TI Designs

EEPROM Emulation and Sensing With MSP FRAM Microcontrollers Design Guide

TI Designs

This TI Design describes an implementation of emulating EEPROM using FRAM technology on MSP low-power microcontrollers combined with the additional sensing capabilities that can be enabled when using a microcontroller (MCU). The design supports both I^2C and SPI interface to a host processor with multiple slave addressing.

Design Resources

TIDM-FRAM-EEPROM	TI Design Files
MSP430FR5994 MCU	Product Folder
MSP-EXP430FR5994	Tool Folder

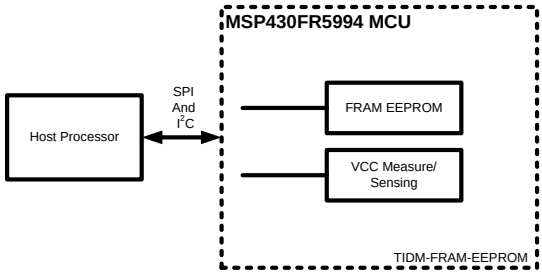
TI E2E™ Community — ASK Our E2E Experts

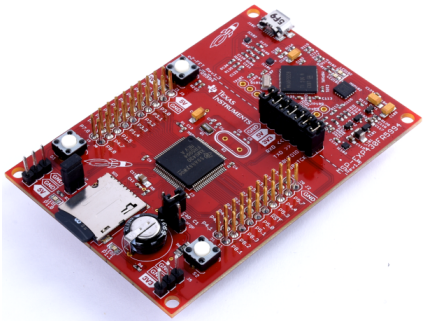
Design Features

- Offers Flexible Electrically Erasable Programmable Read-only Memory (EEPROM) Partition Allocation
- Supports an I^2C interface (100 kHz or 400 kHz) or serial peripheral interface (SPI) (up to 1 Mbps)
- Supports Stand-alone EEPROM Application
- Supports Custom Sensing Solutions Leveraging Multislave Addressing
- Offers Low-Power Consumption

Featured Applications

- Flow Metering
- GPS Tracking
- Electric Point of Sale
- Sensor Transmitters
- Industrial Machinery
- Thermostats





An IMPORTANT NOTICE at the end of this TI reference design addresses authorized use, intellectual property matters and other important disclaimers and information.

1 System Description

Ferroelectric random access memory (FRAM) is a nonvolatile memory with memory that can be written with virtually unlimited cycles. An application can continuously write to memory without stressing it, unlike with flash cells. The MSP430FR5994 MCU has integrated 256KB of FRAM, which is perfect for large applications, applications that require data logging, or saving system information. Typically, a host microcontroller or microprocessor uses an external electrically-erasable programmable read-only memory (EEPROM) to store information externally. External EEPROM stores only information and can have limited cycles.

This design emulates an EEPROM interface for both I^2C and SPI, and a developer can also customize the application to perform intelligent peripheral sensing using the onboard analog front end of the MSP MCU. This external sensing can be temperature or voltage monitoring and other options that the host could periodically read data and offloading some processing to the MSP MCU.

This design provides examples with which I^2C interface or SPI examples can get started on both the slave (the EEPROM emulation) and host (the target processor). The project examples support both CCS and IAR. This design details how to get started by connecting the appropriate jumper wires from the slave to the host. This design provides the average current profile for this application, including an EnergyTrace™++ screenshot.

1.1 MSP430FR5994 MCU

The MSP430FR5994 MCU has 256KB FRAM-based MCU with 8KB of SRAM. The MSP430FR5994 MCU has an onboard low-energy vector math accelerator (LEA) that can efficiently perform certain operations more efficiently than a typical 32-bit MCU. For more information, see http://www.ti.com/product/MSP430FR5994.

1.2 MSP-EXP430FR5994 LaunchPad™ Development Kit

To get started, the MSP-EXP430FR5994 LaunchPad™ is required as shown in Figure 1.

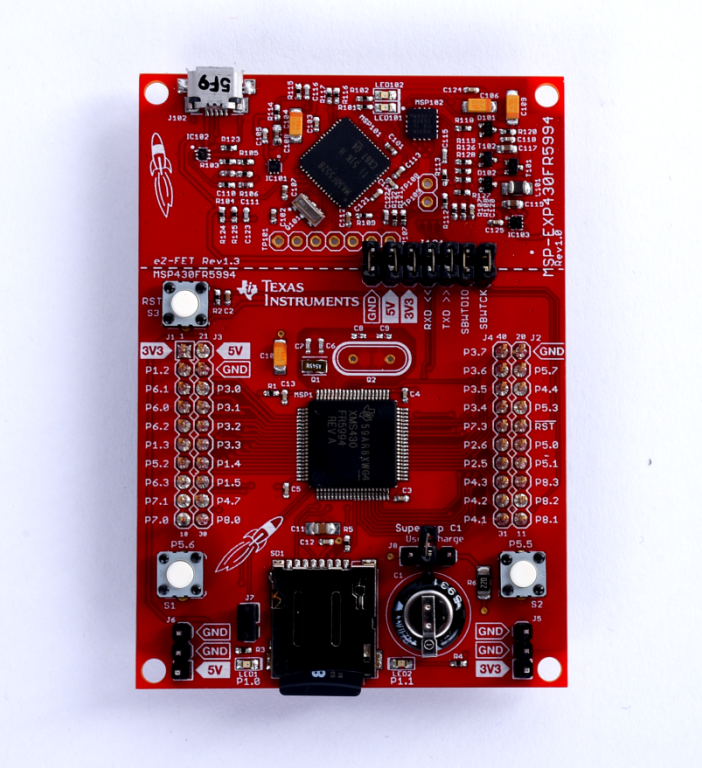


Figure 1. MSP-EXP430FR5994 LaunchPad

2 Block Diagram

Figure 2 shows the block diagram interface for the SPI.Figure 3 shows the block diagram interface for the I²C interface. In both cases, the MSP430FR5994 MCU is configured as a slave processor.

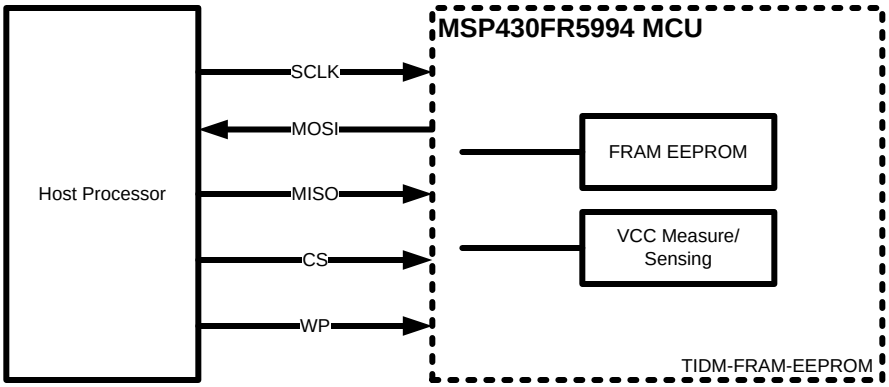


Figure 2. EEPROM SPI Block Diagram

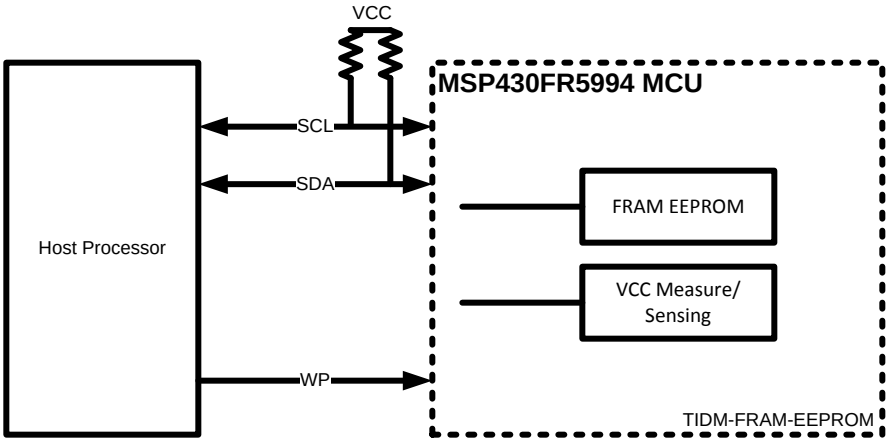


Figure 3. EEPROM I²C Interface Block Diagram

3 System Design Theory

The EEPROM emulation is configured to use I²C or SPI protocol in slave mode. The EEPROM emulation is typically connected to a host processor that would act as the master. Unlike traditional EEPROM, this implementation requires no caching after several hundred bytes. The host could continuously write data to memory when the communication is initiated and data is immediately written to memory, causing higher throughput compared to individually writing each byte.

This section explains the individual serial communication interface to select. Each following section specifies the maximum throughput for each interface and its limitation. This design also emulates industry-standard EEPROM protocols through the I²C interface and SPI as shown in Section 3.1 and Section 3.2. This design also emulates a write protection pin to protect the device from any writes.

In addition to EEPROM emulation, this reference design periodically samples the ADC for the latest VCC and temperature and stores it in FRAM at a low priority. When the host application requests the data, it is immediately available. The sensor data is currently configured to periodically sample every second and can be custom tailored for the application. The sensor reading does not block the EEPROM emulation. The EEPROM emulation is the highest priority function.

Of the 256KB of FRAM on the MSP430FR5994 MCU, 10KB is allocated for main application space to run the EEPROM emulation and data sensor functionality. This reference design emulates a total of 246KB of EEPROM memory. This design uses 3-byte EEPROM addressing method in which only 18 out of 24-bits are used.

3.1 EEPROM Emulation with I²C

The MSP MCU Enhanced Universal Serial Communication Interface (eUSCI) module I²C interface is configured to operate in slave mode with multiple slaves addressing enabled. Depending on the slave address, the eUSCI I²C module automatically matches the slave address and trigger the appropriate interrupt. This reference design uses a slave address of 0x50 for EEPROM emulation and 0x51 for sensor read. These values can be adjusted in the application.

Like any EEPROM timing diagram or protocol, the I²C typically has the start address followed by the read or write bit and the EEPROM address of from where the data is to be written or read. The default application uses 3-byte EEPROM addressing that emulates EEPROM memory greater than 64KB of memory space. The application can be scaled to use 2-byte address size for EEPROM memory less than 64KB.

The application currently supports only standard I²C frequencies of 100 kHz or 400 kHz. The supported EEPROM emulated protocols are as follows:

- Byte write
- Page write
- Acknowledge polling
- Current address read
- Random read
- Sequential read
- Sensor read

3.1.1 Byte Write

Byte writes write a single byte to memory based on the EEPROM address as the location pointer. The write protect pin is first ensured low with a minimum 10-µs latency before starting the I²C. The sequence starts with the I²C slave address, followed by the 3-byte address and then the data to be written. To stop the write, a STOP flag is set indicating the slave to end the byte write.

NOTE: When the stop is issued, the data has written to FRAM. For backward compatibility, the application can call acknowledge polling but I²C will immediately ACK. Figure 4 the I²C byte write.

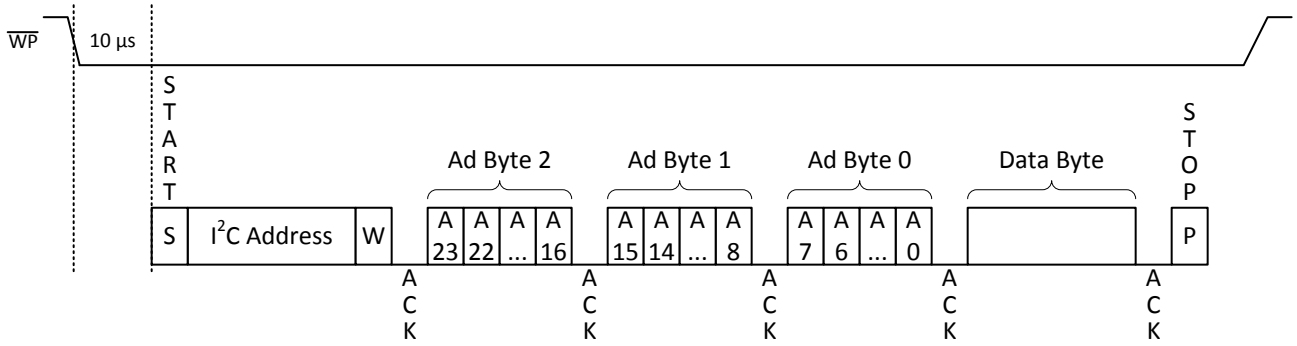


Figure 4. I²C Byte Write Timing Diagram

3.1.2 Page Write

Page write is similar to byte write but instead of writing 1 byte, the host can continuously stream data after writing the EEPROM address as the location pointer. The write protect pin is ensured low with a minimum 10-µs latency before starting the I²C. The sequence starts with the I²C slave address, followed by the 3-byte address, and the data to be written. To stop the write, a STOP flag is set indicating the slave to end the byte write.

NOTE: When the stop is issued, the data has written to FRAM. For backward compatibility reasons, the application can call acknowledge polling but I²C will immediately ACK. For more information,Figure 5 shows the I²C page write.

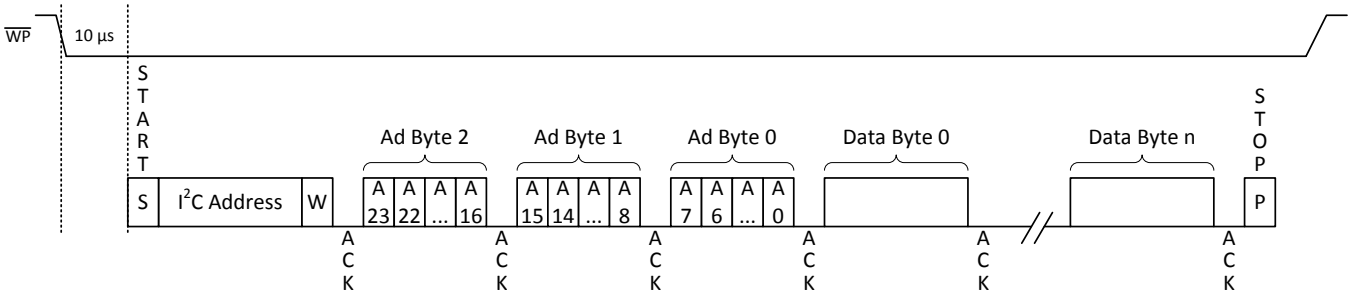


Figure 5. I²C Page Write Timing Diagram

3.1.3 Acknowledge Polling

Acknowledge polling checks if the data has already completed writing to memory by sending the start address following by checking the ACK or NACK flag. If the slave ACK, the device has completed the write operation successfully. A NACK indicates writing is ongoing. This function is available for EEPROM backward compatibility. This protocol is not required because every byte is immediately written to FRAM memory. Unlike traditional EEPROM, the EEPROM can receive a maximum length before the application must poll the EEPROM device if it has successfully written to memory. For more information, see Figure 6.

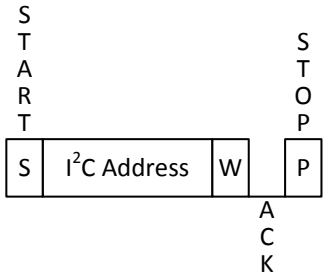


Figure 6. I²C Acknowledge Polling Timing Diagram

3.1.4 Current Address Read

The application contains an internal address counter that maintains the address of the last byte accessed. This counter returns the current byte to where the internal address is currently pointing. After reading, the internal EEPROM address is automatically incremented by 1. Figure 7 shows a timing diagram of how an application reads the value from the current address within the EEPROM memory. To end the read, the master sends a NACK followed by a Stop condition.

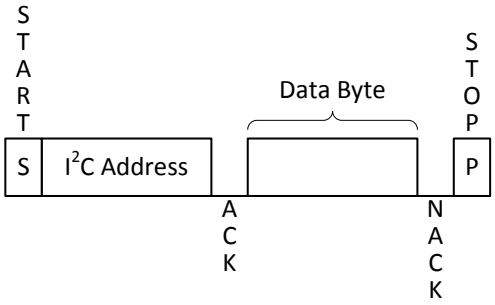


Figure 7. I²C Current Address Read Timing Diagram

3.1.5 Random Read

Random read is when the host reads 1 byte from a specified EEPROM address as part of the initial header. The host generates the Start condition then generates a write and sends the 3-byte address from which the data is read. The host then sends a repeated Start condition and then sends the read flag to start streaming the data out. At the end of the read, the master sends a NACK followed by a Stop condition. The internal counter is automatically incremented after reading the byte. For the timing diagram, see Figure 8.

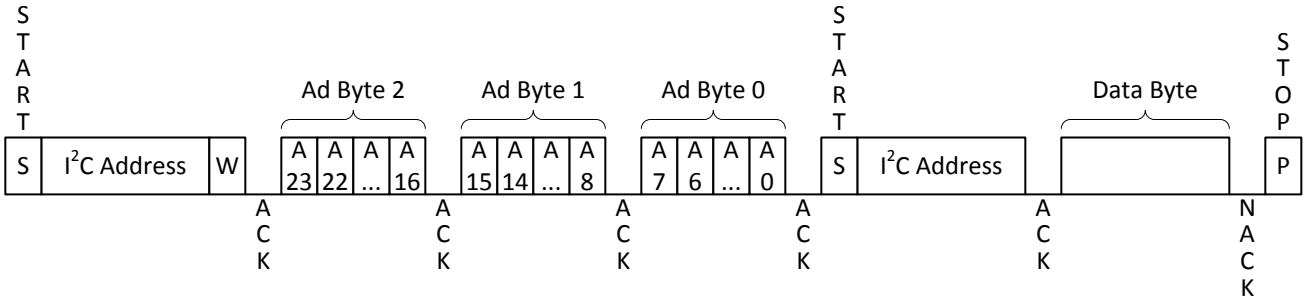


Figure 8. I²C Random Read Timing Diagram

3.1.6 Sequential Read

The sequential read is similar to random read, but instead of reading a single byte, the host can continuously stream the data out from the host until a Stop condition is issued. The host generates the Start write condition and sends the 3-byte address from which the data is read. The host then sends a repeated Start read condition to start streaming the data out. The internal counter is automatically incremented to read the next data byte out. To end the read, the master sends a NACK followed by a Stop condition. For the timing diagram, see Figure 9.

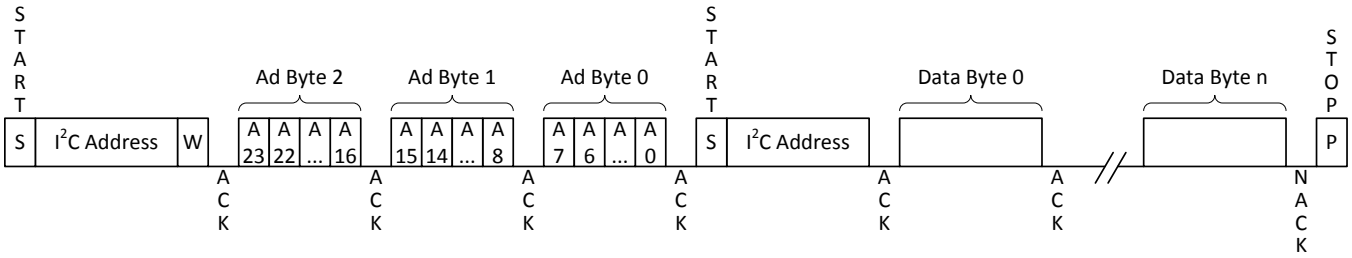


Figure 9. I²C Sequential Read Timing Diagram

3.1.7 Sensor Read

The MSP MCU periodically samples the latest sensor data and stores it within the FRAM. When the master requests the data, it is immediately available. The master generates the Start write condition with the sensor slave address of 0x51 then it generates the sensor opcode. A repeated Start read condition is issued to start clocking the sensor data out. At the end the read, the master sends a NACK followed by a Stop condition. For the timing diagram of reading the sensor data, see Figure 10. Table 1 provides an example of three sensor addresses and the number of bytes that it returns. This example returns 4 bytes of data for each sensor which is then translated into a float value.

Table 1. Sensor Address and Size

Type	Sensor Address	Returned No. of Bytes
VCC measurement	0x01	4 (float)
Temperature in °C	0x02	4 (float)
Temperature in °F	0x03	4 (float)

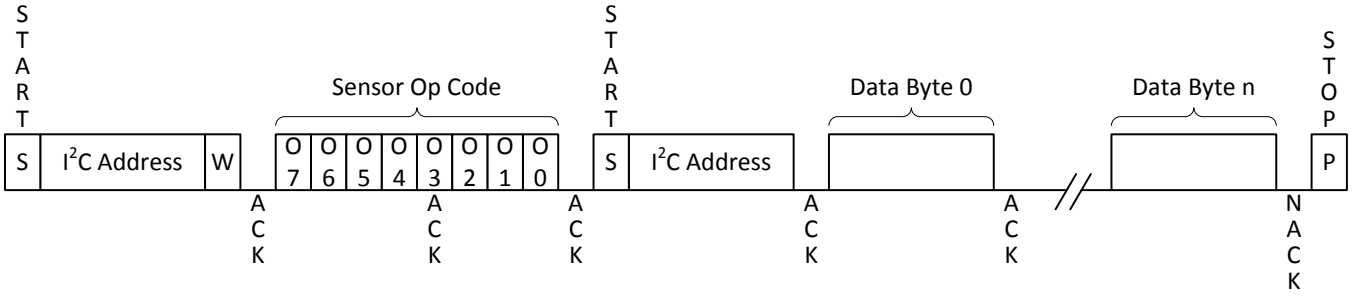


Figure 10. I²C Sensor Read Timing Diagram

3.2 EEPROM Emulation With SPI

This design is also designed to support EEPROM emulation with SPI and supports slave clock polarity high with rising edge as the trigger. This example is capable of supporting SPI clocks up to 1 Mbps using the direct memory access (DMA). All MSP430FR5x and MSP430FR6x devices have onboard DMA. For MSP FRAM MCUs that do not have DMA support, the SPI clock speed is limited to 300 kbps. See Section 7.2 on how to get started with devices that do not have DMA.

In SPI mode, there is a write protection (WP) pin and also a chip-select (CS) pin. When the WP pin is cleared (WP = 0), the EEPROM is allowed to be written. The CS pin is an active-low signal. The CS pin must be pulled low before initiating the SPI clocks and pulling CS high again when completed. CS must be pulled high again before starting a new read or write.

When the CS pin is set low, the host must send an op code as indicated in Table 2 if the host is going to write, read, or read sensor data.

Table 2. SPI Opcodes

Mode	Opcode
Write	0000 0010b
Read	0000 0011b
Sensor read	1000 0000b

After sending the opcode, the host sends a 3-byte address in which only the first 18 bits of the address is decoded to cover the 244KB of EEPROM memory. The following are the supported EEPROM emulated protocols and the timing diagram.

- Byte and page memory write
- Byte and page memory read
- Sensor read operation

3.2.1 Byte and Page Memory Write

Writes a single byte to memory based on the EEPROM address as the location pointer. The WP pin is first ensured low with a minimum of 4-µs latency before setting the CS pin low. When the CS pin is low, ensure a minimum of 13.5 µs before clocking the first clock edge. The op code initiates a write is 0x02 followed by the 3-byte EEPROM address.

Unlike traditional EEPROM, an key integral difference is that there must be a minimum of 20-µs delay after writing the 3-byte address before starting the next clock edge for data to be written. Table 3 summarizes this timing. The MCU requires the time to compute the address pointer and configures the DMA. When started, the incoming data can be 1 byte to multiple bytes continuously streamed between bytes without delay. The internal counter automatically increments to provide the next byte. No caching is required in which every byte is written immediately into memory.

Table 3. SPI Interface Minimum Timing Specification

Description	Minimum Timing
WP setup time to CS low	4 µs
CS setup time	13.5 µs
Delay between EEPROM address to first data	20 µs

The CS pin must be pulled high before starting a new operation. Figure 11 shows a timing diagram for writing a single or multiple byte(s) to memory.

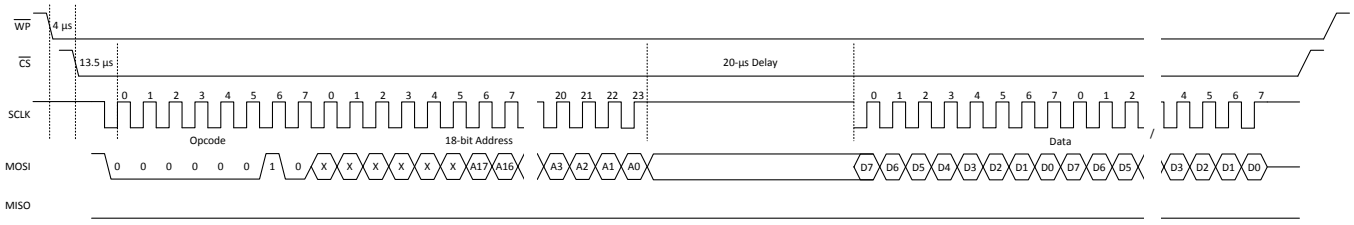


Figure 11. SPI Byte and Page Write Timing Diagram

3.2.2 Byte and Page Memory Read

Byte and page read is when the master reads 1 byte or multiple bytes from a specified EEPROM address as part of the initial header. To initiate a memory read, the CS line must be pulled low first. When the CS pin is low, ensure a minimum of 13.5 µs before clocking the first clock edge. The op code initiates a read is 0x03 followed by the 3-byte EEPROM address.

Unlike traditional EEPROM, an integral difference is that there must be a minimum of a 20-µs delay after writing the 3-byte address before the next clock edge for data to be read out. This difference is because the MCU requires the time to compute the address pointer and configures the DMA. When started, the data can be continuously read between bytes without delay.

The CS pin must be pulled high before starting a new operation. Figure 12 shows a timing diagram for reading a 1 byte or multiple bytes from memory.

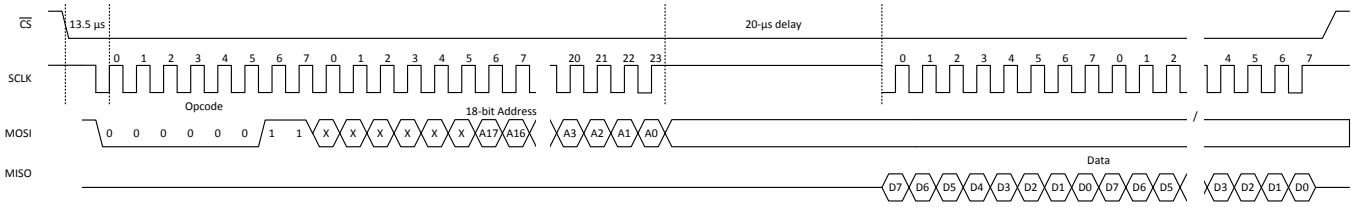


Figure 12. SPI Byte and Page Read Timing Diagram

3.2.3 Sensor Read Operation

The MSP MCU periodically samples the latest sensor data and stores it onboard the FRAM. When the master requests the data, it is immediately available. To begin, the host must pull the CS pin low. When the CS pin is set low, ensure a minimum of 13.5 µs before clocking the first clock edge. The opcode initiates a sensor read is 0x80 followed by the 1-byte sensor address to read. After the sensor address data is sent, an approximately 20-µs delay before starting the next clock edge to read the data out is provided. The returned bytes could vary depending on the sensor data and application.

Table 4 provides three sensor addresses example and the number of bytes that it returns. This example returns 4-bytes of data for each sensor, which is then translated into a float value. See Figure 13 for the timing diagram of reading the sensor data.

Table 4. Sensor Address and Size

Type	Sensor Address	Returned No. of Bytes
VCC measurement	0x01	4 (float)
Temperature in °C	0x02	4 (float)
Temperature in °F	0x03	4 (float)

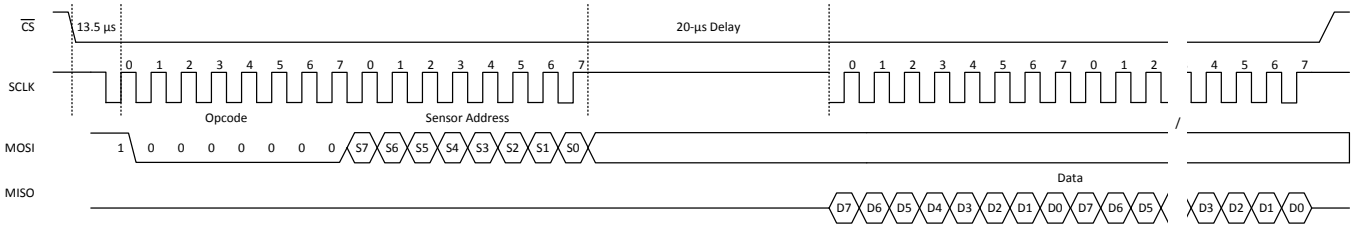


Figure 13. SPI Sensor Data Read Timing Diagram

4 Getting Started Hardware and Setup

This out-of-the-box reference design is tested using the MSP430FR5994 MCU on the MSP-EXP430FR5994 LaunchPad as shown in Figure 1. The user selects a serial interface to communicate with the host. In testing the EEPROM emulation, a set of test code for the host is provided that is designed to be used with the MSP-EXP430FR5969 LaunchPad.

4.1 Getting Started With I²C

To get started with I²C, do as follows:

1. Connect two LaunchPads using jumper wires and a 10k pullup resistor on both SDA and SCL lines as shown in Figure 14. Be sure to connect the GND of the LaunchPads together, as well.

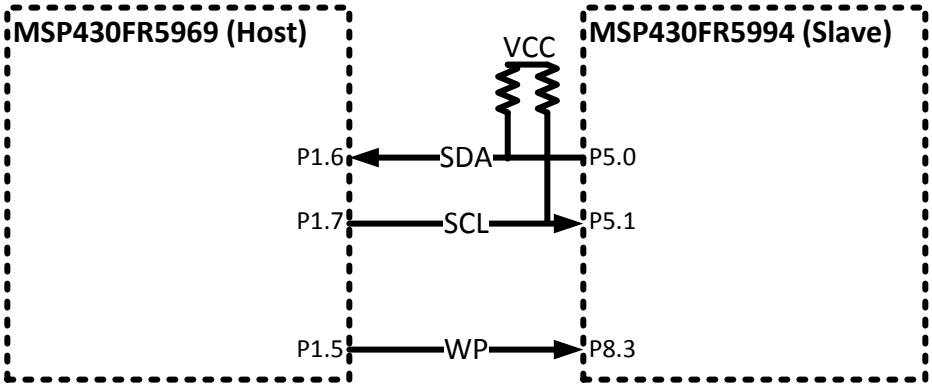


Figure 14. I²C Electrical Wiring Diagram

2. Connect the Micro-USB to the MSP-EXP430FR5994 and MSP-EXP430FR5969 LaunchPads.
3. Download the EEPROM emulation firmware for the I²C interface onto the MSP430FR5994 (eeprom_i2c_fr_emulation) (see Section 5).
4. Exit the debug session for EEPROM emulation firmware.
5. Download and debug the EEPROM test program for the I²C interface onto the MSP430FR5969 (eeprom_emu_i2c_master_test) (see Section 5).
6. Run the test program.

4.2 Getting Started With SPI

To get started with SPI, do as follows:

1. Connect two LaunchPads using jumper wires as shown in Figure 15. Be sure to connect the GND of the LaunchPads together, as well.

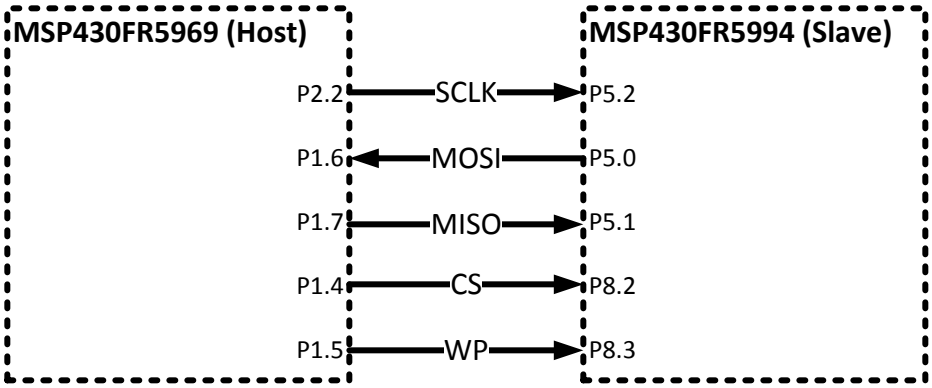


Figure 15. SPI Electrical Wiring Diagram

2. Connect the Micro-USB to both MSP-EXP430FR5994 and MSP-EXP430FR5969 LaunchPads.

3. Download the EEPROM emulation firmware for SPI interface onto the MSP430FR5994 (eeprom_spi_fr_emulation_dma) (see Section 5).
4. Run the debug session for EEPROM emulation firmware and then exit after completion.
   - Running the slave code on the MSP-EXP430FR5994 first is important because it must be initialized to receive the commands from the master.
5. Download and debug the EEPROM test program for SPI interface onto the MSP430FR5969 (eeprom_emu_spi_master_test) (see Section 5).
6. Run the test program.
   - The tests exercise the entire address range, so they take several minutes to run. Do not break or pause before the tests complete or errors may occur from interrupting the DMA.
   - In the console view, the messages visibly compile as the tests progress, for example: “Running Test at SPI Clock 1MHz”.
   - In the case of an error, the red LED on the MSP-EXP430FR5969 illuminates and an error number message displays in the console window.
   - The green LED on the MSP-EXP430FR5969 remains on during testing and then blinks when testing completes. The console also displays the message “No error found”.

5 Getting Started Firmware

In the software package, the folders are structured as follows:

- eeprom_master_test_application – Master or host emulation test code (MSP430FR5969)
  - CCS
    - i2c – CCS test project for I^2C interface
    - spi – CCS test project for SPI
  - IAR – IAR test project that supports both I^2C and SPI interface
  - Src – test application source code for both IAR and Code Composer Studio™ (CCS)
- tidm-fram-eeprom – Slave EEPROM emulation code (MSP430FR5994)
  - CCS_I2C – CCS project for I^2C interface
  - CCS_SPI_DMA – CCS project for SPI using DMA
  - IAR_I2C – IAR project for I^2C interface
  - IAR_SPI_DMA – IAR project for SPI using DMA
  - IAR_SPI_non_DMA - IAR project for SPI without DMA (devices with no DMA)
  - Src – Common source code for both IAR and CCS

NOTE: To support SPI clock rates greater than 300 kHz, a hardware DMA is required and the SPI_DMA example must be used. For slower clock rates, DMA or non-DMA can be used. Some MSP430FRxxx devices do not use DMA hardware. For more information, see device-specific data sheet.

5.1 CCS v6.1 or Newer

5.1.1 EEPROM Emulation (Slave)

The slave project must always be loaded first and the target is the MSP430FR5994 MCU. To get started, do as follows:

1. Open CCS.
2. Start a workspace.
3. Click *Project→ Import CCS Projects*.
4. Click *Browse*.
5. Navigate to tidm-fram-eeprom folder.
6. Select either I²C or SPI EEPROM emulation project (see Figure 16).

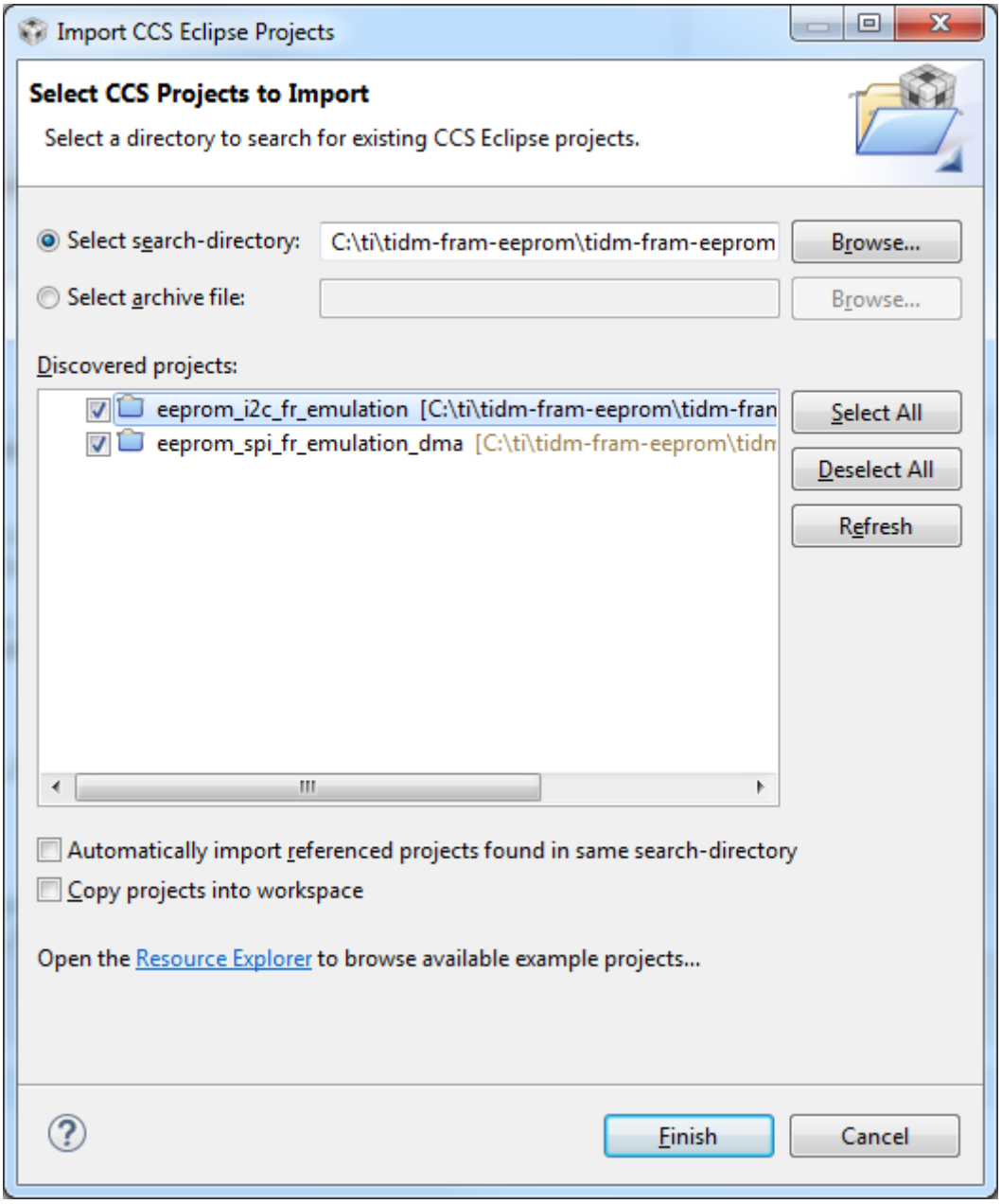


Figure 16. CCS Project Import Wizard for Slave Interface

7. Download and debug the project.
8. Run the slave code first because it requires initialization to receive commands from the master.

5.1.2 EEPROM Test Application (Host)

The host project should only be executed after the slave project has started. The slave requires time to initialize the peripherals and states. Two concurrent debug session (host and slave) could be run by running two CCS workspace. To get started, do as follows:

1. Open another CCS instance with a new workspace.
2. Click *Project→ Import CCS Projects*.
3. Click *Browse*.
4. Navigate to the eeprom_master_test_application folder.
5. Select the equivalent interface to communicate with the slave (see Figure 17).

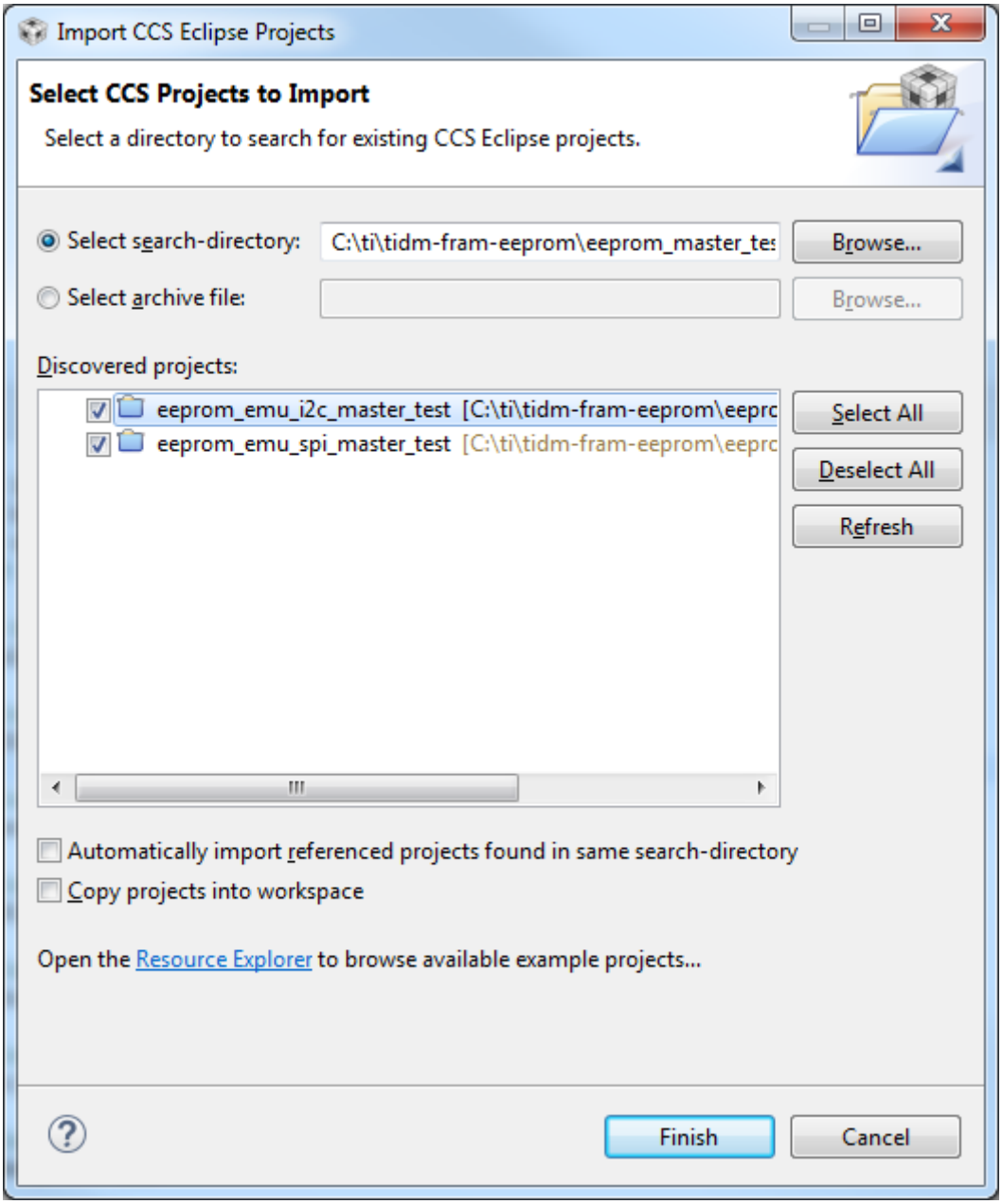


Figure 17. CCS Project Import Wizard for Host Interface

6. Download and debug the project.
7. Open the console window.
8. Run the test application and observe the printf() message output on the console.
   - The tests exercise the entire address range, so they take several minutes to run. Do not break or pause before the tests complete or errors may occur from interrupting the DMA.
   - In the console view, the messages visibly compile as the tests progress, for example “Running Test at SPI Clock 1MHz”.
   - In the case of an error, the red LED on the MSP-EXP430FR5969 illuminates and an error number message displays in the console window.
   - The green LED on the MSP-EXP430FR5969 remains on during testing and then blinks when testing completes. The console also displays the message “No error found”.

5.2 IAR Embedded Workbench® v6.40.2 or Newer

5.2.1 EEPROM Emulation (Slave)

The slave project must always be loaded first and the target is the MSP430FR5994 MCU. To get started, do as follows:

1. Launch IAR™ Embedded Workbench for the MSP430™ MCU.
2. Open the eeprom_fr_emulation.eww workspace project in the tidm-fram-eeprom folder.
   In this workspace, there are three projects (see Figure 18).

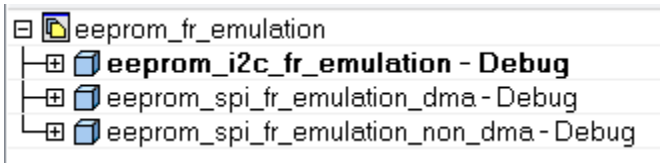


Figure 18. IAR Workspace Project for the Slave Interface

3. Right-click on the project.
4. Select *Set Active*.
5. Download and debug the project.
6. Run the slave code.

5.2.2 EEPROM Test Application (Host)

The host project must only execute after the slave project has started. The slave requires time to initialize the peripherals and states. Two concurrent debug session (host and slave) can be run by running two instances of IAR window.

1. Launch another IAR Embedded Workbench for MSP MCUs.
2. Open the eeprom_emu_master_test.eww workspace project in the eeprom_master_test_application\IAR folder.
   In this workspace, there are two test projects (see Figure 19).

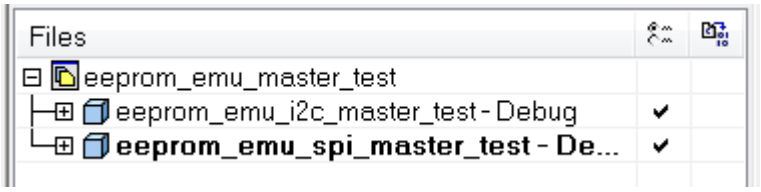


Figure 19. IAR Workspace Project for the Host Interface

3. Right-click on the project.
4. Select *Set Active*.

5. Download and Debug the project.
6. Open the *Terminal I/O* view.
7. Click *Run* to see the printf() status output.

6 Application Performance

The entire EEPROM emulation (slave) application must be compiled for the highest optimization level. In IAR, set the application to *High, Balanced* while CCS is set to Level 4 to ensure the slave can respond to the master with the least latency.

6.1 Application Size

The application code size is optimized for performance and footprint. SPI and I^2C code sizes are approximately 4.3KB in IAR and 4.4KB in CCS.

6.2 Average Current Consumption

The Keysight N6705B is used to perform a power profile analysis on the slave. For the current profile for the I^2C mode, see Figure 20 and Figure 21. Table 5 lists the average current consumption. The internal pull-up resistor mainly contributes to the standby with background sensing average current consumption.

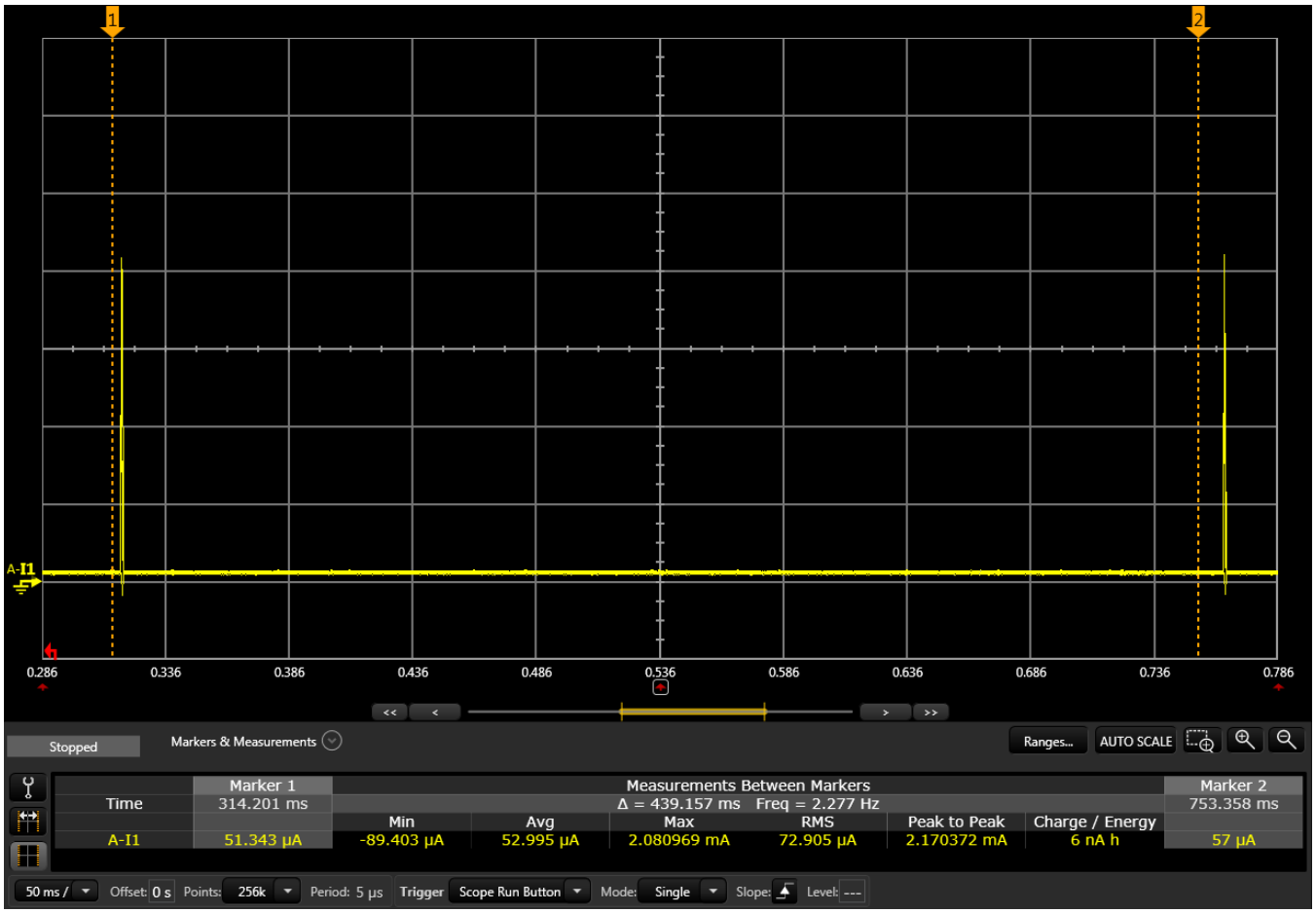


Figure 20. I^2C Standby Current Profile

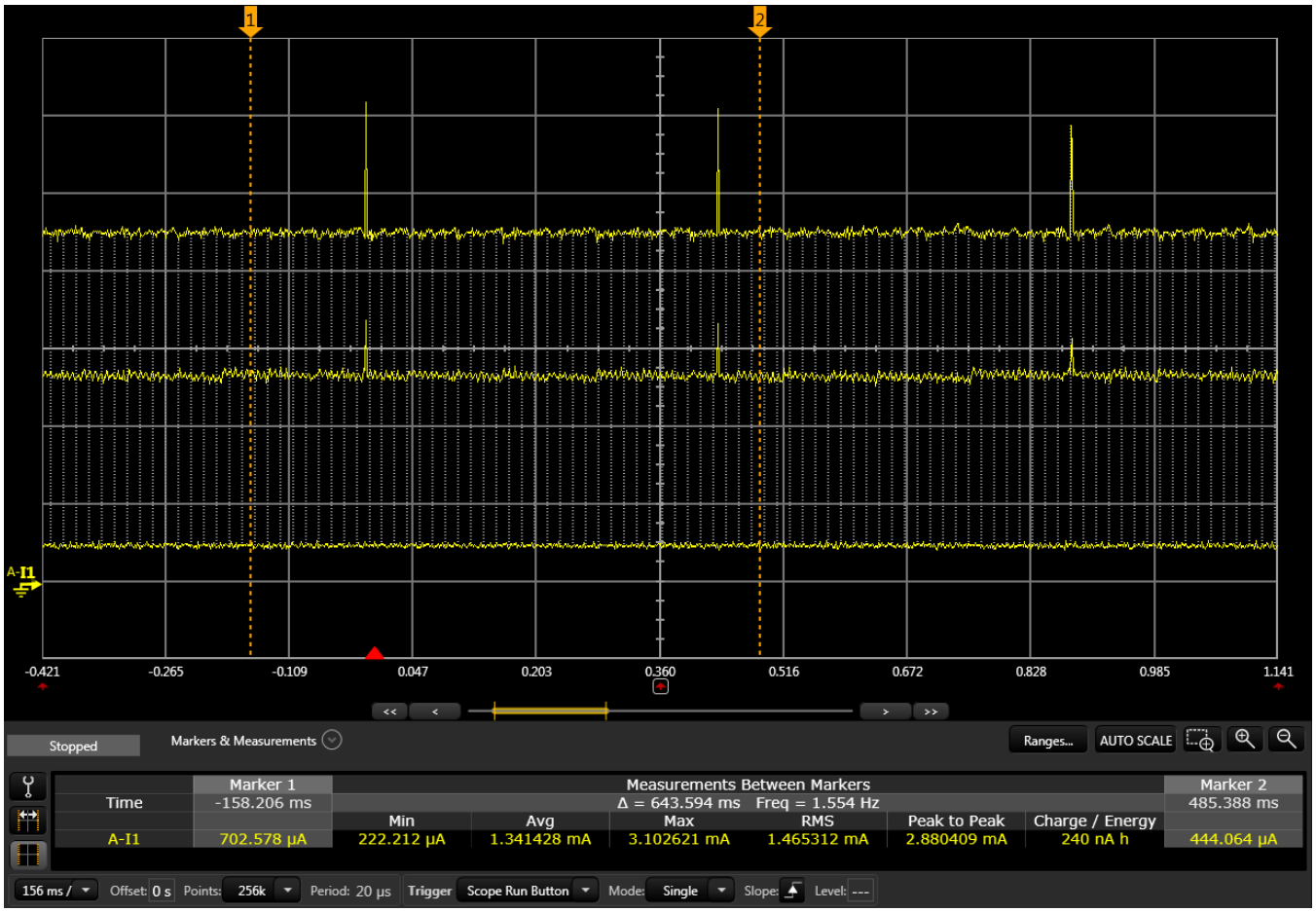


Figure 21. I²C Active EEPROM Read/Write at 400 kHz

Table 5. Average I²C Interface Current Consumption

Task	Current
Standby current with background sensing	53 µA
Reading/Writing at 100 kbps	1.10 mA
Reading/Writing at 400 kbps	1.34 mA
Competitor A Write	5 mA

In SPI mode, Table 6 summarizes the average current consumption. The internal pull-up resistor mainly contributes to the standby with background sensing average current consumption. The current profile would look similar to the I²C as in Figure 20 and Figure 21.

Table 6. Average SPI Interface Current Consumption

Task	Current
Standby current with background sensing	53 µA
Reading/Writing at 250 Mbps	0.994 mA
Reading/Writing at 500 kbps	1.28 mA
Reading/Writing at 1 Mbps	1.51 mA
Competitor A Write	6 mA

6.3 EnergyTrace™++ Technology

EnergyTrace++ technology for MSP MCUs is an energy-based code analysis tool that measures and displays the energy profile of the application and helps optimize it for ultra-low-power consumption. EnergyTrace++ technology, also known as EnergyTrace+[CPU States]+[Peripheral States], brings the capabilities of EnergyTrace++ technology (from just measuring energy) to the next level.

When debugging with devices that contain the built-in EnergyTrace++ support, the technology provides information about energy consumption and the internal state of the MCU. These states include the ON/OFF status of the peripherals and all system clocks (regardless of the clock source) and the low-power mode (LPM) currently in use. This tool helps directly verify whether an application is demonstrating the expected behavior at the correct points in the code, such as ensuring that a peripheral is turned off after a certain activity.

The MSP-EXP430FR5994 LaunchPad supports EnergyTrace++ technology, which is used to analyze the application state. EnergyTrace++ screenshot, as shown in Figure 22, shows the application is in LPM3 during standby. Figure 22 shows EnergyTrace++ state transitions from standby to active reading or writing.

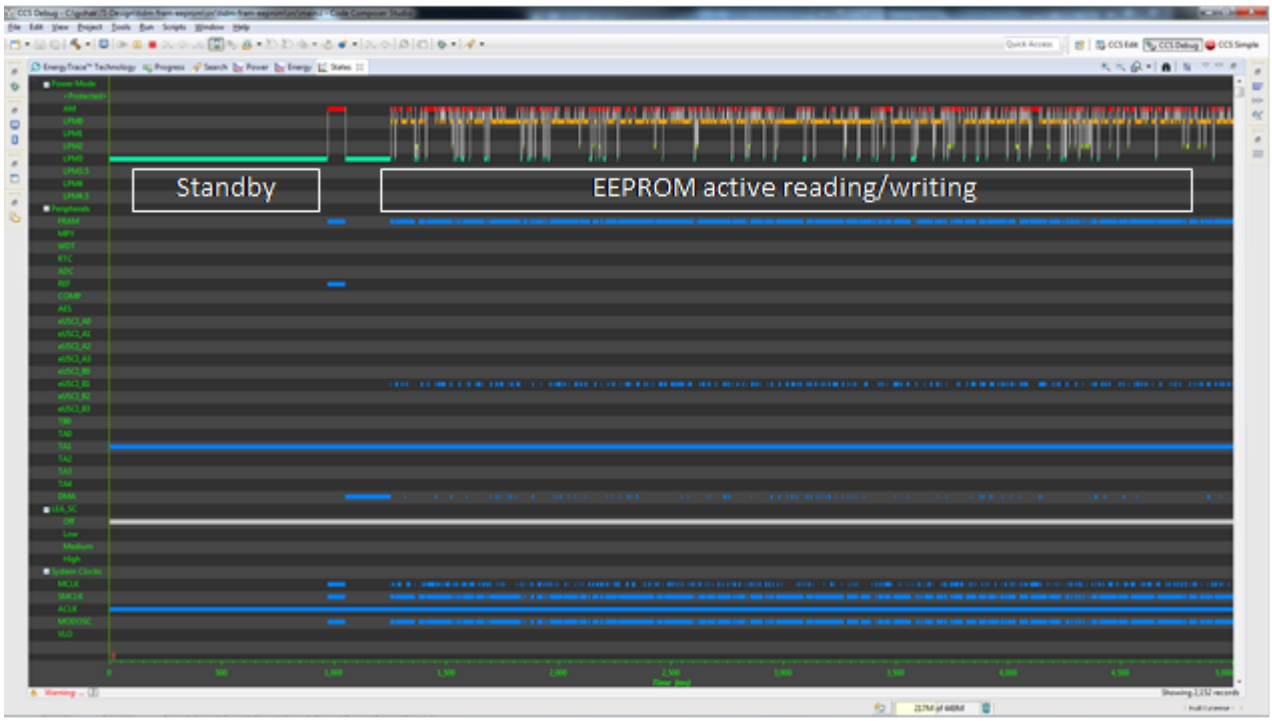


Figure 22. EnergyTrace++ State Transitions From Standby to Active Reading or Writing

7 Software Files

To download the software files for this reference design, see http://www.ti.com/tool/TIDM-FRAM-EEPROM.

7.1 Support for MSP430FR5x/FR6x Devices

This application supports all MSP430FR5x/FR6x with minimal modification. To modify the application, do as follows:

1. Select the new MSP430 device.
2. Customize the eeprom_definitions.h file to ensure the memory allocation is within the boundary of the device memory.

NOTE: EEPROM1_BEGIN, EEPROM1_END, EEPROM2_BEGIN, and EEPROM2_END must be customized.

3. Decide if the device is sufficient to be a 3- or 2-byte address.

7.2 Support for MSP430FR4x/FR2x Devices

The application is modifiable to support MSP430FR4x/FR2x devices. The MSP430FR4x/FR2x does not have hardware DMA and the SPI interface for EEPROM emulation speed is limited to 300 kHz. Within IAR, the example is provided under eeprom_spi_fr_emulation_non_dma. To enable MSP430FR4x/FR2x, modify the eeprom_definitions.h file.

Within the eeprom_definitions.h file, ensure USE_DMA is removed. Because MSP430FR4x/FR2x are less than 64KB of FRAM, the application can use 2-byte addressing. Ensure to properly allocate the EEPROM1_BEGIN and EEPROM_END location. The size of the EEPROM1_BEGIN and EEPROM_END must be aligned on a 4-byte boundary.

MSP430FR4x/FR2x has a global FRAM write protection bit (SYSCFG0.PFWP). Ensure this bit is cleared before writing any data to FRAM and set the bit again to ensure the rest of the FRAM application is protected against unwanted writes.

7.3 Integrating This TI Design to an Application

In adopting this design, consider the critical timing for CS, WP, or DMA interrupts as they must respond as quickly as possible. To ensure application responsiveness, the custom application must be nonblocking (avoid putting large routines in an interrupt service routine).

Note:

NOTE: Timing is dependent on the software implementation. The delays might require adjustment on the host side if additional critical timing software is added to the slave.

If the size of the customer application is greater than 10KB (what is allocated currently), the size can be adjusted by modifying the EEPROM1_BEGIN address in the eeprom_definitions.h file.

8 References

- Texas Instruments, *Texas Instruments E2E Community*, http://e2e.ti.com/
- Texas Instruments, *MSP430FR599x, MSP430FR596x Mixed-Signal Microcontrollers* (SLASE54)
- Texas Instruments, *MSP430FR58xx, MSP430FR59xx, MSP430FR68xx, and MSP430FR69xx Family User's Guide* (SLAU367)
- Texas Instruments, *MSP-EXP430FR5994 tool page*, http://www.ti.com/tool/msp-exp430fr5994

8.1 Trademarks

Revision History A

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Changes from Original (March 2016) to A Revision	Page
• Added disclaimer: "Be sure to connect the GND of the LaunchPads together".	10
• Added disclaimer: "Be sure to connect the GND of the LaunchPads together".	10
• Added expanded information regarding step to "run the debug session for EEPROM emulation firmware and then exit after completion".	11
• Added expanded level of information regarding step to "run the the test program".	11
• Added expanded information regarding step to "run the slave code first".	13
• Added further steps after step to "download and debug the project".	14

IMPORTANT NOTICE FOR TI REFERENCE DESIGNS

Texas Instruments Incorporated ('TI") reference designs are solely intended to assist designers ("Designer(s)") who are developing systems that incorporate TI products. TI has not conducted any testing other than that specifically described in the published documentation for a particular reference design.

TI's provision of reference designs and any other technical, applications or design advice, quality characterization, reliability data or other information or services does not expand or otherwise alter TI's applicable published warranties or warranty disclaimers for TI products, and no additional obligations or liabilities arise from TI providing such reference designs or other items.

TI reserves the right to make corrections, enhancements, improvements and other changes to its reference designs and other items.

Designer understands and agrees that Designer remains responsible for using its independent analysis, evaluation and judgment in designing Designer's systems and products, and has full and exclusive responsibility to assure the safety of its products and compliance of its products (and of all TI products used in or for such Designer's products) with all applicable regulations, laws and other applicable requirements. Designer represents that, with respect to its applications, it has all the necessary expertise to create and implement safeguards that (1) anticipate dangerous consequences of failures, (2) monitor failures and their consequences, and (3) lessen the likelihood of failures that might cause harm and take appropriate actions. Designer agrees that prior to using or distributing any systems that include TI products, Designer will thoroughly test such systems and the functionality of such TI products as used in such systems. Designer may not use any TI products in life-critical medical equipment unless authorized officers of the parties have executed a special contract specifically governing such use. Life-critical medical equipment is medical equipment where failure of such equipment would cause serious bodily injury or death (e.g., life support, pacemakers, defibrillators, heart pumps, neurostimulators, and implantables). Such equipment includes, without limitation, all medical devices identified by the U.S. Food and Drug Administration as Class III devices and equivalent classifications outside the U.S.

Designers are authorized to use, copy and modify any individual TI reference design only in connection with the development of end products that include the TI product(s) identified in that reference design. HOWEVER, NO OTHER LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE TO ANY OTHER TI INTELLECTUAL PROPERTY RIGHT, AND NO LICENSE TO ANY TECHNOLOGY OR INTELLECTUAL PROPERTY RIGHT OF TI OR ANY THIRD PARTY IS GRANTED HEREIN, including but not limited to any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license to use such products or services, or a warranty or endorsement thereof. Use of the reference design or other items described above may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

TI REFERENCE DESIGNS AND OTHER ITEMS DESCRIBED ABOVE ARE PROVIDED "AS IS" AND WITH ALL FAULTS. TI DISCLAIMS ALL OTHER WARRANTIES OR REPRESENTATIONS, EXPRESS OR IMPLIED, REGARDING THE REFERENCE DESIGNS OR USE OF THE REFERENCE DESIGNS, INCLUDING BUT NOT LIMITED TO ACCURACY OR COMPLETENESS, TITLE, ANY EPIDEMIC FAILURE WARRANTY AND ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT OF ANY THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

TI SHALL NOT BE LIABLE FOR AND SHALL NOT DEFEND OR INDEMNIFY DESIGNERS AGAINST ANY CLAIM, INCLUDING BUT NOT LIMITED TO ANY INFRINGEMENT CLAIM THAT RELATES TO OR IS BASED ON ANY COMBINATION OF PRODUCTS AS DESCRIBED IN A TI REFERENCE DESIGN OR OTHERWISE. IN NO EVENT SHALL TI BE LIABLE FOR ANY ACTUAL, DIRECT, SPECIAL, COLLATERAL, INDIRECT, PUNITIVE, INCIDENTAL, CONSEQUENTIAL OR EXEMPLARY DAMAGES IN CONNECTION WITH OR ARISING OUT OF THE REFERENCE DESIGNS OR USE OF THE REFERENCE DESIGNS, AND REGARDLESS OF WHETHER TI HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

TI's standard terms of sale for semiconductor products (http://www.ti.com/sc/docs/stdterms.htm) apply to the sale of packaged integrated circuit products. Additional terms may apply to the use or sale of other types of TI products and services.

Designer will fully indemnify TI and its representatives against any damages, costs, losses, and/or liabilities arising out of Designer's non-compliance with the terms and provisions of this Notice.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265