Maximizing Write Speed on the MSP430™ FRAM

Priya Thanigai, William Goh   *MSP430 Applications*

ABSTRACT

Nonvolatile low-power ferroelectric RAM (FRAM) is capable of extremely high-speed write accesses. This application report discusses how to maximize FRAM write speeds specifically in the MSP430FRxx family using simple techniques. The document uses examples from bench tests performed on the MSP430FR5739 device, which can be extended to all MSP430™ FRAM-based devices, and discusses tradeoffs such as CPU clock frequency and block size and how they impact the FRAM write speed.

Related software can be downloaded from http://www.ti.com/lit/zip/slaa498.

Contents

List of Figures

List of Tables

MSP430 is a trademark of Texas Instruments.
All other trademarks are the property of their respective owners.

1   Introduction to FRAM

FRAM is a nonvolatile memory technology that behaves similar to the widely used volatile static RAM (SRAM). It has all the advantages that can be attributed to SRAM, such as being bit-addressable with no requirements for pre-erase, while being nonvolatile. The MSP430FR5739 is the world's first embedded FRAM solution with a 16-bit ultra low power MCU.

There are a few significant differences between the MSP430FR5739 and other MSP430 flash offerings, some of which are outlined in *Migrating From the MSP430F2xx Family to the MSP430FR57xx Family* (SLAA499).

The purpose of this application report is to provide a brief introduction to FRAM while discussing typical application use cases and how they can be implemented to improve FRAM write speed. There are two main differences between FRAM and SRAM:

- FRAM is nonvolatile; that is, it retains contents on loss of power.
- The embedded FRAM on MSP430 devices can be accessed (read or write) at a maximum speed of 8 MHz.

In comparison to MSP430 flash[1], FRAM:

- Is very easy to use
- Requires no setup or preparation such as unlocking of control registers
- Is not segmented and each bit is individually erasable, writable, and addressable
- Does not require an erase before a write
- Allows low-power write accesses (does not require a charge pump)
- Can be written to across the full voltage range (2.0 V to 3.6 V)
- Can be written to at speeds close to 8 MBps (maximum flash write speed including the erase time is approximately 14 kBps)
- Requires wait states when accessed at speeds > 8 MHz

[1] All MSP430 flash data is from the *MSP430F22x2, MSP430F22x4 Mixed-Signal Microcontrollers* data sheet (SLAS504).

A comparison between flash, FRAM, and static RAM is given in Table 1.

Table 1. Comparison Between Embedded Memory Technologies

	FRAM	SRAM	Flash
Write speed per word	125 ns	< 125 ns	85 µs
Erase time	No pre-erase required	No pre-erase required	23 ms for 512 bytes
Bit-wise programmable	Yes	Yes	No
Write endurance	10^{15} write/erase cycles	N/A	10^5 write/erase cycles
Nonvolatile	Yes	No	Yes
Internal write voltage	1.5 V	1.5 V	12 V to 14 V (charge pump required)

2   Writing to FRAM

2.1   The FRAM Write Cycle

All FRAM accesses are limited to 125 ns per access or 8-MHz access frequency. However, the MSP430FRxx family supports system speeds (MCLK) up to 16 MHz or 24 MHz, (see the device-specific data sheet) which is faster than the FRAM access frequency. Therefore, the FRAM controller requires wait states to prevent the FRAM from being accessed at speeds faster than 8 MHz. For the MSP430FR57xx devices, the wait-state generator can be controlled automatically or manually—see the *MSP430FR5xx Family User's Guide* ([SLAU272](https://www.ti.com)). For all other MSP430 FRAM devices, the wait state must be manually configured—see the *MSP430FR58xx, MSP430FR59xx, MSP430FR68xx, and MSP430FR69xx Family User's Guide* ([SLAU367](https://www.ti.com)).

2.2   Theoretical Calculation on the Maximum FRAM Write Speed

The MSP430FRxx data sheets specify FRAM word or byte write time t_{WRITE} as 125 ns [3] [4] [5] [6]. This leads to a maximum write time of 8 megawords per second or 16 megabytes per second (MBps). However, this maximum speed is theoretical because it does not account for the time taken for data handling overhead. Small blocks of data require frequent processing for pointer updating and data retrieval when compared to larger blocks. Therefore, one method to increase the speed of FRAM write is to minimize CPU intervention for data handling and use DMA to optimize data transfers.

A second method to optimize the write speed is by increasing the CPU speed beyond 8 MHz, allowing data handling operations that are cached in SRAM to be executed at speeds faster than 8 MHz. These two methods are discussed in greater detail in the following sections.

2.3   Increasing FRAM Write Speed Using DMA

Because FRAM writes are very high speed, the biggest bottleneck lies in the data handling and processing overhead. This may be due to a communication protocol that is limited to a specific speed or due to application overhead caused by data movement and pointer updates. Figure 1 presents an example use case where 2 to 512 bytes are written to FRAM using a simple code example that performs a move and an increment operation at a CPU clock frequency (MCLK) = 8 MHz. This example use case is then compared to writing the same number of bytes but using DMA instead. It can be seen that for smaller block sizes using DMA does not provide any noticeable advantage, but as the block size increases, using DMA minimizes CPU intervention per block, thereby, reducing the overall write time and dramatically increasing FRAM write speed. Figure 1 illustrates, for a block size of 512 bytes, using DMA yields FRAM write speeds that are 4x that when DMA is not used.

Therefore, DMA is the preferred approach to efficiently benchmark the maximum achievable speed when writing to FRAM.
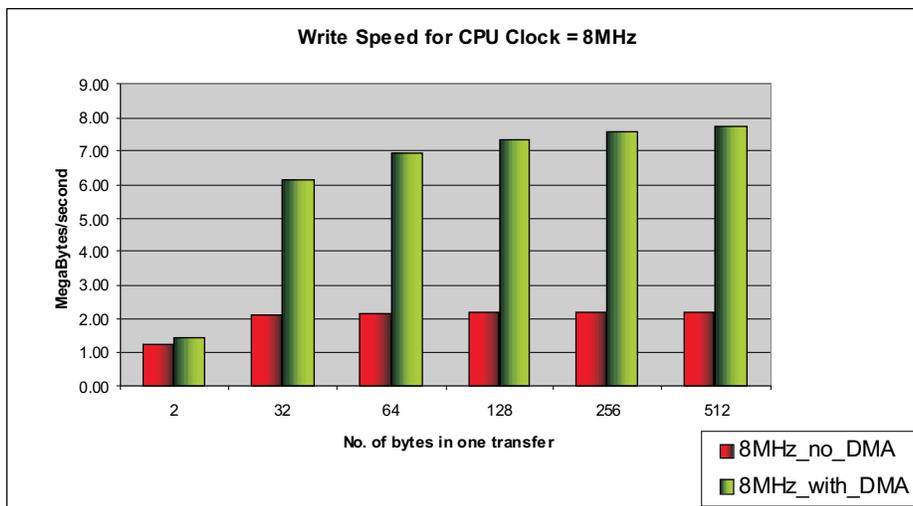

Figure 1. Comparing Write Speeds With and Without DMA Usage

Each DMA transfer takes 2 MCLK cycles. When writing to memory using DMA, a block-wise approach is preferred because the initial overhead for setting up the block is spread over the size of the block. An example code that uses block-wise DMA to write to FRAM is shown. Because FRAM writes are similar to RAM writes there is no special memory handling required.

The code used to benchmark the time taken to write an FRAM block does the following:

- Initializes DMA registers and configures block size
- Toggles general-purpose input/output (GPIO) at the start of a DMA transfer
- Sets the DMA trigger to transfer a block
- Toggles GPIO to indicate the end of DMA transfer

Because the CPU is held while the DMA block transfer is in progress and not released until it is complete, it is not required to check the DMA interrupt flag at the end of a transfer. The GPIO output pin is used to measure the time per DMA block write. The code is written in assembler to avoid any potential overhead or other possible side effects caused by C compilers.

```
SetupData
          mov.w   #0x1D00,R4              ; Data is stored in SRAM @ 0x1D00
          mov.w   #0x1234,0(R4)
SetupPort                                 ; output toggle on P3.6
          bis.b   #BIT6,&P3DIR


SetupDMA
          movx.a  #0x1D00,&DMA0SA         ; Start block address- some RAM location
          movx.a  #0xD000,&DMA0DA         ; Destination block address - FRAM
scratchpad
          mov.w   #1,&DMA0SZ              ; Block size in words 1 to 4096
                                          ; DMA in repeated block, dst increment, word mode
          mov.w   #DMADT_5+DMASRCINCR_0+DMADSTINCR_3,&DMA0CTL
          bic.w   #DMADSTBYTE+DMASRCBYTE,&DMA0CTL
          bis.w   #DMAEN, &DMA0CTL        ; Enable DMA0
Mainloop
          bis.b   #BIT6, &P3OUT          ; Set P3.6 at the start of transfer
          bis.w   #DMAREQ,&DMA0CTL       ; Clear P3.6 at the end of transfer
          bic.b   #BIT6,&P3OUT           ; Trigger block transfer
          jmp     Mainloop
```

The plot of measured FRAM write speed for various DMA block sizes is shown in Figure 2. These measurements were taken with CPU clock frequency = 8 MHz.

FRAM Write Speed Vs DMA Block Size
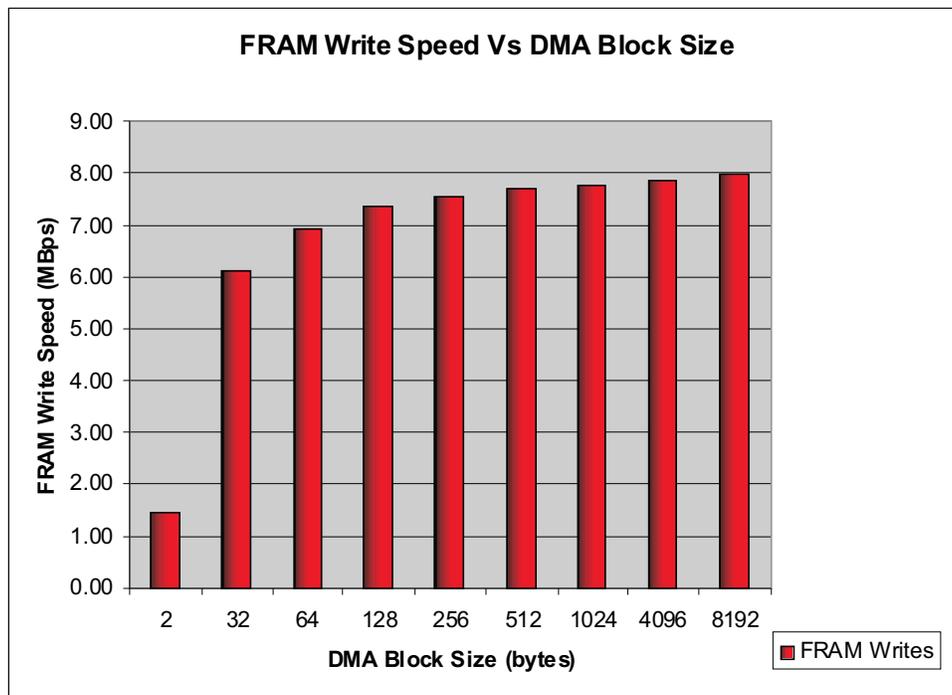
Figure 2. FRAM Write Speed for Various DMA Block Sizes (CPU Clock Frequency = 8 MHz)

Figure 2 shows that for a large block size of 8192 bytes, the write speed of FRAM is close to 8 MBps. This is half the theoretical maximum FRAM write speed according to Section 2.2. The reason for this is that for every FRAM word write – two CPU cycles are spent in the actual data transfer. The smaller the block size, the more time is spent in toggling the GPIO output and re-triggering the DMA when compared to the actual FRAM access. While a large transfer of 8KB is most likely not suitable in all applications, it can be seen that even for a moderate block size of 512 bytes (similar to flash), the FRAM write speed is ~7.7 MBps and is about 500 times faster than a corresponding 512-byte block flash write.

Table 2 shows the actual write times and corresponding write speeds for various DMA block sizes.

Table 2. FRAM Write Speed for Various DMA Block Sizes (CPU Clock Frequency = 8 MHz)

DMA Size (bytes)	MCLK = 8 MHz	MCLK = 8 MHz
	Write Time µs	Write Speed MBps
2	1.35	1.45
32	5.1	6.13
64	9.02	6.93
128	17.04	7.34
256	33.14	7.54
512	64.74	7.72
1024	128.3	7.79
4096	509.2	7.86
8192	1002	7.98

2.4   *Increasing FRAM Write Speed by Maximizing CPU Clock Frequency*

The FRAM controller uses a 2-way associative cache that has a 64-bit line size. The cache uses static RAM to store pre-fetched instructions. Figure 3 shows a block diagram of the FRAM controller. The function of the FRAM controller is to pre-fetch four instruction words depending on the current PC location. The actual execution of these instructions is carried out in the cache. Once the end of the cache buffer is reached, the FRAM controller preserves the four current words in one page of the cache and fetches the next four words. If a code discontinuity is encountered at the end of a two-page cache, the cache is refreshed and the following four words of instruction are retrieved from FRAM. However, if the application code loops back to a location already present in the cache at the end of the cache, the relevant instruction is simply executed directly from the cache instead of re-fetching code from FRAM.

Note that only FRAM accesses are subject to the 8 MHz access limitation. A system clock of up to 16 MHz or 24 MHz (see the device-specific data sheet) can be used when executing from the cache, which significantly increases the throughput of code execution. Wait states are inserted only when the FRAM needs to be accessed to refresh the cache, that is, when a code discontinuity is encountered. Also, the FRAM controller makes use of the cache efficiently in the background in a way that is transparent to the user.
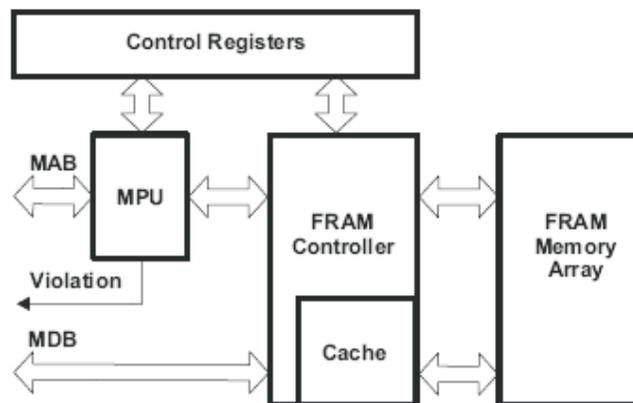
Figure 3. FRAM Controller Block Diagram

However, it is good practice to write short, efficient loops to achieve the maximum benefit of executing code from the cache and to minimize cache refreshes. Executing from the cache makes the system more optimized for speed and power because:

*   Power is significantly lower when executing code from cache
*   The execution throughput is not limited to 8 MHz, thereby, maximizing the FRAM write speed

When writing to FRAM, bench tests were performed using different MCLK frequencies of 8 MHz, 16 MHz, and 24 MHz (only MSP430FR57xx devices support 24 MHz) using DMA with varying block sizes. The results are shown in Figure 4.
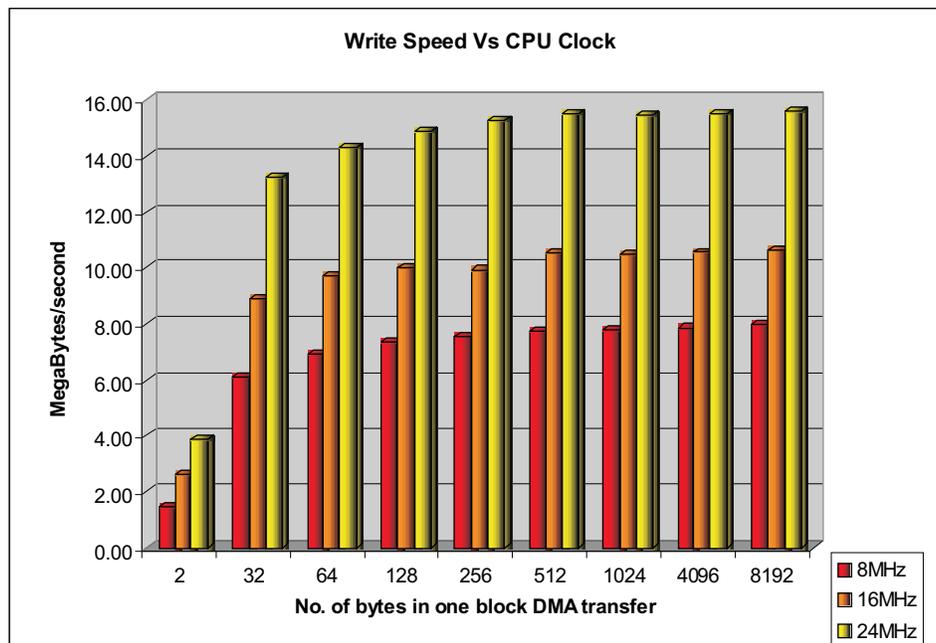
Figure 4. FRAM Write Speed vs. CPU Clock Frequency (MCLK)

In the chart, data for f_{MCLK} = 8 MHz is the same as in Figure 2. The orange and yellow bars illustrate the maximum FRAM write speed when f_{MCLK} = 16 MHz and 24 MHz, respectively.

Consider the data point for DMA block size = 512 bytes:

Speed at f_{MCLK} = 8 MHz: 7 MBps

Speed at f_{MCLK} = 16 MHz: 10 MBps

Speed at f_{MCLK} = 24 MHz : 15 MBps

This data shows that by increasing the MCLK frequency by a factor of 3, the FRAM write speed does not increase proportionally. That is because only the access to RAM and the instruction fetches in cache are executed at 24 MHz. Any access to FRAM is still performed at 8 MHz. Therefore, only a portion of the entire block write benefits from the f_{MCLK} increase. However, the throughput increase (not 1:1) is still significant, resulting in almost double the FRAM write speed at three times the MCLK frequency.

Consider a second data point where DMA transfer size is 8KB. At f_{MCLK} = 24 MHz, the FRAM write speed is very close to the theoretical maximum of 16 MBps. However, before it is assumed this as the de facto standard, a few words of caution:

- The speed of FRAM access at f_{MCLK} > 8 MHz and the length of wait states is dependent on factors such as temperature and process variation. The length of the wait states can be fixed by using manual wait state control option; however, there is still some possibility for variation in the overall FRAM write speed. While 16 MBps is achievable, it is not guaranteed and there may be a per device variation.

- The data presented above is an average from bench testing a limited number of device samples. The intent was to test the boundaries of FRAM write speed and not to provide a specification for the maximum write speed. The maximum write speed is highly application dependent, based on data and code overhead, block sizes and other system parameters.

3   Conclusion

FRAM is undoubtedly the fastest nonvolatile embedded memory option available today. Being embedded with the ultra-low power MSP430 architecture makes it a perfect choice for applications needing extremely fast write speeds, low power and high endurance. Some of these applications include data logging using remote sensors, energy harvesting applications, and critical response time applications. The key factors that influence the maximum possible FRAM write throughput were discussed and the tradeoffs presented. In analyzing the bench test results for the MSP430FR5739, it is seen that the achievable practical write speed is very close to the theoretical maximum. It is up to the user to determine the available resources and design their application in a way that can achieve the fastest possible write speed for FRAM.

4   References

1. *MSP430FR57xx Family User's Guide* ([SLAU272](#))
2. *MSP430FR58xx, MSP430FR59xx, MSP430FR68xx, and MSP430FR69xx Family User's Guide* ([SLAU367](#))
3. *MSP430FR573x Mixed-Signal Microcontrollers* ([SLAS639](#))
4. *MSP430FR572x Mixed-Signal Microcontrollers* ([SLASE35](#))
5. *MSP430FR59xx Mixed-Signal Microcontrollers* ([SLAS704](#))
6. *MSP430FR58xx Mixed-Signal Microcontrollers* ([SLASE34](#))
7. *MSP430F543xA, MSP430F541xA Mixed-Signal Microcontrollers* ([SLAS655](#))
8. *Migrating From the MSP430F2xx Family to the MSP430FR57xx Family* ([SLAA499](#))
9. *MSP430F22x2, MSP430F22x4 Mixed-Signal Microcontrollers* ([SLAS504](#))

IMPORTANT NOTICE

Products		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		