NI-845x EEPROM Programmer VI

Author: Aldrin A. Albano (labviewcoder@gmail.com)

Release 2.0 (10-8-2020)

Updated (05-24-2025)

# Summary

The NI-845x EEPROM Programmer is a Windows 7, 10 & 11 LabVIEW application. It is intended to be used by engineers and technical users knowledgeable in EEPROM binary data programming and structure.

The following functions were supported in version 1.0 to 1.5:

* ERASE
* ERASE ALL
* WRITE
* WRITE ALL
* READ
* READ ALL
* DEBUG

And these functions are added in revision 2.0:

* EEPROM content dump to binary (BIN) file
* Binary file upload to EEPROM
* Security hash (checksum) of BIN file with MD5 cypher algorithm

Each of these modes are described in their own sub-sections in [Modes of Operation](#_Modes_of_Operation).

It is recommended to have a wide bandwidth oscilloscope for initial troubleshooting of design. The appendix also discusses how to use a 4-channel oscilloscope with enough bandwidth to capture the timing of the SPI CLK, CS, DO and DI signals to the UUT.

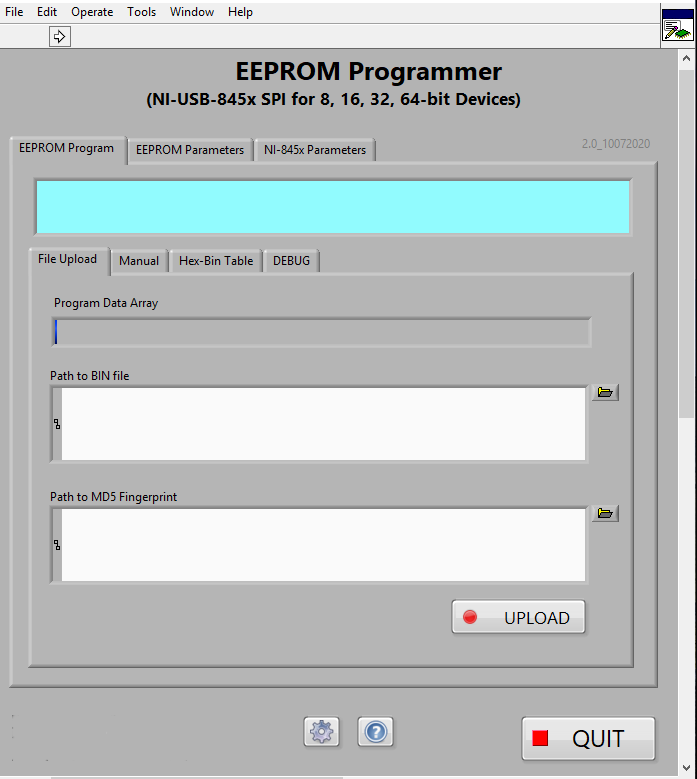


Figure 1 EEPROM Programmer Front Panel (revision 2.0)

There are three main sections of the EEPROM programmer GUI:

* NI-845x Parameters
* EEPROM Parameters
* EEPROM Program.

It is recommended to configure the NI-845x device programmer for the EEPROM slave device parameters on initial power-up. Once the parameters are verified, the INI configuration file MUST be updated.

## Application Build Requirement

The NI-845x EEPROM Programmer is a LabVIEW application compiled to a Windows executable (EXE). The application is only tested on Windows 7 but should run in Windows 10 & 11 as per National Instruments NI-845x driver DLL supports.

At this time, there is no National Instruments NI-845x driver DLL for the LabVIEW Community Edition for Linux.

To build the source (VI) a minimum LabVIEW Full Development 2015 must be installed. This can be installed from the NGC Software Store or manually from the network repository by a credentialed user (a minimum QQ required and an administrator privileged to the local PC)

These following are also required to open and compile/build the source VI:

1. **JKI’s VI Package Manager (VIPM)**

JKI’s VI Package Manager (VIPM) should be downloaded from JKI.net and install the latest version available.



Figure 2 VI Package Manager from JKI.net

Once VIPM is installed, run and use VIPM to install these libraries:

1. **JKI State Machine**
2. **Open G Toolkit**
3. **UI Tools**
4. **MGI Tools & Library**

Once these build toolsets are properly installed, restart the PC. Re-open LabVIEW development application and open the source code (.VI)

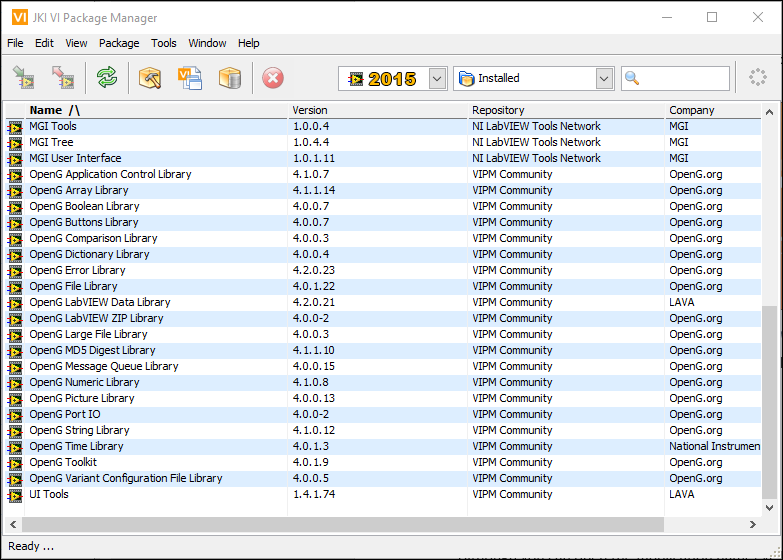


Figure 3 LabVIEW Libraries available from JKI VI Package Manager (VIPM)

There are excellent open-sourced libraries available for LabVIEW from the VI Package Manager (VIPM). These libraries are similar to C, C++ or Python standard libraries available freely and open-licensed. It is often recommended to use these standard libraries to prevent “reinventing the wheel” per se. Standard libraries enhance productivity, more secure and minimizes design flaws & bugs.

**Note 1:** Although one can open the ***[application name].vi***, I recommend opening the ***[application name].proj*** instead as this is the only way to both open & edit the source code and at the same session, build and compile an executable (EXE)

**Note 2:** Regular users/operators are advised to use the compiled/built “EXE” application to prevent changes to source code.

# Usage

## GPIB Instrument Resource Alias

Also referred to as instrument handle or nickname, this is used to programmatically refer to a GPIB, USB or RS-232 serial interface test equipment. Here we use the alias “NI-8452” from the Measurement and Automation Explorer (NI MAX).

When initially setting up a new bench for the EEPROM programmer, this alias “NI-8452” will not be there. You will have to create this using NI MAX (see below). You can optionally use the raw “USB:xxxx” instrument ID under the “NI845x Device ID” control.

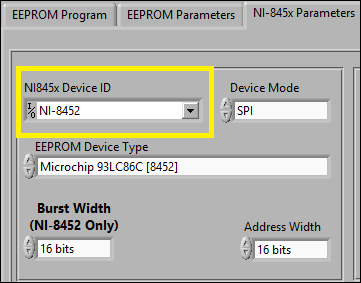


Figure 4 GPIB Instrument Resource Alias

The main reasons for using an alias is to simplify the naming convention of test equipment. This is done through the National Instrument Measurement and Automation Explorer (NI MAX)

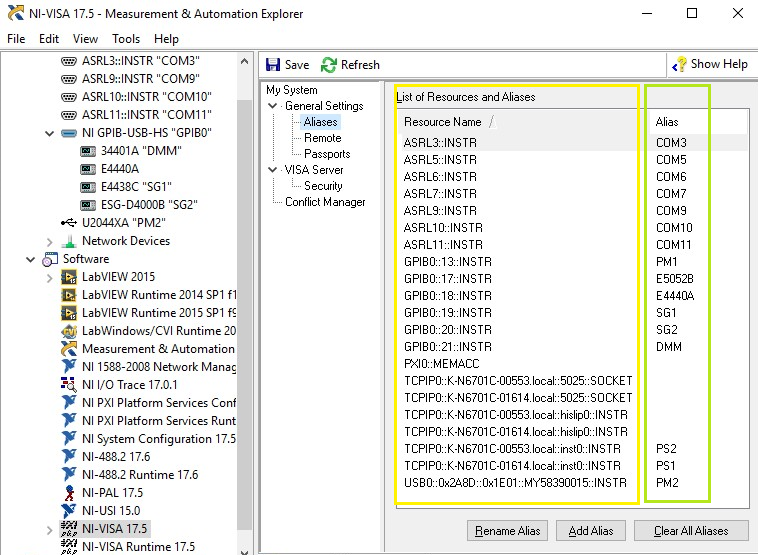


Figure 5 NI MAX Equipment Alias Configuration

Note: Aliases will vary from bench to bench. Click on the drop-down button to refresh the current test equipment configuration for your bench. Select the appropriate model from the list. Make sure to re-save the test [configuration INI](#_Saving_Configuration_INI_1) whenever changing default settings.

In the above list, the left column are the raw equipment VISA resource addresses. These can be long alpha-numeric string up to 32 characters long

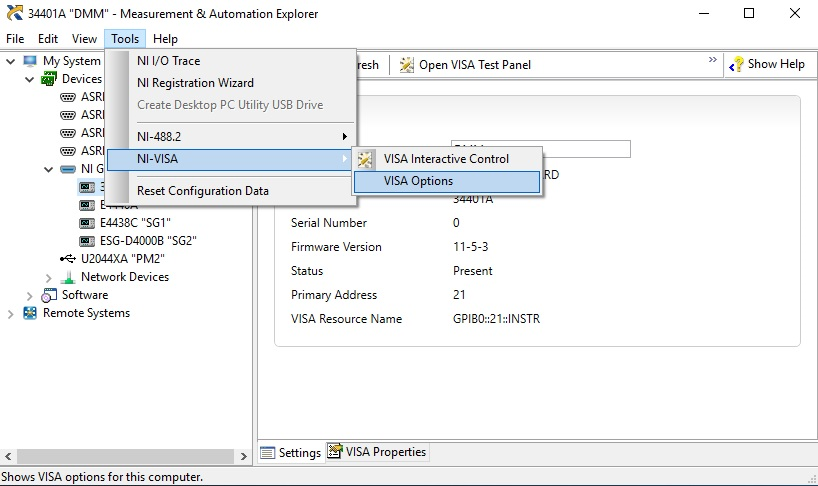


Figure 6 NI MAX Modifying Equipment Alias

If you have good reasons to modify an existing test equipment alias, select **[Tools > NI-VISA > VISA Options]** [[1]](#footnote-2)and find the equipment on the list. Verify multiple (similar model) test equipment such as signal generators, power supplies or DMMs do not have conflicting GPIB addresses.

Each GPIB bus can support up to 31 addresses with up to 15 active at any one time. Address 0 is always reserved to the GPIB bus master (typically the adapter which can be a USB or PCIe based). It is possible to have multiple GPIB bus masters but the first adapter is always addressed “0”, the next is “1”, so forth.

**Note:**

Be aware that changing a test equipment alias to make the bench setup compatible with your LabVIEW test application will certainly break other test applications that have been “hard coded” or custom developed for a very specific bench configuration. These test applications should not be authored this way as the application is not flexible between test bench setups alias configurations.

## NI-845x Programmer & Device Configuration

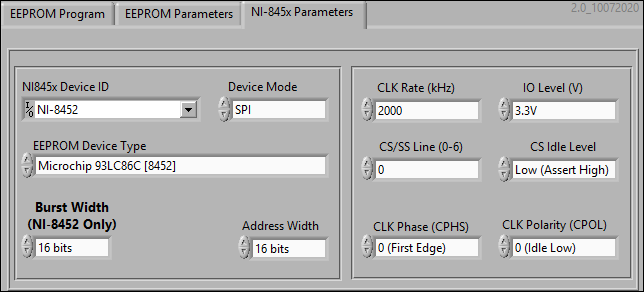


Figure 7 NI-845x Device Configuration

The ST/Microchip 93C86 can be organized into 8- or 16-bits in hardware. In the Test board design the EEPROM “ORG” pin is set to “N/C” or not connected. This makes the chip into a 16-bit ORG mode in the Test board.

**Note 1:** The current revision of this EEPROM programmer GUI only supports SPI (4 wire) or Microwire (3 wire) device protocols. I2C or DIO are not currently supported.

**Note 2:** The NI-8451 is an 8-bit only SPI/I2C programmer. It means it can only send 8-bit burst per operation. The NI-8452 can be configured to send 4, 8, 16, 32 or 64-bit wide bursts. With EEPROM memory sizes greater than 256 memory cells, use NI-8452 device programmer.

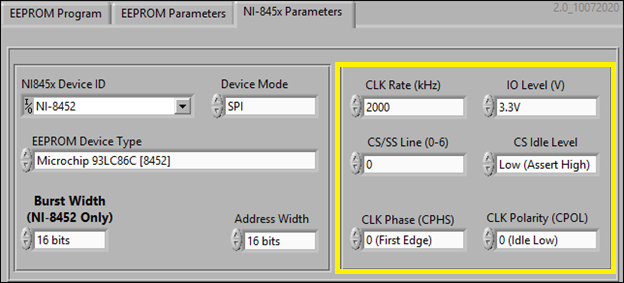


Figure 8 EEPROM Device DC & Clock Requirements

The device programmer must be configured for the EEPROM device signaling and DC levels as per device data sheet or programming specifications.

The ST/Microchip 93C-series EEPROMs require a 2 MHz clock input and Vcc voltage of 5V nominal. The I/O data “HIGH” level a minimum of 2.8V to 3.5V. The NI-845x can supply +2.5V and +3.3V so select the latter (+3.3V) as the former (+2.5V) is too low and will not work for this EEPROM family.

The NI-845x can control multiple SPI devices (using CS0 to CS7). Select CS0 if only programming one EEPROM. If using other CS lines make sure that is configured here as well.

The ST/Microchip 93C-series EEPROMs CS logic idles “LOW” or ~0 VDC and asserted “HIGH” or +3.3V. Note that this is the opposite of most SPI EEPROM devices in the industry (such as Atmel or Cypress) where the CS assert is LOW.

Lastly, clock phase and polarity is where the data synchronizes, value determined and idle level. For most devices, the data is timed with the rising edge of CLK. This is termed CPHS 0, CPOL 0.

***Reminder: once these are set for your device programmer save the current configuration by clicking on the*** [***cog icon to save to INI file***](#_Saving_Configuration_INI)***.***

## OPCODEs

The ST/Microchip 93C-series EEPROMs family share the same opcodes. Other EEPROM families, such as Altera’s, have different opcodes for the same operations so it is recommended to read their data sheet and change these accordingly.

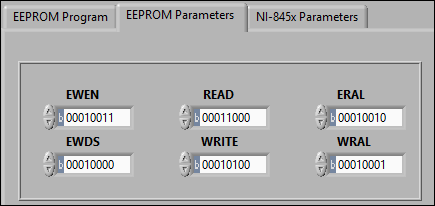


Figure 9ST/Microchip Device Op-Codes

Note: opcodes are usually written in an 8-bit unsigned byte form. The lowest order of the bits (LSb) starts from right-most character (1 or 0) and towards the left (MSb)

***Reminder: once these are set for your device programmer save the current configuration by clicking on the*** [***cog icon to save to INI file***](#_Saving_Configuration_INI)***.***

## Big Endian/Little Endian

Some hardware configurations required the bit order reversed where the LSb is on the left. This is different from byte order endianness where a high endian order means the MSB (byte) is stored in memory first at the lowest storage address.

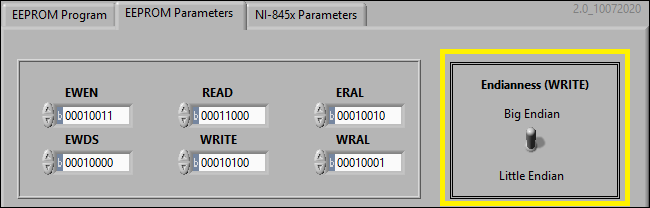


Figure 10 Select Big/Little Endian Programming Mode

The NI-845x device programmers are by default a big endian programmer meaning they will transmit the MSB first to the EEPROM address chosen. If the EEPROM is configured for 16-bits, the MSB (byte) is stored in the lowest order bits of the 16-bit word. Traditionally, the lowest order bits are the LSb bits but can be flipped the other way (store MSb bits there) so please check your design requirement to confirm.

For example, if you are storing the 16-bit hex “0xABCD” the MSB is “AB” and stored at the low-order bits of the 16-bit word. The LSB “CD” would be stored at the high-order bits of the memory address.

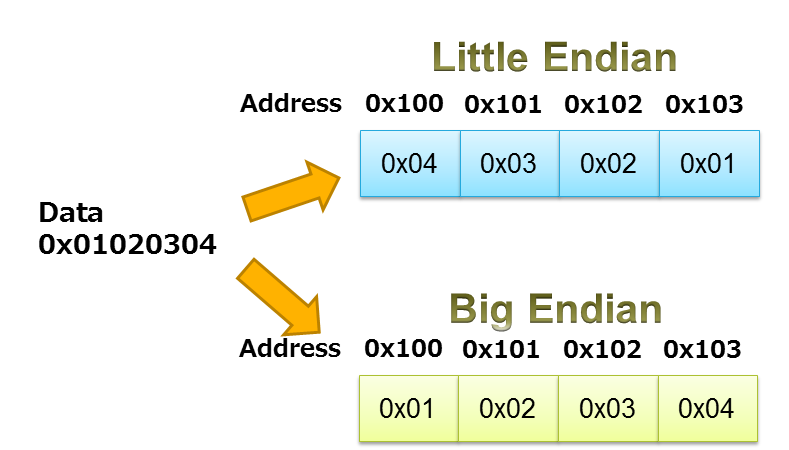
[[2]](#footnote-3)

Figure 11 Data Endianness

Note that the order of the bytes within the data word are ordered one way or another. This is normally is a requirement by the processor not the memory. In this case the processor in the network Phy chip of the Test board requiring a big endian arrangement.

Also note, the bits in each byte are not reversed nor flipped. The byte “0xAB” will be store in that order as “0xAB” not “0xBA” when flipped or “0xD5” when bit order is reversed or mirrored.

Things get confusing very quickly so make sure to check the device requirement before attempting to program from the NI-845x programmers. Keep in mind that these NI-845x programmers will always be big endian (MSB) sent first if addressing a 16-, 32- or 64-bit addresses.

Consequently the READ operation will always return the MSB from the address selected in 16-, 32- or 64-bit sizes. One thing to note, the LabVIEW array used will always package the elements in 8-bit byte per element. So, for a 16-bit data organizations, the NI-845x will return two array elements, a 32-bit words will be returned as 4 array elements, so forth.

## Saving Configuration INI

The NI-845x EEPROM Programmer VIprogrammer GUI is built to automatically load/save the configuration INI from/to:

***C:\Users\[User ID]\Documents\LabVIEW Data\Config Files\<VI Name>.ini***

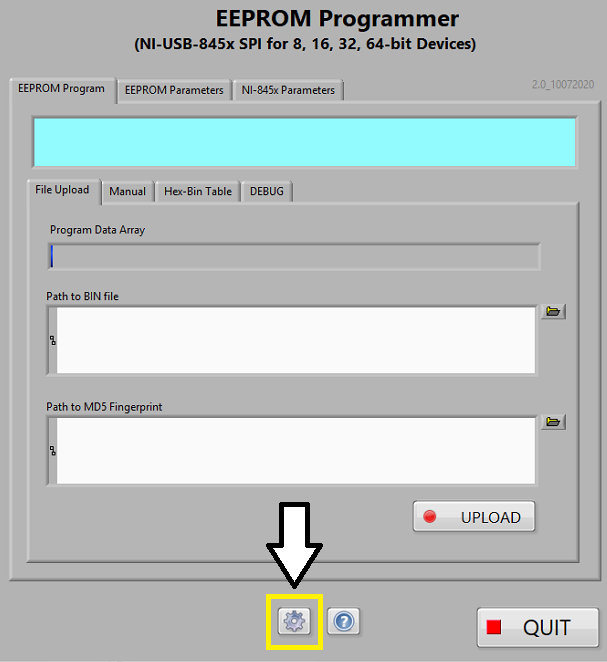


Figure 12 Configuration INI SAVE Button

It is recommended that whenever changing test parameters during use. The configuration INI file is also saved when exiting the EEPROM programmer GUI with the [QUIT] or window close [X] buttons.

## Invert Op-Code

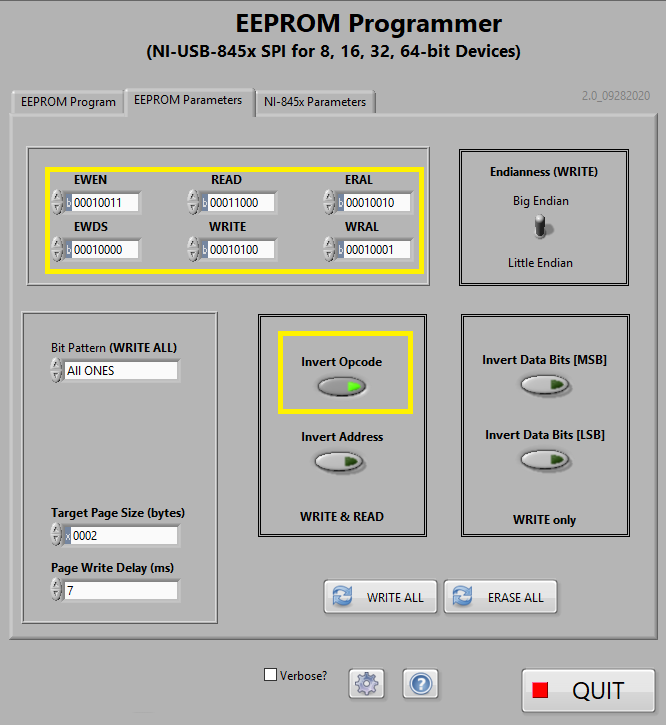


Figure 13 Invert Op-Code Mode

The ST/Microchip 93C86 chip addressing scheme uses the two most-significant bits (bits 6 & 7) of the opcode octet to support the address space of 10-bits.

This requires the opcode to be inverted for proper bitwise manipulation. The 2 most-significant bits (bits 8 & 9) of the address space is then “summed” into the opcode (8-bit) space as the opcode is only 5-bits long. This special routine takes care of packing the opcode space correctly for addresses greater than 255 (0xFF).

**Note: This is not required for EEPROM chips that are 256 (0xFF) memory cells such as the Microchip 93LC56 or EEPROMs that uses full 16-bit address and data bursts.**

Keep in mind the NI-845x sends bursts of 8, 16, 32 or 64 bits selectable but NI LabVIEW driver for the NI-845x have a fixed-width 8-bit array of unsigned 8-bit bytes elements.

## Invert Address and Data Bits

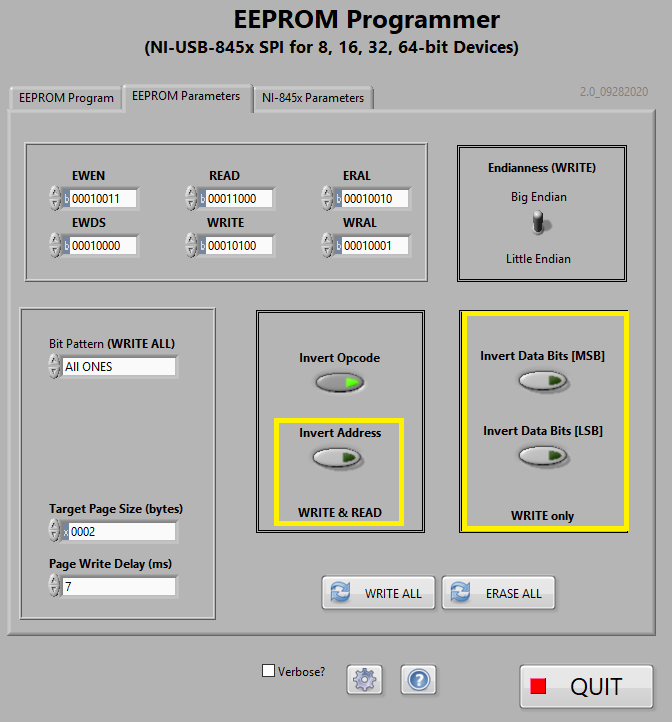


Figure 14 Invert Address and Data Bits

These are, more or less, intended for debugging the EEPROM using an oscilloscope (see Appendix). The ST/Microchip 93C86 or 93LC56 chips do not required these to be set.

## Target Page Size & Write Delay

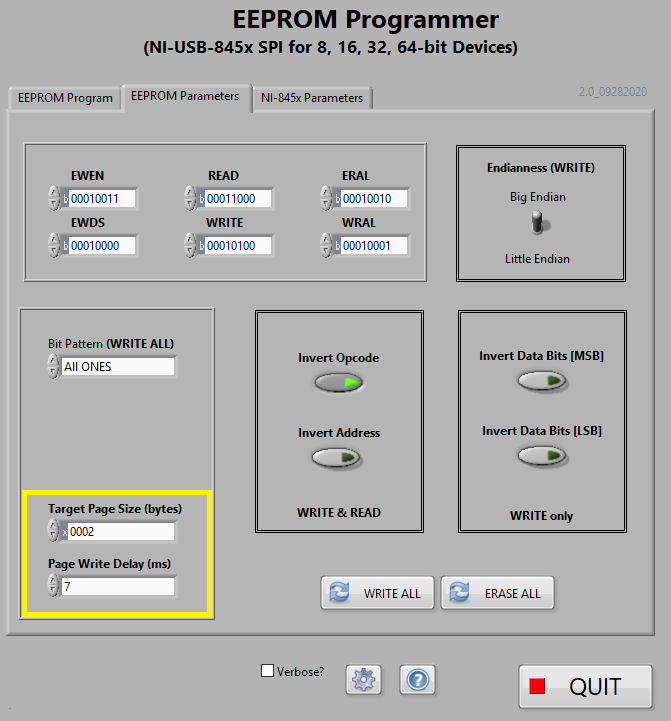


Figure 15 Target Page Size & Write Delay

The ST/Microchip 93C86 or 93LC56 chips does not have a capability to write in certain size blocks as EEPROMS from Atmel, for example. ST/Microchip 93C86 or 93LC56 chips set “Target Page Size” to 2-bytes (0x0002). This is of course, the size of each memory address which is 16-bits (2 bytes).

As for the total duration for transmitting the op-code + address + data and the processing of the instruction transmitted, the minimum is 5ms.

Some instructions take longer to process, such as WRITE-ALL, READ-ALL and ERASE-ALL and therefore it is recommended to leave the “Page Write Delay” to 7 milliseconds (ms).

# Modes of Operation

Operations that can be performed with the NI-845x SPI/I2C device programmer is dependent on the capabilities of the slave device.

For example, we have an ST (Microchip) 93C86 installed. The data sheet details the device capabilities below.

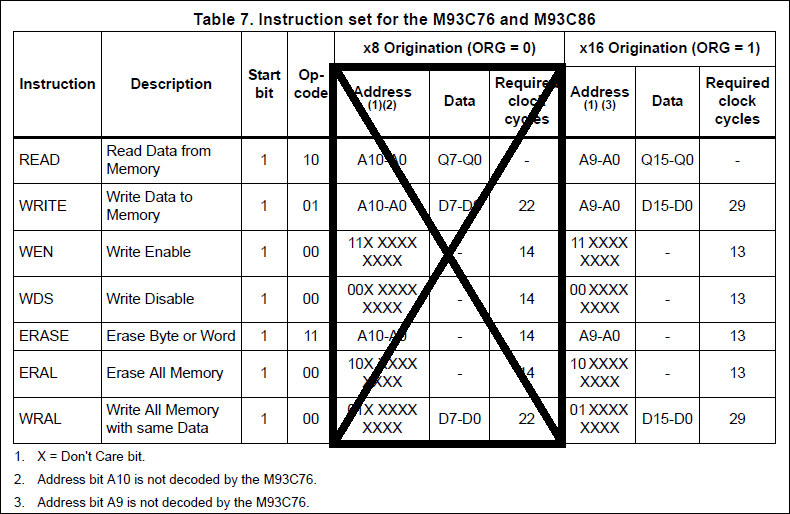


Figure 16 ST (Microchip) 93C86 Instruction Set

In the module, the EEPROM “ORG” pin is floated (not connected). This configures the chip’s memory organization to 16 data bits. Its opcode is 5 bits. Its addressing is 10 bits.

Note 1: As the address is 10 bits, the two MSb must be merged with the 3-bit opcode in the first byte transmitted. These two MSb bits of the address occupies the lowest LSb bits of the first byte transmitted to the slave device.

Note 2: Keep in mind the NI-845x I2C/SPI programmer transmits the MSB first. This is important to understand when reading the EEPROM memory arrays. The READ ALL will return the address space starting from address 0x0000 to maximum. The READ will return one 16-bit word as written.

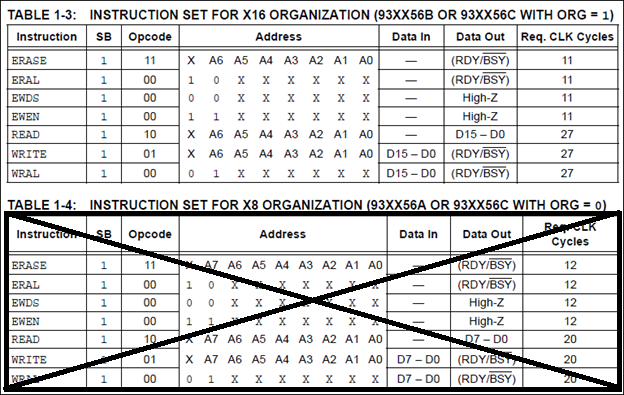


Figure 17 Microchip 93LC56BT Instruction Set

The Microchip 93LC56BT is not used in the Test module. It is only included here for reference as the GUI application was developed with this chip.

The 93LC56BT “ORG” pin was floated and therefore it configured itself as a 16-bit data widths. Opcode is 3 bits. Address is 8-bits.

## ERASE ALL

**Warning: ERASE ALL button does exactly what is says and will erase the EEPROM memory array entirely. There is no reverting to prior state. Use only for new EEPROM or need to reprogram an existing chip.**

From the Microchip 93C86/LC86 data sheet:

*The Erase All (ERAL) instruction will erase the entire memory array to the logical ‘1’ state. The ERAL cycle is identical to the erase cycle, except for the different opcode. The ERAL cycle is completely self-timed.*

*The rising edge of CLK before the last data bit initiates the write cycle. Clocking of the CLK pin is not necessary after the device has entered the ERAL cycle.*

Therefore, use this button carefully.

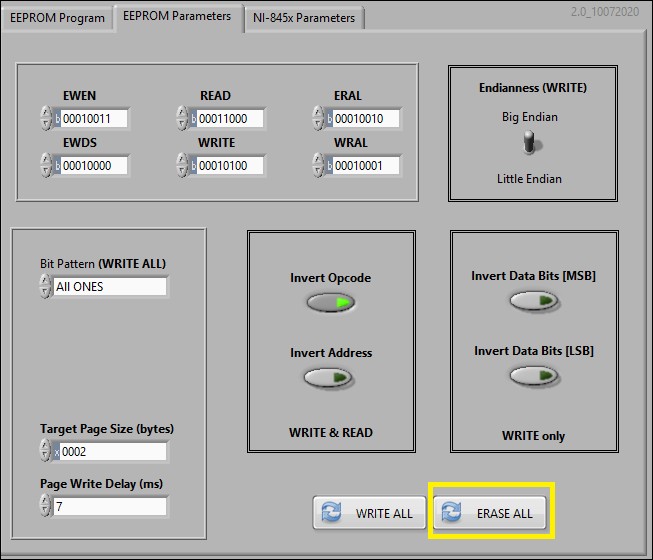


Figure 18 ERASE ALL Operation

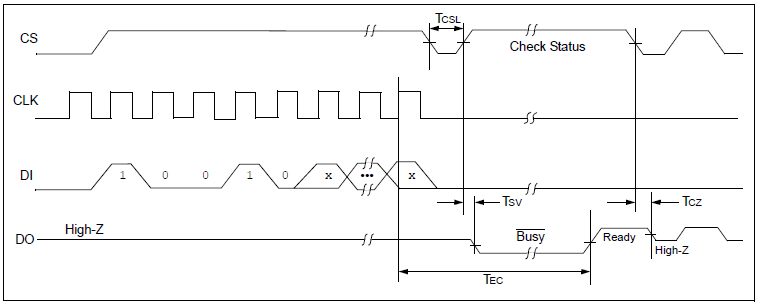


Figure 19 ERASE ALL (ERAL) Timing Sequence

As the timing sequence shows, upon assertion of CS (high) the opcode is transmitted MSb first then the address. The address field is ignored. It can be filled with 1 or 0. In our case we fill the address field with 1s or 0x3FF (d1023), as the Microchip 93C86/LC86 address field is 10 bits wide.

Then we dessert and reassert the CS after the last bit of address is sent to start the self-timed erase operation. Reasserting within 1us is the fastest that the NI-845x can toggle the CS line.

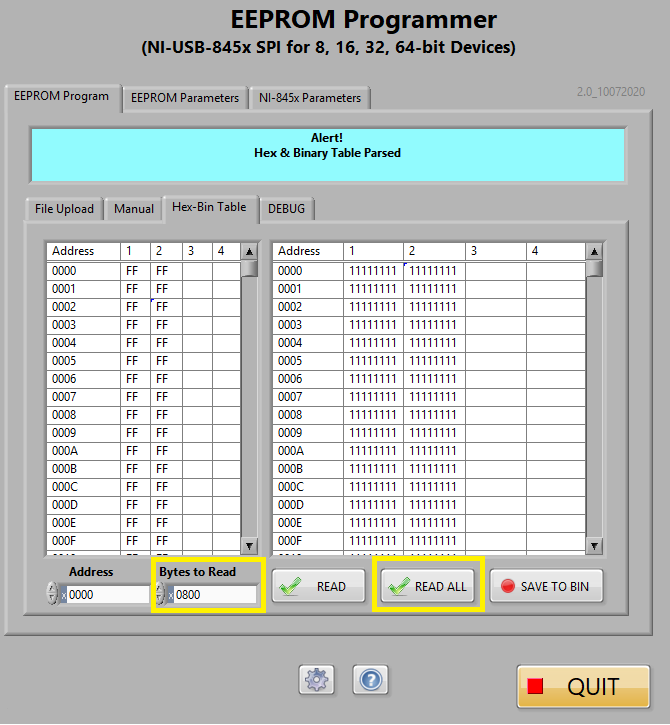


Figure 20 READ ALL post ERASE ALL operation

Press the READ ALL to confirm the ERASE ALL succeeded. The Hex and Binary tables would now show “0xFFFF” and “b11111111 11111111” on all memory address slots.

Setting the “Bytes to Read” to the EEPROM’s maximum specified memory size. In this case, the MC/ST 93C86 EEPROM memory size is 0x400 and are organized in 16-bit data width. This translates to a memory size of 16,384 bits, 2048 bytes or 1024 words.

The hex and binary tables here are organized in 8-bit bytes per column. Each memory address contains 16-bits or 2-bytes in big-endian order. [Big endian ordering means the MSB is stored at the lowest memory location.](#_Big_Endian%252FLittle_Endian)

## WRITE

We can use WRITE mode to write to one or several memory addresses. Here, we need to type a 16-bit hexadecimal data to write to the 16-bit hexadecimal address. The 16-bit data word is split into two 8-bit bytes into the array arranged in the “endianness” selected.

**Note:** The NI-845x I2C/SPI programmer driver only allows unsigned 8-bit values for each element in the array. The driver will use these 8-bit elements to form the transmitted bursts to the slave device according to the selected #-bits burst width.

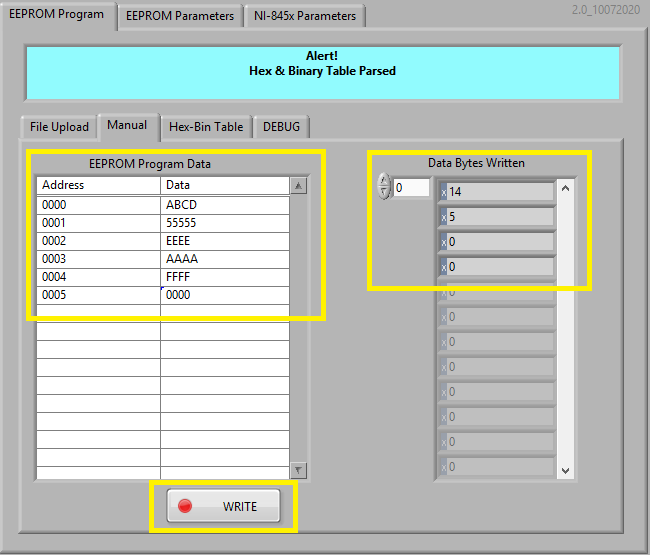


Figure 21 WRITE operation (Big Endian)

In this example, memory address 0x0000 was written with 0xABCD in “big endian” mode. In [big-endian mode](#_Big_Endian%252FLittle_Endian), memory address 0x0000 would be written with the MSB (byte) at the lowest order bits of the 16-bit word.

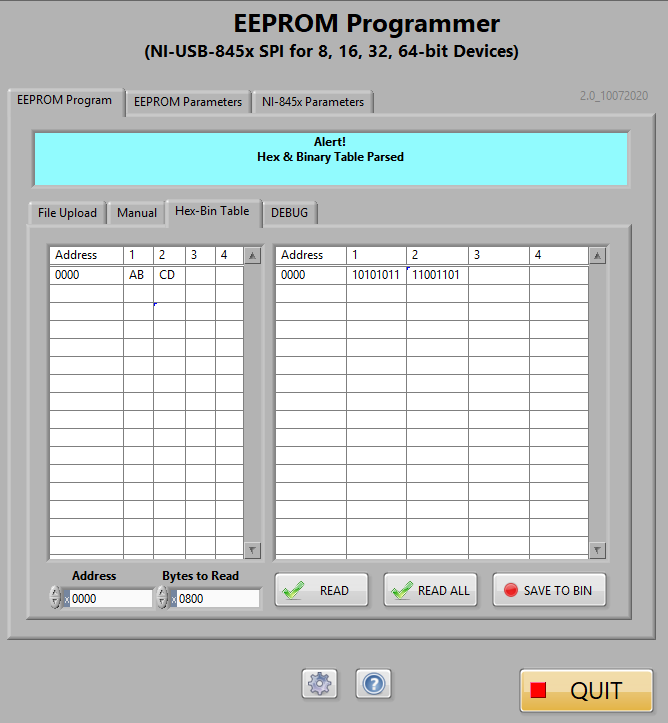


Figure 22 READ operation post WRITE operation.

Reading memory address 0x0000, the driver returns two array elements. The NI-845x will always return the MSB (byte) to the first array element. The memory array is not altered. The array elements are displayed in the order of the endianness of the previous WRITE.

When the WRITE mode was big-endian, we can expect the MSB (byte) written first and at the lowest bit order of the 16-bit data word. Correspondingly, the NI-845x also return the MSB (byte) of the data word first.

## WRITE ALL

The WRITE ALL mode is a debugging mode. You can program pre-determined test patterns such as:

* All Ones
* All Zeros
* 0x1010 pattern
* 0x5555 pattern
* 0xABCD pattern
* 0xEEEE pattern

These are used mainly to determine there no dead or stuck bits and to visually observe the order of the endianness of the programmed data.



Figure 23 WRITE ALL ONES to EEPROM

### From the Microchip 93C86/LC86 data sheet:

*The Write All (WRAL) instruction will write the entire memory array with the data specified in the command. The self-timed auto-erase and programming cycle is initiated by the rising edge of CLK on the last data bit. Clocking of the CLK pin is not necessary after the device has entered the WRAL cycle. The WRAL command does include an automatic ERAL cycle for the device. Therefore, the WRAL instruction does not require an ERAL instruction, but the chip must be in the EWEN status.*

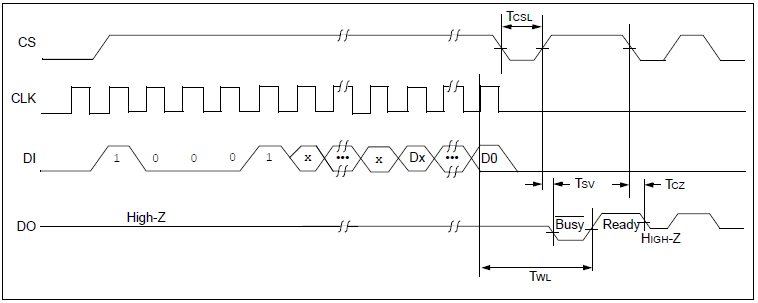


Figure 24 WRITE AL (WRAL) Timing Sequence

The WRITE ALL address field is a dummy field. The address can be 1s or 0s but we chose to use 0x0000 as with the ERASE ALL function. The data field must contain the pattern to use in each memory address.

Keep in mind that the ST/Microchip 93C86/LC86 is organized for 16-bit data widths (D0 – D15) and MSb is transmitted first by the NI-845x programmer.

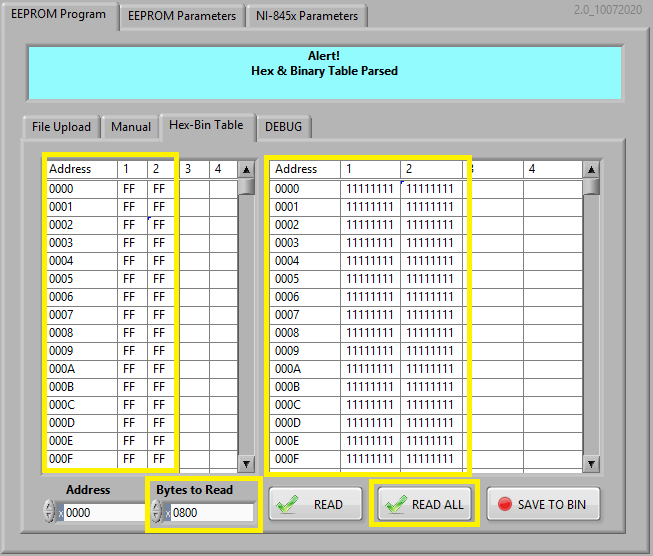


Figure 25 READ ALL post WRITE ALL zeros

Finally, you can use the READ ALL function with the Hex and Binary table display to quickly browse the entire content of the device memory array.

Alternatively, you can use the READ function to read one memory address at a time.

## READ

READ mode is a read operation of **only one memory address**.

The ST/ Microchip 93C86/LC86 used in the ICING module is organized as a 16-bit per memory address (D0 – D15) by not connecting the EEPROM’s ORG pin to Vcc. There is an internal 10k Ohm resistor that does the function of an external pull-up resistor. Likewise, the PE (Program Enable) pin is also not connected and pulled up to Vcc in same manner as the ORG pin.

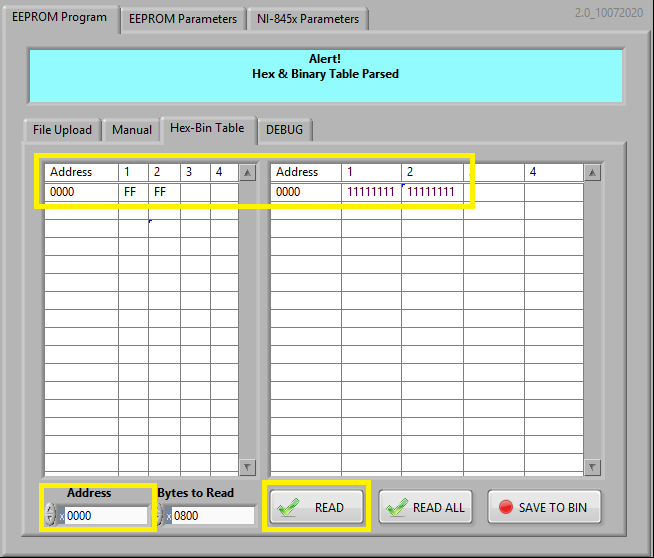


Figure 26READ One Memory Array of 16-bits

The opcode is b110 or x06 is logically left-shifted to become 0x18 as indicated on the OPCODE fields on the front panel. This is a requirement for the ST/Microchip 93C/LC series EEPROMs as they have 10-bit address field and a 5-bit opcode.

Note: data endianness is for WRITE operation only. READ will parse each memory array as written and the NI-845x devices will transmit the MSB from each 16-bit data word of every address.

### From the Microchip 93C86/LC86 data sheet:

*The READ instruction outputs the serial data of the addressed memory location on the DO pin. A* ***dummy zero bit precedes*** *the 8-bit (If ORG pin is low or A-Version devices) or 16-bit (If ORG pin is high or B-version devices) output string.*

***The output data bits will toggle on the rising edge of the CLK*** *and are stable after the specified time delay (TPD).* ***Sequential read is possible when CS is held high****. The memory data will automatically cycle to the next register and output sequentially.*

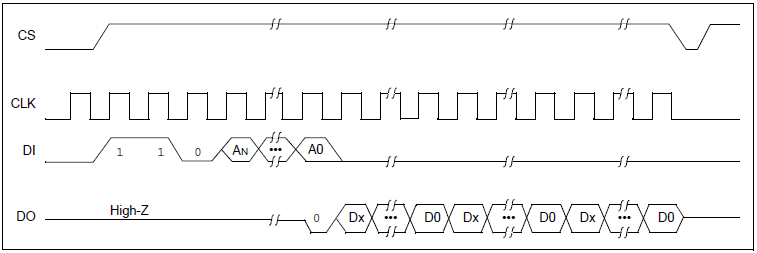


Figure 27 READ Timing Sequence

The READ can read one memory address space or sequence of addresses. In fact, we use this feature to add a READ ALL mode. All READ ALL requires is keep the CS asserted until ALL memory locations are read.

This requires the DO be monitored (or estimated from oscilloscope measurement) as the DO is set BUSY (high) during the READ operation or READY (low). For 93C86/93LC86 with 16kb addressable memory, the full memory is read in ~6ms.

## READ ALL

There is no separate opcode for READ ALL. It is just a slight modification of the READ instruction by extending the length of the CS assert until all data burst from DI (or MOSI) are sent to the device UUT.

Refer to the READ mode for most of READ functionalities. As to where the READ ALL differs is described below.

When the CS is kept asserted and there are additional addresses + dummy data (0xFF) to be clocked in the following happens. The EEPROM will enter into a self-timed mode and parses its DI (MOSI) and as long as there is address + dummy data (0xFF) on the wire and the CS is still asserted, the UUT will return the content of the memory addressed in 8-bit (NI-8451) or 16-bit (NI-8452) bursts.

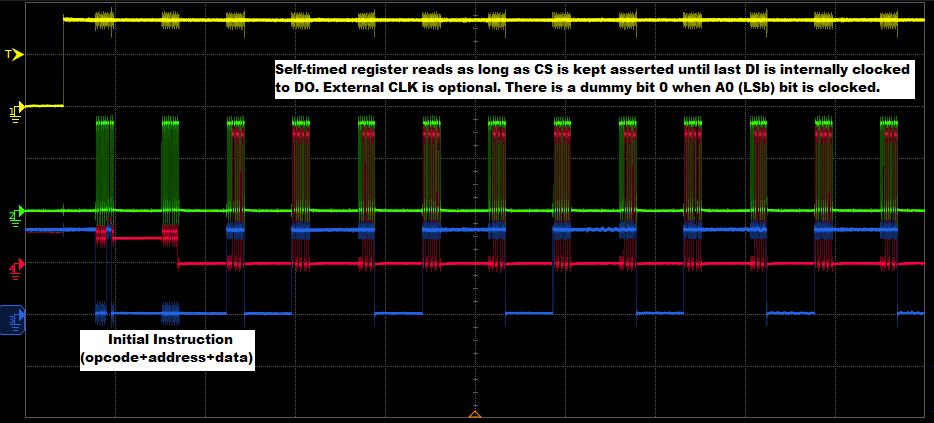


Figure 28 READ ALL Address + Data Bursts (NI-8451 8-bit bursts)

As with the READ, there is a dummy bit 0 returned when the last bit of the address (A0 or LSb) is clocked in. The READ parser automatically removes this initial dummy bit 0 by shifting all received bits by 1 to the left (LSL or logical shift left) and reforming the data arrays before displaying the data in the Hex and Binary tables.

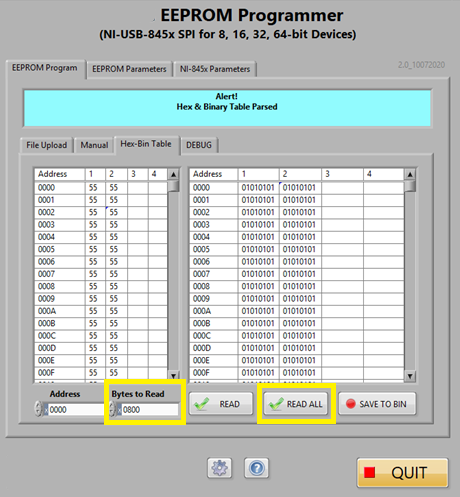


Figure 29 READ ALL operation post a WRITE ALL “0x5555”

READ ALL reads all memory content from memory address 0x000 up to the set “Bytes to Read” value. Set this value to the maximum memory address space of the EEPROM slave device.

## DEBUG

The Debug panel is useful when data from UUT is not making sense. It should be used in conjunction with an oscilloscope to debug UUT issues as out-of-ordered bits, incorrect addresses, etc.

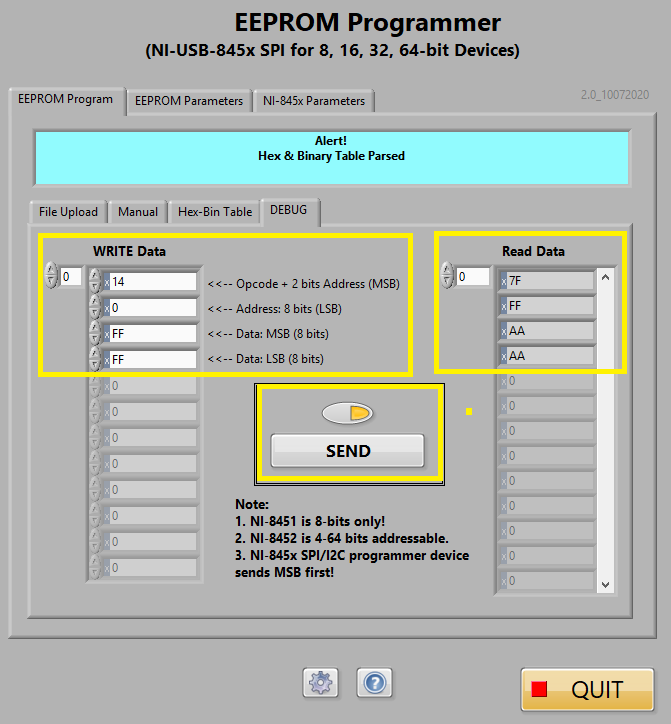


Figure 30 Debug Panel

Most EEPROMs follow this convention: opcode + address + data sent in that order.

For addresses and data widths greater than 8-bits (0xFF or d255) the MSB is typically sent first. Refer to your UUT data sheet for details.

Here, the Microchip 93C or 93LC series EEPROMs follow this convention. CS is asserted high for each instruction set. In this particular example, an NI-8451 is only capable of 8-bit bursts so can only address up to 0xFF or d255 memory addresses. Data from each memory address are 16-bit wide but are broken up into two 8-bit bursts, again MSB (byte) is transmitted first.

When debugging the EEPROM it is recommended that a 4-channel oscilloscope is available to verify the signals are properly formatted to and from the chip. See Appendix for oscilloscope screen captures.

**Note:** You must enable the Debug mode by clicking on the enable button. Use the [SEND] button to send the WRITE array elements to the chip in order of entry.

The manually inputted array elements that comprises the “opcode + address + data” will override the manual data entry table. Therefore, the Debug mode should be used to send one instruction set at a time.

It is possible to script multiple instructions into the “Write” array manually but not recommended.

If you need to send multiple “WRITE” instructions it is easier to use the “Manual” write mode.

Last but not least, observe correct data endianness. The array elements will be transmitted in order. You have to plan byte endianness when using the Debug mode.

# Programming the Test module EEPROMs (Manual)

Programming the EEPROMs is accomplished by knowing what address and data you want to write. Both values are unsigned 16-bit word formats (e.g. 0xNNNN) in hexadecimal notation (see figures below)

Before you program the EEPROM, verify that you can read and erase fist. Click on ERASE ALL to erase the entire content of the chip. This will set the memory array bits to “1”. An alternative, is to WRITE ALL “All Ones” or “All Zeros” depending on system requirement.

Then click on READ ALL to read the entire memory array and display the contents in the Hex and Binary tables. The memory array is displayed in either Big Endian or Little Endian as selected on the front panel options.

Note that READ or READ ALL does not alter the memory array order. It is merely displaying the memory as read and in the order of endianness as it was programmed with WRITE function.

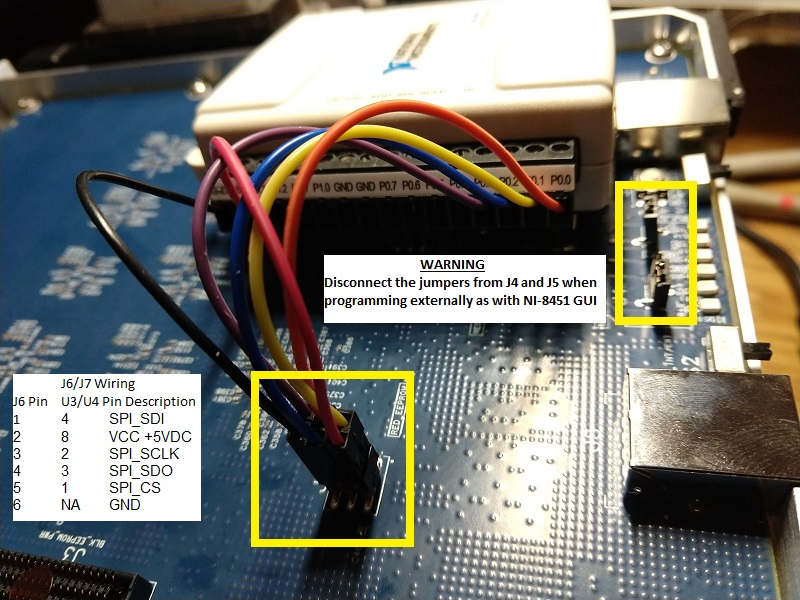


Figure 31 NI-845x Programming Harness

Note: unplug the Red and EEPROM1external power jumpers shown above to prevent +5V leaking into the Phy chip during programming. Observe the correct polarity of the programming harness to the EEPROM headers.

Note: NI-8451 shown above. NI-8452 wiring is to a 40-pin IDE header. Consult the device programmer user manual.

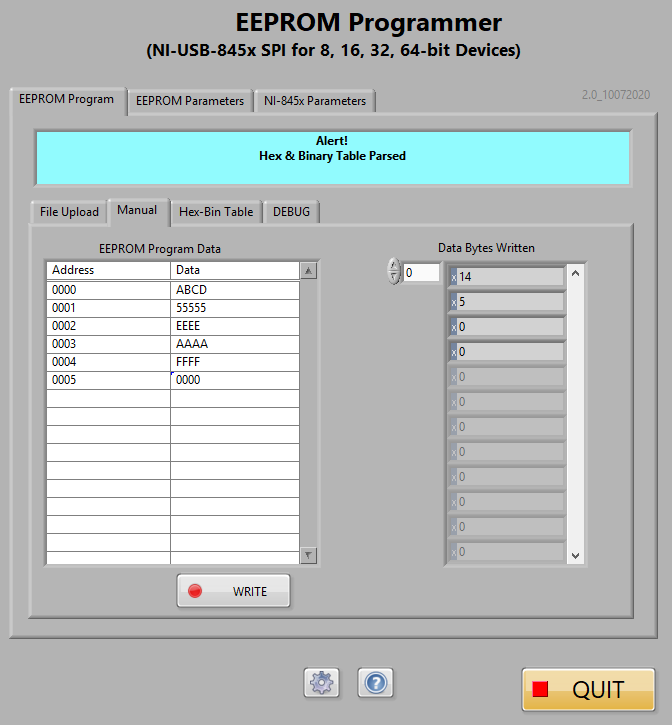


Figure 32 EEPROM Programming Data for WRITE

It is recommended to use the “Manual” EEPROM program mode as it is easier to simply type the hexadecimal addresses and data values in 16 bit format (above) rather than using the “DEBUG” mode to type individual 8-bit array elements.

This is the recommended way to program the EEPROM prior to generating a binary (BIN) and checksum hash (MD5). When the EEPROM has been successfully provisioned then a BIN and MD5 can be easily created.

**Note:** If you already have a working BIN & MD5 files skip manual programming and use the binary file upload to easily provision the EEPROM. The MD5 security has will validate the BIN file has not been tampered, corrupted or invalid (wrong file).

# Programming the Test module EEPROMs (Automated)

The EEPROM Programmer GUI revision 2.0 implements a binary file upload to the EEPROM. The binary (BIN) file is generated Hex-Bin Table sub-panel using **[READ-ALL]** and **[SAVE TO BIN]** functions.

The BIN and MD5 (a security hash of the binary file) is saved at:

***C:\Users\<User ID>\Documents\LabVIEW Data***

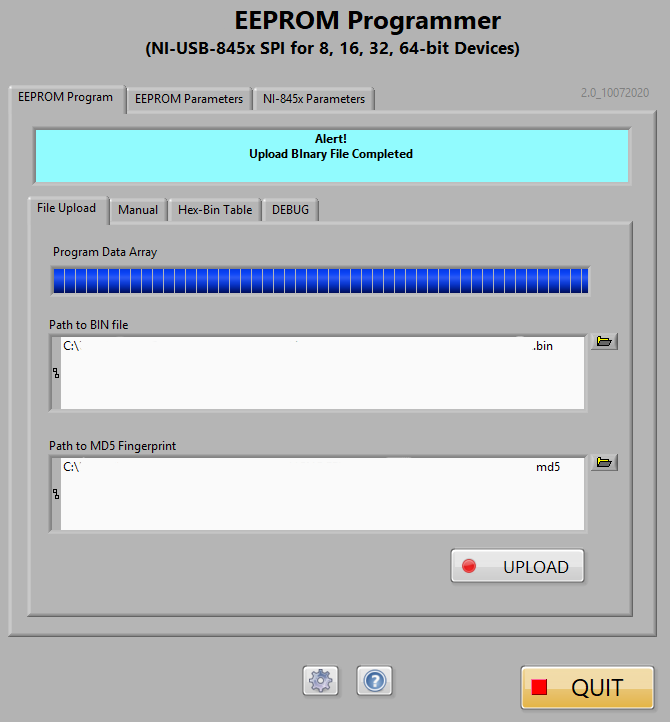


Figure 33 EEPROM Programming using binary (BIN/MD5) file Upload

## TestEEPROM

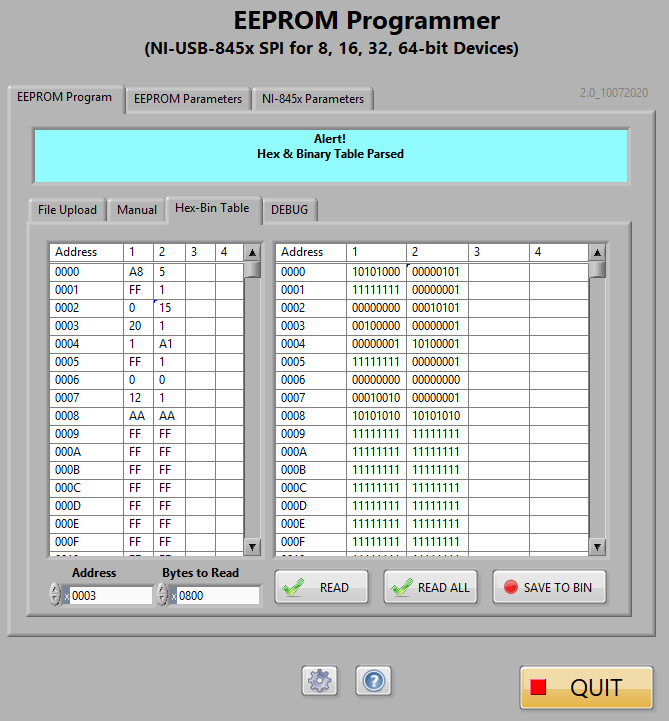


Figure 34 TestEEPROM Memory Array READ ALL from EEPROM

After a successful WRITE of the data, use READ ALL to parse the entire content of the memory array and displayed in the Hex and Binary table as shown above.

## Red EEPROM

The RED EEPROM is programmed in same manner as the TestEEPROM.

Power off the EEPROMs by unplugging the NI-8452 from the USB port. Move the programming harness over to the header pins for the Red EEPROM.

Use “File Upload” sub-panel to select the RED EEPROM binary file and security hash (MD5) file.

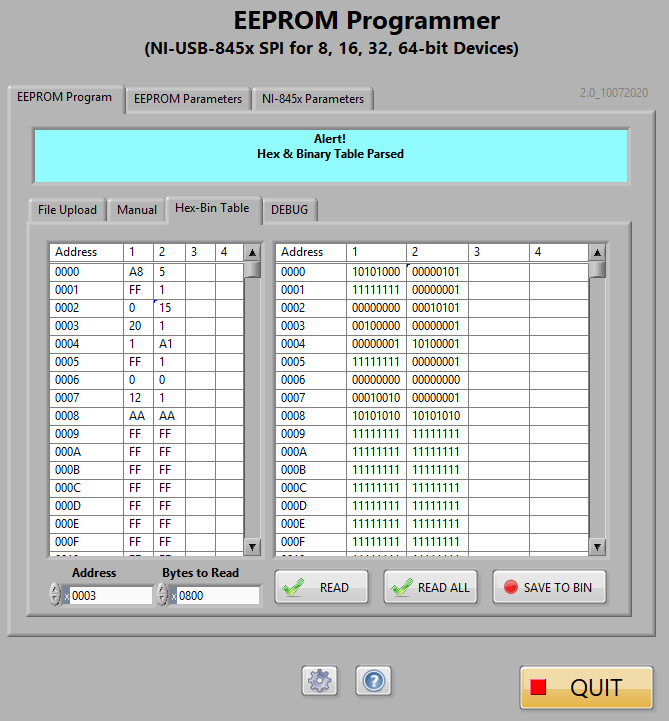


Figure 35 RED EEPROM Memory Array READ ALL from EEPROM

After a successful WRITE of the data, use READ ALL to parse the entire content of the memory array and displayed in the Hex and Binary table as shown above.

# Appendix

## Oscilloscope Measurement with NI-8451 SPI Device Programmer

For the 93LC56BT we have to use the NI-8451 and set to 8-bit bursts to accommodate the 8-bit op-code, 8-bit address space and 8-bit wide data lengths.

## READ

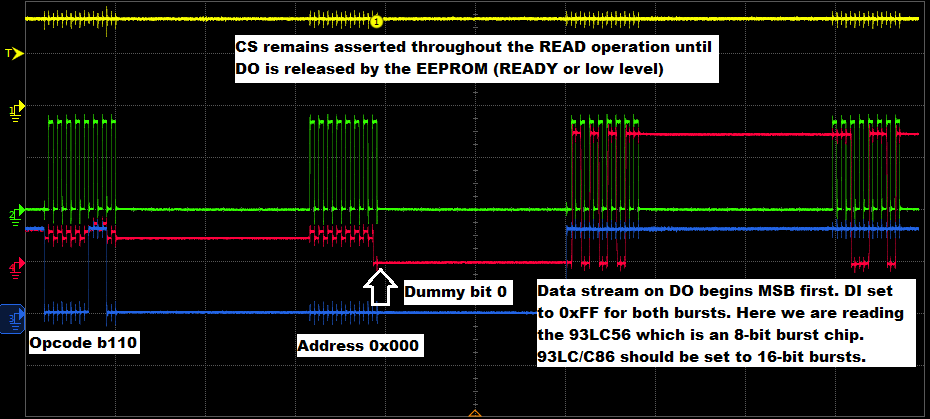


Figure 36 READ Operation in Detail (Oscilloscope)

The READ operation requires four 8-bit bursts with the NI-8451 to read the 93LC56BT here.

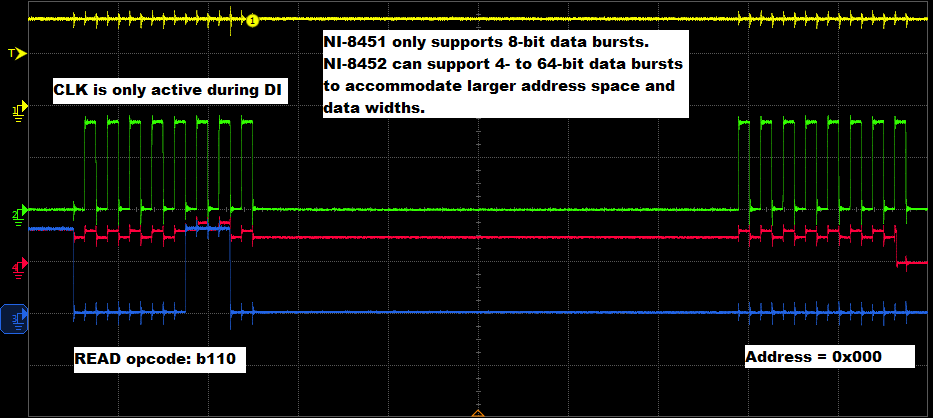


Figure 37 READ Opcode and Address in Detail (Oscilloscope)

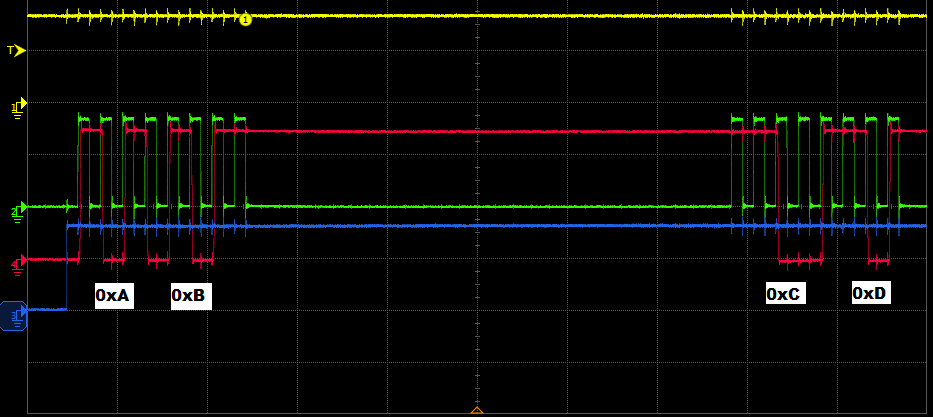


Figure 38 READ Data Endianness in Detail (Oscilloscope)

**Note: The endianness of the READ is determined by the WRITE operation.**

Here we read memory address 0x0000 that was previously written with 0xABCD in “Big Endian” mode meaning the MSB (0xAB) was written to the lowest address space. The LSB (0xCD) was written at the next address space.

The NI-8451 will always transmit the MSB from a memory address. Therefore, “0xAB” is transmitted first as that is the data in the lower address space.

If the WRITE operation used “Little Endian” mode, the MSB (0xCD) would be written at the lowest address space and, consequently, the READ will transmit “0xCD” first from the lowest address space.

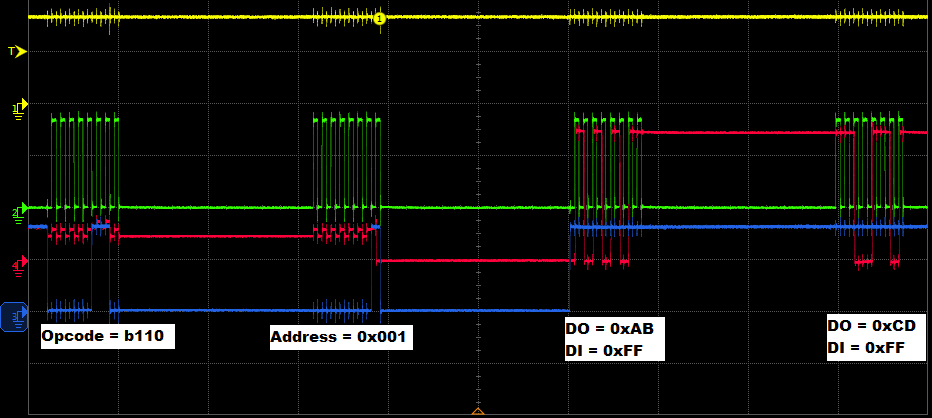


Figure 39 READ Data Address 0x001

Reading the next address space shows the address field burst is increased by one at LSb end of burst. The MSb is always transmitted first by the NI-8452 device programmer.

### READ ALL

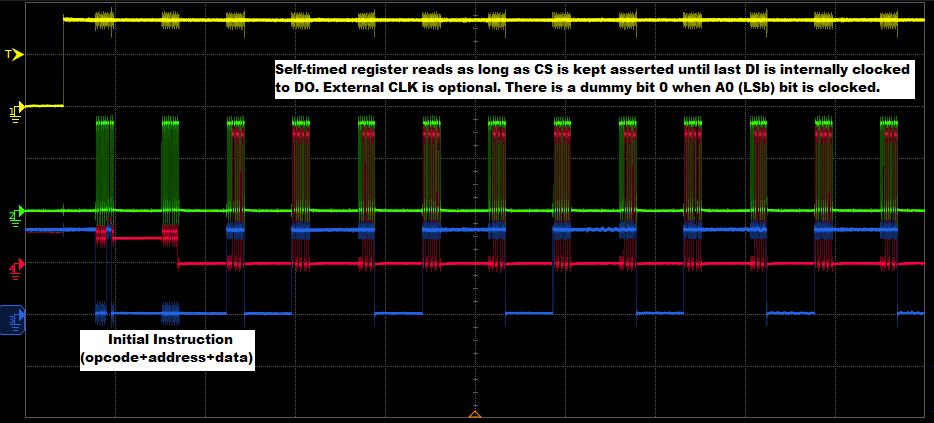


Figure 40 READ ALL Address + Data Bursts (NI-8451 8-bit bursts)

There is no separate opcode for READ ALL. It is just a slight modification of the READ instruction by extending the length of the CS assert until all data burst from DI (or MOSI) are sent to the device UUT.

[Refer to the READ-ALL operation description.](#_READ_ALL)

## Oscilloscope Measurement with NI-8452 SPI Device Programmer

For the 93C86/93LC86 we have to use the NI-8452 and set to 16-bit bursts to accommodate the 10-bit address space and 16-bit wide data lengths.

### EWEN

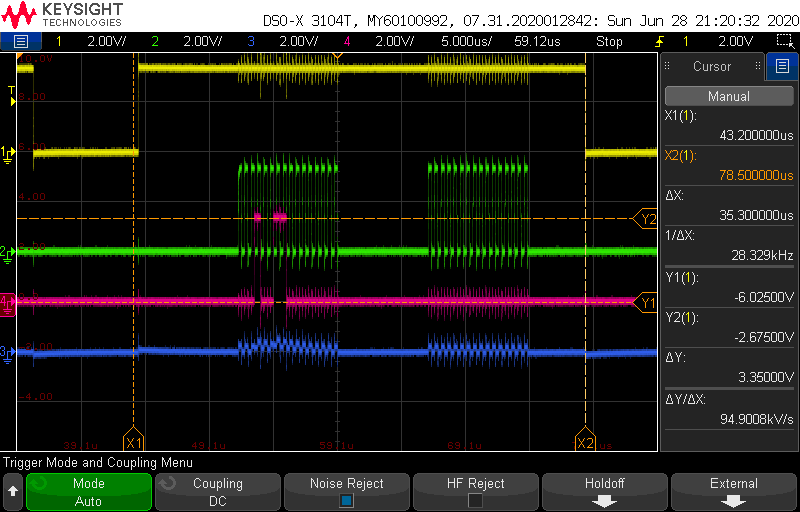


Figure 41 EWEN (Erase and Write Enable) Opcode + Address + Data

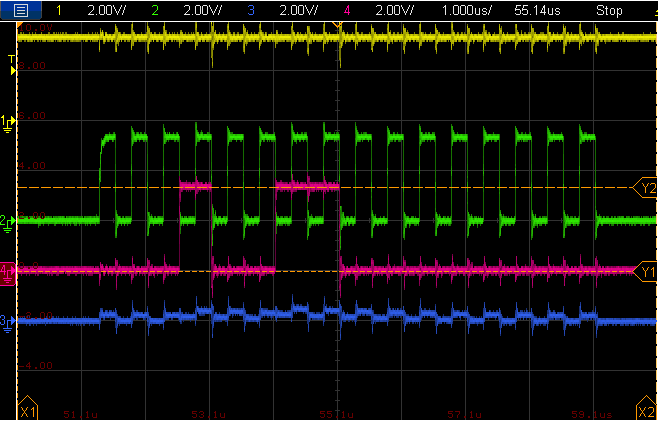


Figure 42 EWEN Opcode

### READ

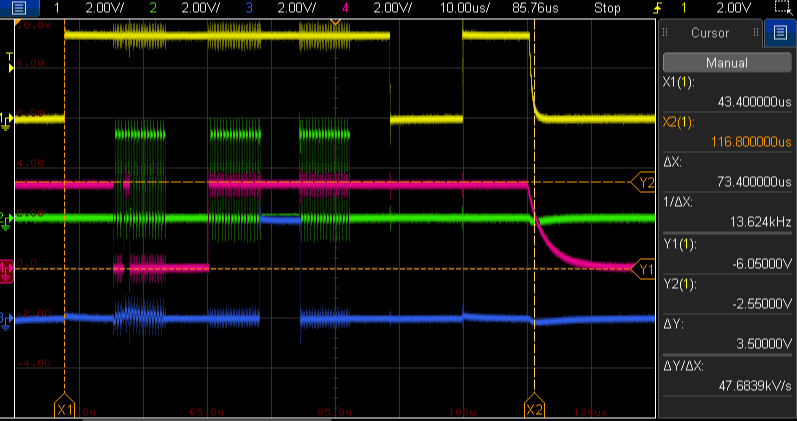


Figure 43 READ address 0x0001 Opcode + Address + Data (Full Span)

We can see that it takes ~73us to complete one READ operation as indicated by the DO going back to idle (low) after the 2nd CS de-assertion.

Zooming closer to inspect the opcode + address 16-bit burst shows the READ opcode “00011000” followed by 10-bit address field of all “0s” which points to address 0x0000.

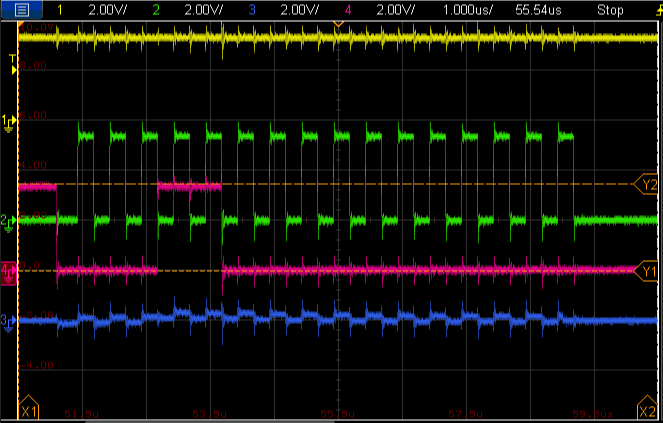


Figure 44 READ\_0x0001 Opcode + Address (NI-8452 16-bit burst)

Meanwhile, the data field (also 16-bits) are set to all “1s” of 0xFFFF. This is optional and can be set to any value. Here we used all “1s”.

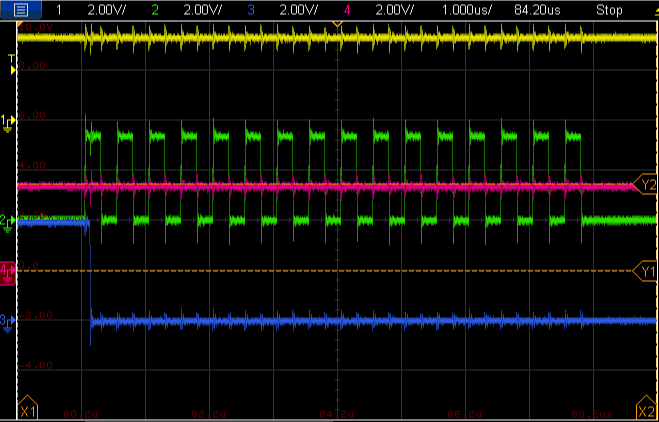


Figure 45 READ\_0x0001 Data (NI-8452 16-bit burst)

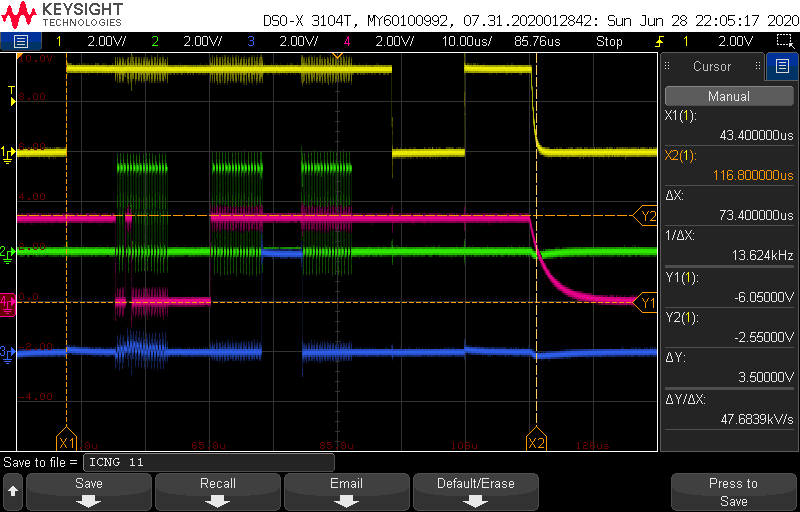


Figure 46 READ 0x0001 Opcode + Address + Data

The 2nd 16-bit burst is the data word. The 3rd 16-bit burst is needed to properly receive enough data from the NI-845x array elements to parse into the hex and binary tables.

The most significant bit (MSb) is transmitted first from the data burst for both WRITE and READ operations. This is the default mode for the NI-845x SPI & I2C programming devices.

The slight delay out of the device (SDO) is just response latency.

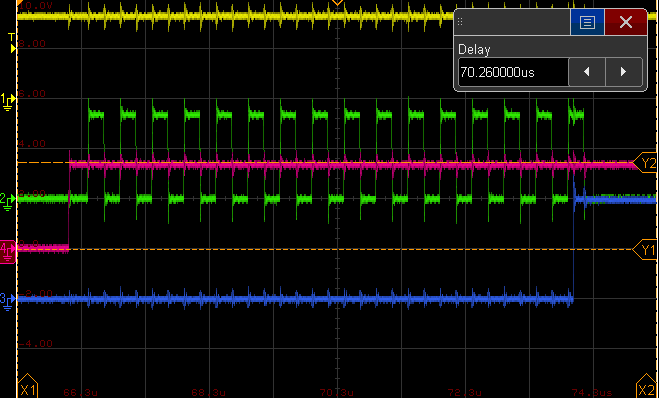


Figure 47 READ 16-bit Data Word (0x0001 written to address 0x0000)

DO (data out) from the EEPROM is outputted after the last DI and CLK rising edge (slight delay)

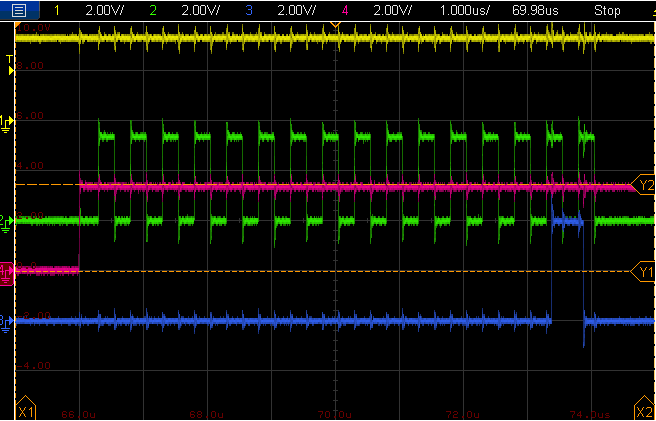


Figure 48 READ 16-bit Data Word (0x0002 written to address 0x0000)

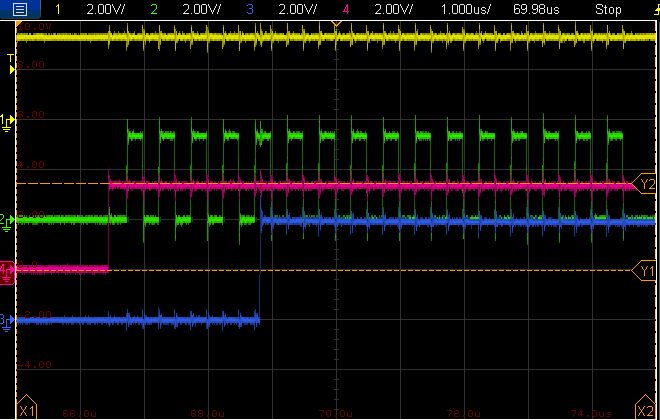


Figure 49 READ 16-bit Data Word (0x0FFF written to address 0x0000)

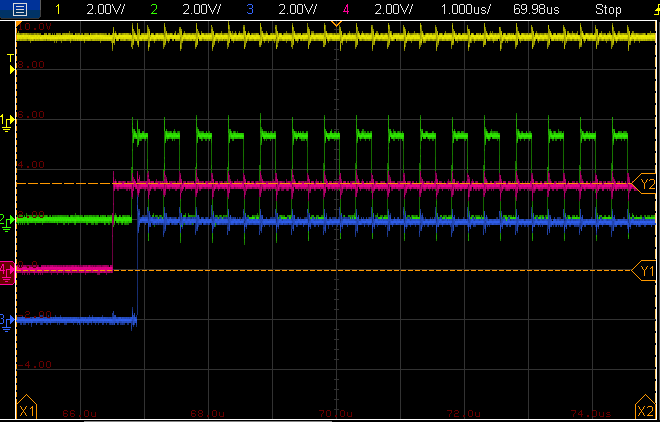


Figure 50 READ 16-bit Data Word (0xFFFF written to address 0x0000)

### WRITE

WRITE operations need EWEN instruction first before writing to memory array can be performed. Otherwise, the memory array is left untouched even if the WRITE opcode + address + data is well-formed. Refer to the EWEN operation section.

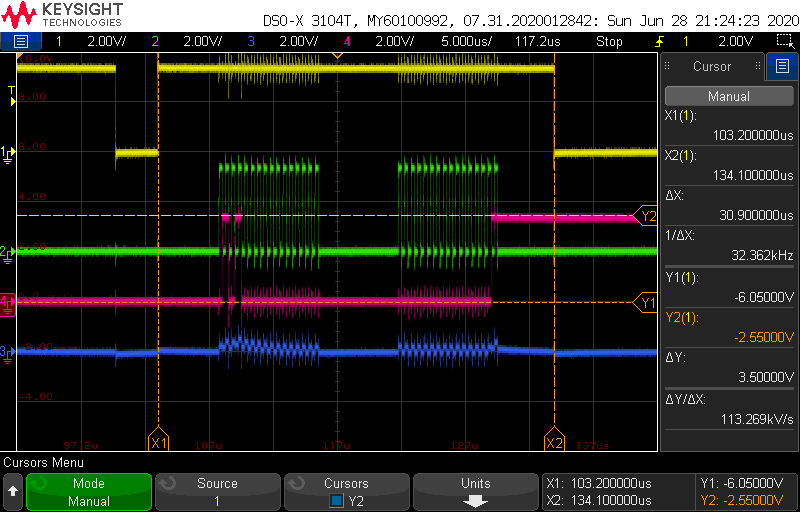
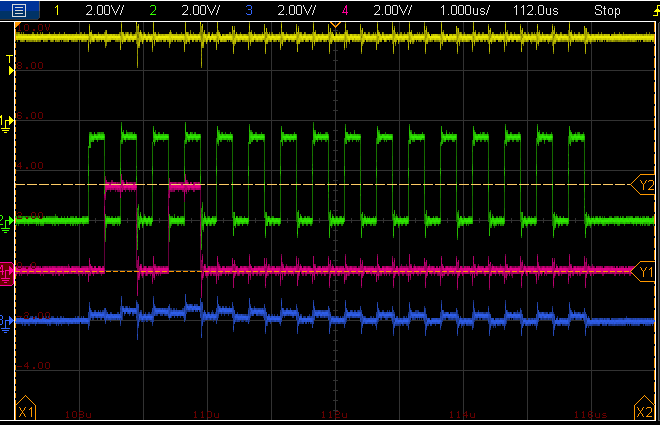


Figure 51



1. VISA stands for Virtual Instrumentation System API (Application Programming Interface). It is an abstraction layer between a higher-level programming language (e.g. LabVIEW, Visual BASIC or ANSI C) and the test equipment GPIB (IEEE-488) interface bus. NI MAX implements the VISA API protocol and INSTR resources and Aliases are VISA features. [↑](#footnote-ref-2)
2. https://www.iar.com/support/resources/articles/migration-techniques-for-different-endianness/ [↑](#footnote-ref-3)