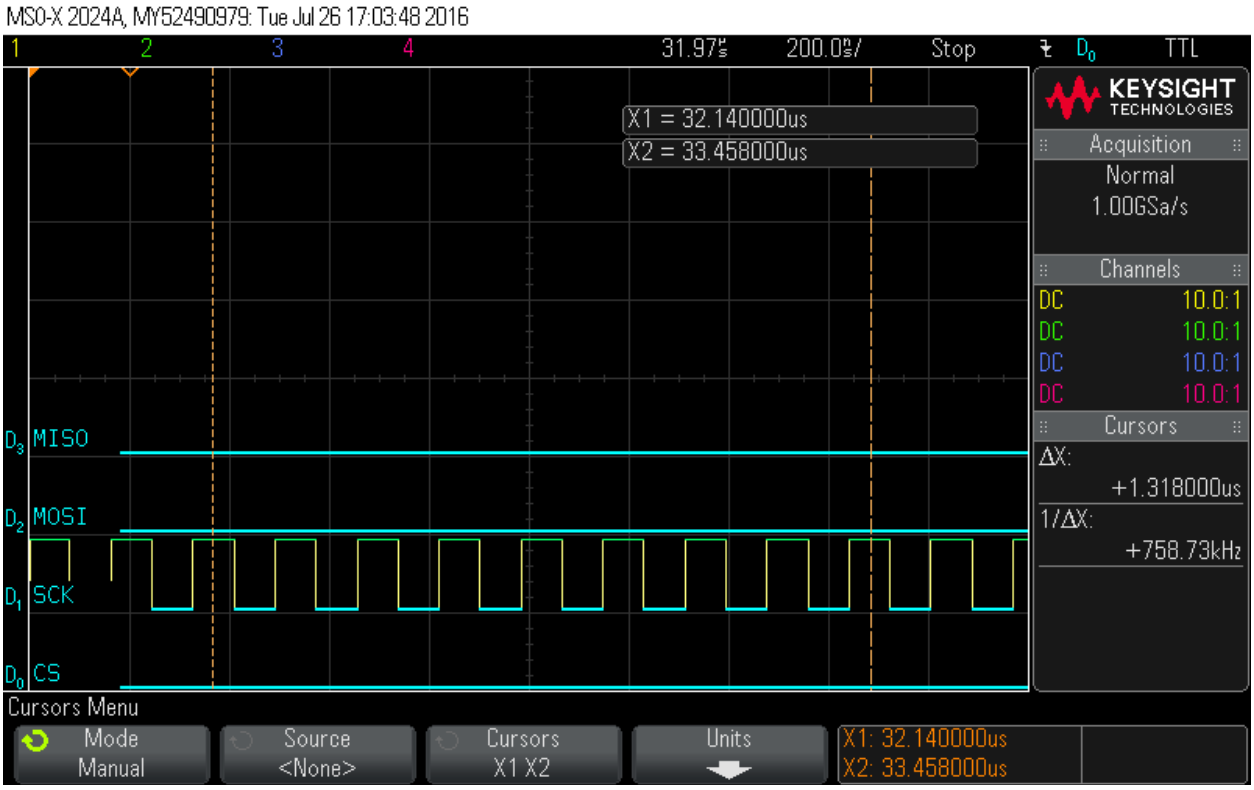
Step 11: Clock in 16 bytes of zero values - Byte 7



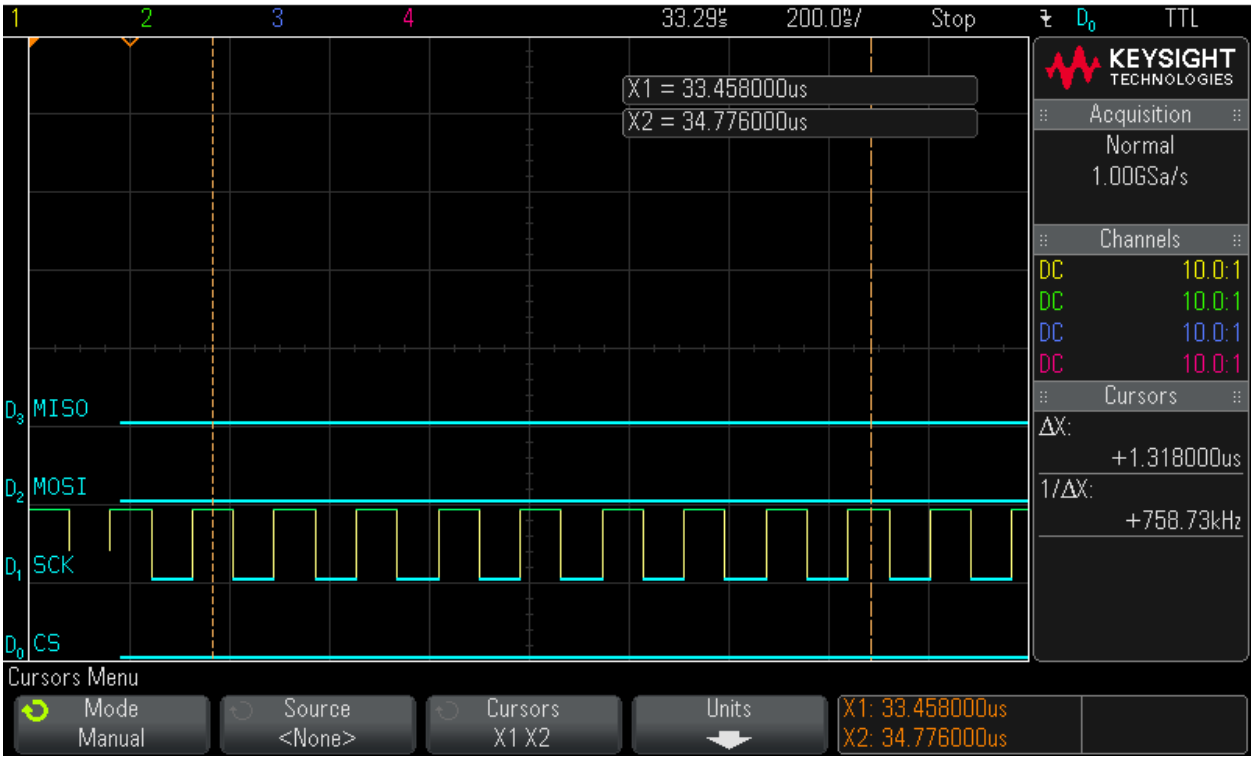
Step 12: Clock in 16 bytes of zero values - Byte 8



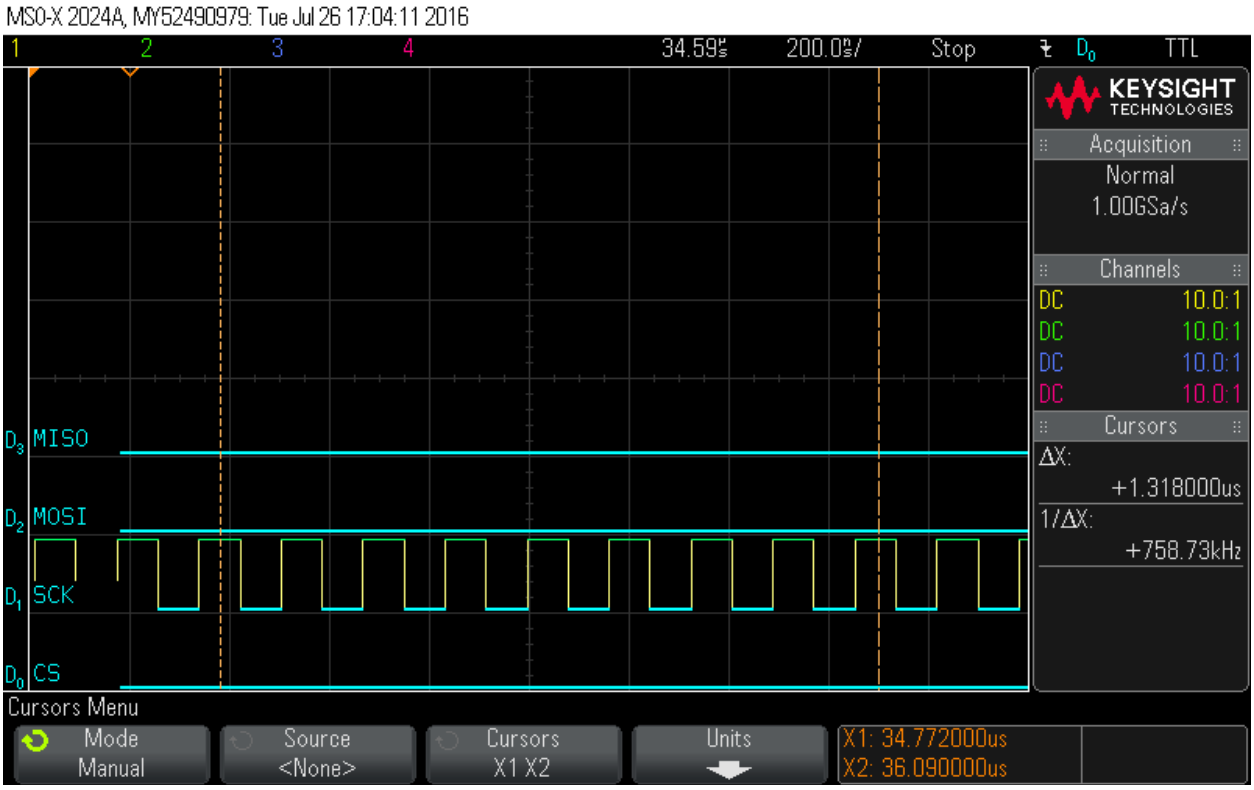
Step 13: Clock in 16 bytes of zero values - Byte 9



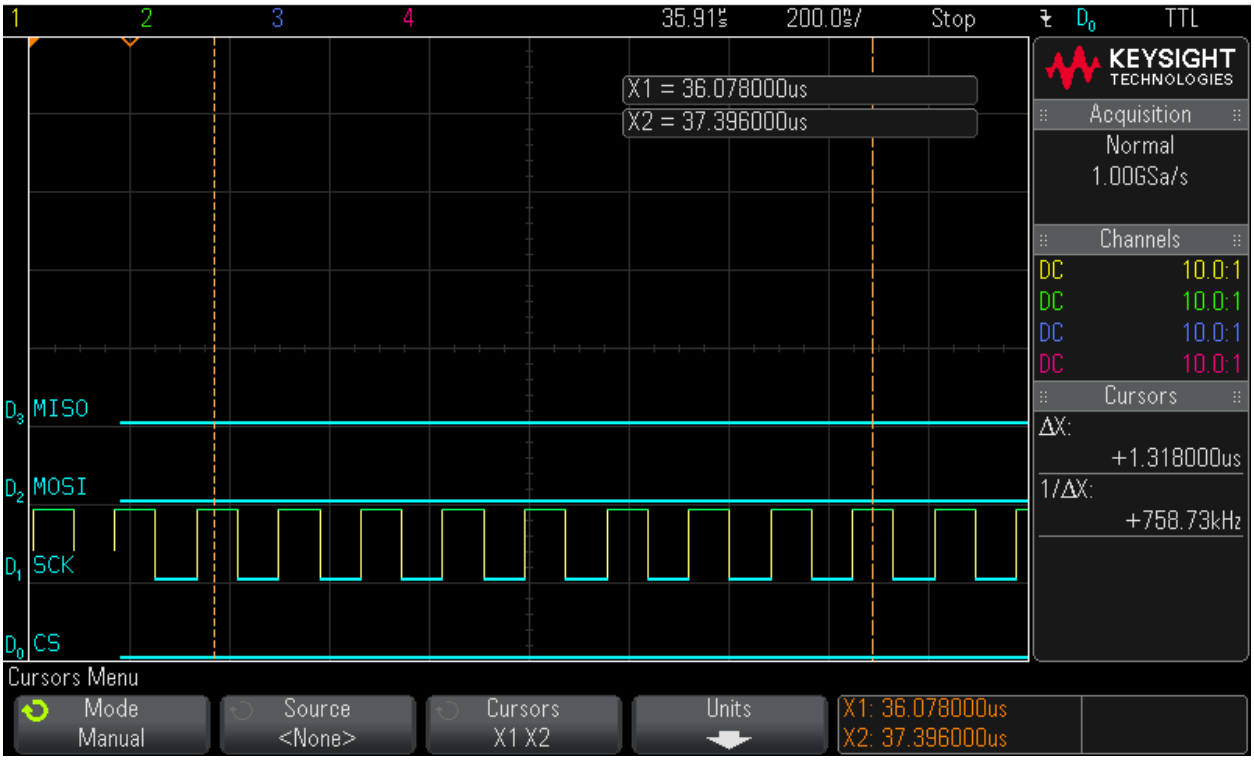
Step 14: Clock in 16 bytes of zero values - Byte 10



Step 15: Clock in 16 bytes of zero values - Byte 11

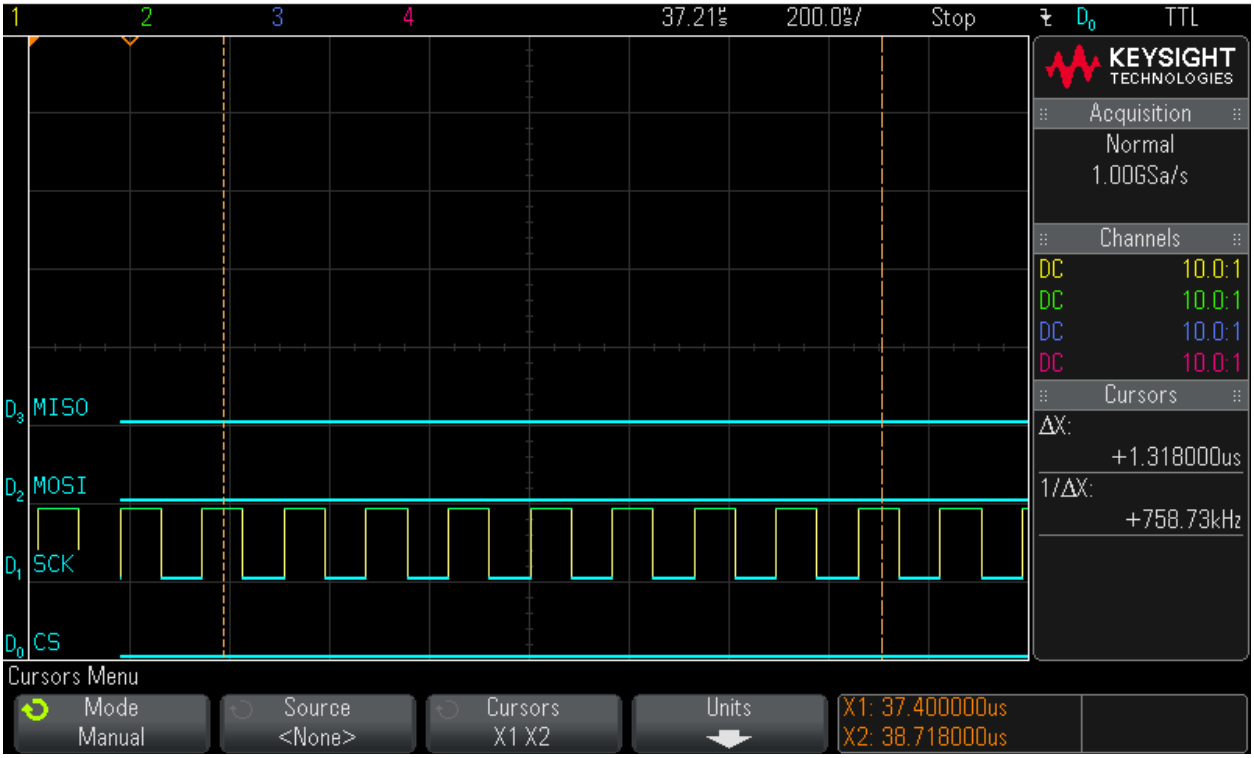


Step 16: Clock in 16 bytes of zero values - Byte 12



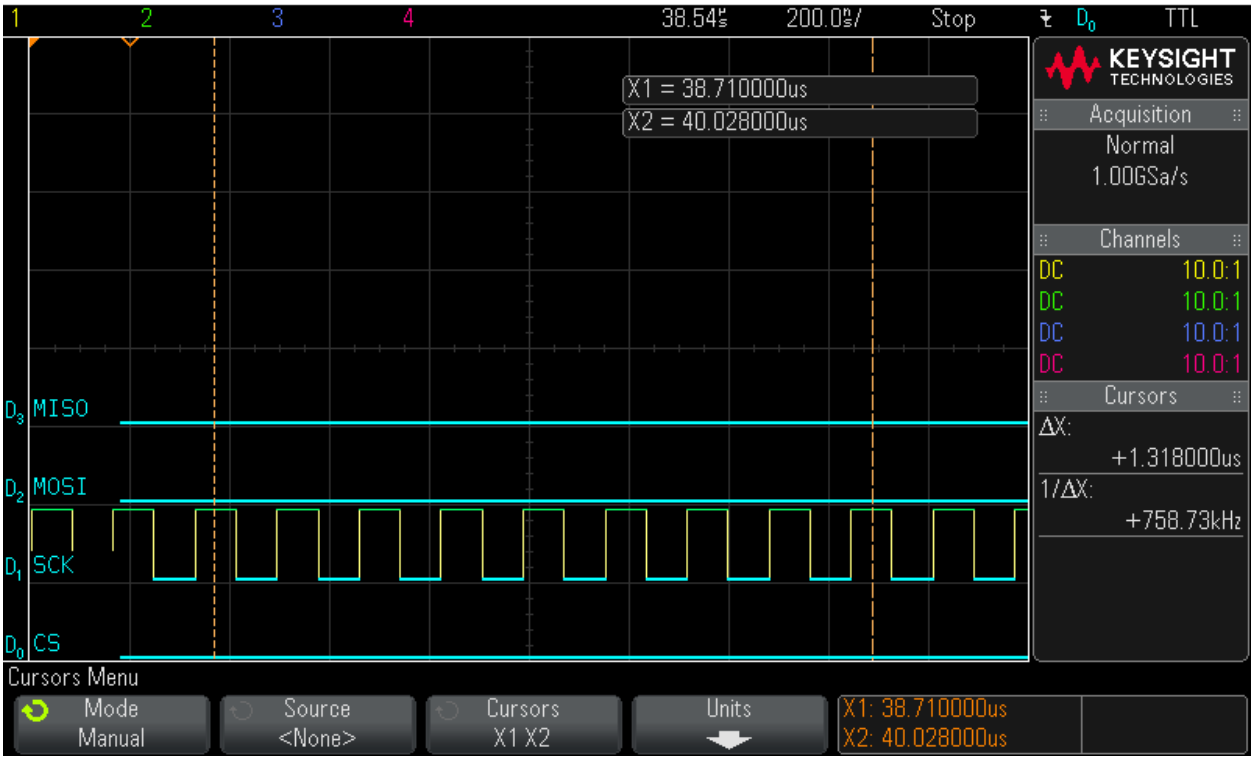
Step 17: Clock in 16 bytes of zero values - Byte 13

MSO-X 2024A, MY52490979: Tue Jul 26 17:04:32 2016



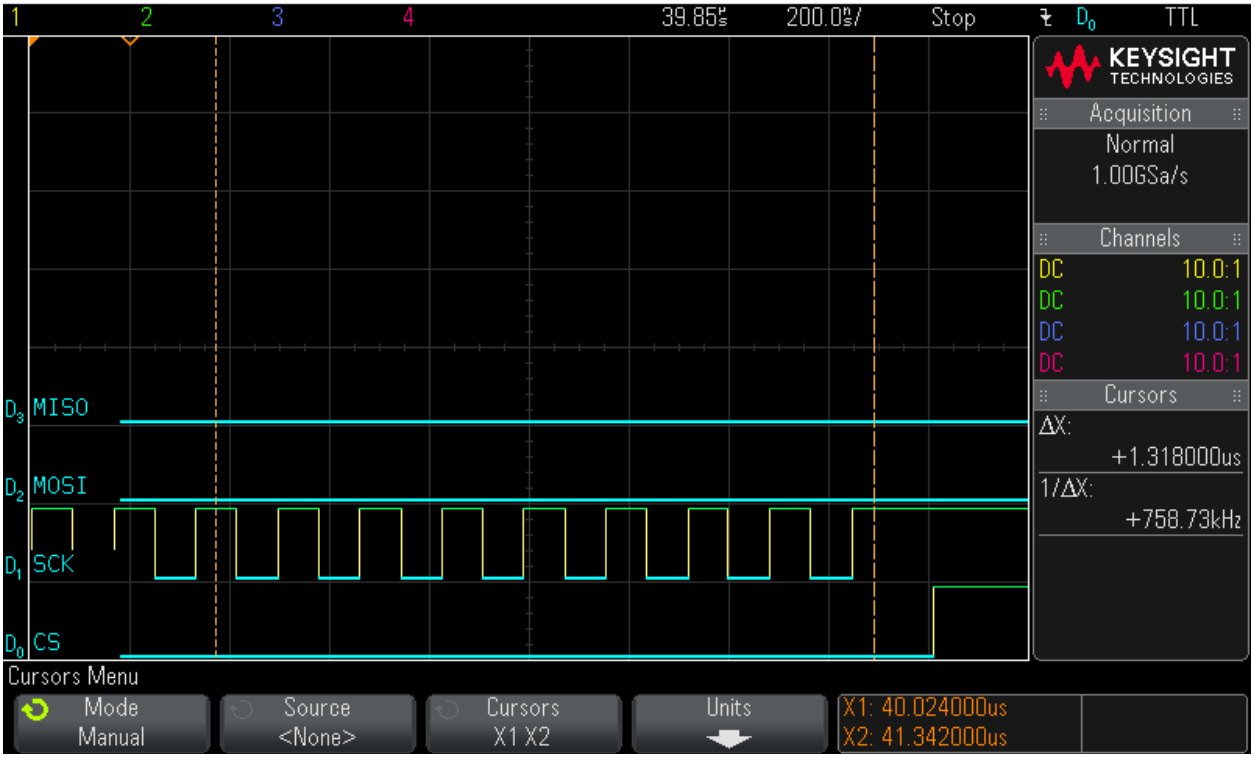
Step 18: Clock in 16 bytes of zero values - Byte 14

MSO-X 2024A, MY52490979: Tue Jul 26 17:04:42 2016



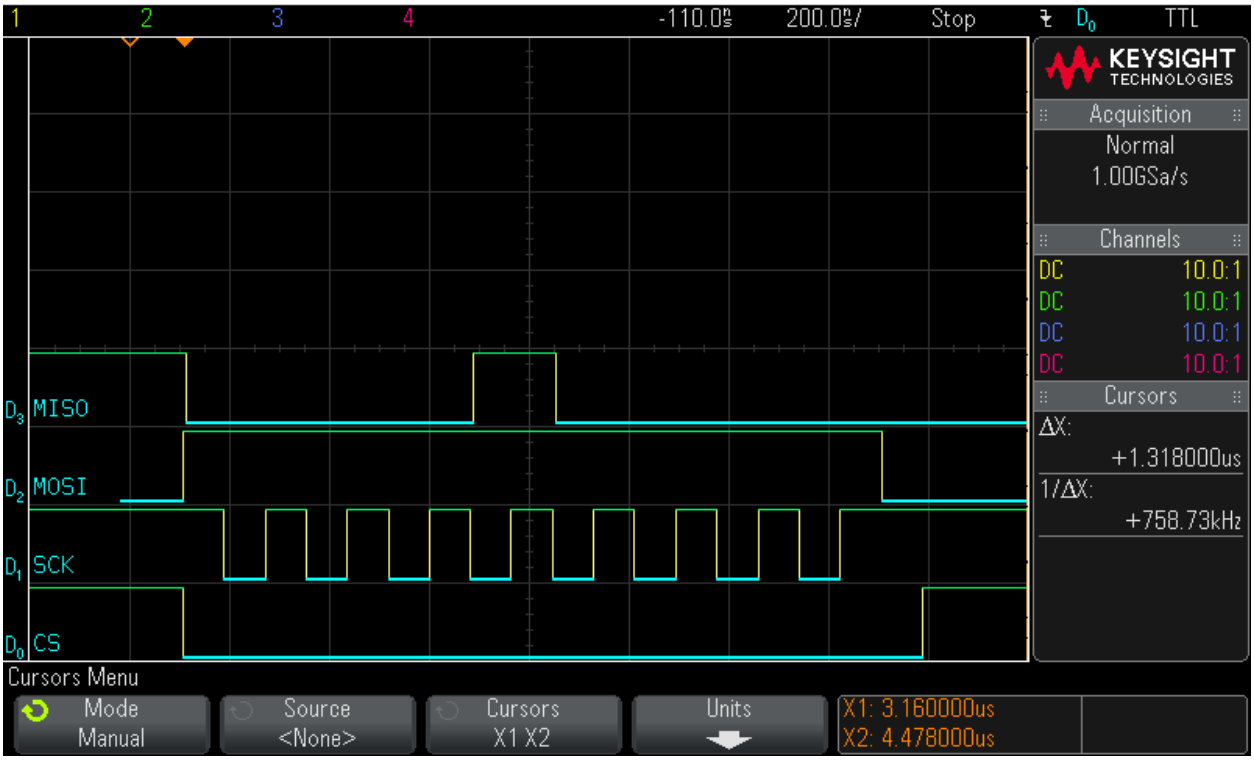
Step 19: Clock in 16 bytes of zero values - Byte 15

MSO-X 2024A, MY52490979: Tue Jul 26 17:04:56 2016



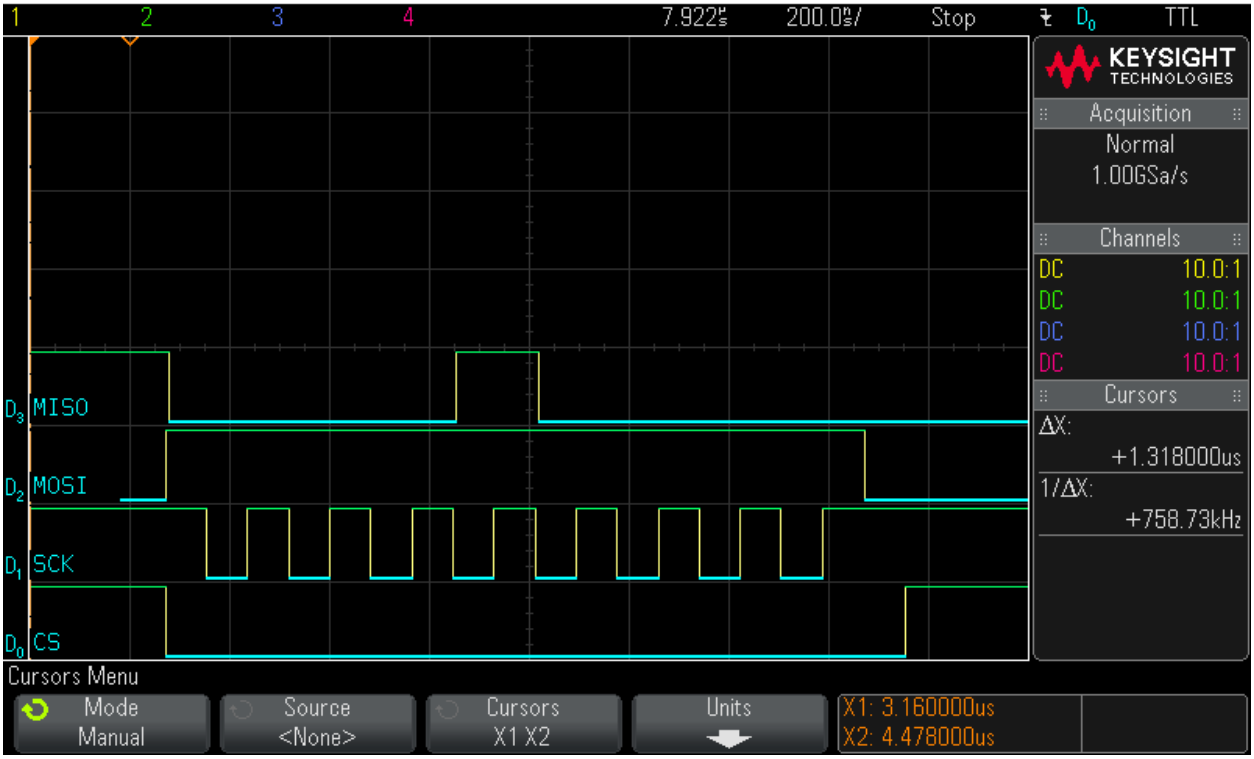
Step 20: Hardware Status Check

MSO-X 2024A, MY52490979: Tue Jul 26 17:25:33 2016



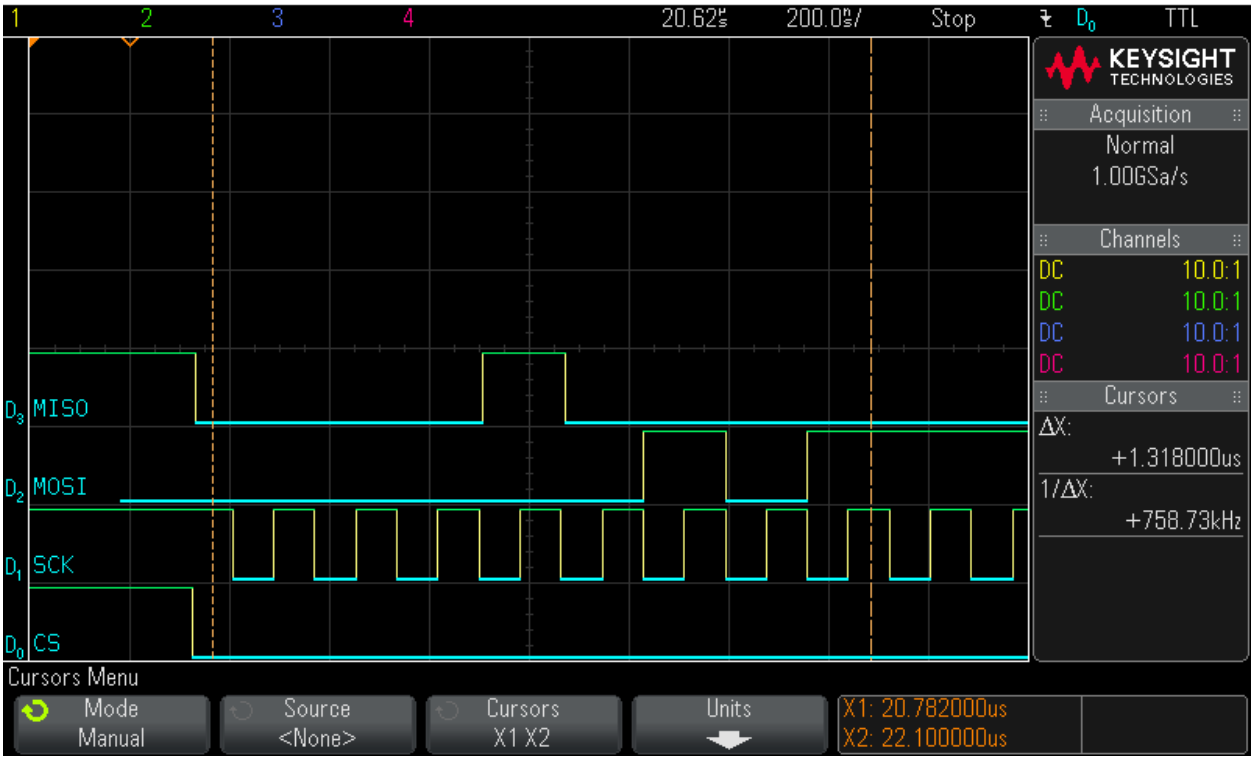
Step 21: Hardware Status Check

MSO-X 2024A, MY52490979: Tue Jul 26 17:25:43 2016

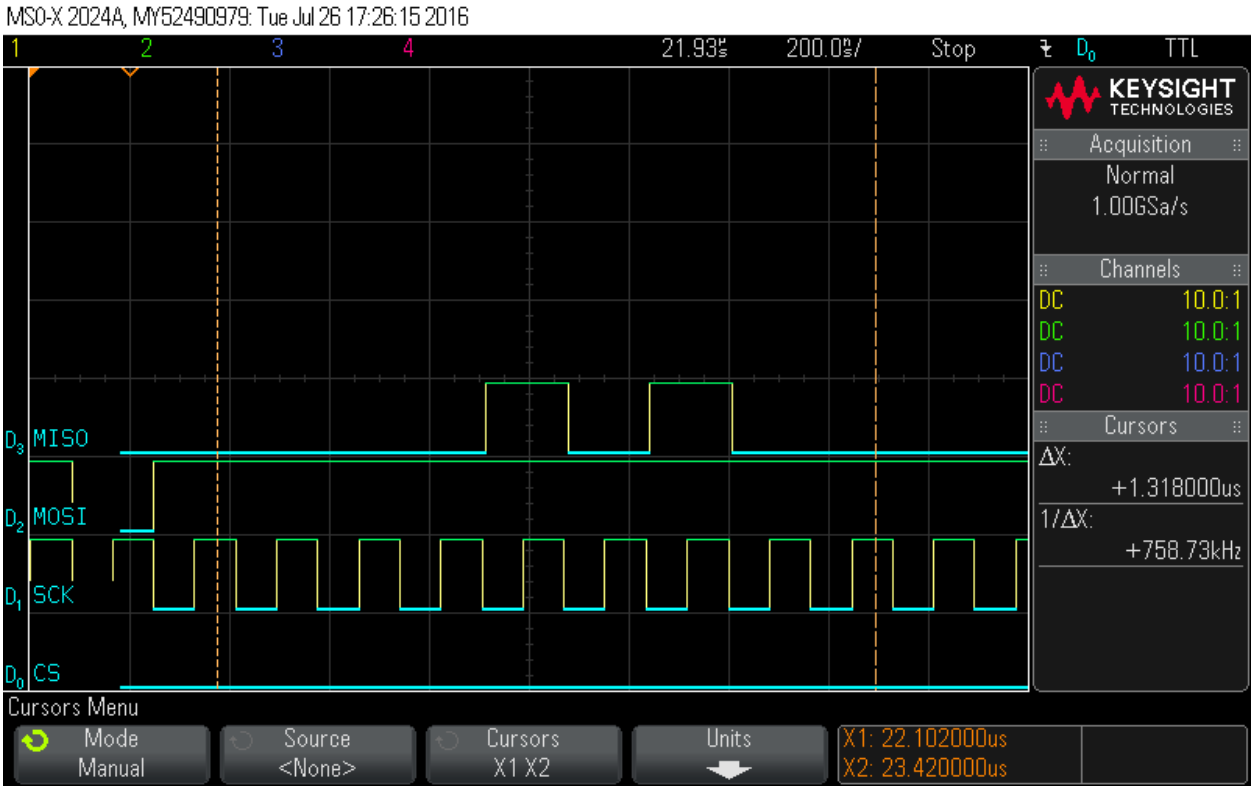


Step 22: Clock in read command (0x5)

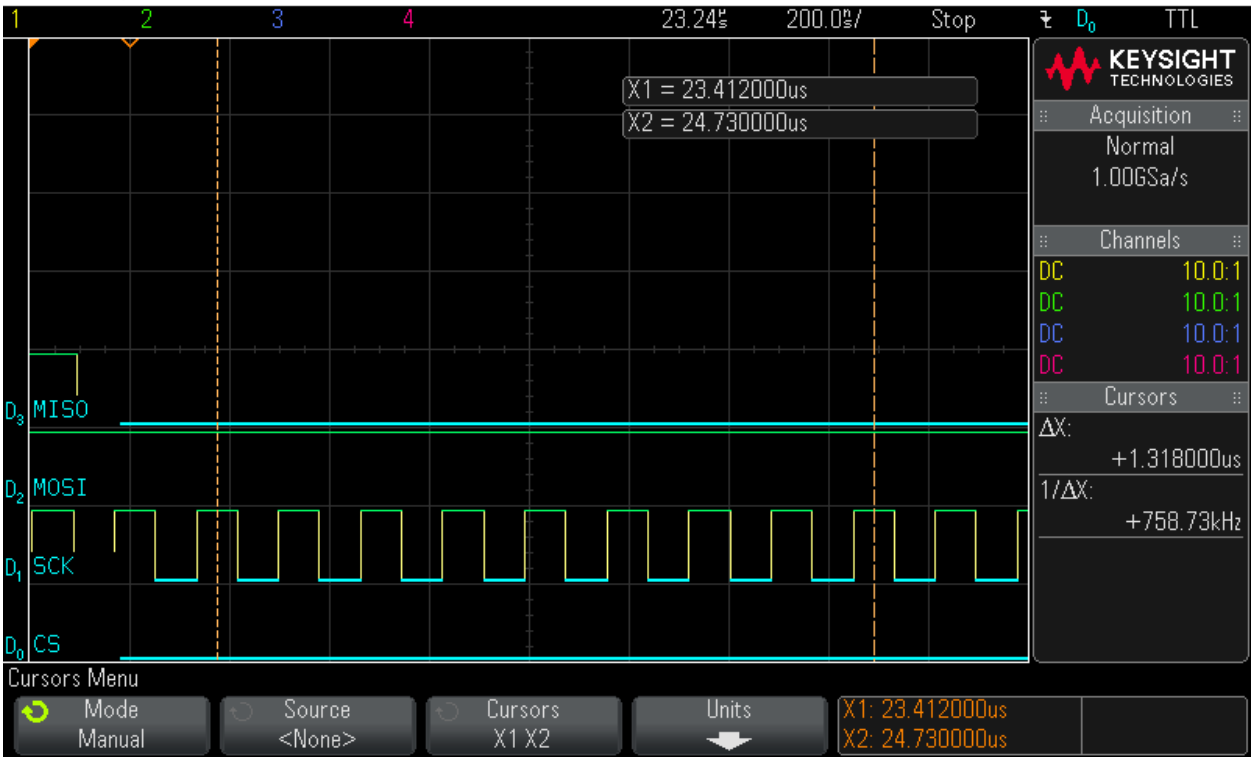
MSO-X 2024A, MY52490979: Tue Jul 26 17:26:00 2016



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

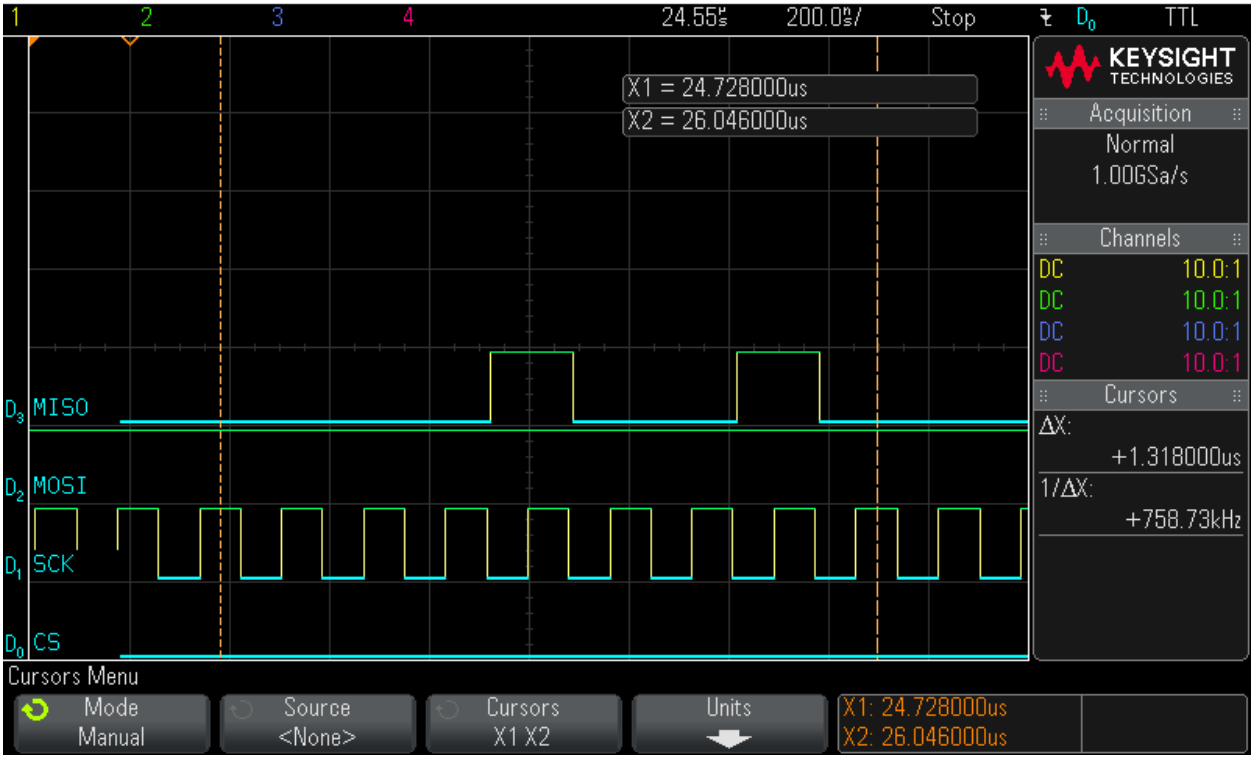


Step 24: Reading out 16 Bytes of FSN data – Byte 1 = 0x0



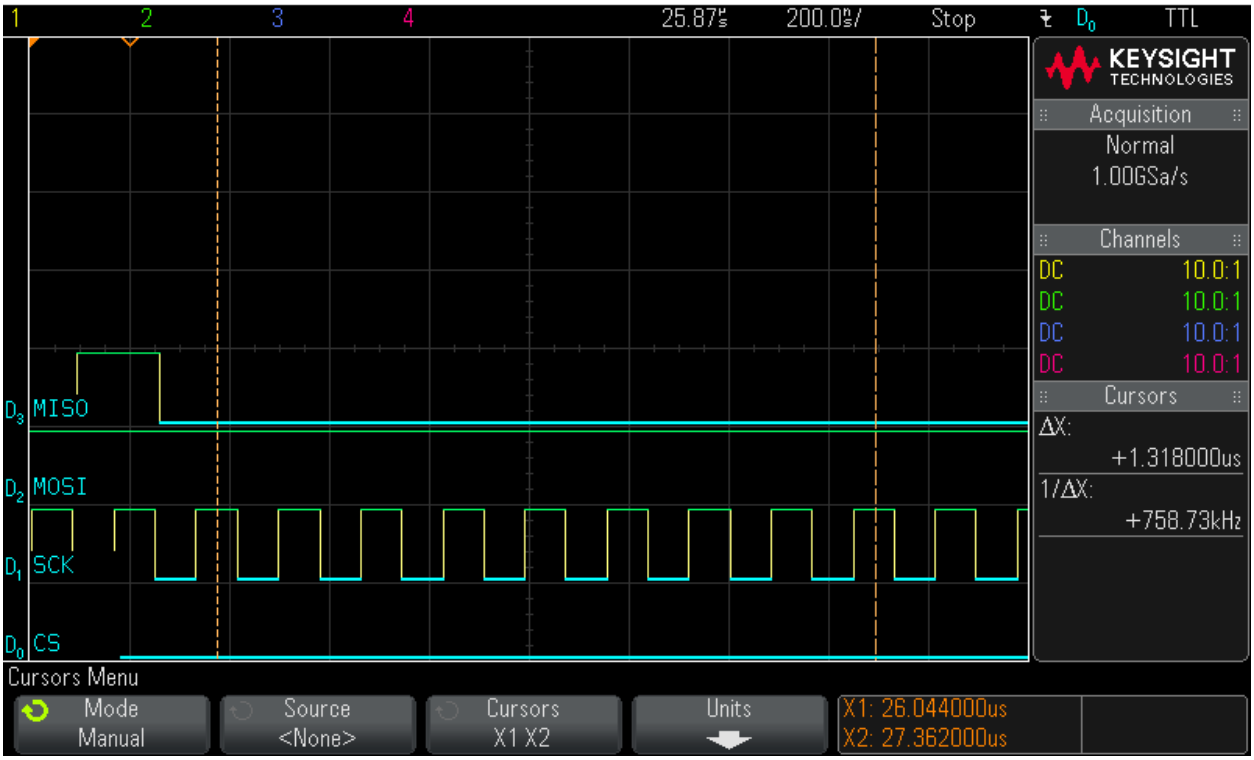
Step 25: Reading out 16 Bytes of FSN data – Byte 2 = 0x12

MSO-X 2024A, MY52490979: Tue Jul 26 17:26:39 2016



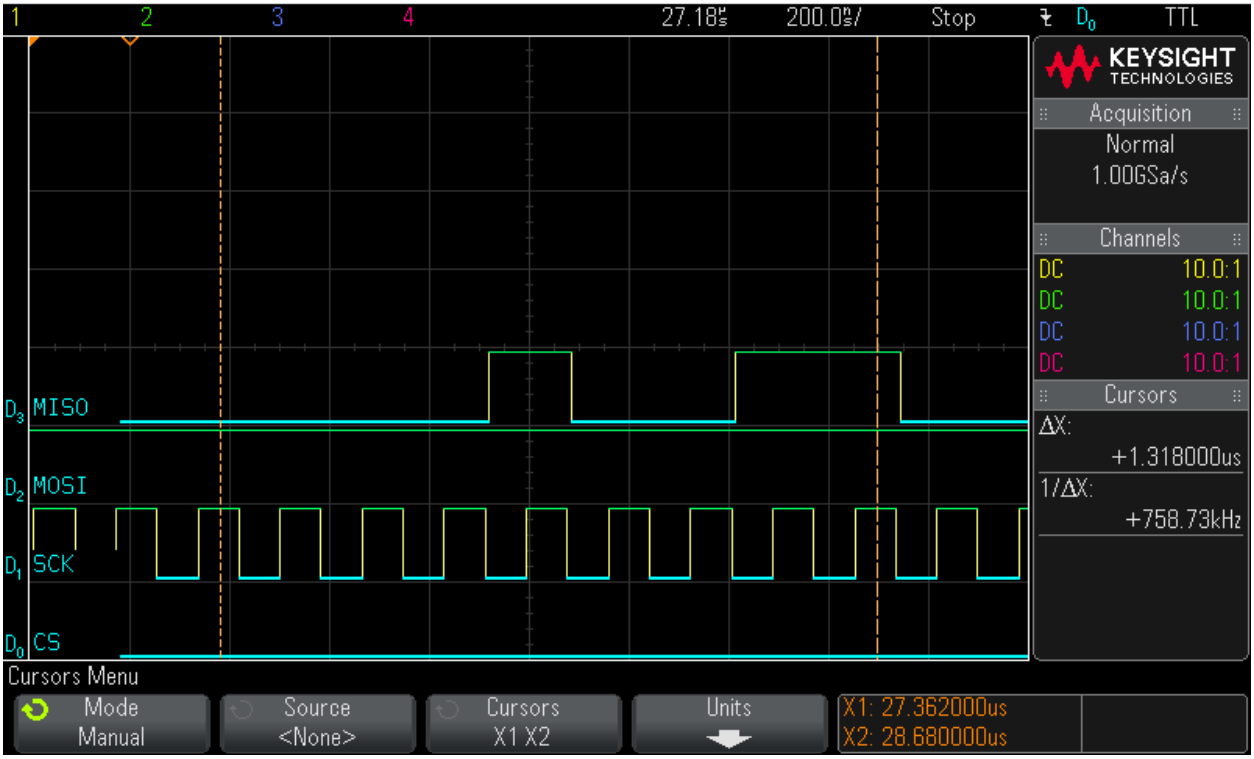
Step 26: Reading out 16 Bytes of FSN data – Byte 3 = 0x0

MSO-X 2024A, MY52490979: Tue Jul 26 17:26:49 2016



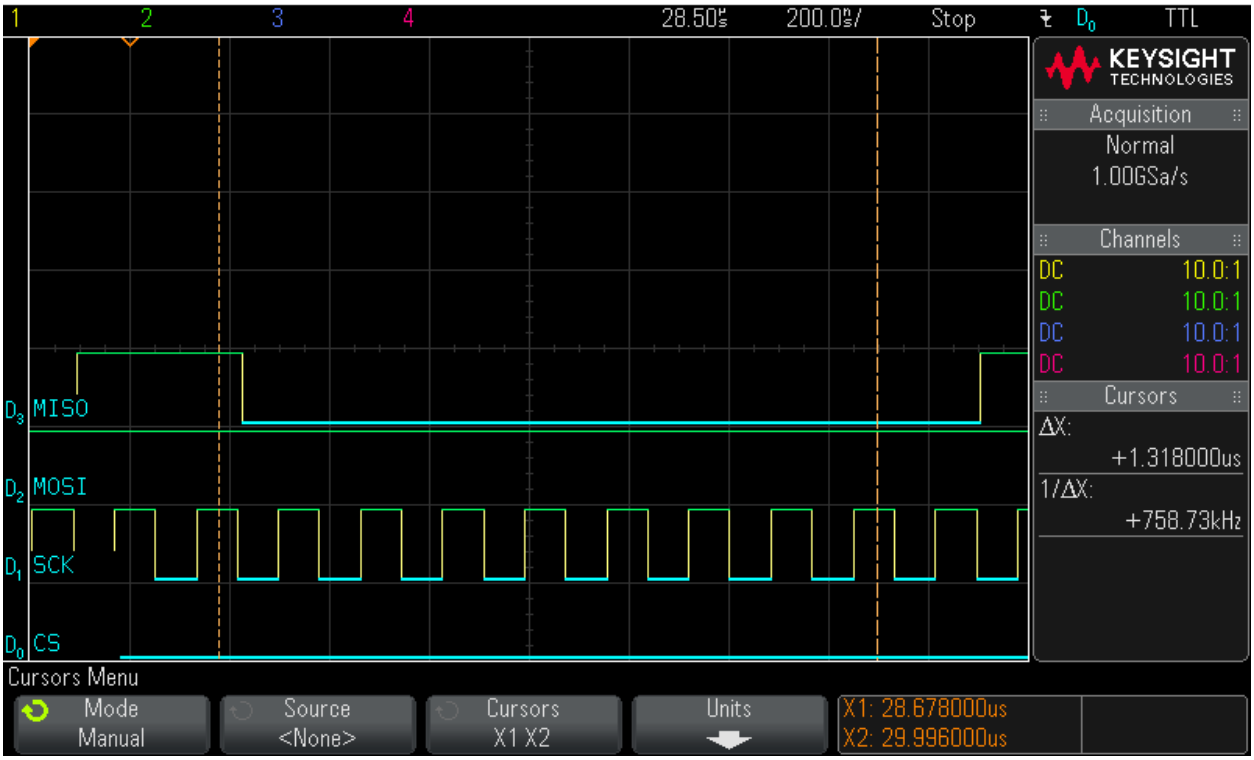
Step 27: Reading out 16 Bytes of FSN data – Byte 4 = 0x13

MSO-X 2024A, MY52490979: Tue Jul 26 17:27:01 2016

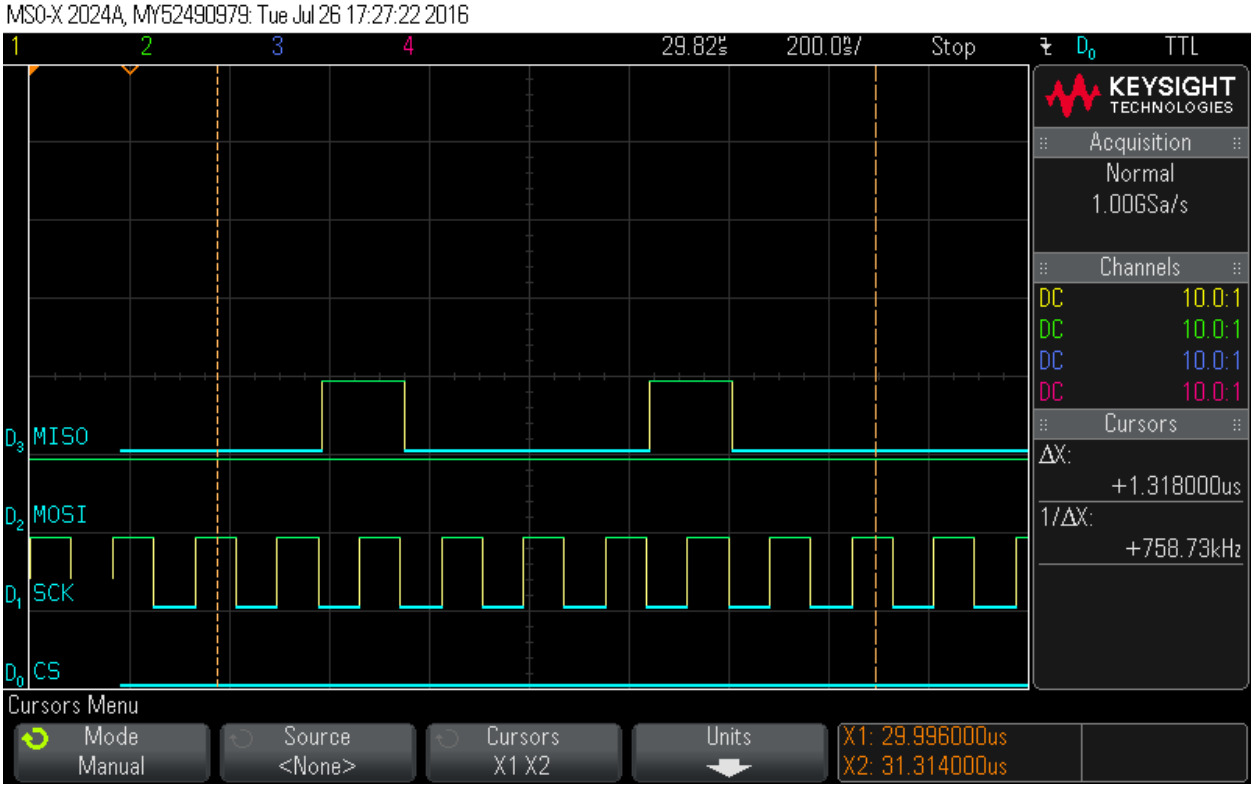


Step 28: Reading out 16 Bytes of FSN data – Byte 5 = 0x0

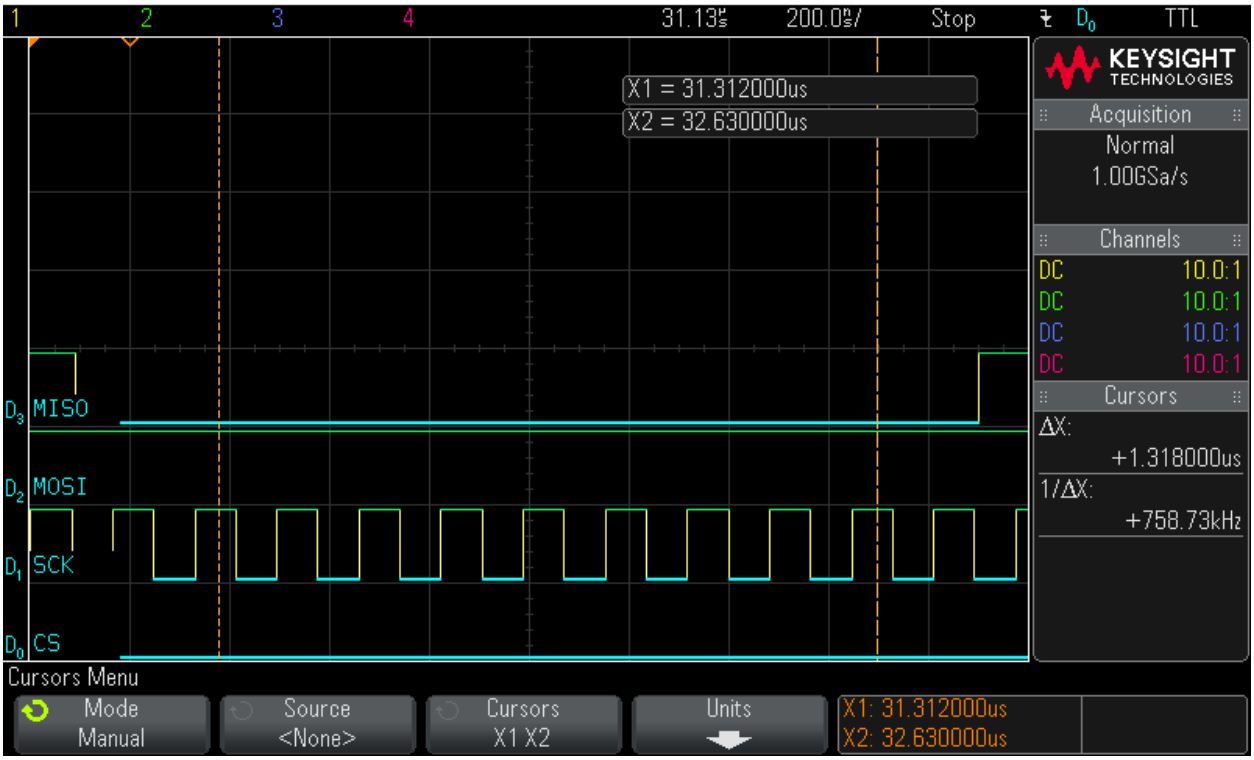
MSO-X 2024A, MY52490979: Tue Jul 26 17:27:12 2016



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

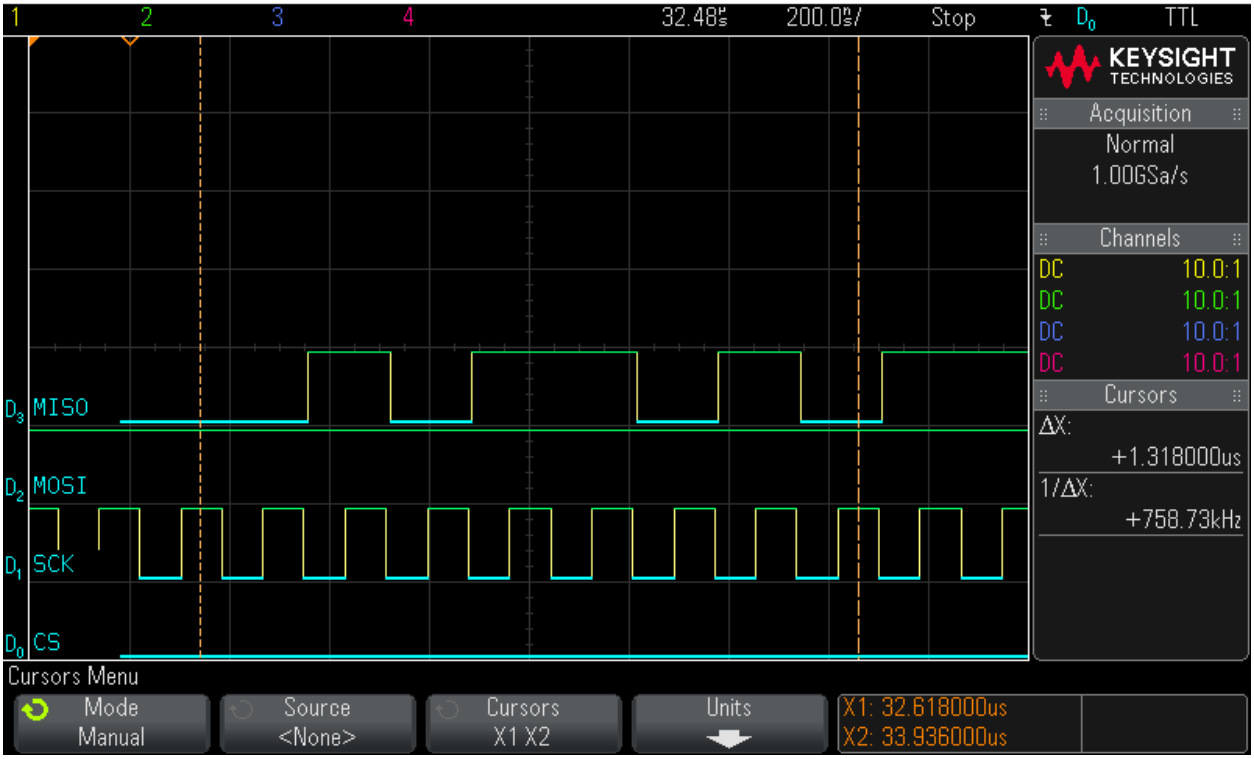


Step 30: Reading out 16 Bytes of FSN data – Byte 7 = 0x0



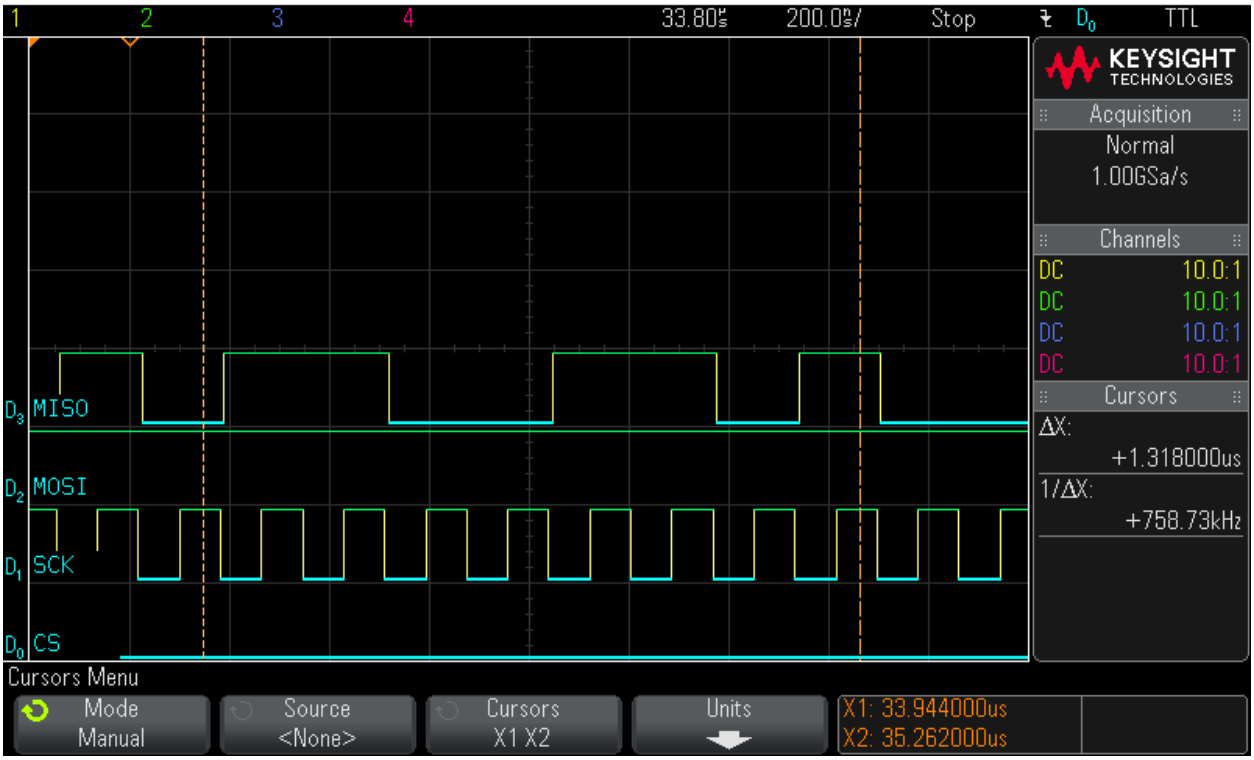
Step 31: Reading out 16 Bytes of FSN data – Byte 8 = 0x5A

MSO-X 2024A, MY52490979: Tue Jul 26 17:27:43 2016



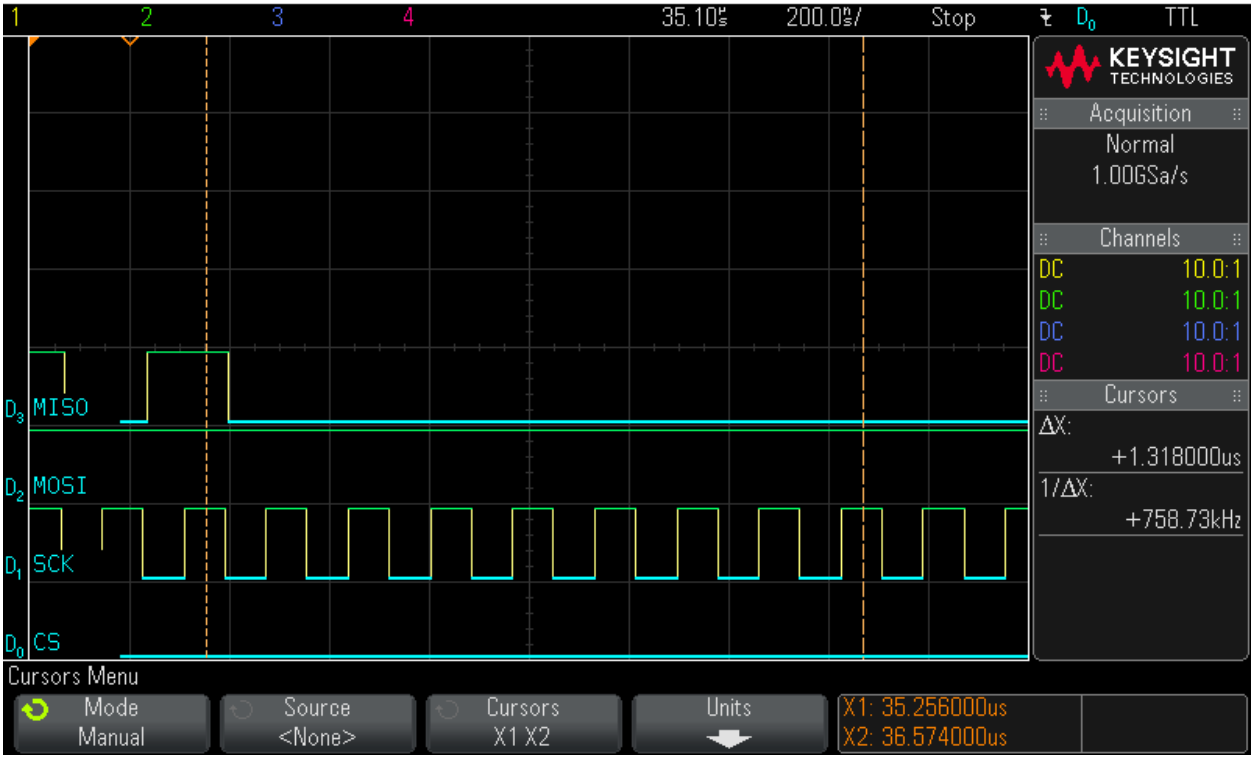
Step 32: Reading out 16 Bytes of FSN data – Byte 9 = 0xCD

MSO-X 2024A, MY52490979: Tue Jul 26 17:27:55 2016



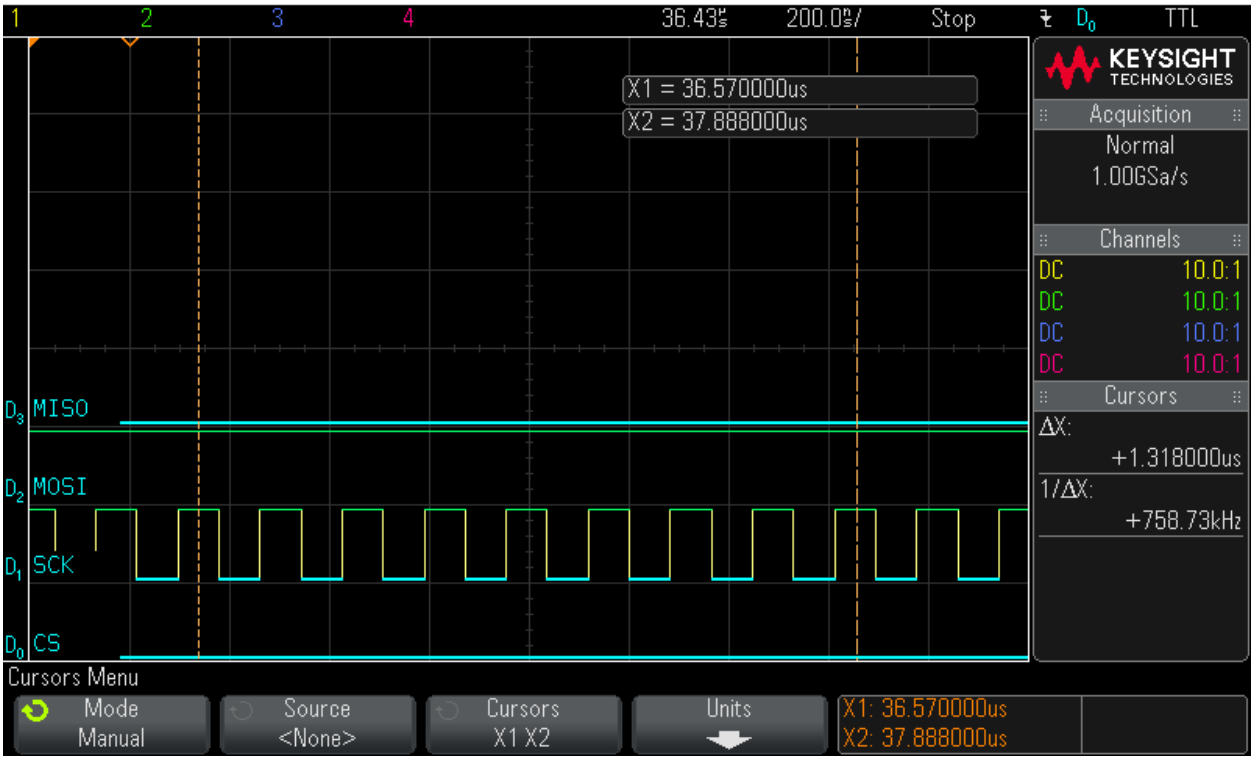
Step 33: Reading out 16 Bytes of FSN data – Byte 10 = 0x0

MSO-X 2024A, MY52490979: Tue Jul 26 17:28:03 2016



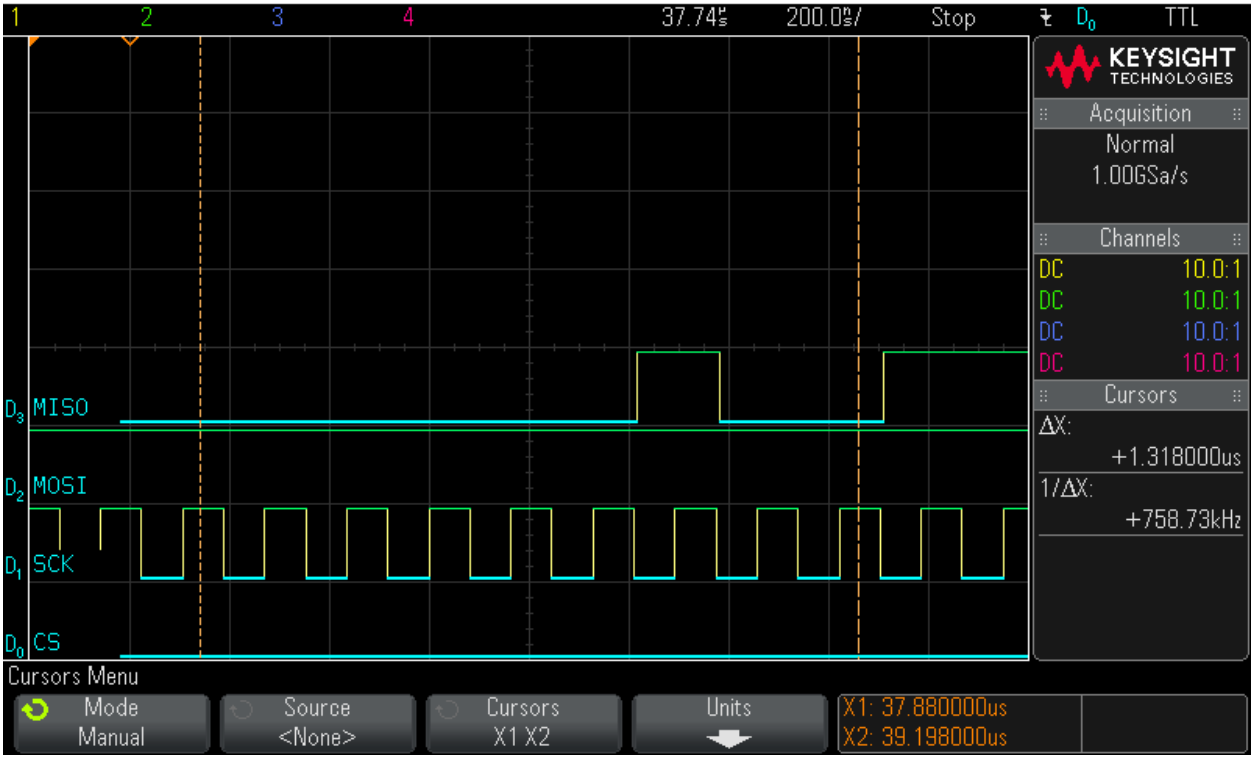
Step 34: Reading out 16 Bytes of FSN data – Byte 11 = 0x0

MSO-X 2024A, MY52490979: Tue Jul 26 17:28:13 2016



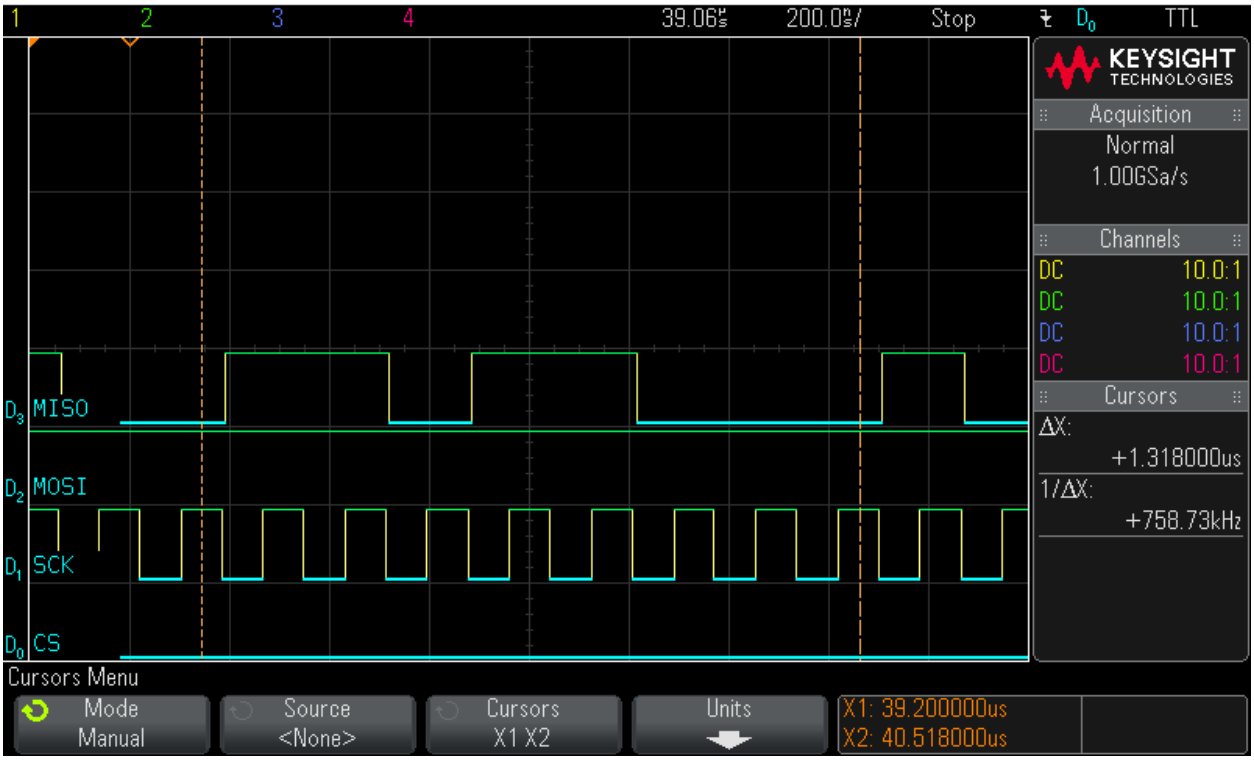
Step 35: Reading out 16 Bytes of FSN data – Byte 12 = 0x4

MSO-X 2024A, MY52490979: Tue Jul 26 17:28:25 2016



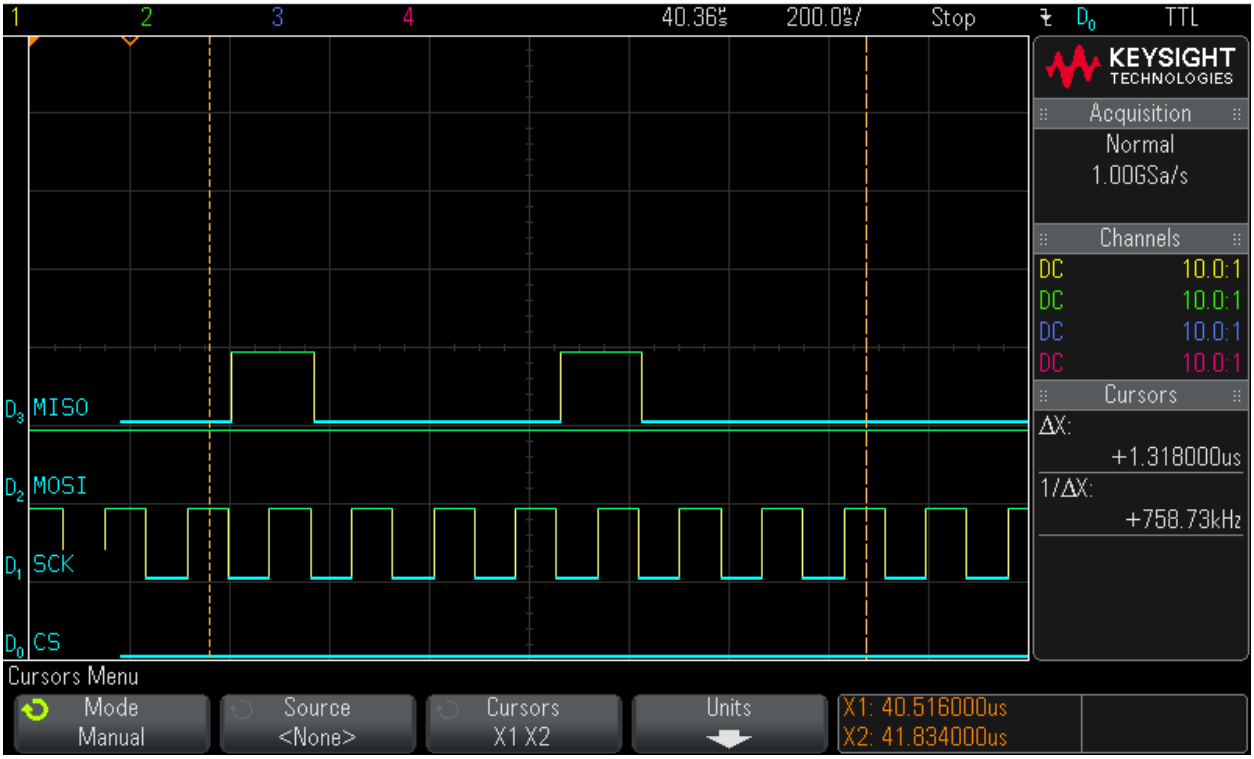
Step 36: Reading out 16 Bytes of FSN data – Byte 13 = 0xD8

MSO-X 2024A, MY52490979: Tue Jul 26 17:28:38 2016



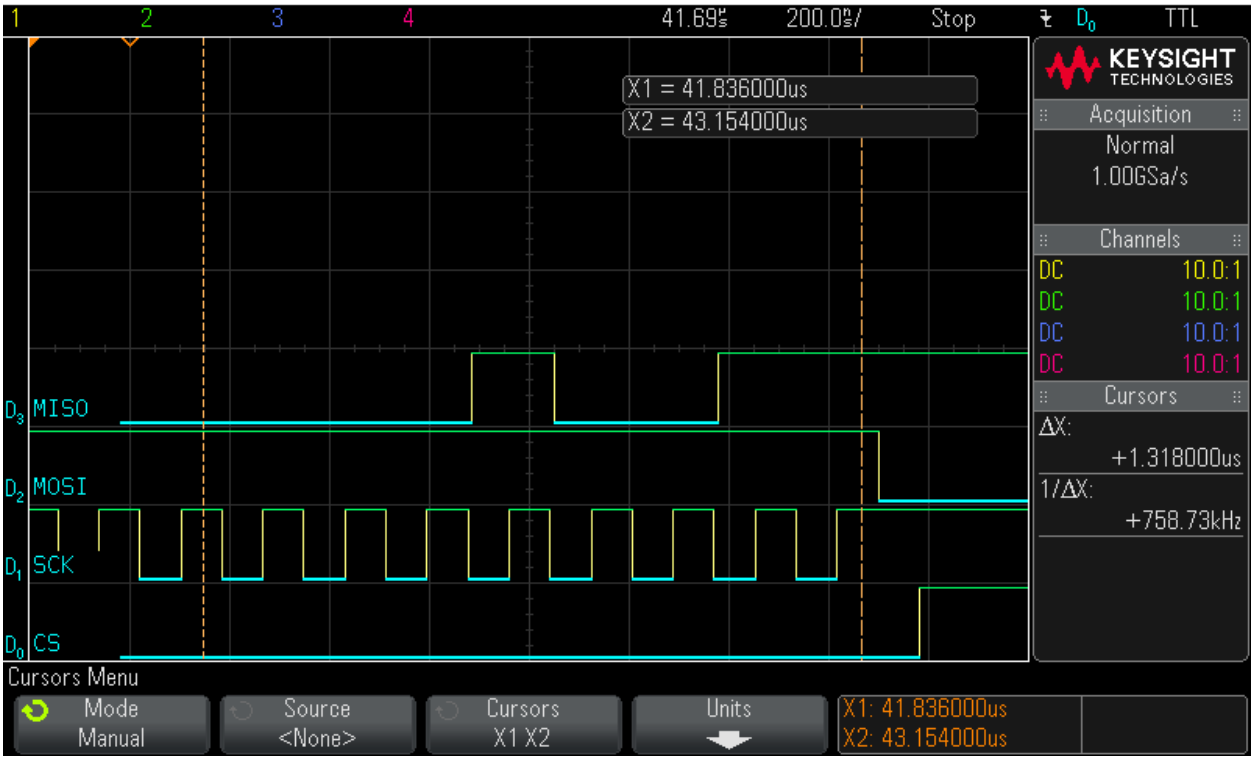
Step 37: Reading out 16 Bytes of FSN data – Byte 14 = 0x88

MSO-X 2024A, MY52490979: Tue Jul 26 17:28:48 2016



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

MSO-X 2024A, MY52490979: Tue Jul 26 17:29:01 2016



Program frame waveform:

When performing “Program, Verify or Authenticate” actions, the data to be clocked into the device starts at the beginning of the “datastream” block as shown below.
Obviously, 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.

```
--------------------------------------------------------------------------------
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000
numOfBlocks
2000407E93B00500640000200000
datastream
7C5D1C2B3D7519B3924AABEE4ED56C6286545B7B3266FDF5975C868564A76E06DC5E14F6C16E6DE0
C7F14DFC0DA0412A00030002000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000002F8FFFF030100C02FD94D3253303530000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
8C41036EF3EF6CC81E93FD29C4F2890B6FBA831F5F4A2BAB62092E6A9C604E7E9AC405860A7B097A
5B80D956D163A0012389AC4DB418B59558AB1A2F1CF5703EFB1915B8C20DA85FB053B7DA68E5490C
6B3E0239D719E17F9F2869F240AC43034122933EC8A0D2F58CBBD85B5AA8B224749635569CD51E95
CD18AC8359DCA64994043B10DA5B9A7596CC21727F01D3C8D20279283AD77D2726F1549822F1B52A
01030002000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
--------------------------------------------------------------------------------
```

The scope plots below show how the first data frame is clocked. In this example, the following data is clocked:

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

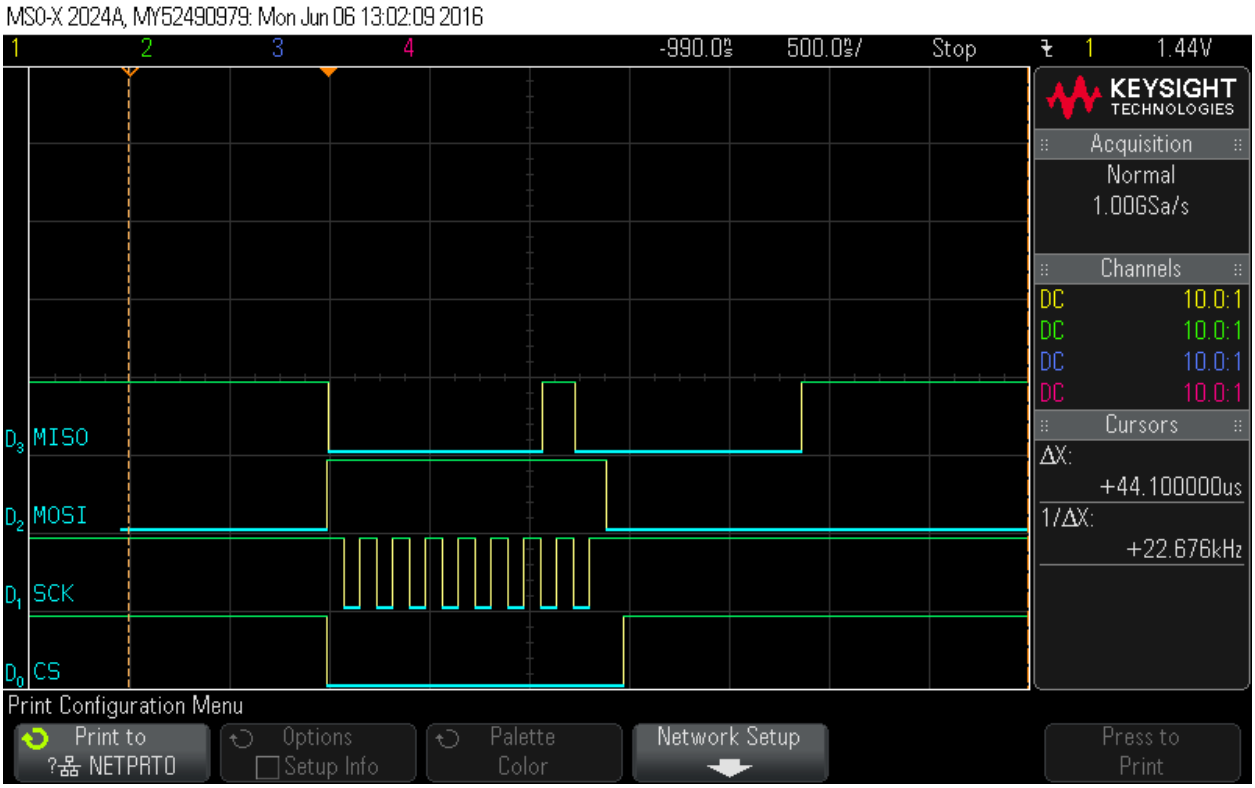
The SPI mode used is mode 3 and the data is clocked byte 0 MSB first.

Please note the following:

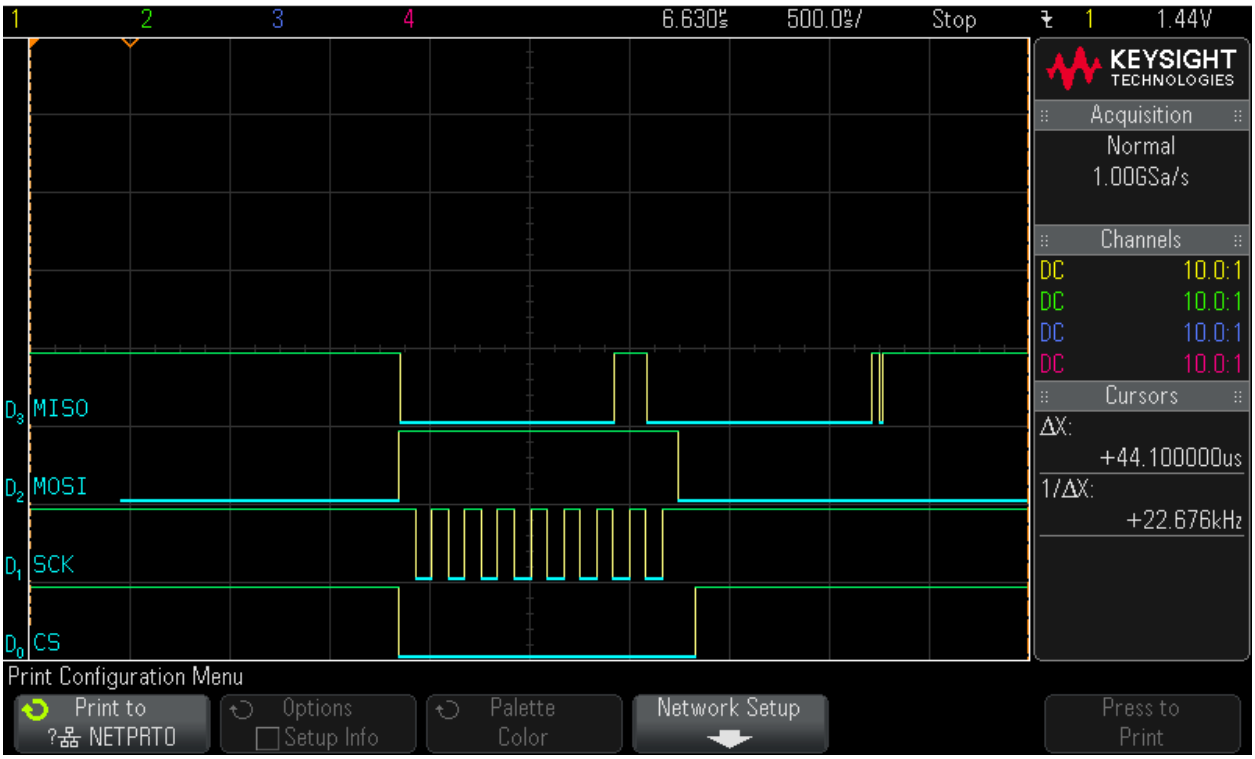
1. Before performing any data shift, the target device SPI buffer status is checked by shifting 0xff. This is the only instruction that is 8 bit long and the data is read out at the same time as it is shifted in. The result of the first shift is ignored.
2. When shifting data, into the device, the first 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 with the exception of spi hardware status check are made of one byte of command followed by 16 bytes of data. 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.

Notes 1, 2, 3 above are taken care of by the programming algorithm.

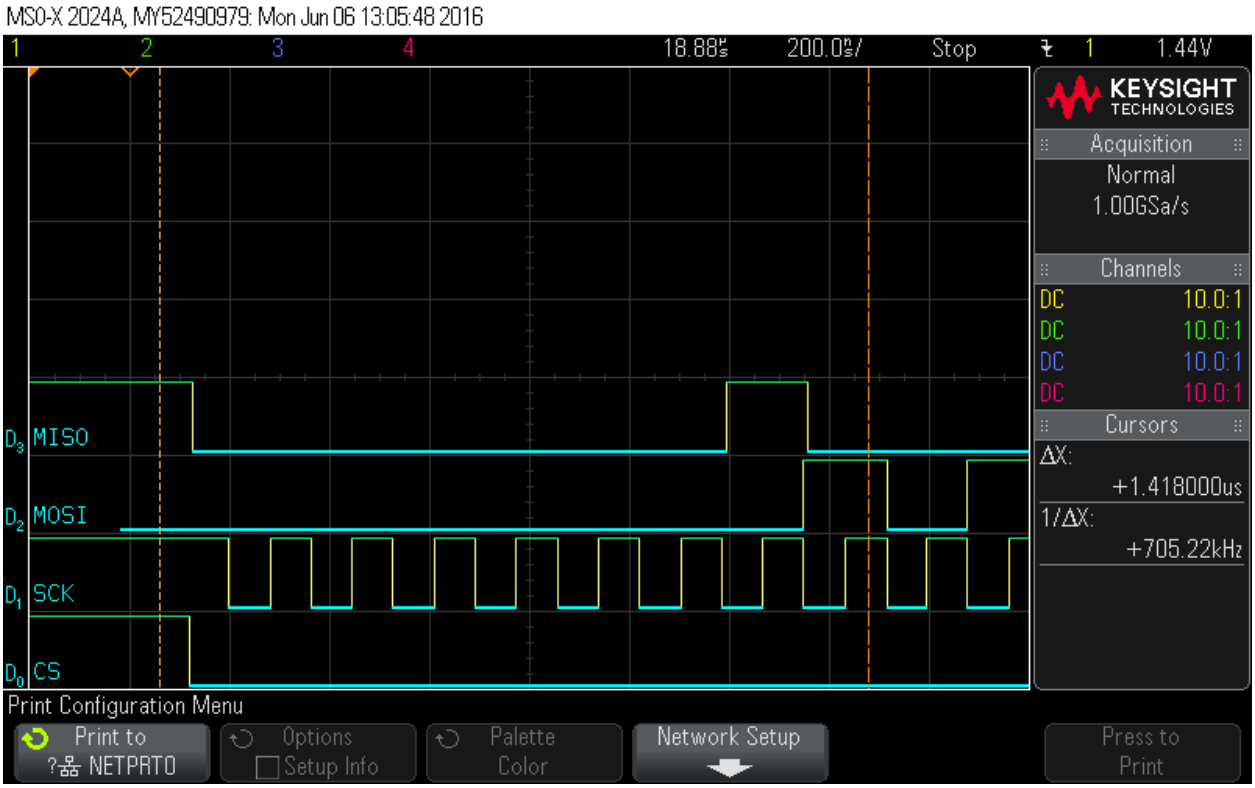
Step 1: Hardware Status Check



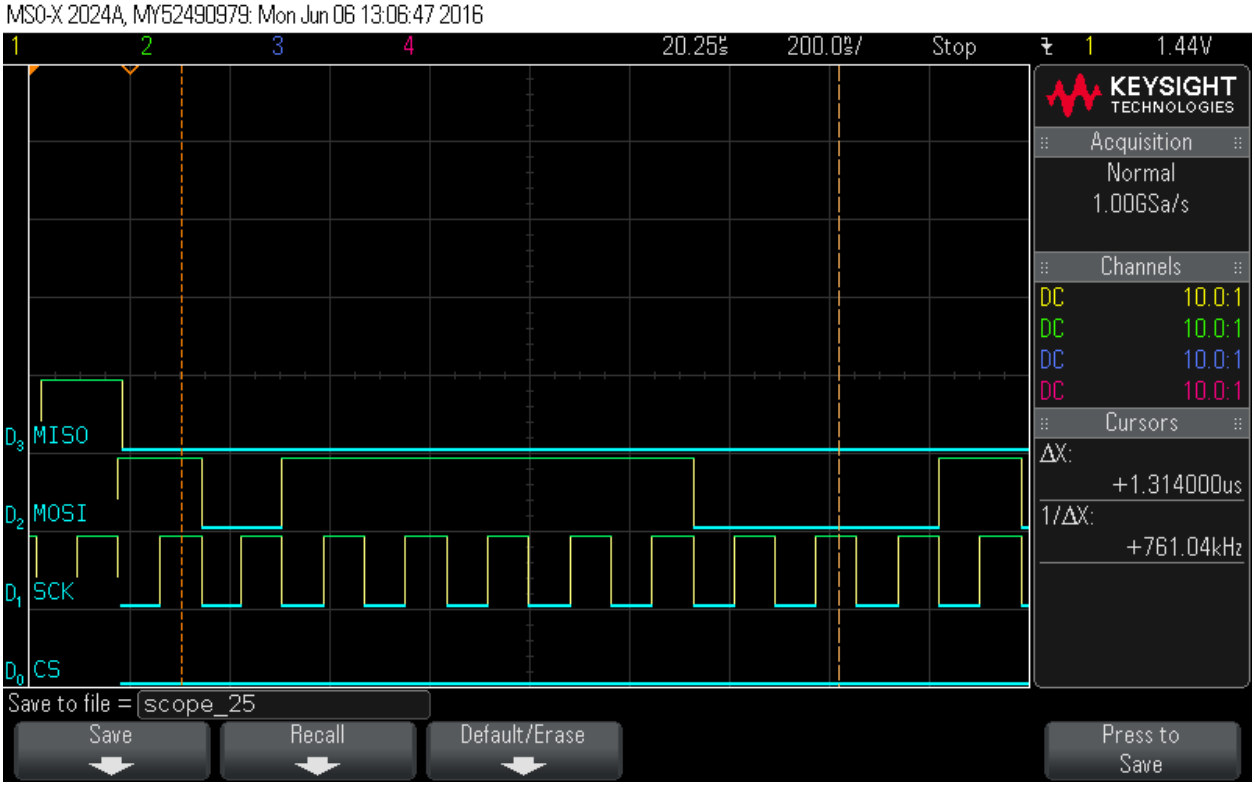
Step 2: Hardware Status Check



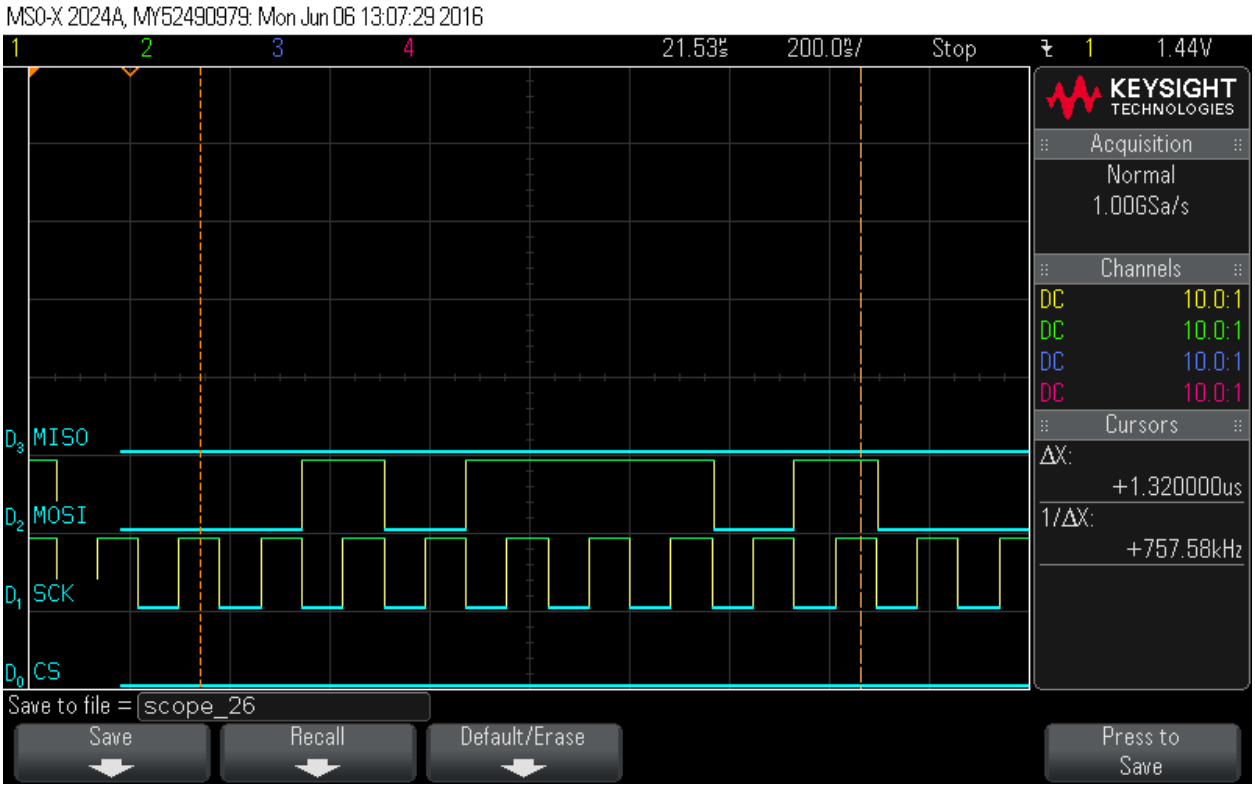
Step 3: Shift in the first frame. Command = 0x1. Data to follow. **Note CS signal**



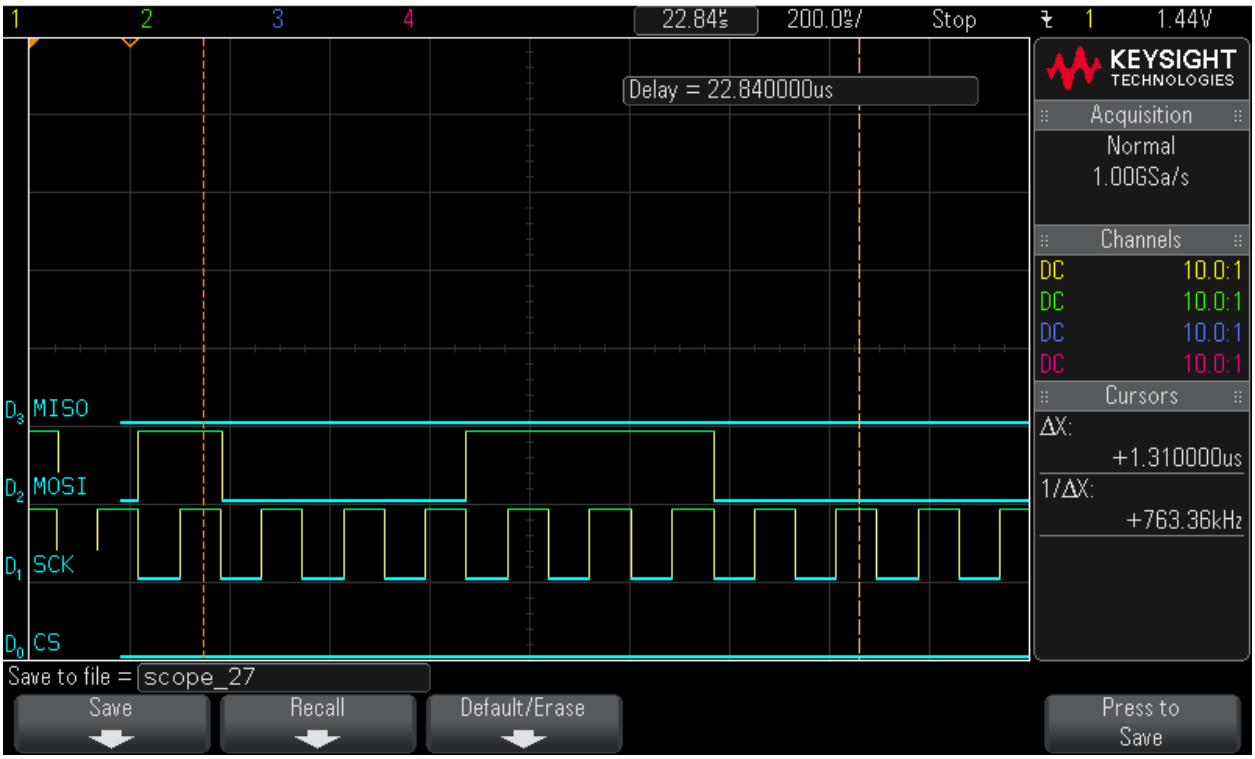
Step 4: Data Byte0 = 0x7C



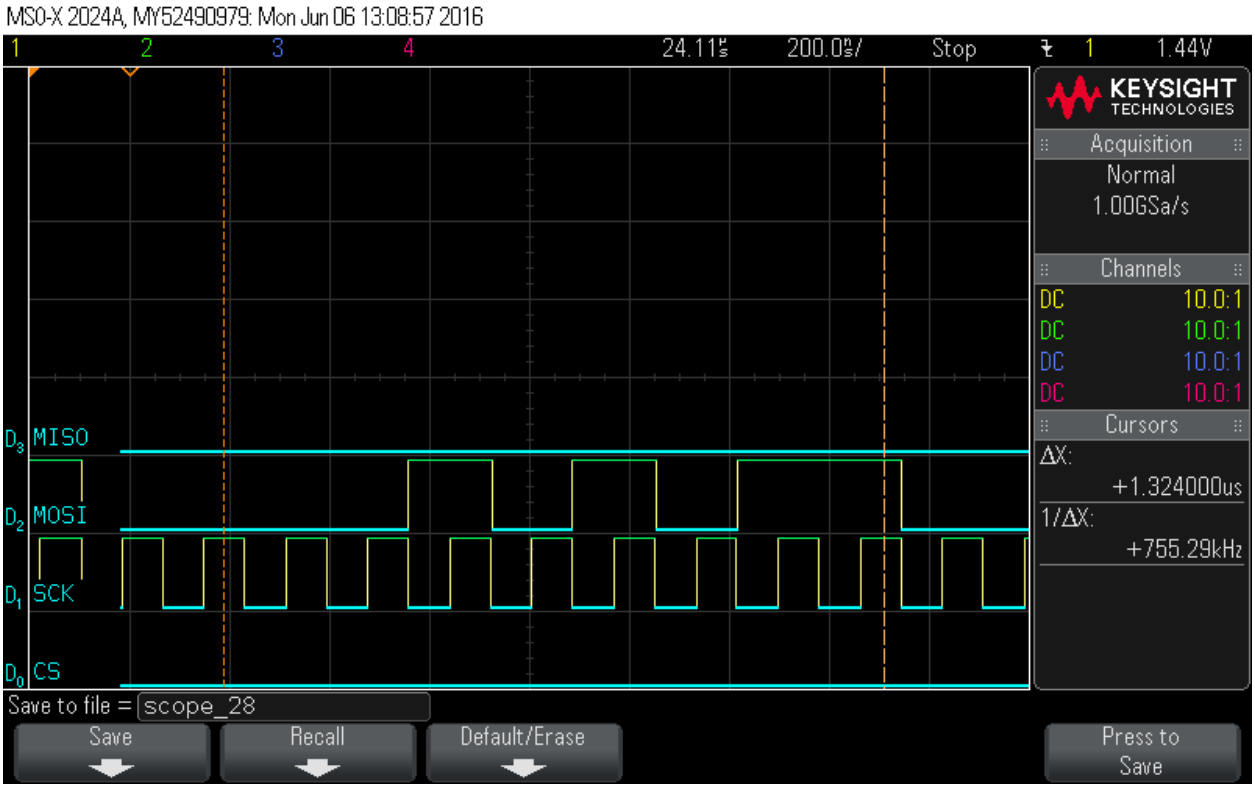
Step 5: Data Byte1 = 0x5D



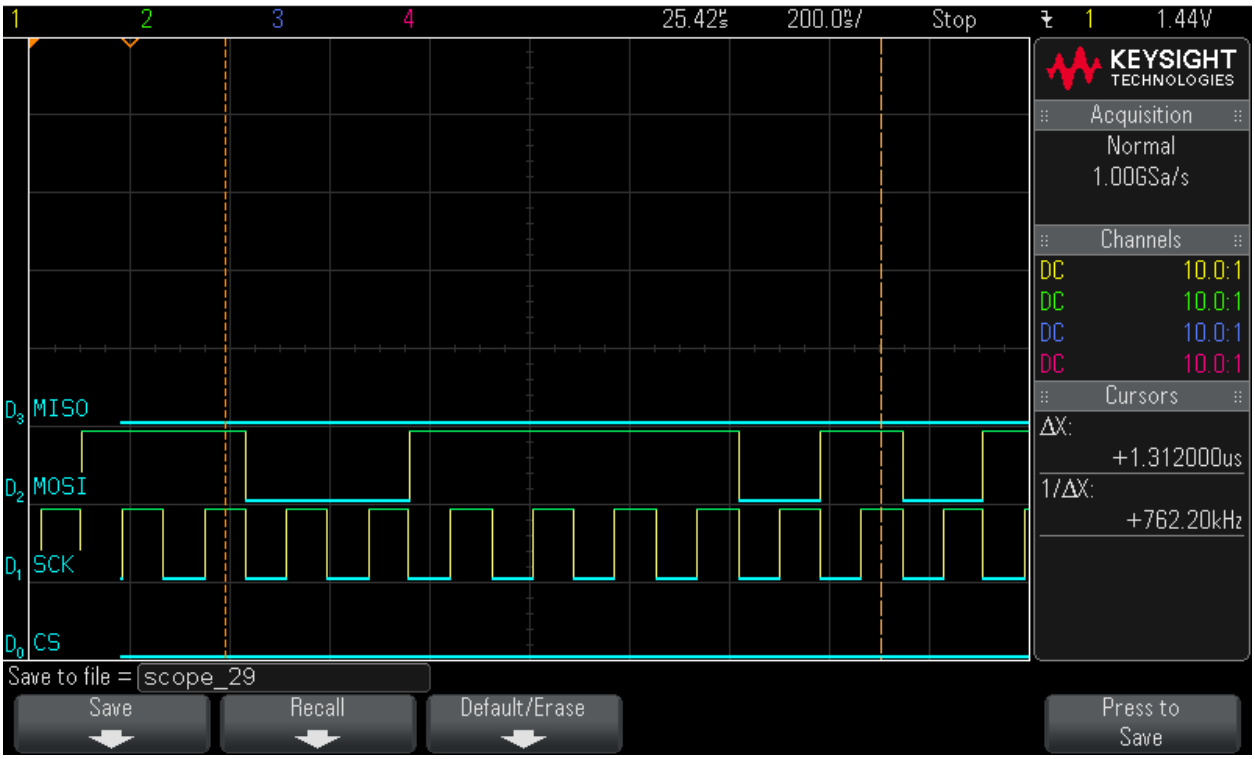
Step 6: Data Byte2 = 0x1C



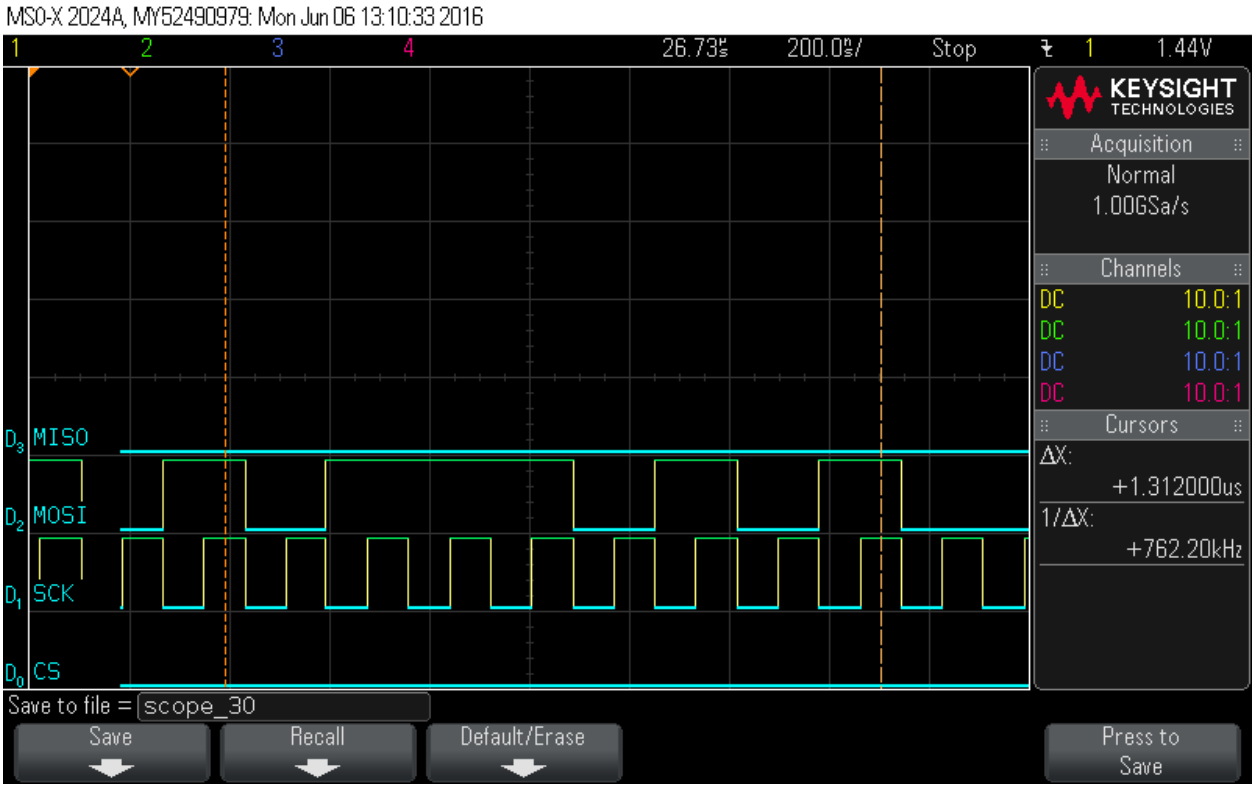
Step 7: Data Byte3 = 0x2B



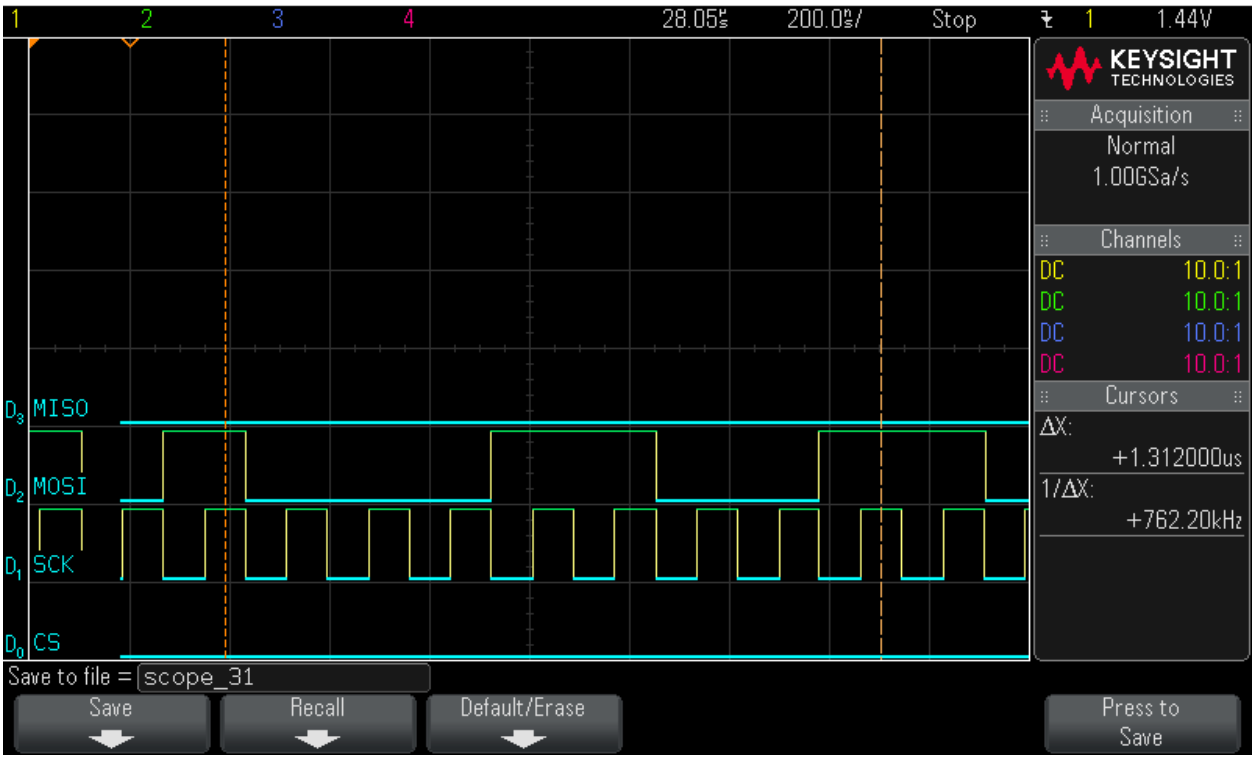
Step 8: Data Byte4 = 0x3D



Step 9: Data Byte5 = 0x75

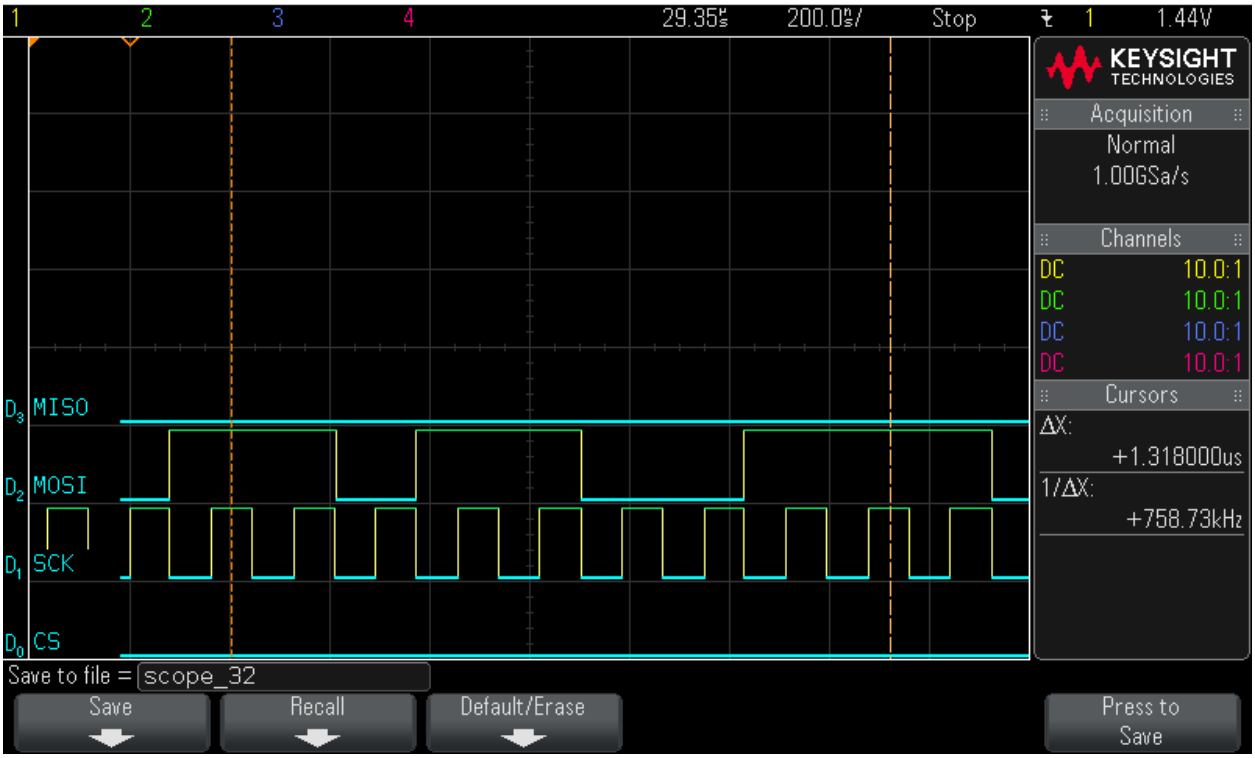


Step 10: Data Byte6 = 0x19



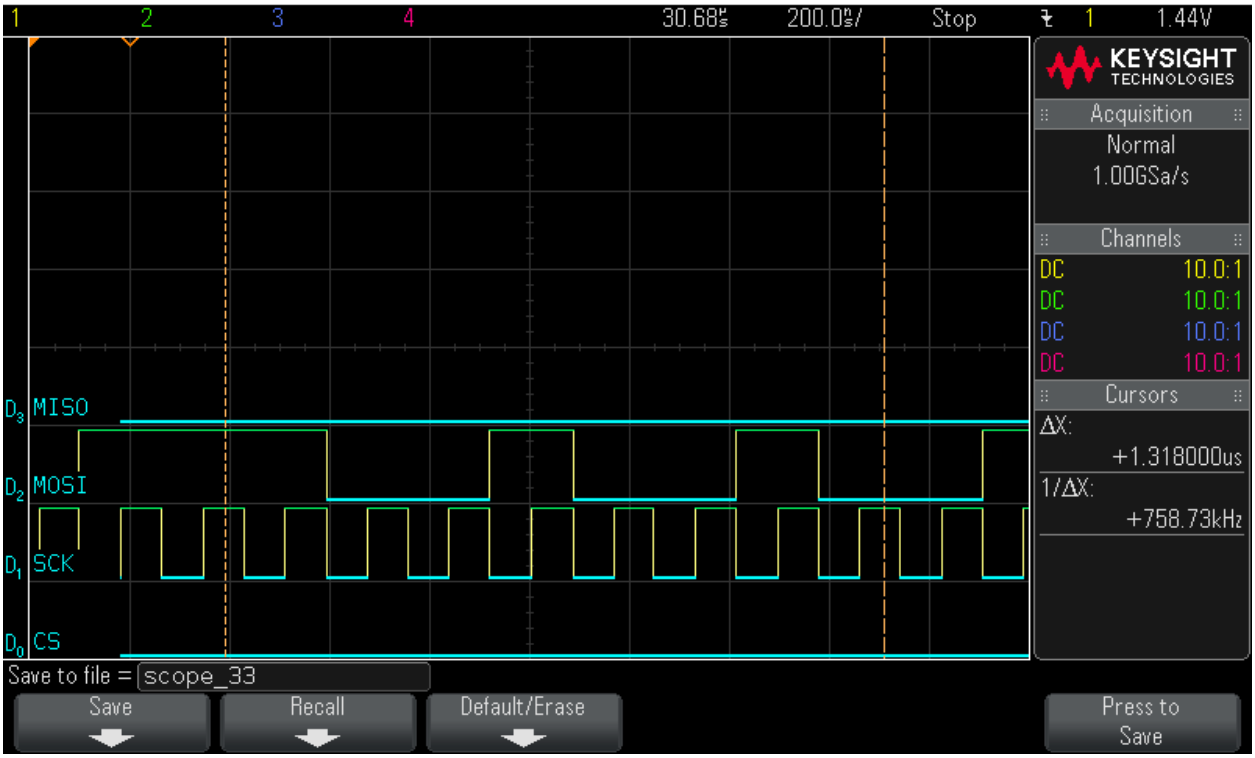
Step 11: Data Byte7 = 0xB3

MSO-X 2024A, MY52490979: Mon Jun 06 13:11:46 2016

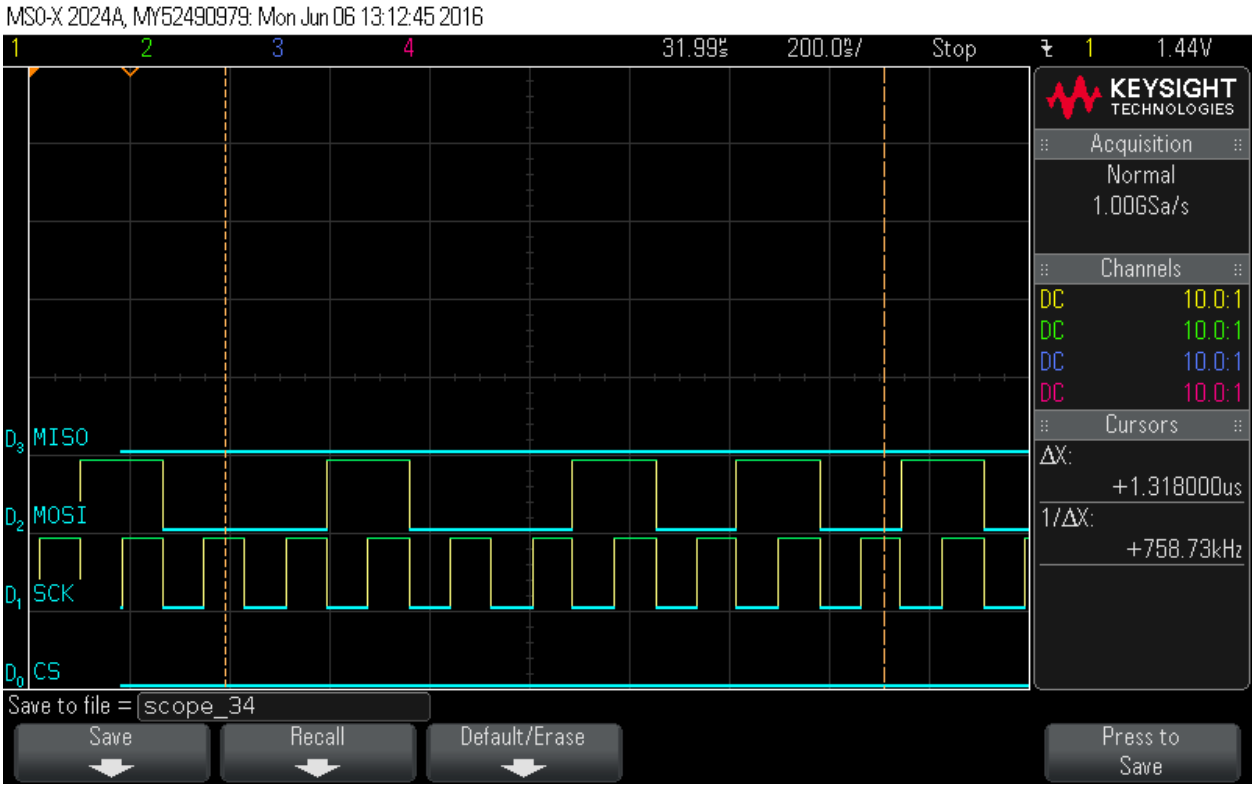


Step 12: Data Byte8 = 0x92

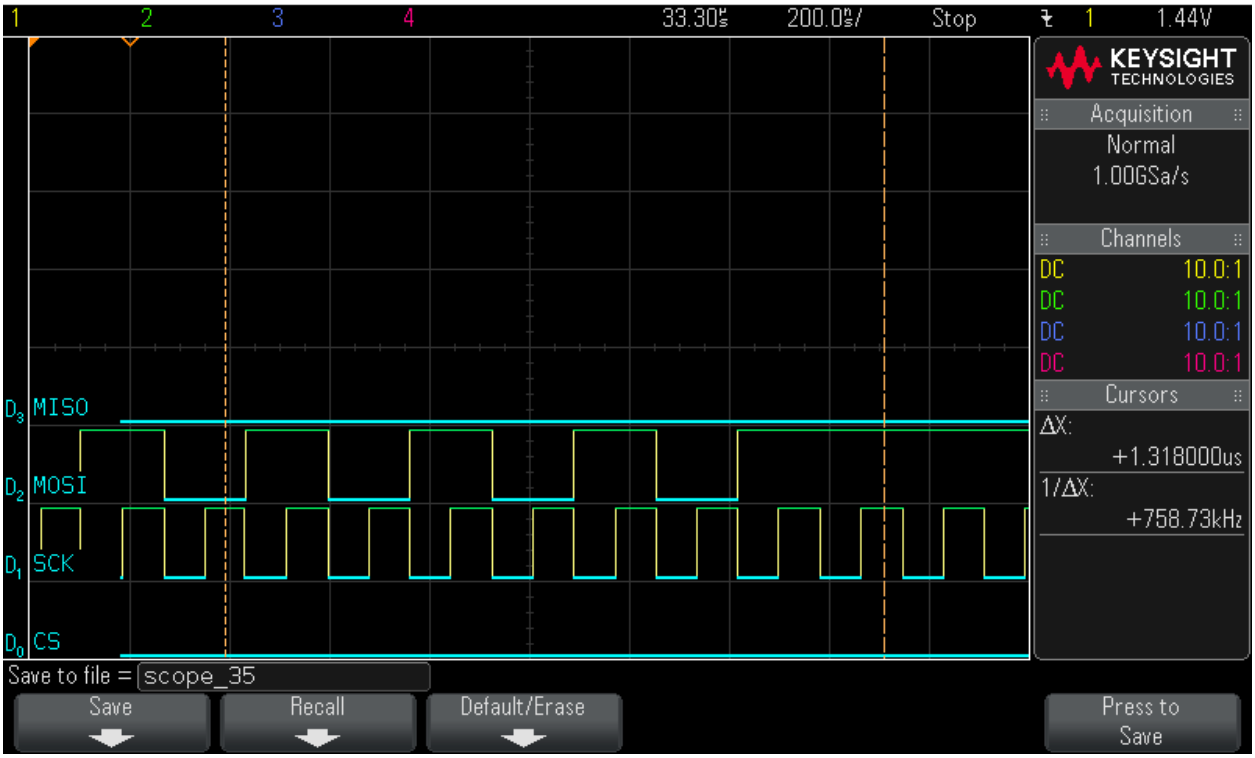
MSO-X 2024A, MY52490979: Mon Jun 06 13:12:20 2016



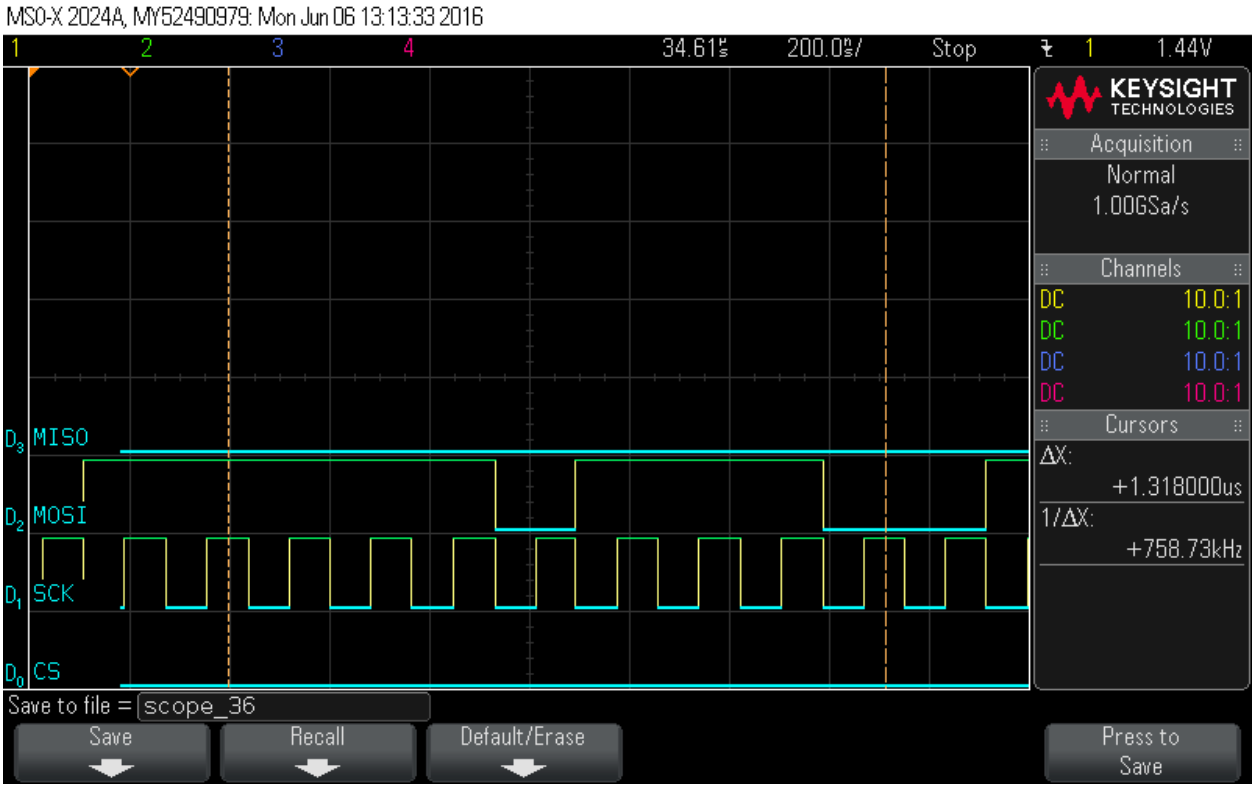
Step 13: Data Byte9 = 0x4A



Step 14: Data Byte10 = 0xAB

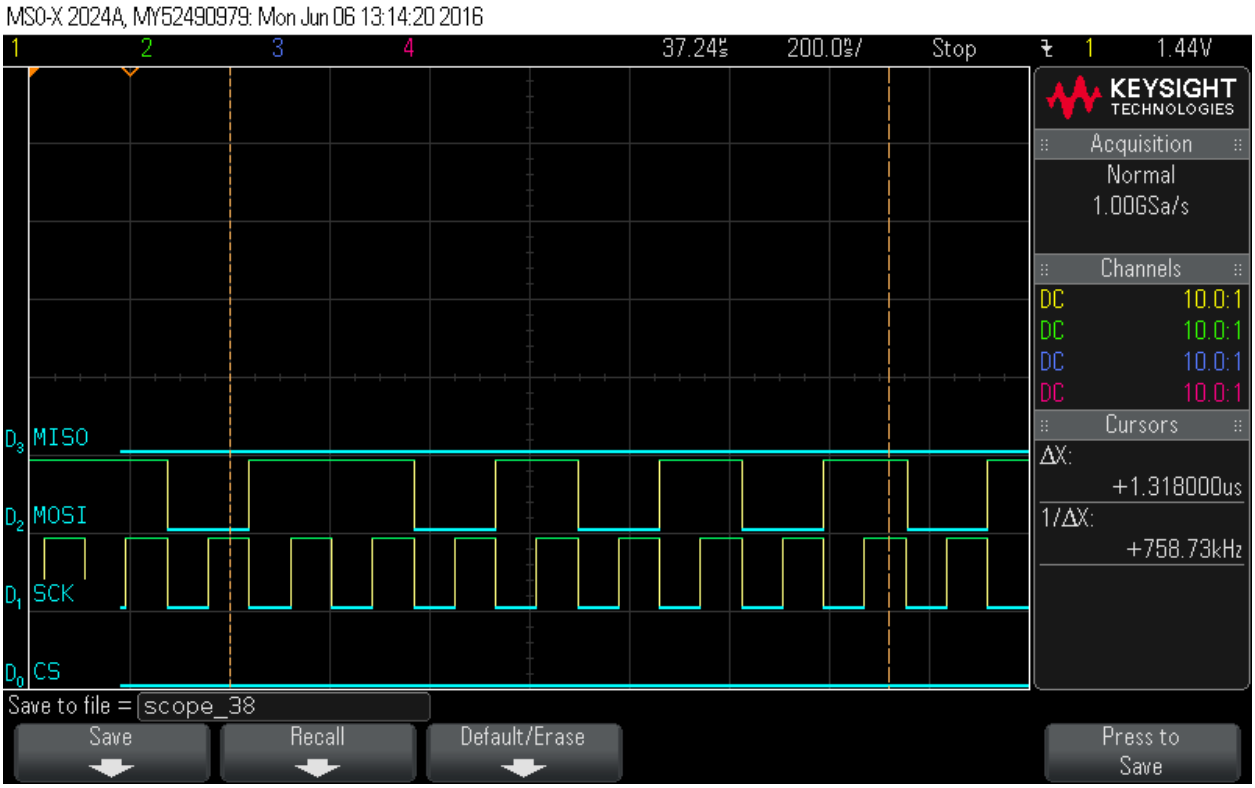


Step 15: Data Byte11 = 0xEE

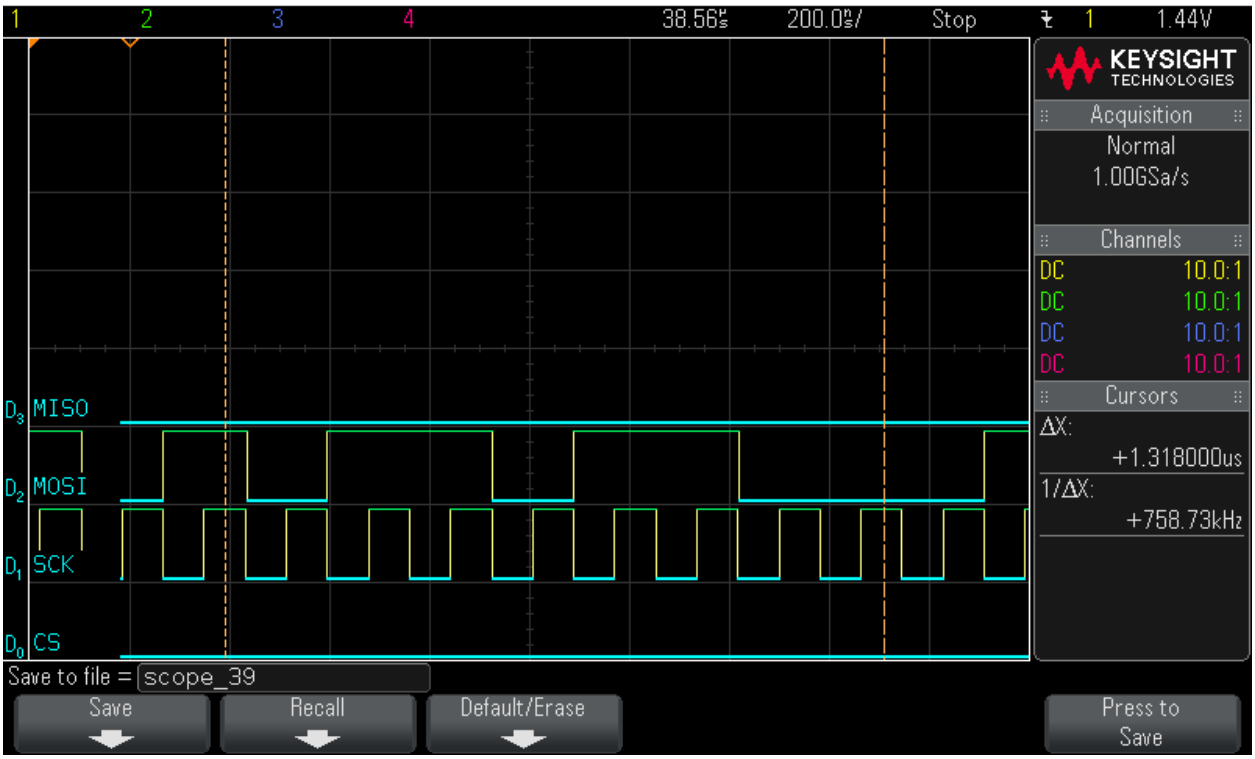


Step 16: Data Byte12 = 0x4E

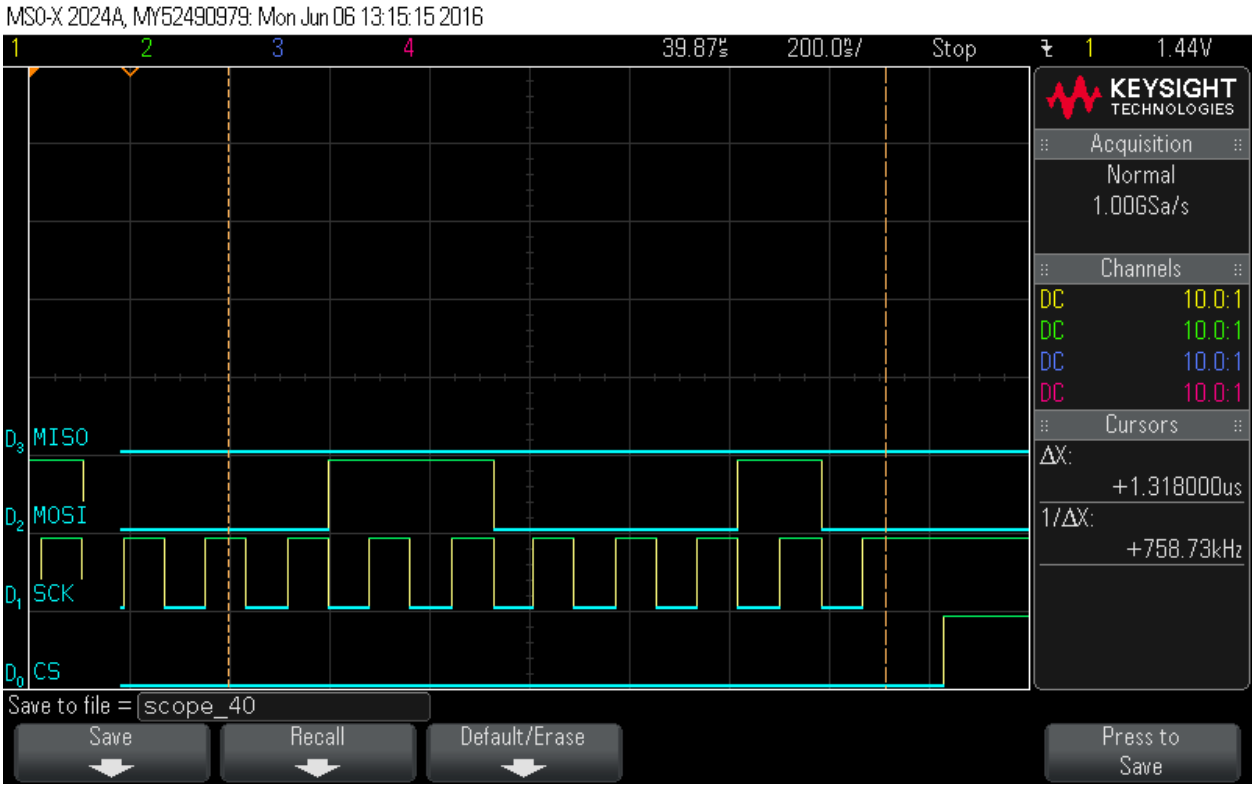
Step 17: Data Byte13 = 0xD5



Step 18: Data Byte14 = 0x6C

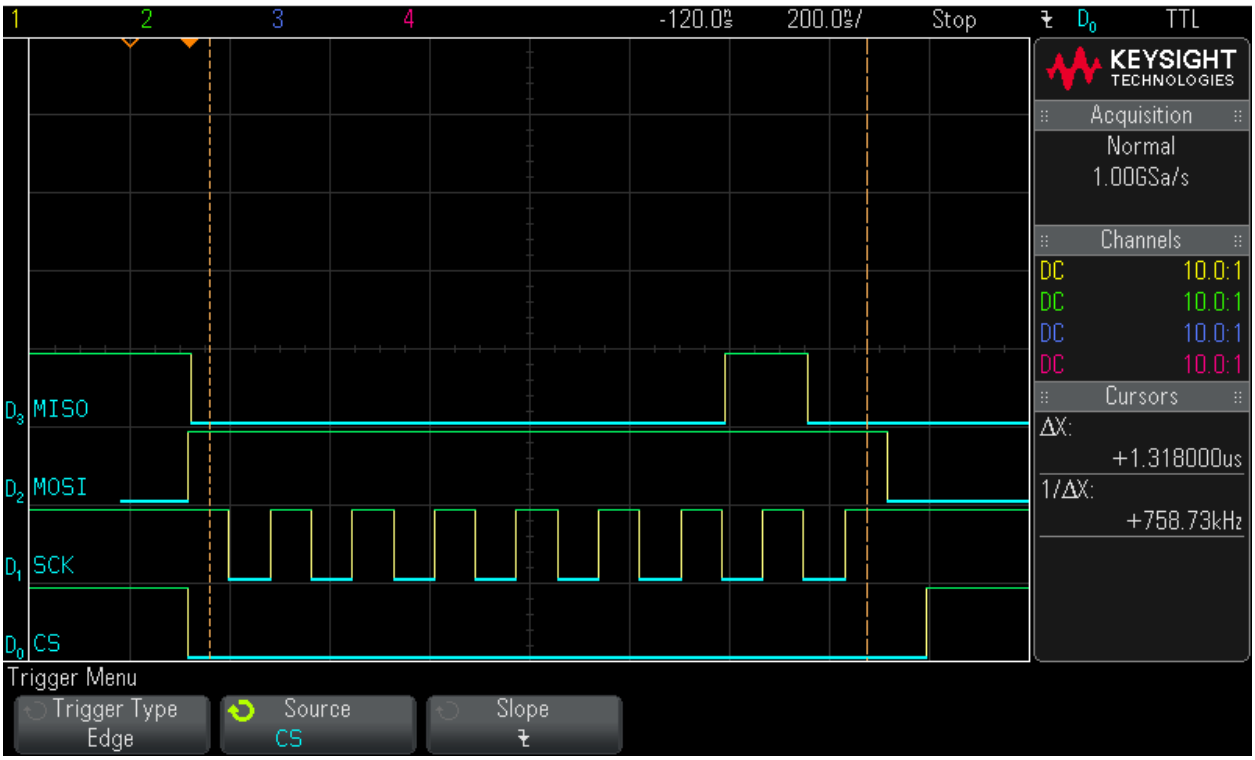


Step 19: Data Byte15 = 0x62. **Note CS signal**

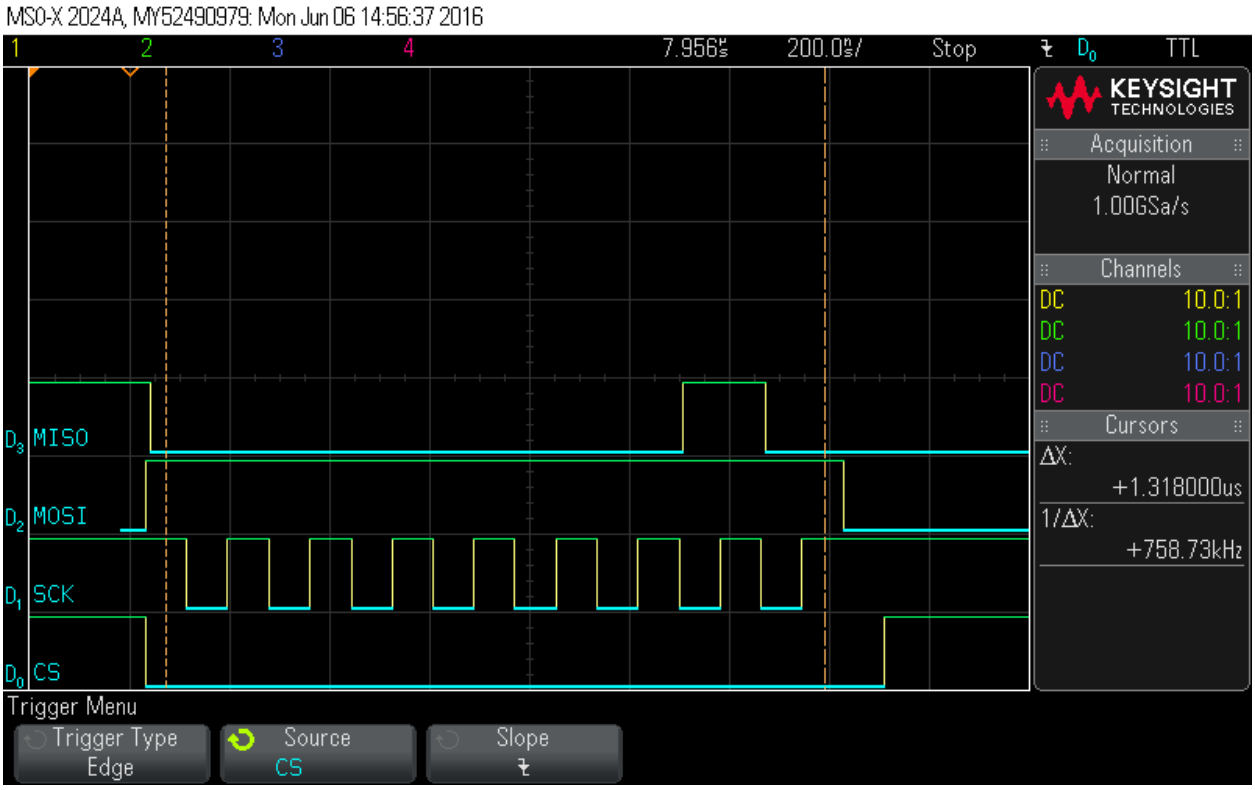


At this point, the first frame of data is clocked in. The next operation is to check the status.

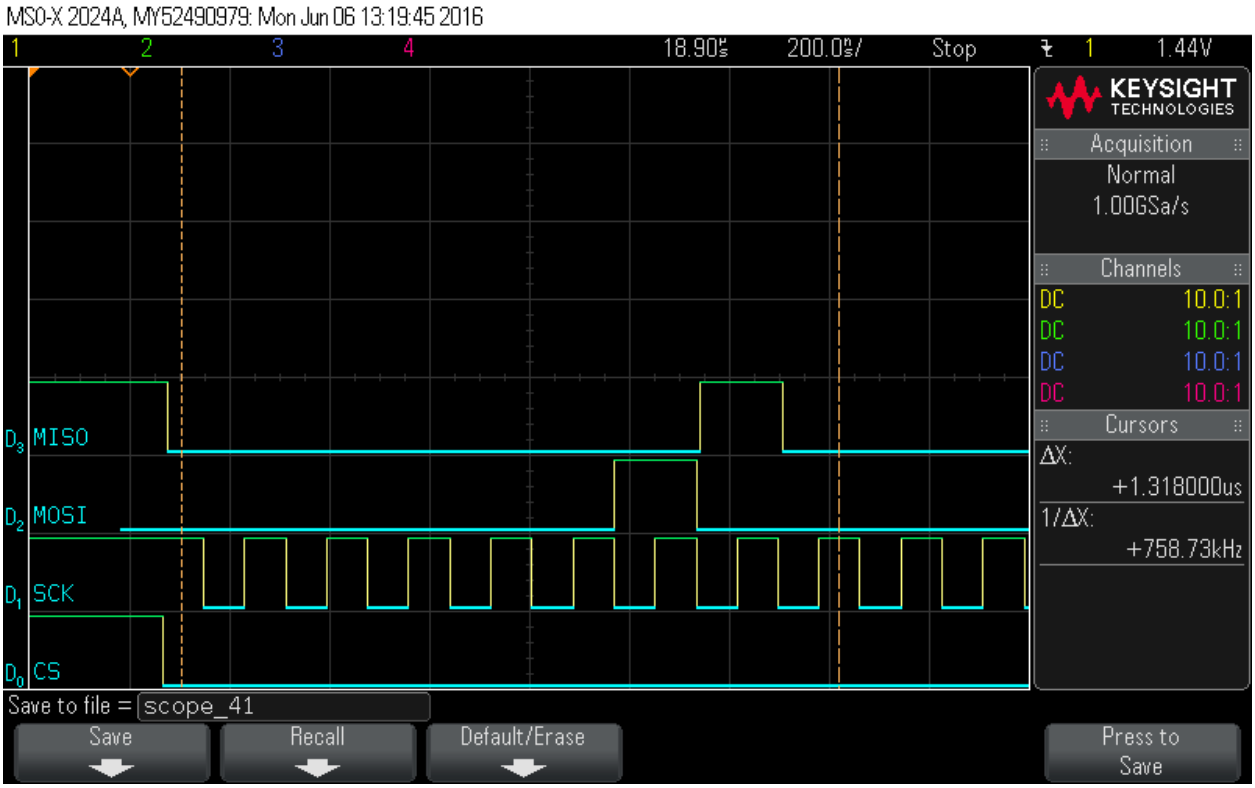
Step 20: Hardware Status Check



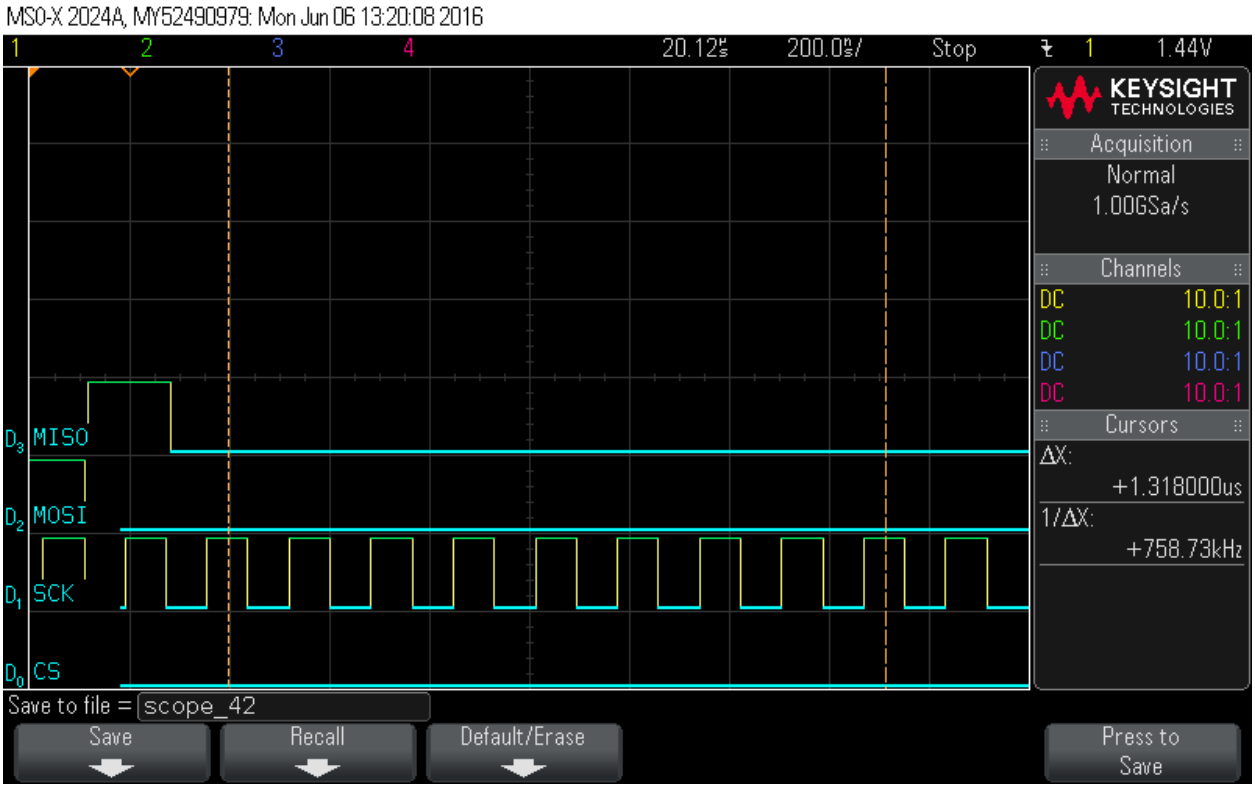
Step 21: Hardware Status Check



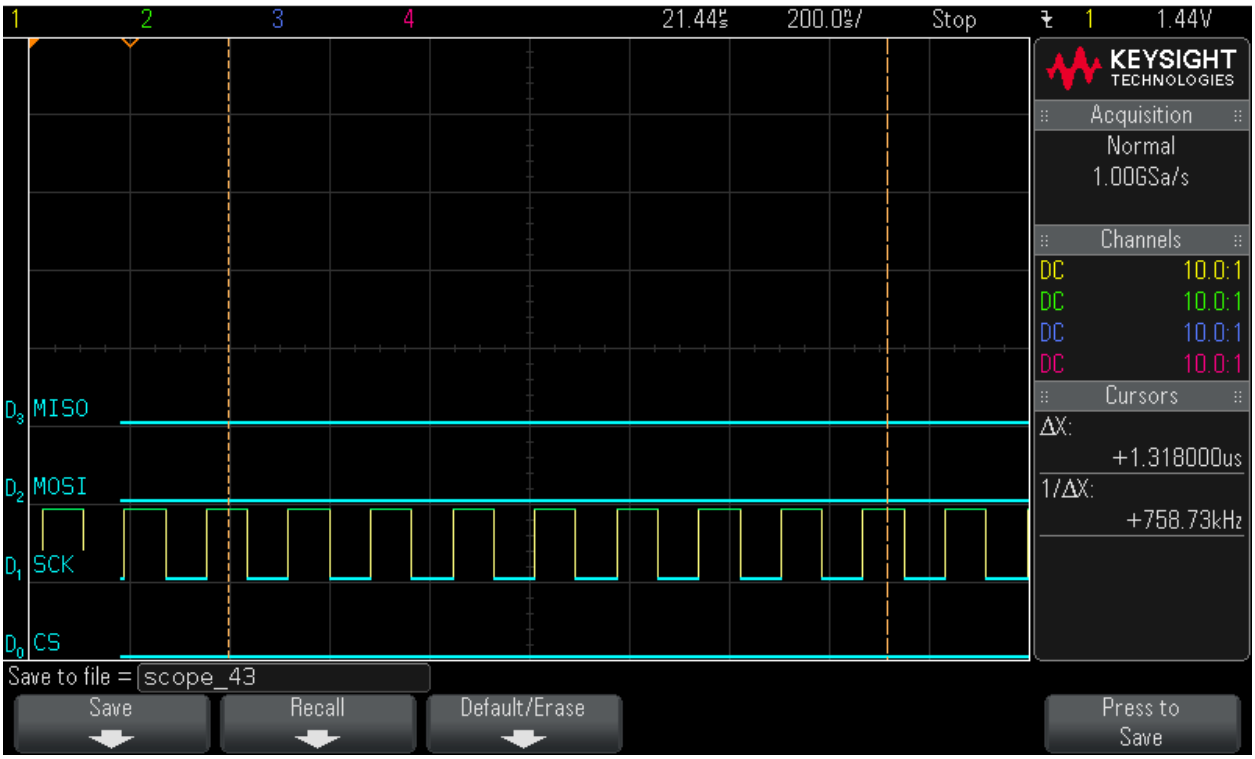
Step 22: Frame Status command. Command = 0x4. **Note CS signal**



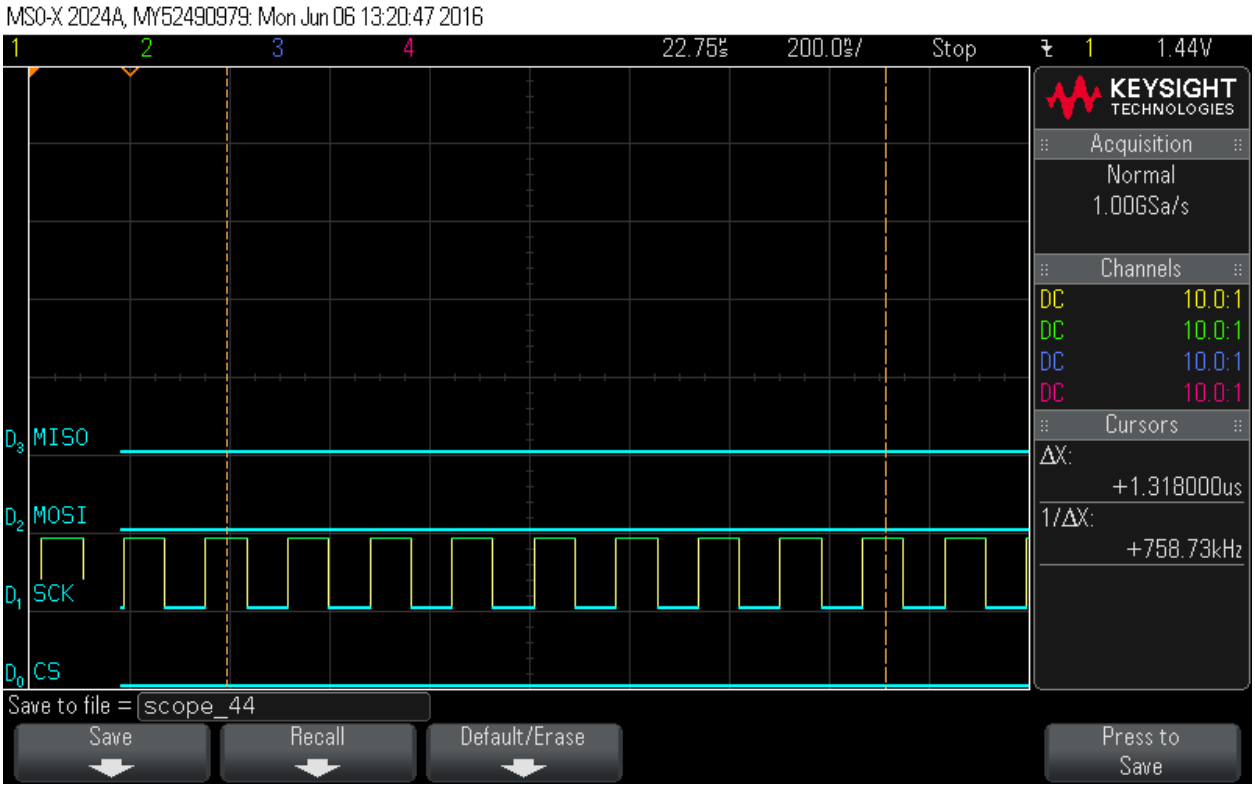
Step 23: Data Byte0 = 0x0



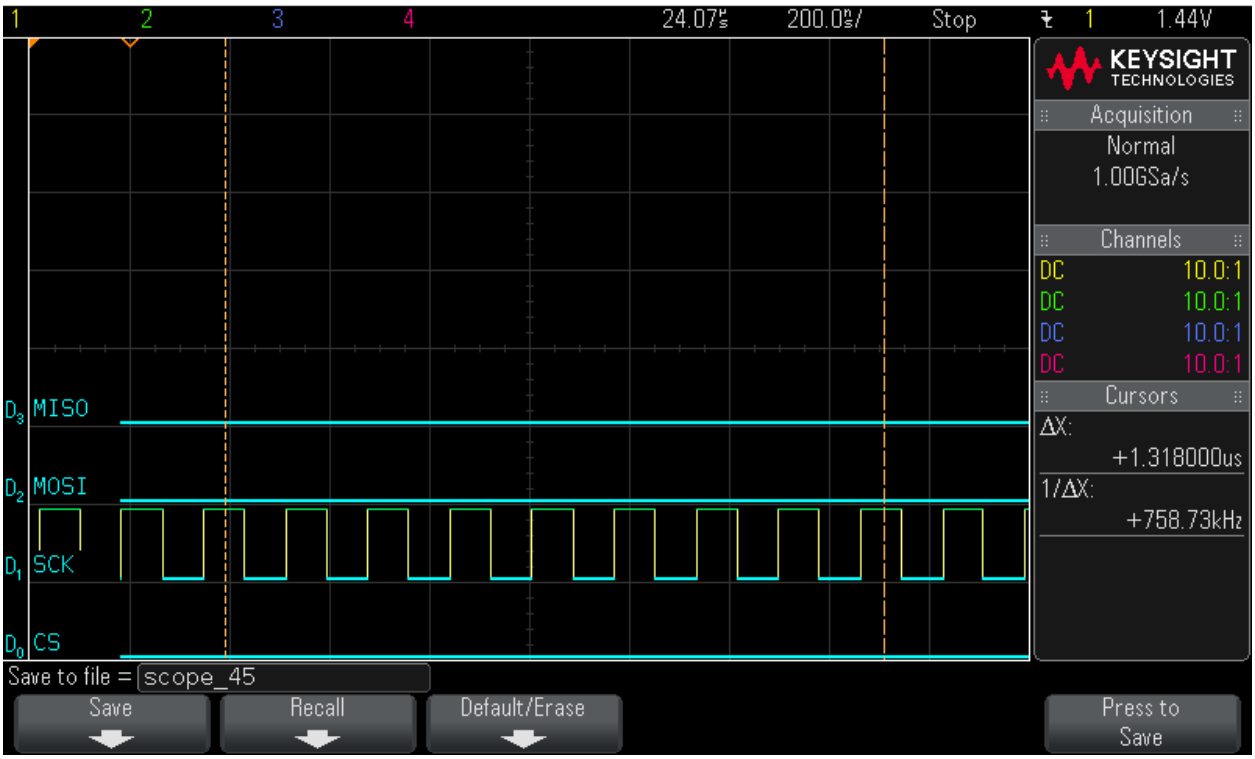
Step 24: Data Byte1 = 0x0



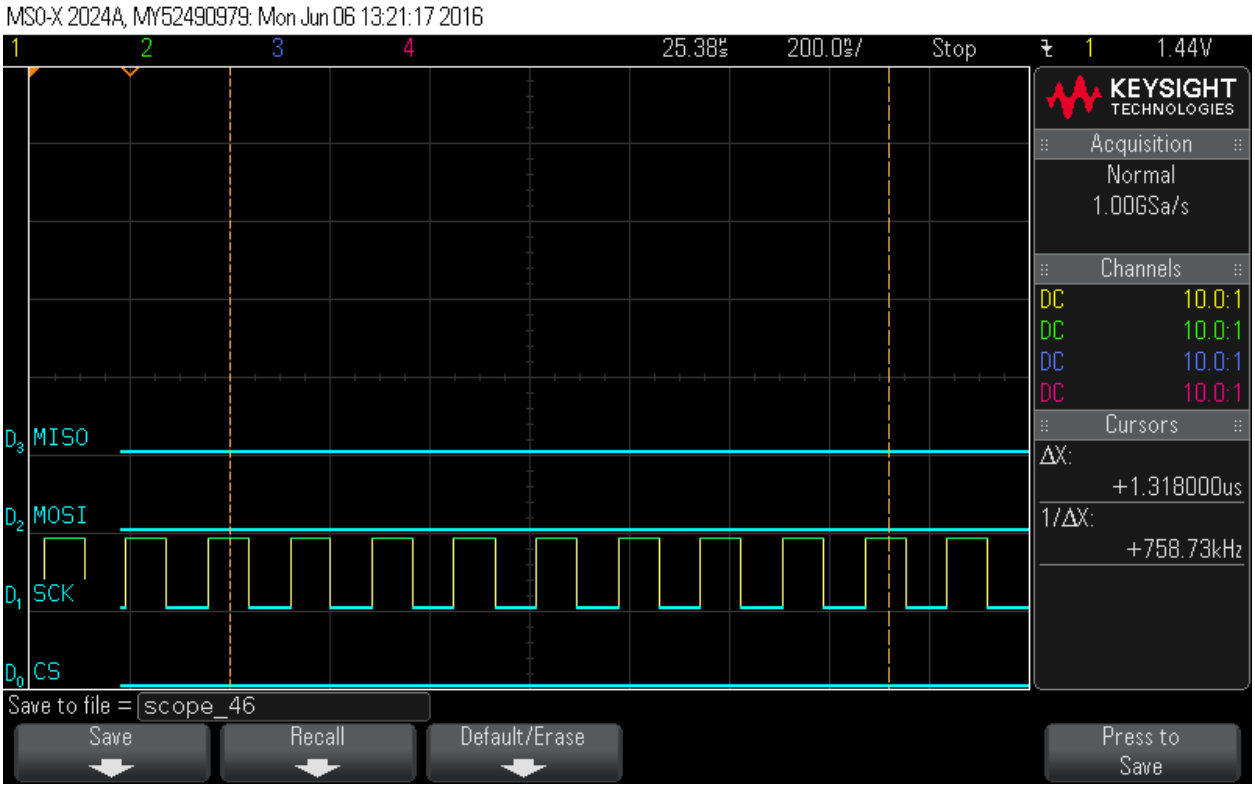
Step 25: Data Byte2 = 0x0



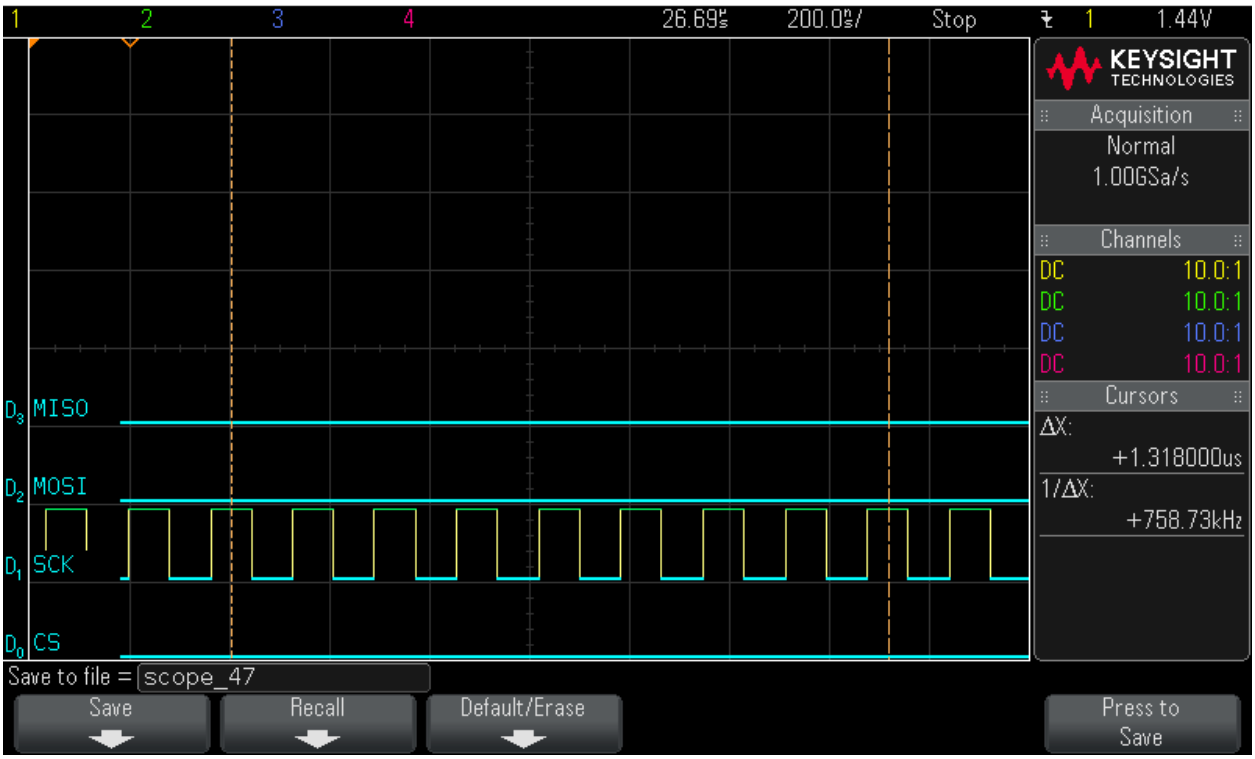
Step 26: Data Byte3 = 0x0



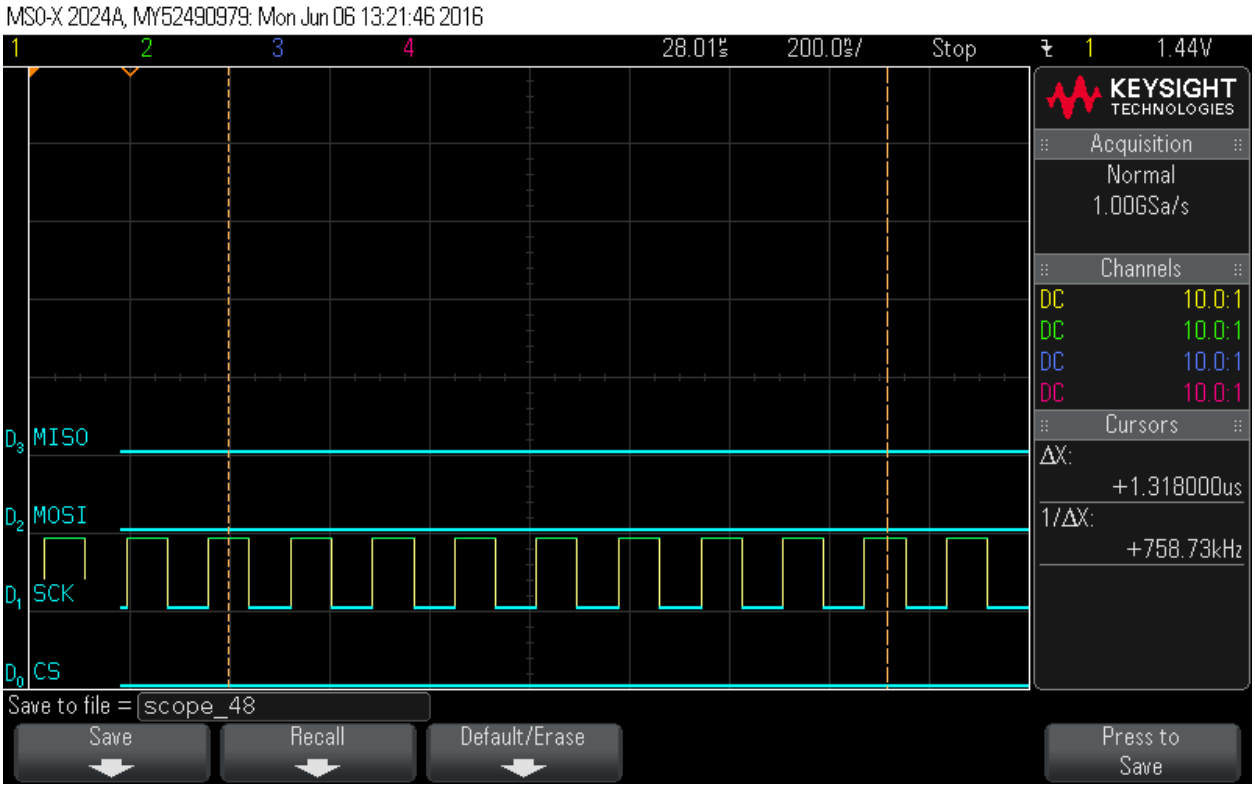
Step 27: Data Byte4 = 0x0



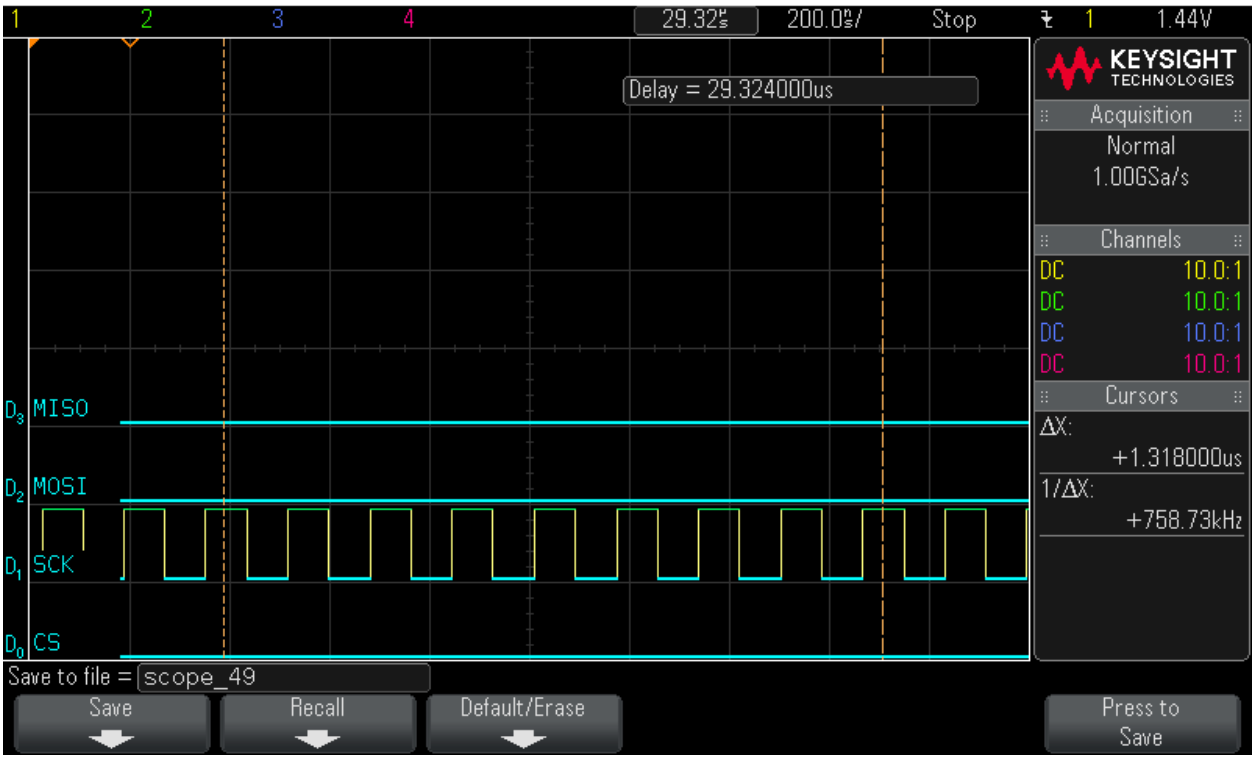
Step 28: Data Byte5 = 0x0



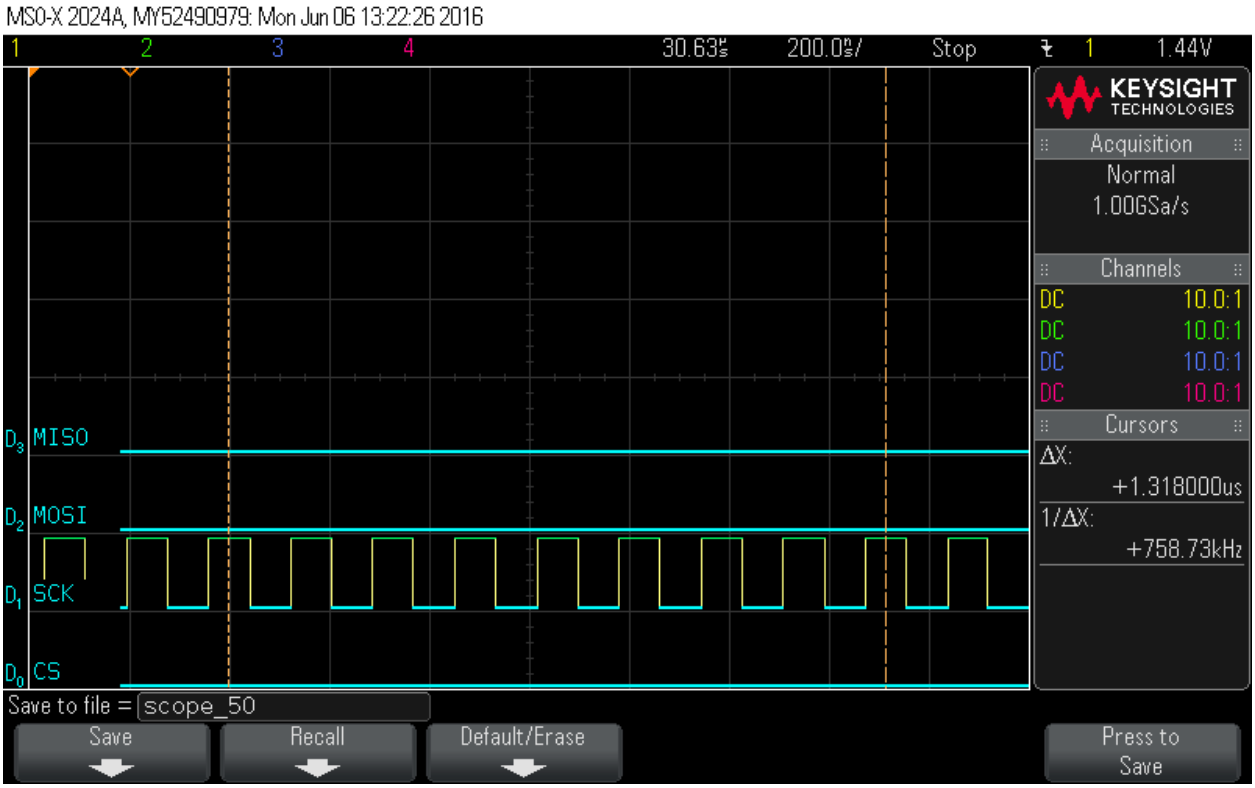
Step 29: Data Byte6 = 0x0



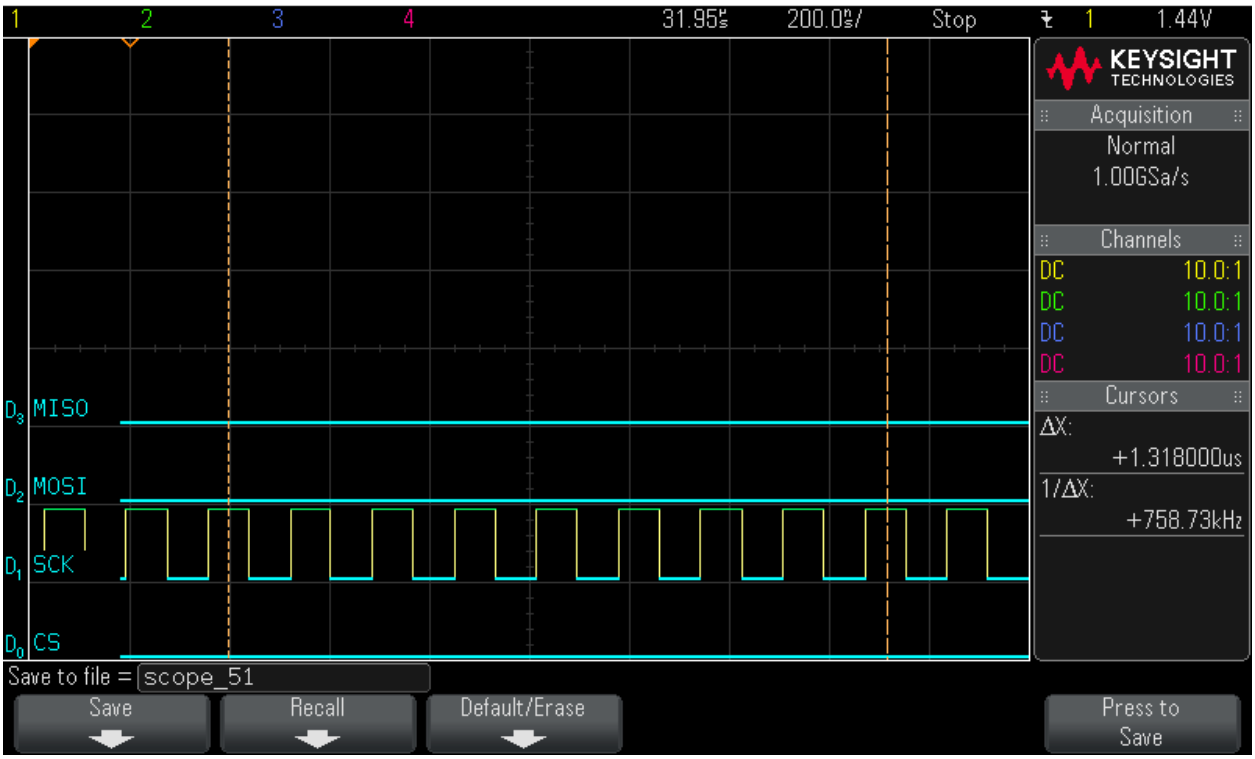
Step 30: Data Byte7 = 0x0



Step 31: Data Byte8 = 0x0

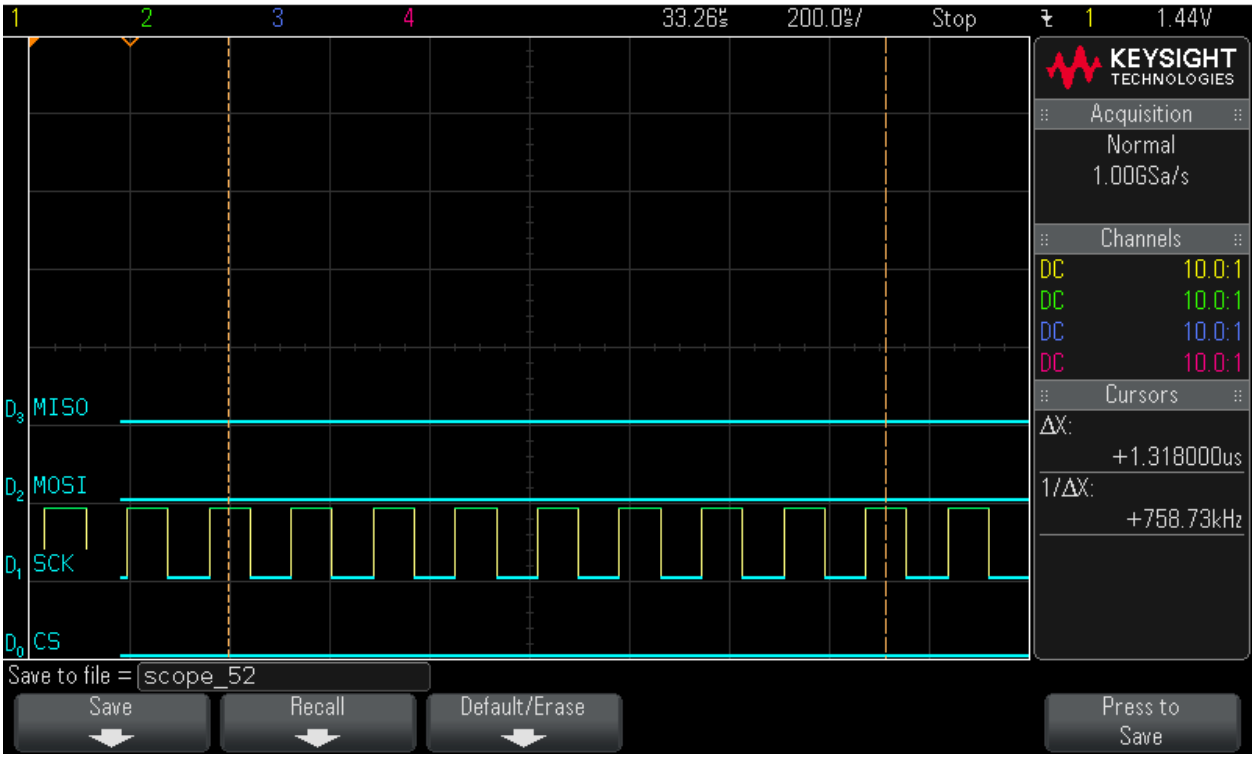


Step 32: Data Byte9 = 0x0



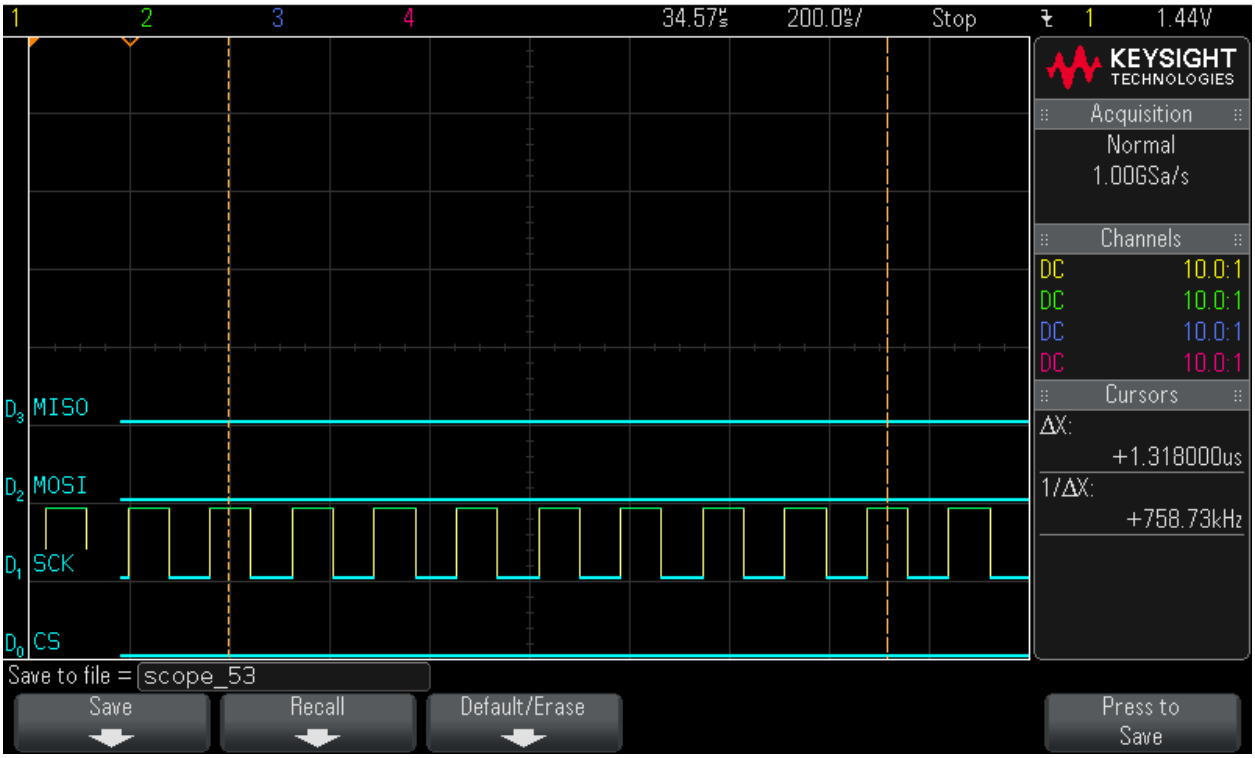
Step 33: Data Byte10 = 0x0

MSO-X 2024A, MY52490979: Mon Jun 06 13:22:54 2016



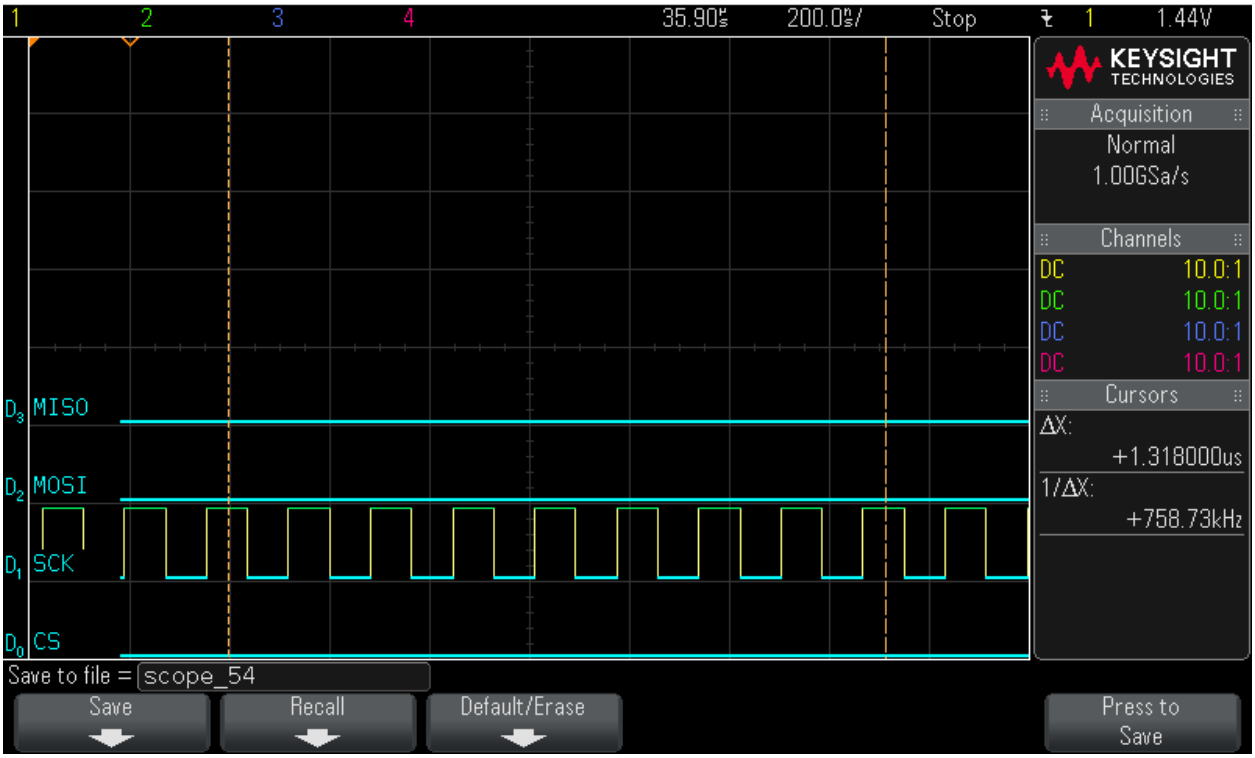
Step 34: Data Byte11 = 0x0

MSO-X 2024A, MY52490979: Mon Jun 06 13:23:05 2016



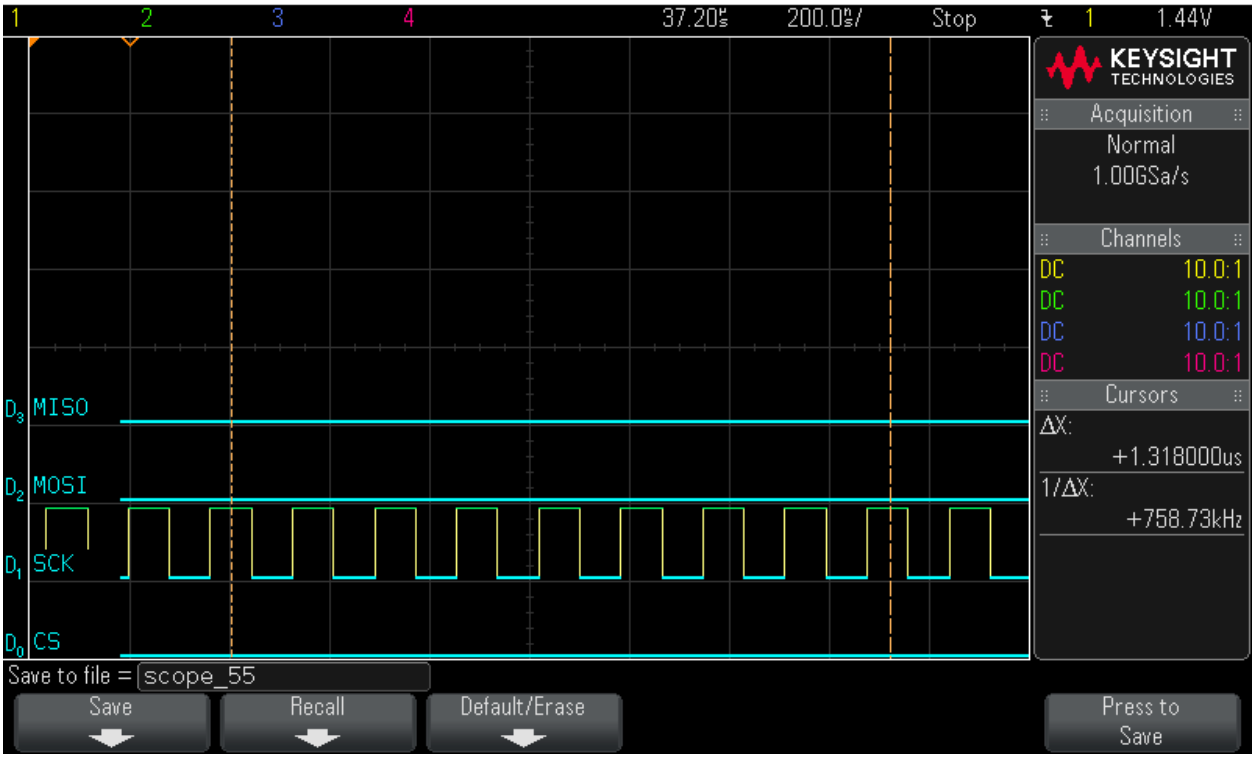
Step 35: Data Byte12 = 0x0

MSO-X 2024A, MY52490979: Mon Jun 06 13:23:19 2016

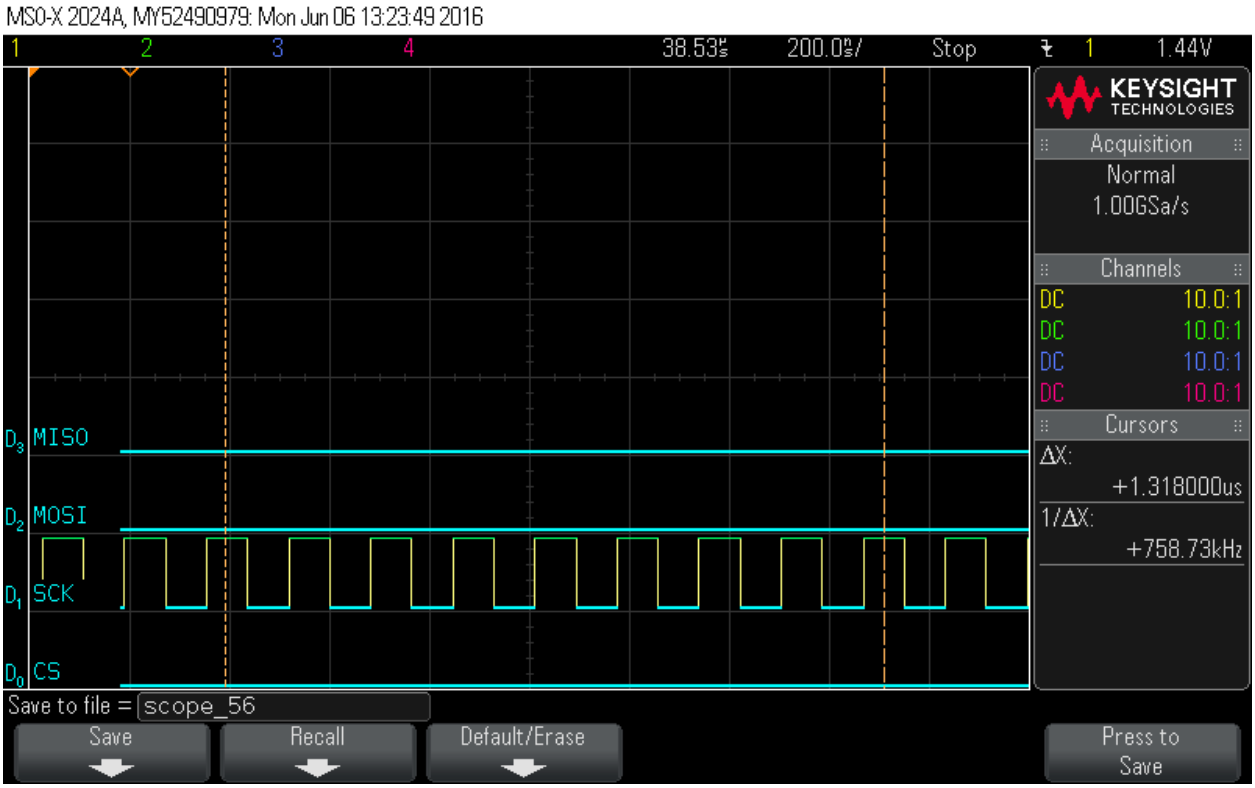


Step 36: Data Byte13 = 0x0

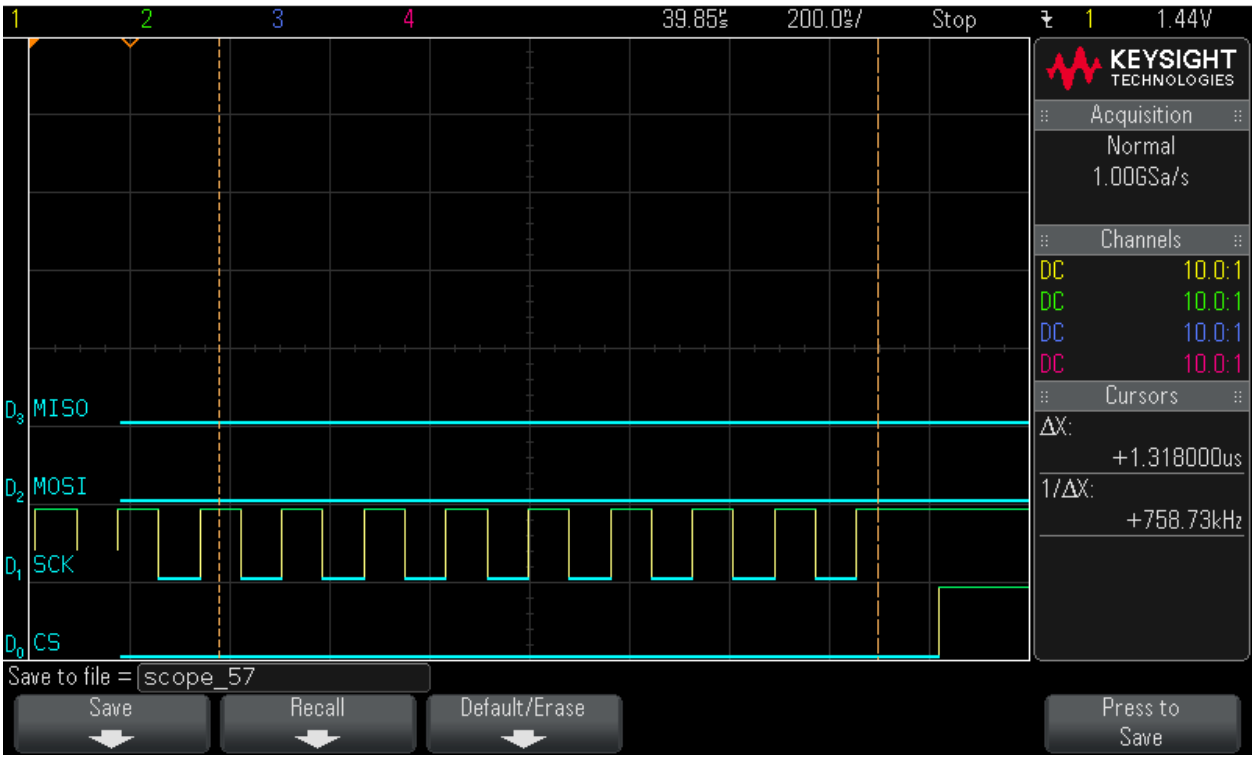
MSO-X 2024A, MY52490979: Mon Jun 06 13:23:32 2016



Step 37: Data Byte14 = 0x0

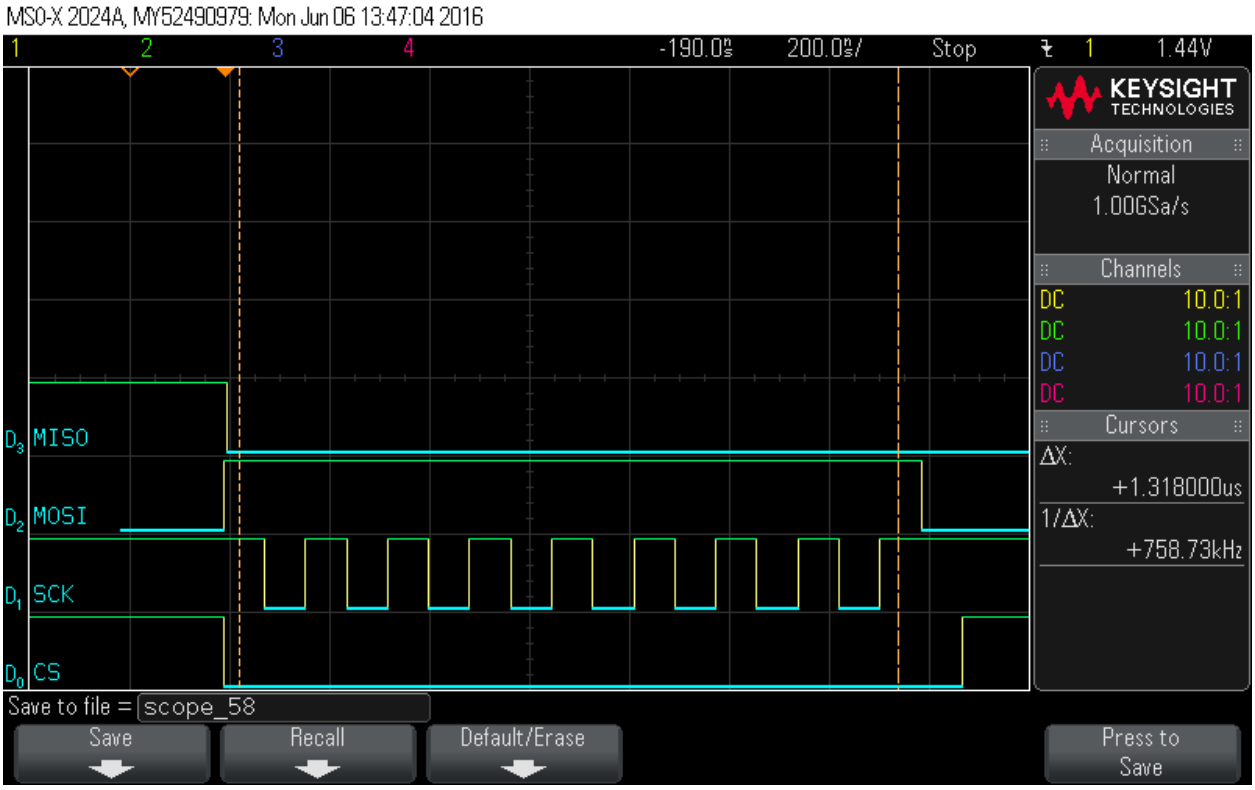


Step 38: Data Byte15 = 0x0

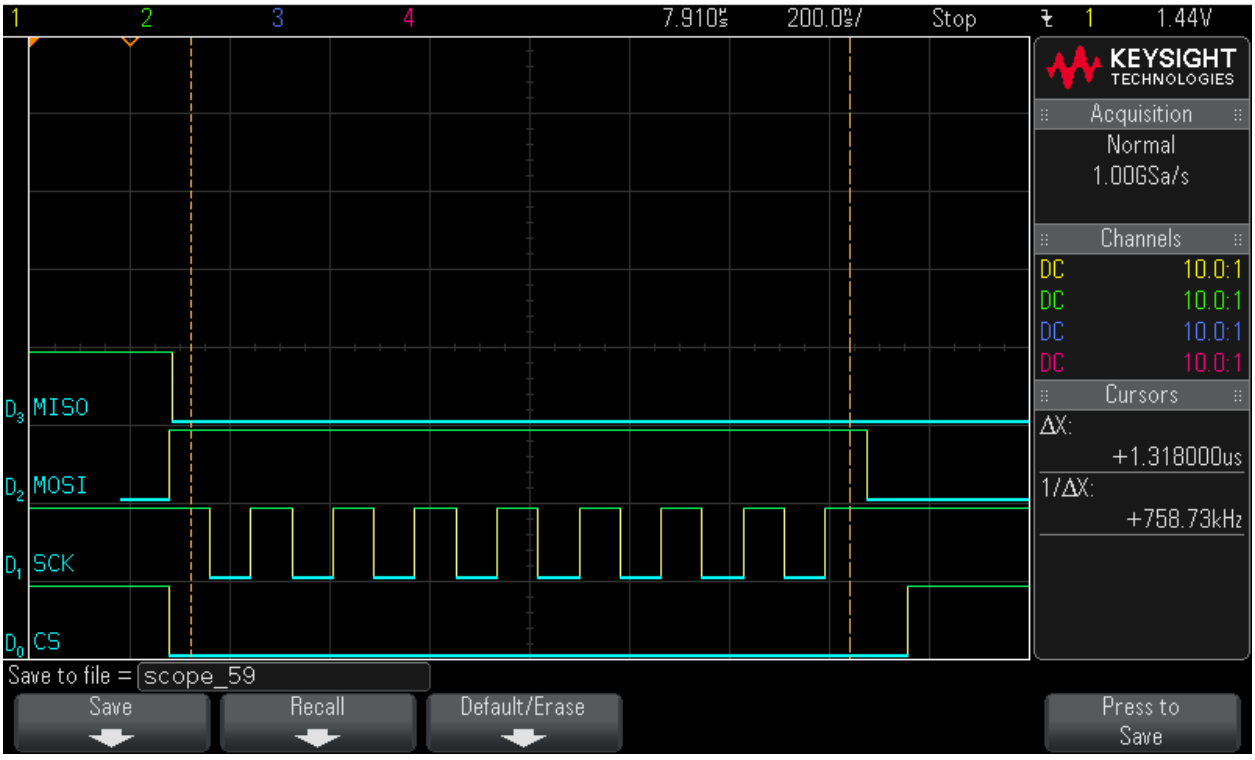


Instruction is loaded. Issue read instruction using 0x5 command

Step 39: Hardware Status Check

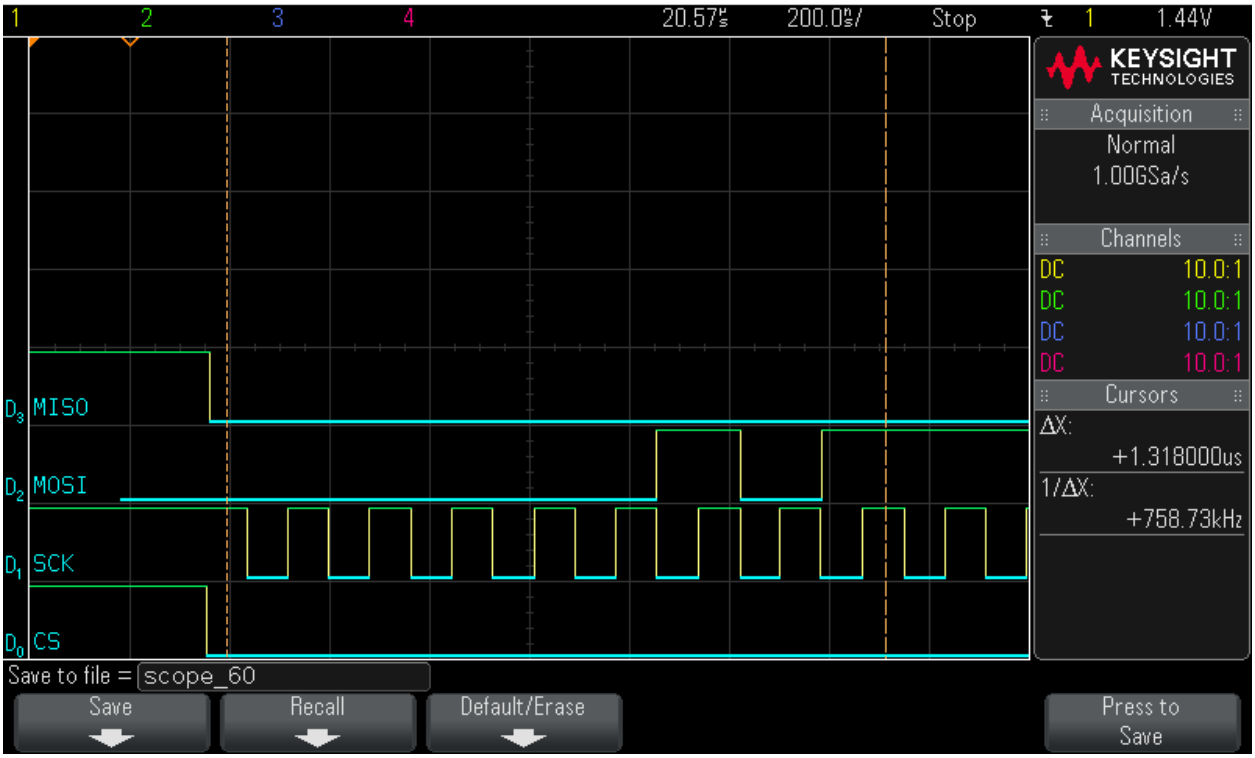


Step 40: Hardware Status Check



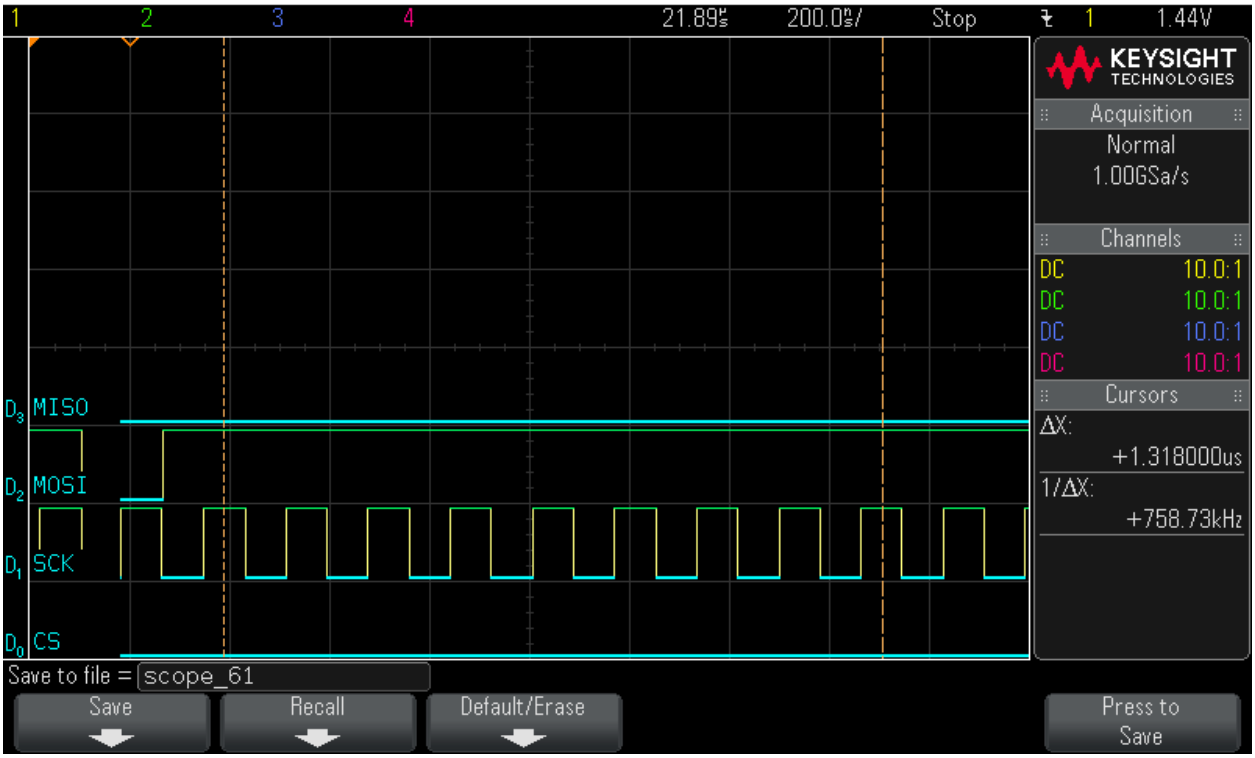
Step 41: Read command. Command = 0x5. **Note CS signal**

MSO-X 2024A, MY52490979: Mon Jun 06 13:47:45 2016

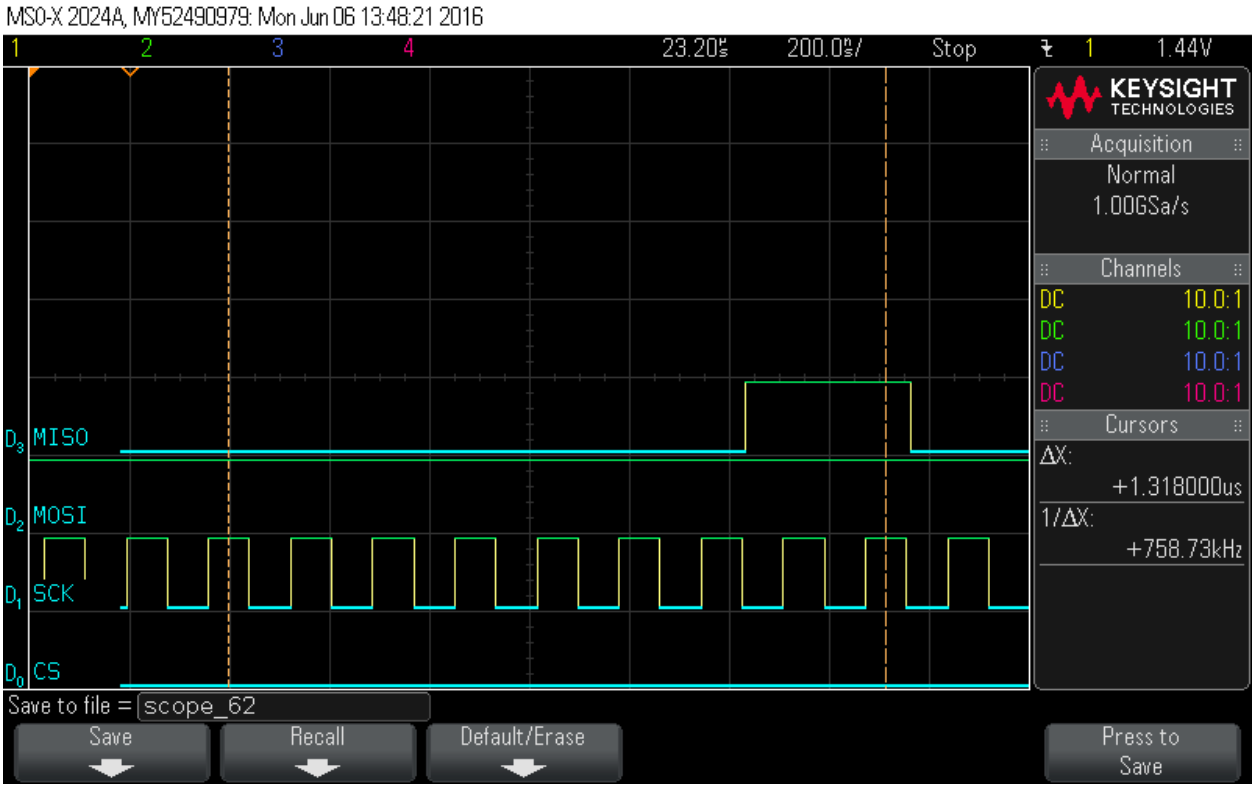


Step 42: Data Byte0 read

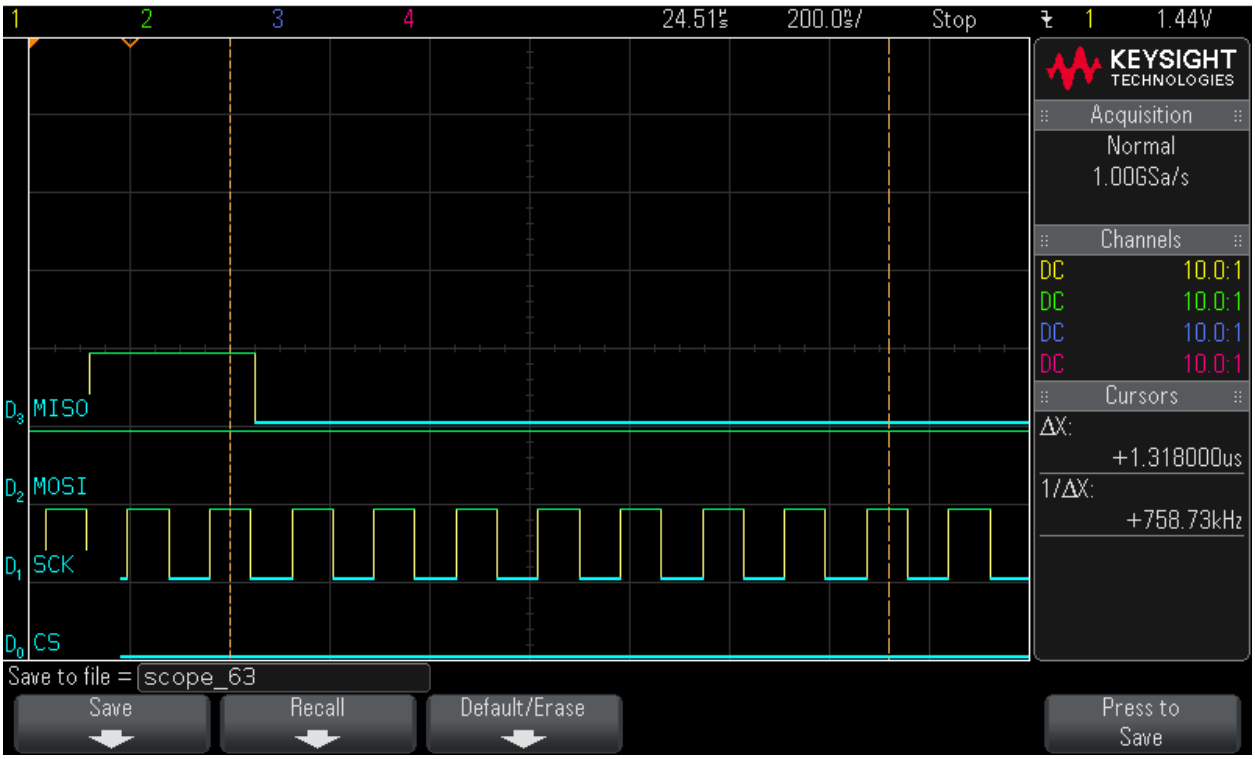
MSO-X 2024A, MY52490979: Mon Jun 06 13:48:02 2016



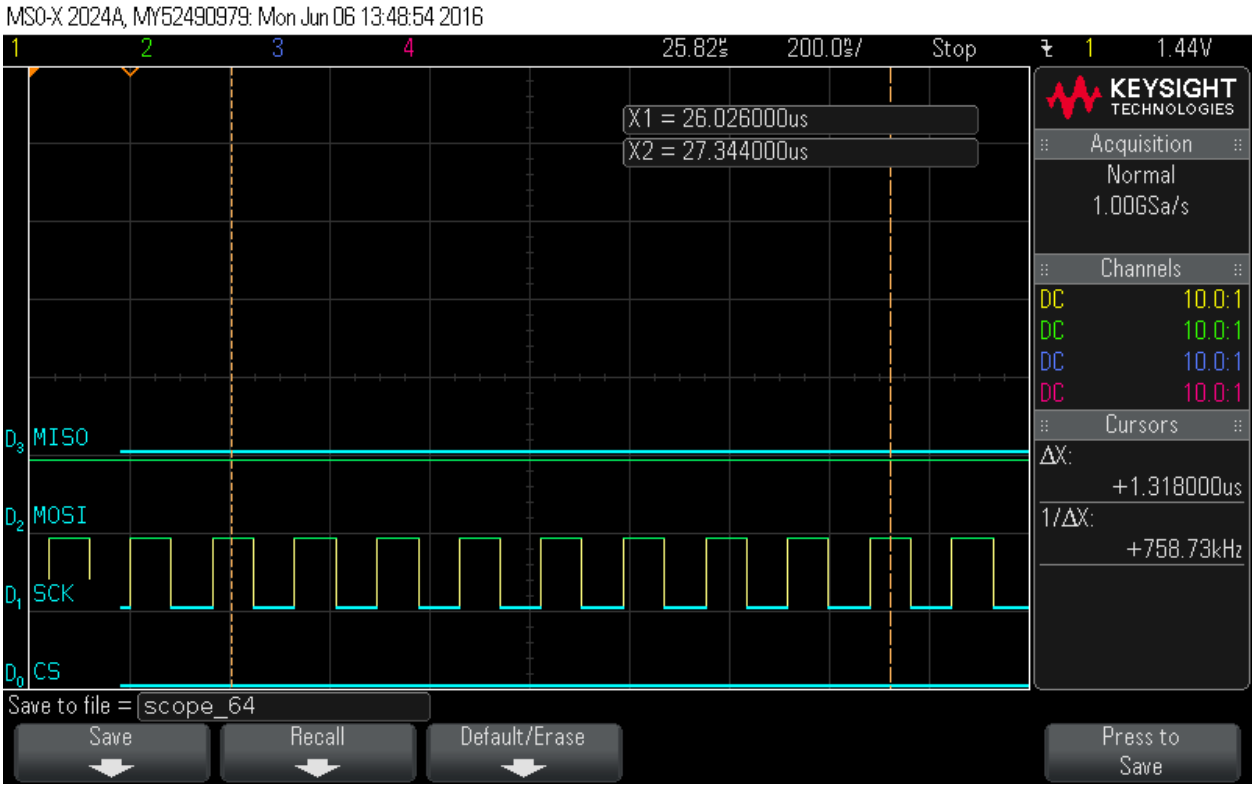
Step 43: Data Byte1 read



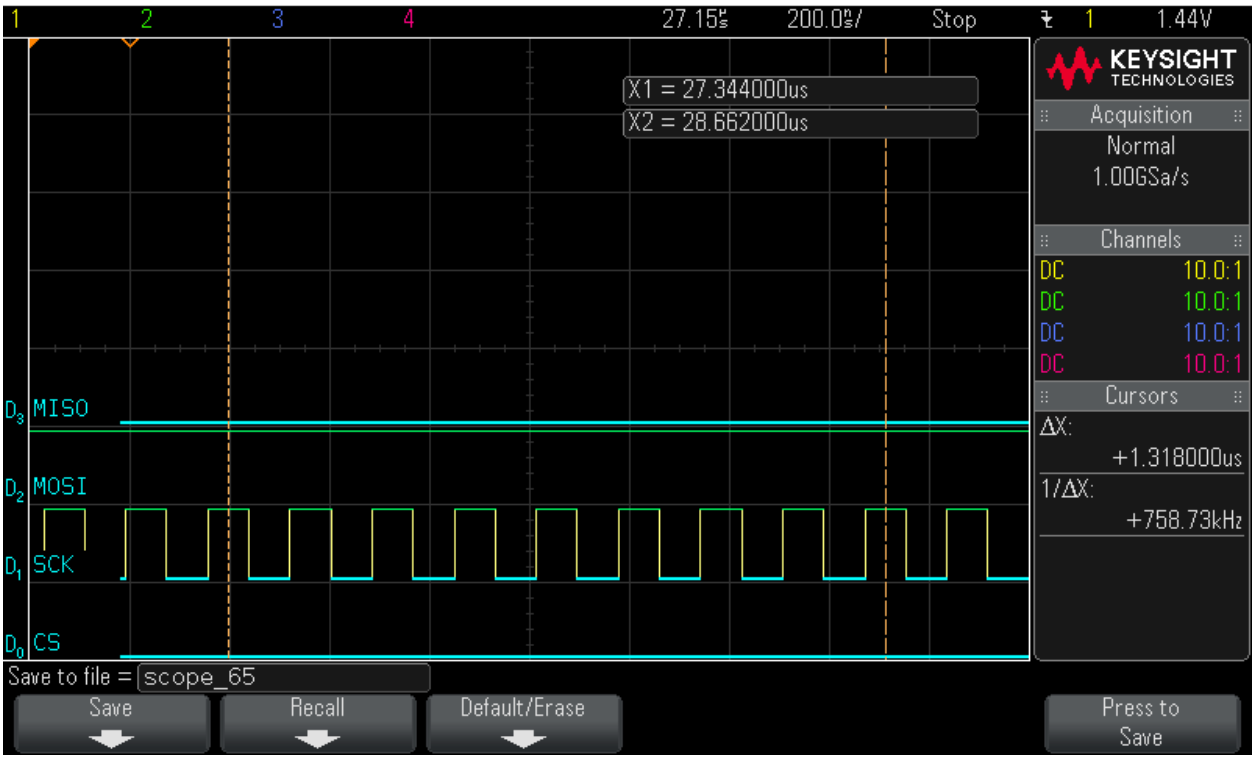
Step 44: Data Byte2 read



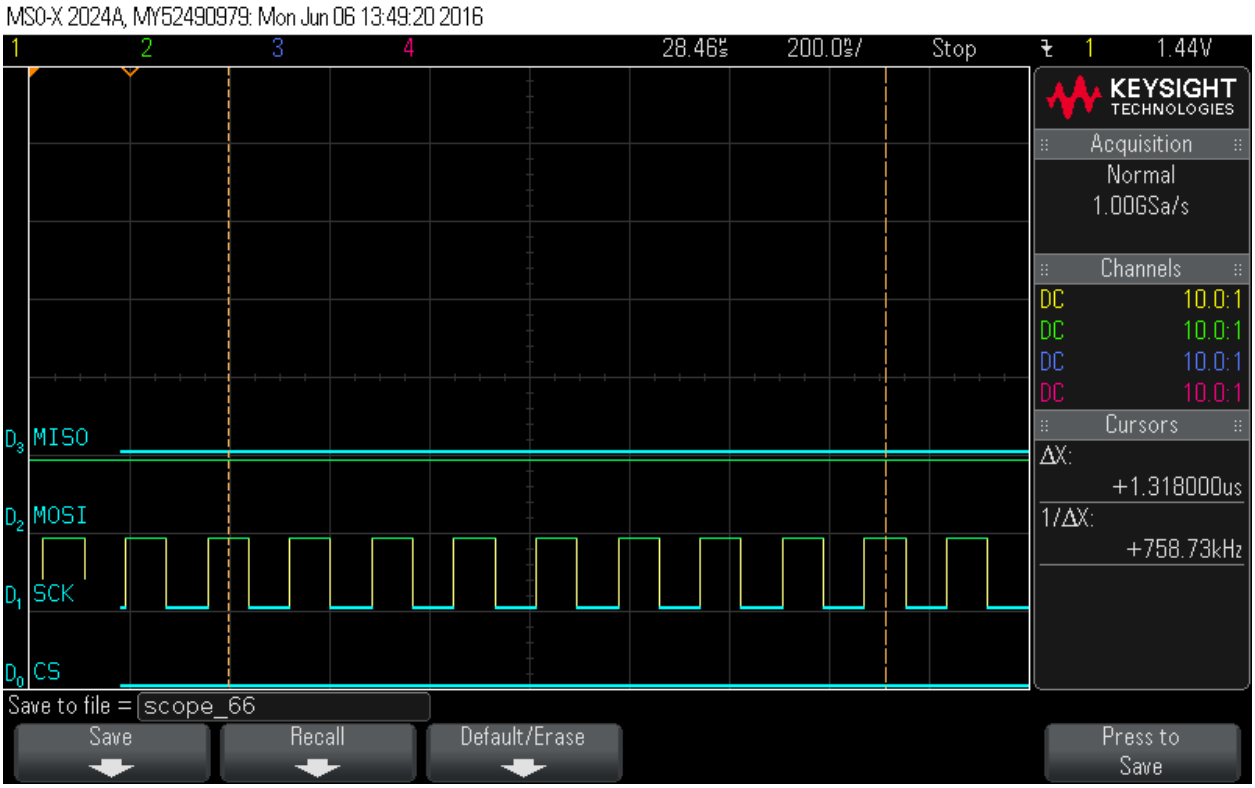
Step 45: Data Byte3 read



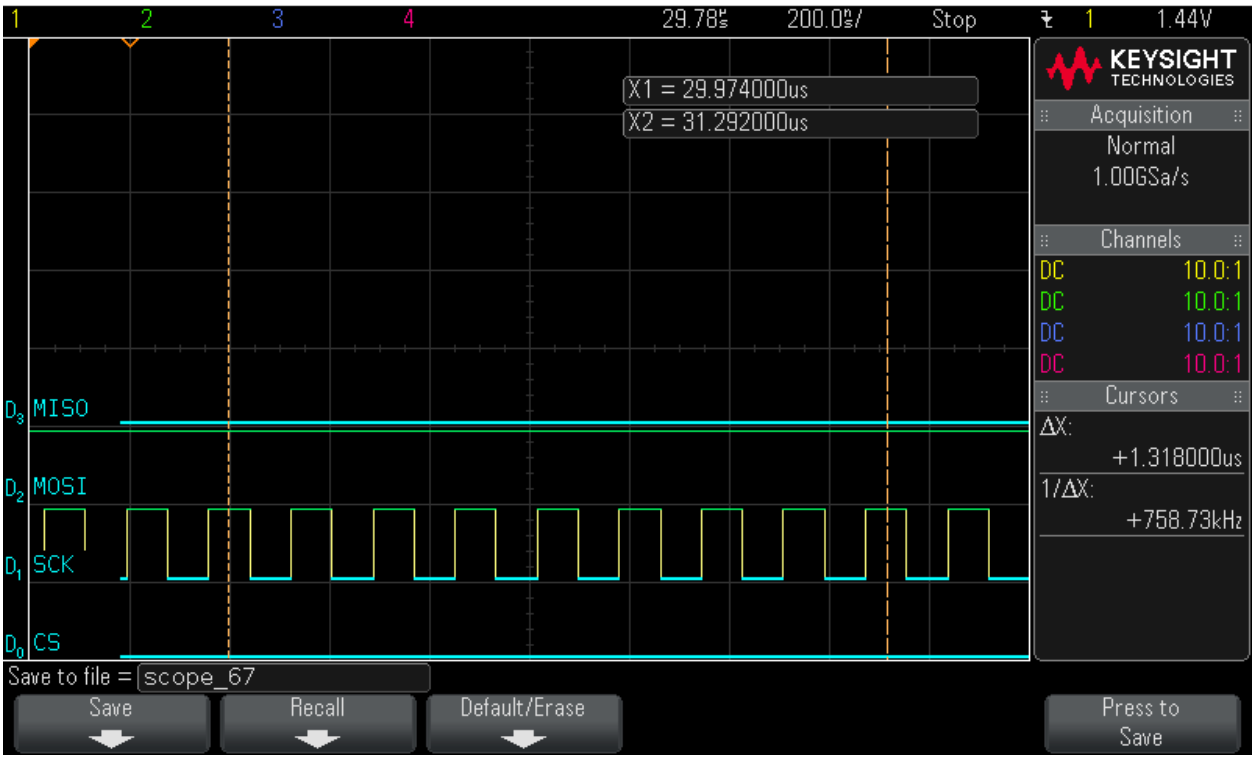
Step 46: Data Byte4 read



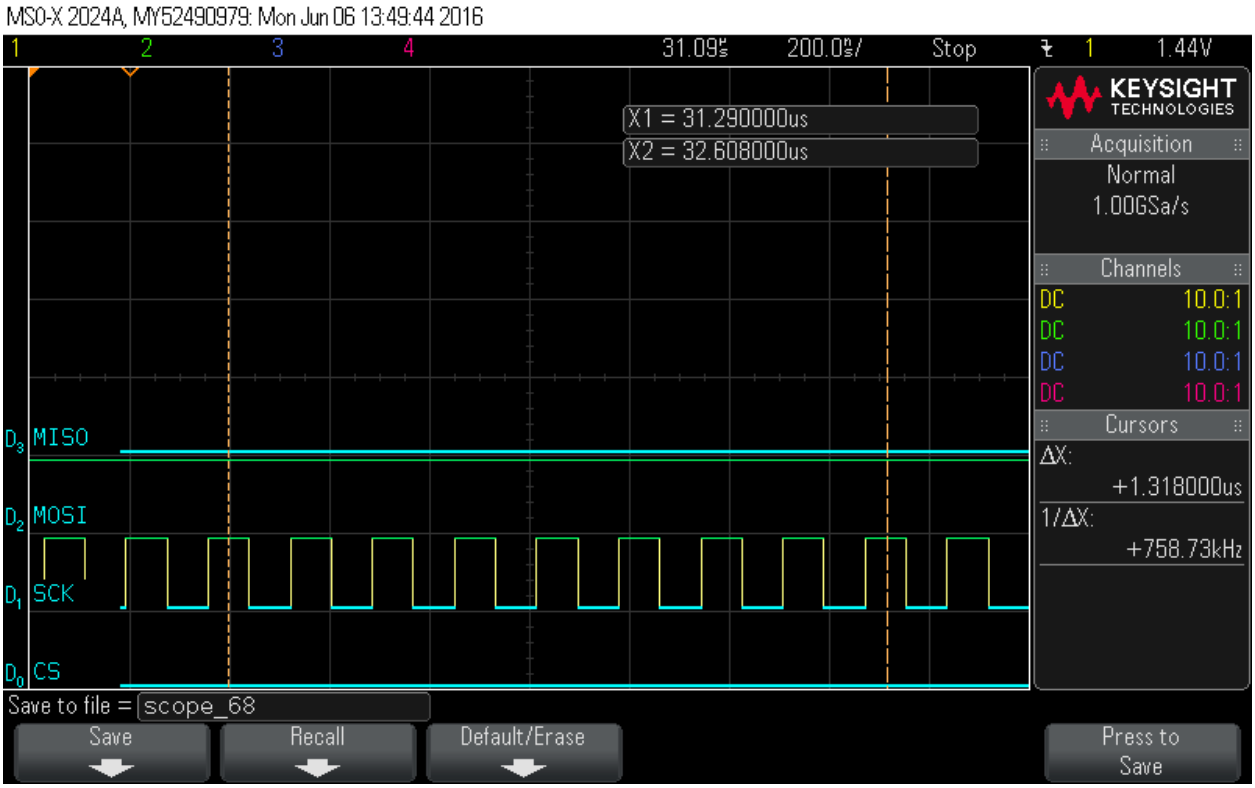
Step 47: Data Byte5 read



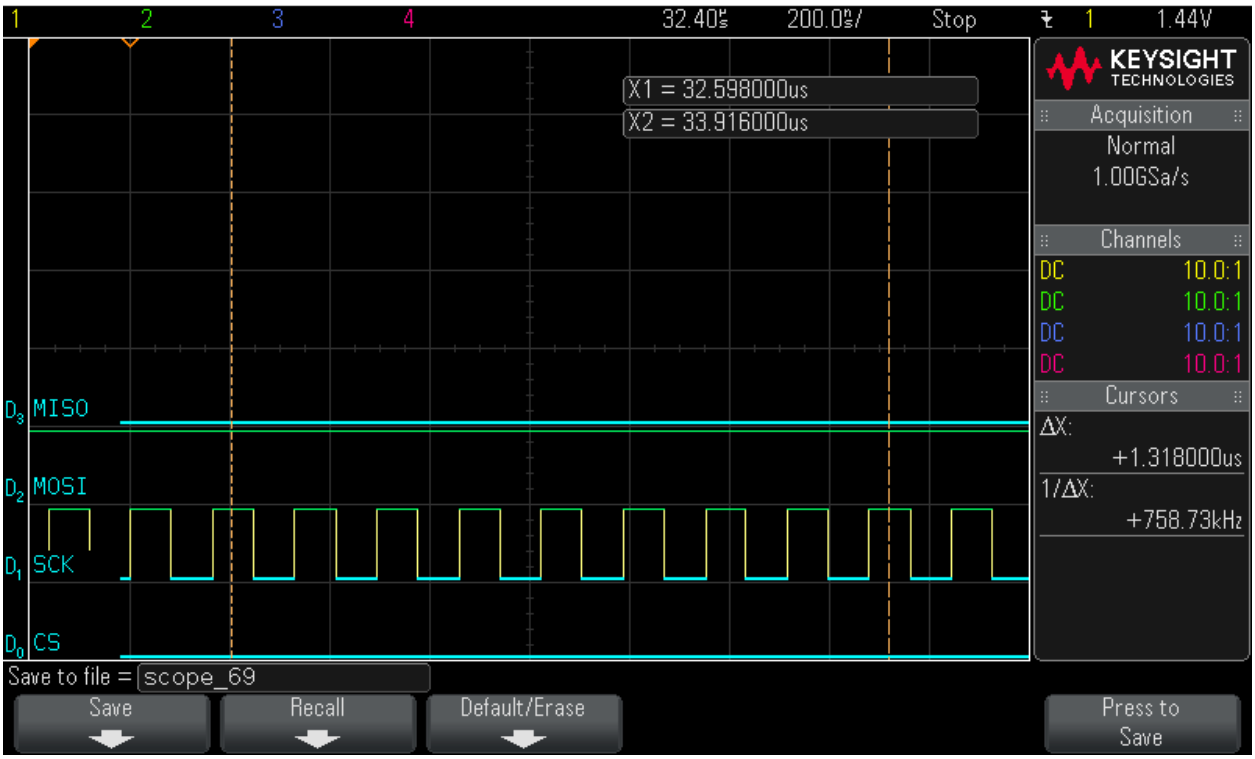
Step 48: Data Byte6 read



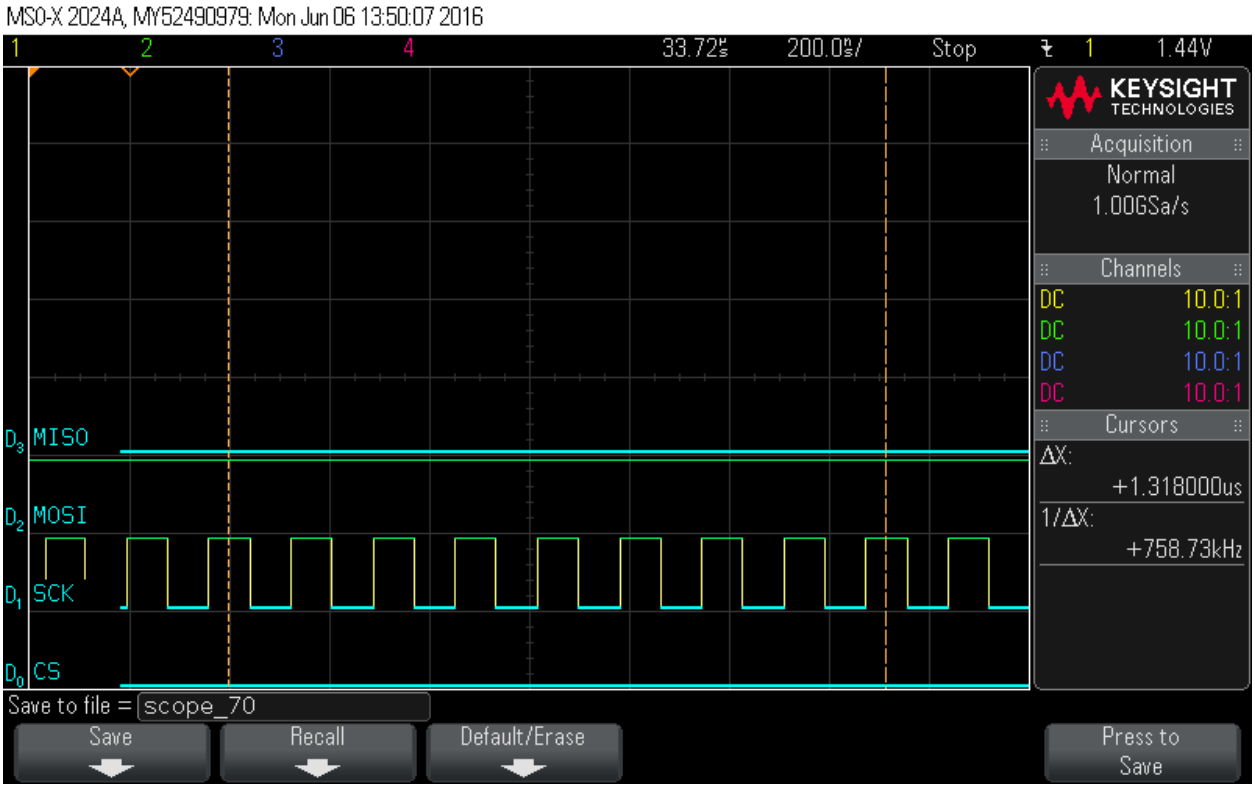
Step 49: Data Byte7 read



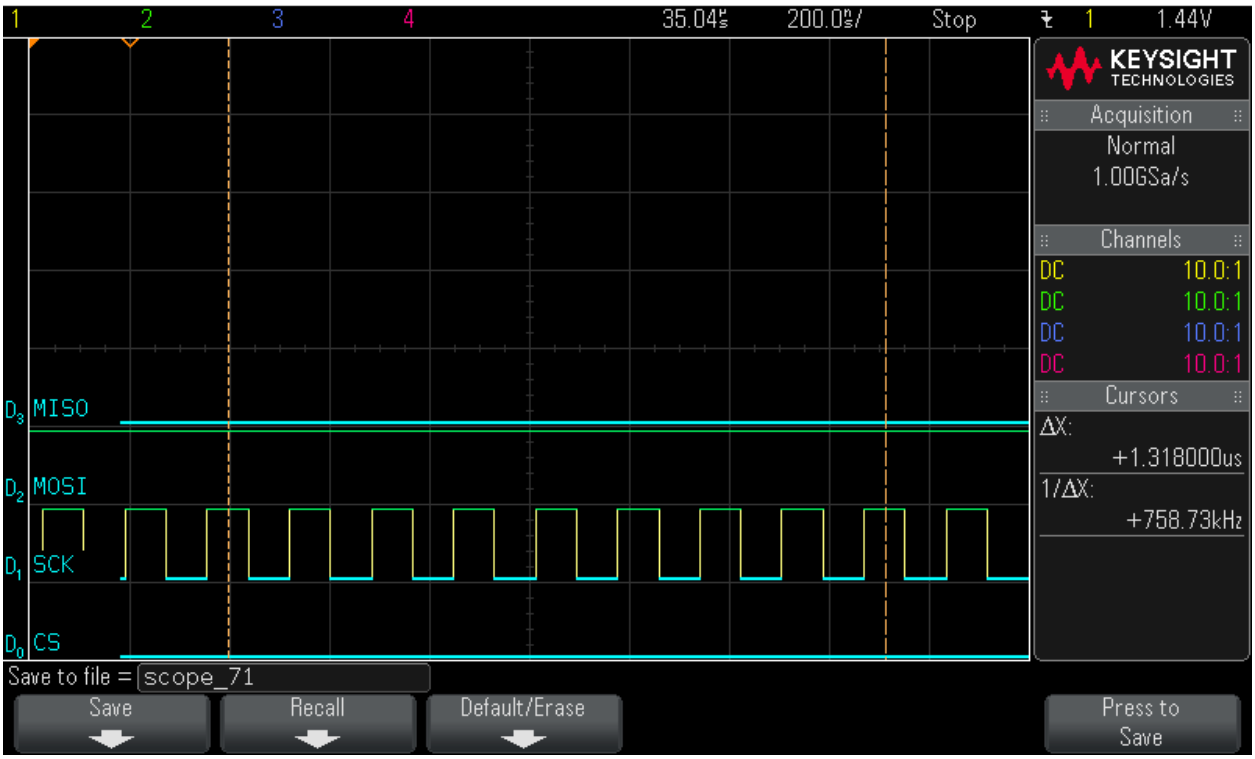
Step 50: Data Byte8 read



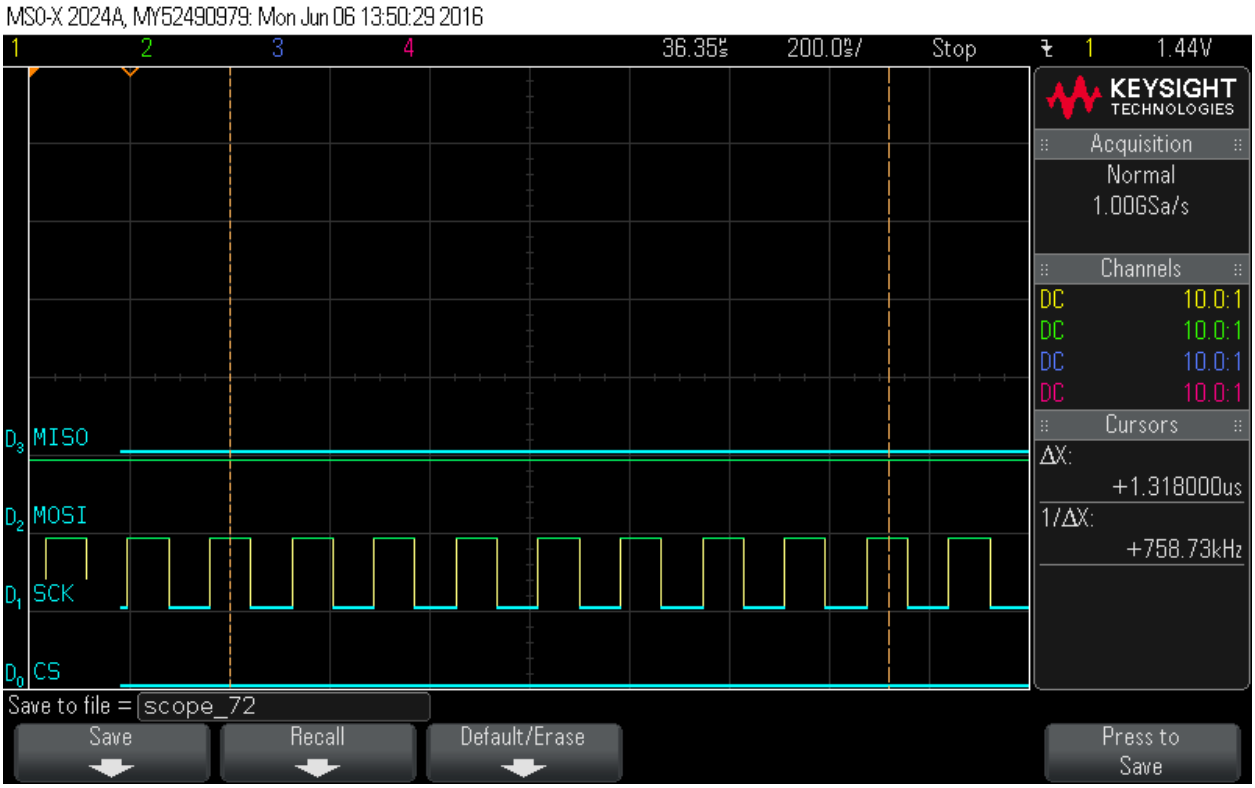
Step 51: Data Byte9 read



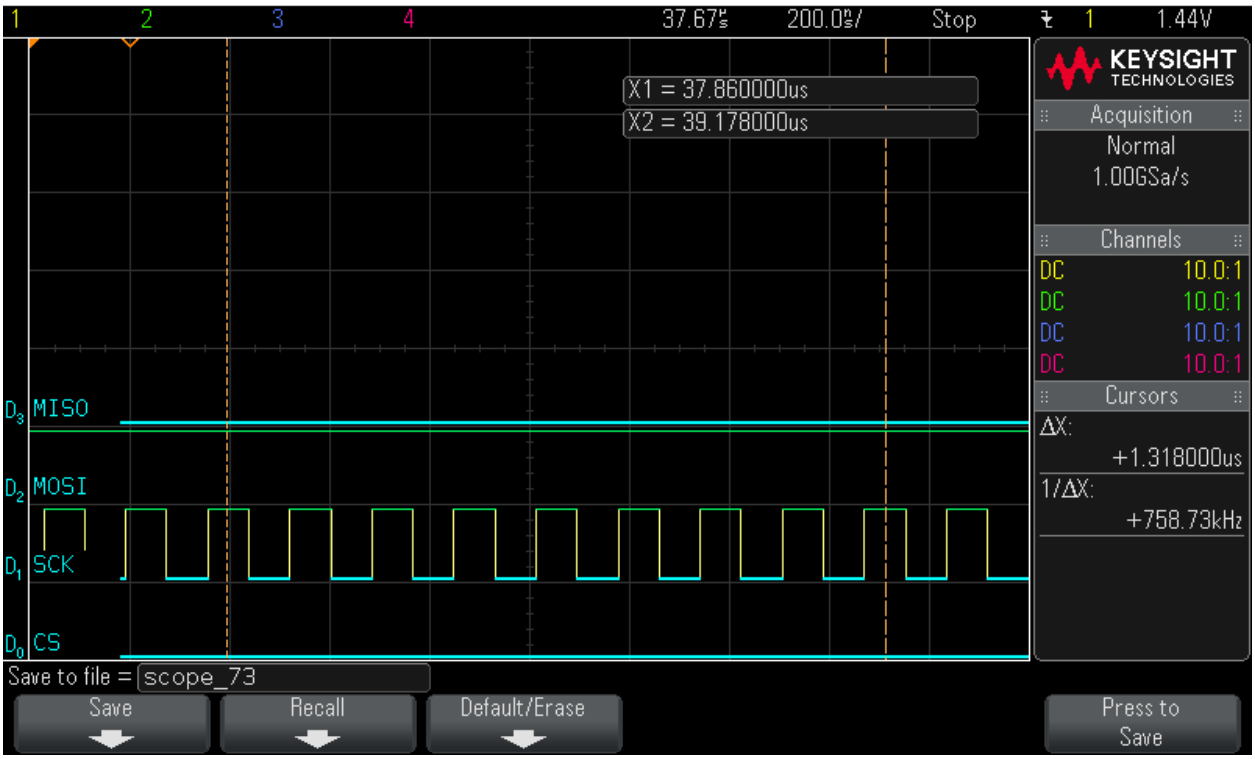
Step 52: Data Byte10 read



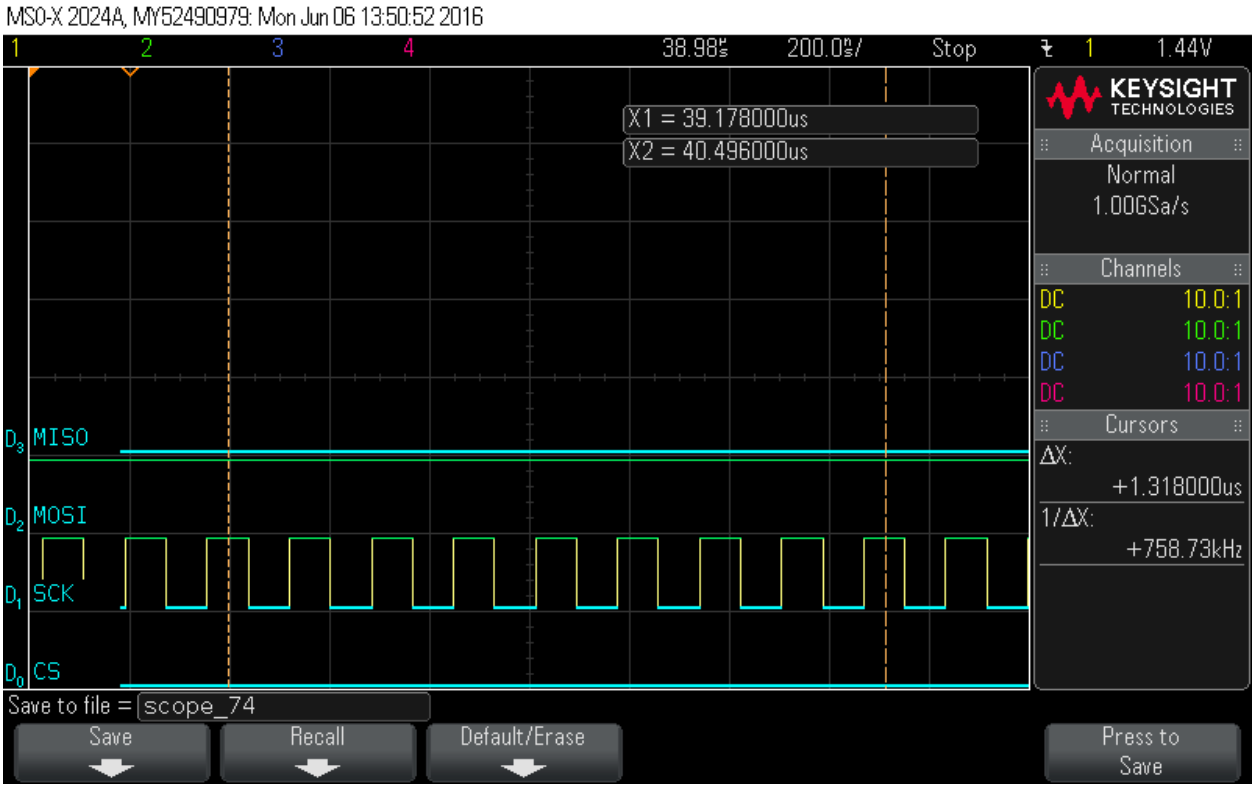
Step 53: Data Byte11 read



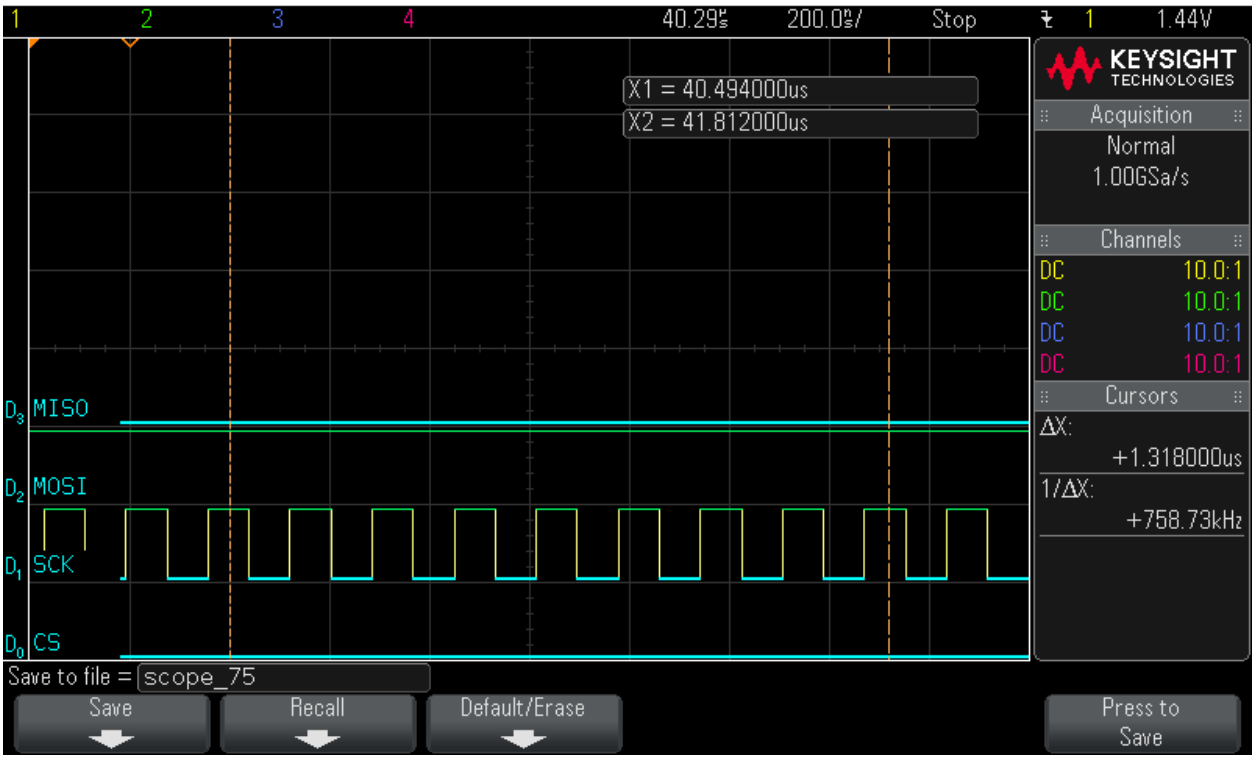
Step 54: Data Byte12 read



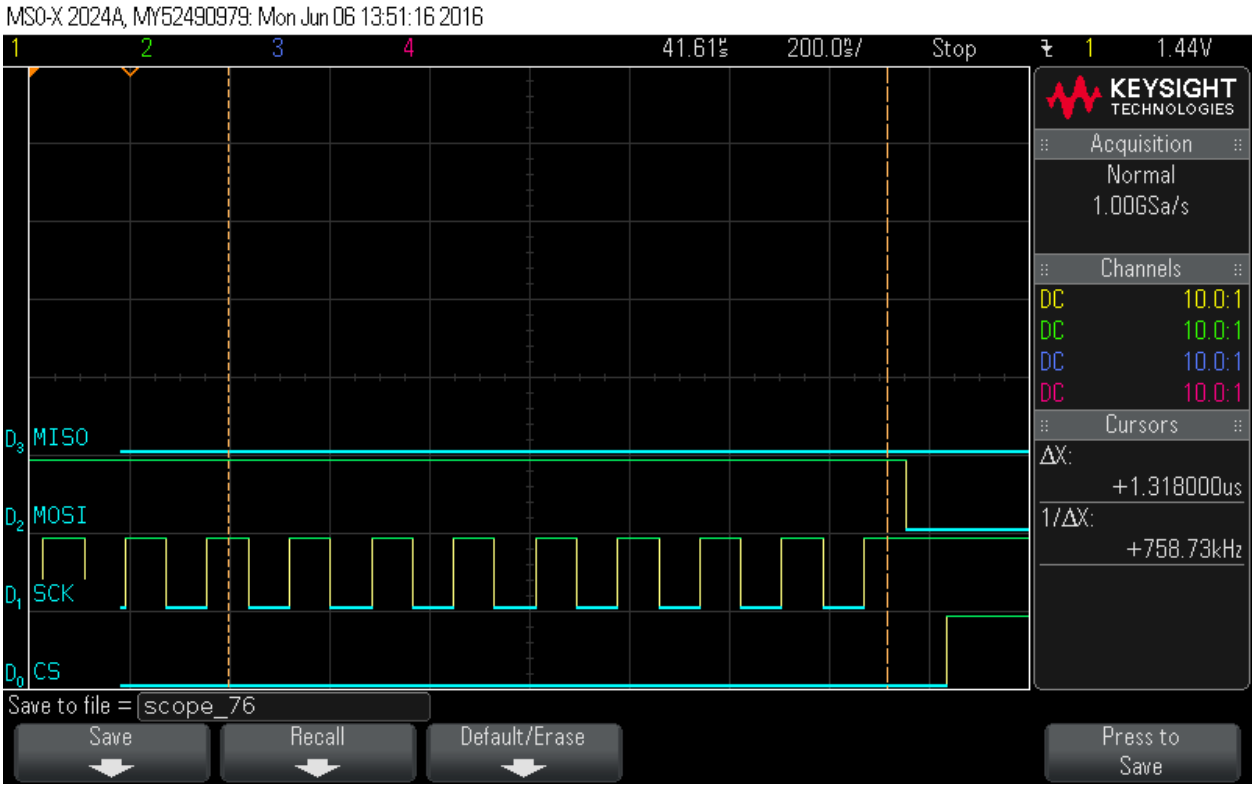
Step 55: Data Byte13 read



Step 56: Data Byte14 read



Step 57: Data Byte15 read



B – Product Support

Microsemi SoC Products Group backs its products with various support services, including Customer Service, Customer Technical Support Center, a website, electronic mail, and worldwide sales offices. This appendix contains information about contacting Microsemi SoC Products Group and using these support services.

Customer Service

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**

Customer Technical Support Center

Microsemi SoC Products Group staffs its Customer Technical Support Center with highly skilled engineers who can help answer your hardware, software, and design questions about Microsemi SoC Products. The Customer Technical Support Center spends a great deal of time creating application notes, answers to common design cycle questions, documentation of known issues, and various FAQs. So, before you contact us, please visit our online resources. It is very likely we have already answered your questions.

Technical Support

For Microsemi SoC Products Support, visit http://www.microsemi.com/products/fpga-soc/design-support/fpga-soc-support.

Website

You can browse a variety of technical and non-technical information on the Microsemi SoC Products Group home page, at www.microsemi.com/soc.

Contacting the Customer Technical Support Center

Highly skilled engineers staff the Technical Support Center. The Technical Support Center can be contacted by email or through the Microsemi SoC Products Group website.

Email

You can communicate your technical questions to our email address and receive answers back by email, fax, or phone. Also, if you have design problems, you can email your design files to receive assistance. We constantly monitor the email account throughout the day. When sending your request to us, please be sure to include your full name, company name, and your contact information for efficient processing of your request.

The technical support email address is soc_tech@microsemi.com.

My Cases

Microsemi SoC Products Group customers may submit and track technical cases online by going to My Cases.

Outside the U.S.

Customers needing assistance outside the US time zones can either contact technical support via email (soc_tech@microsemi.com) or contact a local sales office.

Visit About Us for sales office listings and corporate contacts.

Sales office listings can be found at www.microsemi.com/soc/company/contact/default.aspx.

ITAR Technical Support

For technical support on RH and RT FPGAs that are regulated by International Traffic in Arms Regulations (ITAR), contact us via soc_tech_itar@microsemi.com. Alternatively, within My Cases, select **Yes** in the ITAR drop-down list. For a complete list of ITAR-regulated Microsemi FPGAs, visit the ITAR web page.

Microsemi Corporate Headquarters
One Enterprise, Aliso Viejo,
CA 92656 USA

Within the USA: +1 (800) 713-4113
Outside the USA: +1 (949) 380-6100
Sales: +1 (949) 380-6136
Fax: +1 (949) 215-4996

E-mail: sales.support@microsemi.com

About Microsemi

Microsemi Corporation (Nasdaq: MSCC) offers a comprehensive portfolio of semiconductor and system solutions for communications, defense & security, aerospace and industrial markets. Products include high-performance and radiation-hardened analog mixed-signal integrated circuits, FPGAs, SoCs and ASICs; power management products; timing and synchronization devices and precise time solutions, setting the world's standard for time; voice processing devices; RF solutions; discrete components; Enterprise Storage and Communication solutions, security technologies and scalable anti-tamper products; Ethernet solutions; Power-over-Ethernet ICs and midspans; as well as custom design capabilities and services. Microsemi is headquartered in Aliso Viejo, Calif. and has approximately 4,800 employees globally. Learn more at **www.microsemi.com**.