Application Report
SLAA728–December 2016

Optimizing Active Mode Current Consumption on MSP Devices

Dietmar Walther *Quality*

ABSTRACT

Today's advanced architectures of low-power microcontrollers have created a real challenge for the conscientious engineer. How does one, without an intimate knowledge of each competitor's architecture, discern spin from a true level-set of the expected power consumption? This application report focuses on active mode power consumption when the MSP CPU is executing loops. All other power-saving features like power modes, speed or frequency adjustments, and intelligent use of peripherals are not described in detail. The goal is to introduce the differences between the different MSP families with respect to architecture and memory interface. The focus of this document is to provide short and crisp answers to the question why the current consumption might differ from one code to another without changing obvious functions of the application software.

Contents

List of Figures

Trademarks

1 Introduction

Today's requirements for ultra-low power applications are continuously increasing. Therefore, large efforts during chip development have been made to achieve lower power consumption by using advanced architectures and better process nodes. Besides hardware-related aspects, coding can also play a significant role in achieving low power numbers in the end application. The focus of this paper is on code execution while the CPU is active. Features like low-power modes or utilizing peripherals to save energy are not considered in this application report. As the code development is mainly done by the user of a microcontroller and not the silicon vendor, this document provides some inputs about how MSP devices can be programmed for ultra-low power consumption, especially in active mode. Besides the active mode, also different low power modes are provided to further extend the power budget of MSP based applications.

This document does not deal with all flavors of using different low-power modes, utilizing advanced peripherals, or environmental impact like temperature or supply voltage. It clearly focuses on pure active mode current consumption driven by the CPU and simple code execution.

The following factors stay constant for all the considerations discussed later:

- Only active mode is used without any interrupts.
- No peripherals like ADC, DMA, GPIOs, or LEA are considered or used.
- Supply voltage, temperature, frequency, and subvoltage domains are not considered.

To address the above influential aspects, a much more complex and comprehensive explanation considering all the different device families would be required.

1.1 F1xx, F2xx, and F4xx Families

The MSP430F1xx, F2xx, and F4xx families are flash-based ultra-low-power microcontrollers having a 16-bit memory interface together with a 16-bit CPU. This means for each data or instruction fetch, a memory access takes place. Therefore, a software loop of a defined cycle located at the same flash address always results in the same current consumption. There are two different factors influencing the active mode current consumption on MSP4301xx, 2xx, and 4xx devices:

- The access ratio between RAM and flash memory
- The address location of the executed code influencing the number of toggling address bus bits

The ratio of accesses between RAM and flash has the bigger impact and needs to be considered if active mode (AM) current consumption between devices is compared. The AM current consumption specification that is in the data sheet is defined as a 50/50 ratio. In the end, it is a simple loop that jumps from flash to RAM and back to flash at the standard frequency of approximately 1 MHz. If the ratio is increased for RAM accesses, the current consumption becomes lower, and if accesses for flash are increased, the current consumption becomes higher. However, no details about the RAM-to-flash ratio are documented because the real use cases have a larger number of combinations. Therefore a typical use case of 50/50% was used to characterize and specify the current consumption to reflect a typical use case scenario.

The second influencing factor is the number of toggling address lines, because this defines the number of charging processes inside the device; however, this has a less significant impact to current consumption. For example, if a "while(1)" loop (or "jmp $" loop) is placed at address 0x7FFE. In this case, nearly all address bits toggle, because the MAB is loaded with the next address which is 0x8000. If the "while(1)" loop is placed instead at 0x8000 only one address line toggles when the MAB is loaded with 0x8002.

If a user wants to utilize the whole low-power capabilities of devices of these MSP device families, the above two aspects need to be considered. This can explain how different compiled codes can lead to slightly different current consumption if the executive part is placed at different addresses.

1.2 F5xx and F6xx Families

On the flash-based F5xx and F6xx families, the two factors mentioned in Section 1.1 are valid as well. However, one additional factor comes into play as well. Despite their extended 16-bit based CPU architecture, these devices have a 32-bit flash interface. This means for 16-bit word access to the flash, 32 bits of data are read and stored in the read buffer, which is then accessed by the CPU. Customers looking to recreate the active current consumption measurements (for example, in the MSP430F5438A data sheet) may be tempted to compile the following, seemingly correct application test case for the F5xx active mode current consumption: "while(1)" loop at 8 MHz. With the data sheet at their side, they would observe one of the following scenarios:

- They measure a value that is much less than 230 µA/MHz and may attribute the delta to a loose specification or a good lot of silicon.
- They measure a value that is much greater than the documented 230 µA/MHz and may label the MSP430™ MCU as violating the specification.

In either case, the customers face an inaccurate representation of the 5xx core's capabilities in active mode. This would be mainly due to the use of the while(1){ } loop, which is and should be recognized as a deprecated method of testing active current consumption.

The reason why the "while(1)" loop is an insufficient test has to do with the method in which the 5xx accesses flash memory, 32 bits at a time. An empty "while(1)" loop compiles into the assembly instruction "jmp $" which is a 16-bit instruction (0x3FFF). "jmp $" tells the CPU to spin without executing anything, and the alignment of this instruction can make a big difference on the measured current consumption, making this not a meaningful test case.

The following paragraphs gives some insight into the details for seeing a different current consumption using different alignment for the "jmp $" loop:

- If the "jmp $" is the first instruction of the 32-bit fetch, the Program Counter (which auto-increments to the address of the next instruction) auto-increments within the data that has already been fetched to the read buffer, and the CPU recognizes there is no need to make an additional fetch from flash. The measured current consumption reflects only the CPU spinning in a loop doing nothing resulting in a value much less than 230 µA/MHz.
- If the "jmp $" is the second instruction of the 32-bit fetch, then the Program Counter falls outside of the data that has already been fetched into the read buffer, so the CPU core continually fetches the next 32 bits from flash as it executes the instruction. The measured current consumption reflects the CPU spinning in a loop doing nothing, but accessing flash every time resulting in a value much greater than 230 µA/MHz.
- The address where the "jmp $" instruction is executed plays a significant role with respect to the changing MAB bits that toggle during the execution. If the 16-bit instruction is located at 0x7FFE, all bits toggle during the transition to 0x8000, which leads to an increased current consumption.
- Also the data stored at the address after the "jmp $" instruction influences the current consumption because of the toggling MDB. Here the worst case would be 0xC000 which is the opposite of 0x3FFF representing the "jmp $" instruction.

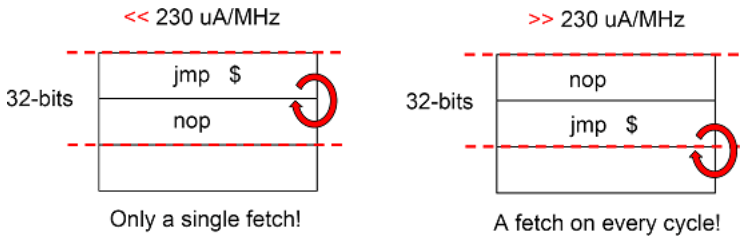

Figure 1. 32-Bit Flash Fetch and JMP $ Alignment

Any real-world application example falls somewhere between zero fetches and a fetch for every instruction executed. Hence a piece of code has been generated that more closely reflects the expected behavior of the core using the Code Composer Studio™ IDE assembly format (see Figure 2). This code example mixes different instruction formats, address modes, and types to establish a code base that is representative of the F5xx core's capabilities. This is also the code example that is used to measure current consumption for the quoted data sheet parameters in the MSP430F5xx devices, including the integrated USB and RF families like the F552x and CC430.

```
                .global  ACTIVE_MODE_TEST

                .text

ACTIVE_MODE_TEST
                MOV     #0x2000, R4
                MOV     #0x4, 0(R4)
                MOV     &0x2000, &0x2002
                ADD     @R4, 2(R4)
                SWPB    @R4+
                MOV     @R4, R5
IDD_AM_L1
                XOR     @R4+, &0x2020
                DEC     R5
                JNZ     IDD_AM_L1
                JMP     ACTIVE_MODE_TEST

                .end
```

- **Code mixes instruction formats**
  - Type I, Type II, JMP, Emulated
- **Code mixes address modes**
- **Types of Instructions**
  - ALU: 60%
  - Data Read: 40%
  - Data Write: 40%
  - Control/Jump: 26%

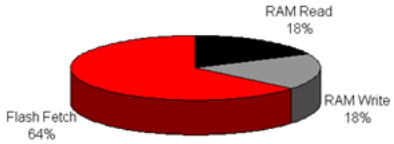

Figure 2. ASM Test Case for Active Mode Consumption

The above explained principle using a simple "while(1)" loop can be extended to slightly more complex loops. One example would be simple wait loops that are often used during standard C code development. On these loops, the 32-bit alignment plays a significant role as well. Even if the impact to the current consumption is not as large as compared to the "while(1)" loop, it is significant enough to be considered by the developer. As an example, a simple blinking LED loop with two delay loops is used. Figure 3 shows that the LED toggle loop consists of two delay elements. The first loop of 1-s delay at the beginning, and a second loop of 0.5-s delay in the LED toggle loop that is executed four times. The assumption is that the current consumption during the execution of each of the loops is identical. But this is only true if the alignment of these loops is done according to the 32-bit flash interface implemented in F5xx and F6xx devices.

```
void leftLEDBlink() {
    int i;
    __delay_cycles(1000000); // ~1 second delay
    for(i=0;i<4;i++)
    {
        __delay_cycles(500000); // ~0.5 second delay
        P1OUT^=(BIT0);    // toggle the led
    }
}
```

Figure 3. Simple LED Blink Test Loop

For better understanding, it is important to look at the output of the compiler to see where the code is placed and how the compiler converted the C code into assembly code. From a translation point both loops are looking identical with respect to instruction set, looking to the assembly code in Figure 4.

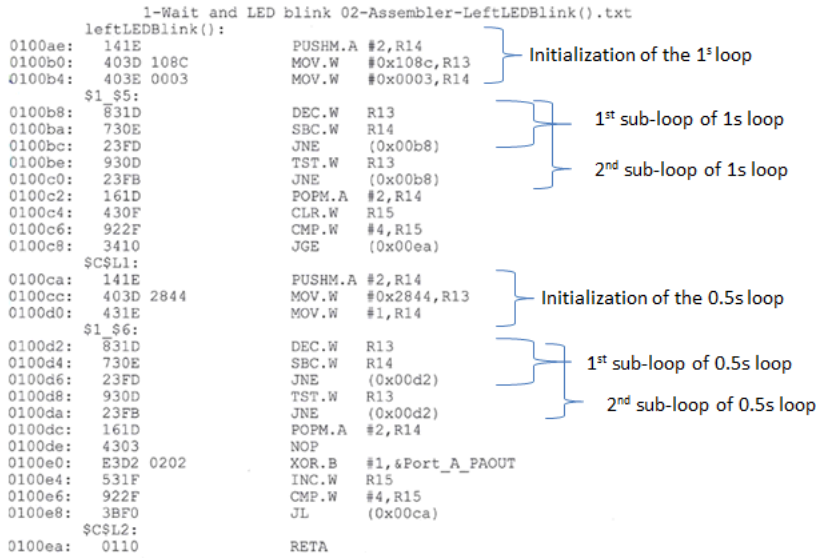

Figure 4. Assembly of Simple LED Blink Test Loop

Considering the placement inside the memory, Figure 5 shows that the 1-s loop (first bold marked data) is aligned to 32 bit, while the 0.5-s loop (second bold marked data) is not. Each green or blue colored block represents a new access to the flash due to the implemented 32-bit read buffer sourcing the 16 bit CPU. Both of the loops consist of 5 × 16-bit words which always means that the flash needs to be accessed at least three times, because of the 32-bit read buffer between flash and CPU. The critical point is at which part of the loops the flash gets most accessed. The device architecture, in combination with the placement of the loops inside the flash memory, requires one additional flash access cycle (four times in total) in the 0.5-s delay loop, resulting in a higher current consumption.

```
@100A0
xx xx xx xx xx xx xx xx xx xx xx xx xx xx 1E 14
3D 40 8C 10 3E 40 03 00 1D 83 0E 73 FD 23 0D 93
FB 23 1D 16 0F 43 2F 92 10 34 1E 14 3D 40 44 28
1E 43 1D 83 0E 73 FD 23 0D 93 FB 23 1D 16 03 43
D2 E3 02 02 1F 53 2F 92 F0 3B 10 01 xx xx xx xx
q
```

Figure 5. Placement of the Delay Loops in the Memory Map

This results in a high current consumption of the 0.5-s loop. Figure 6 shows the dynamic current consumption measured over a 1-Ω shunt resistor at DVCC = 3.0 V.

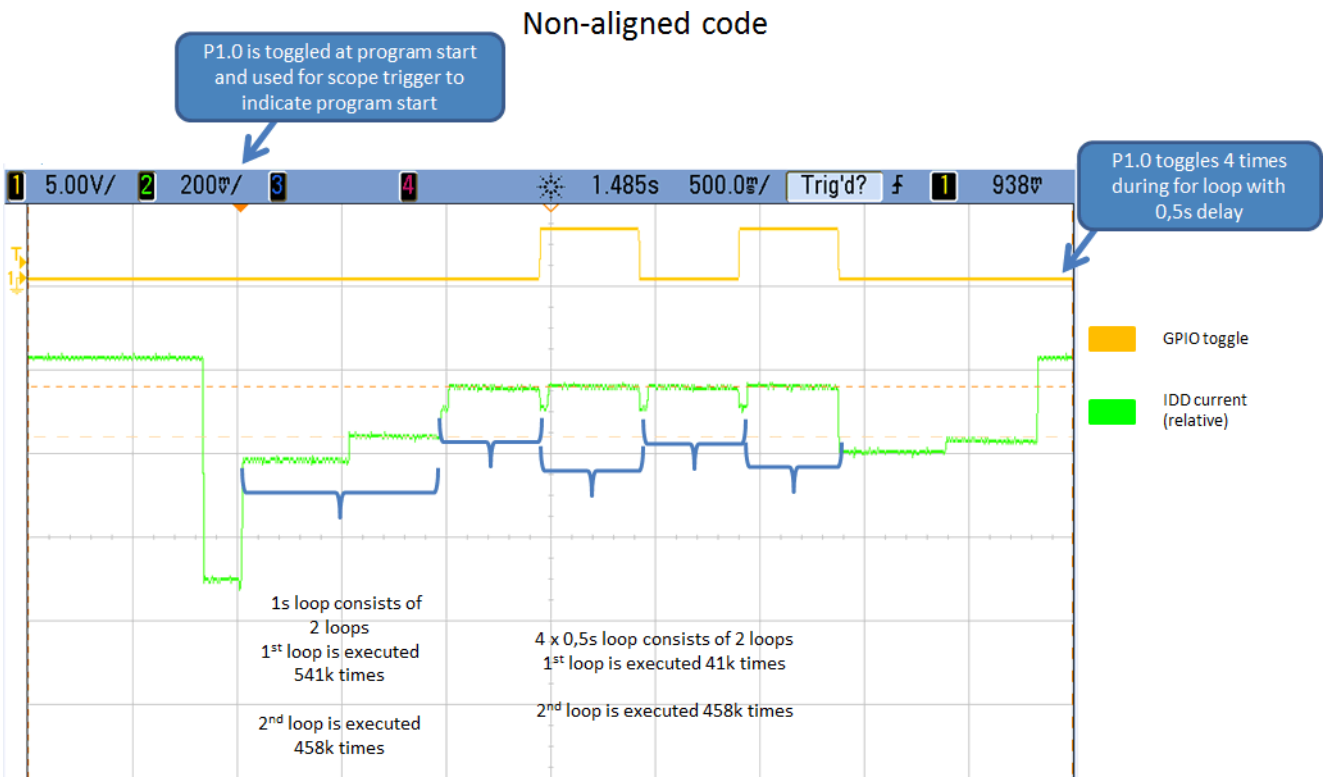

Figure 6. Dynamic Current Consumption for Nonaligned Loops

If the loop is "manually" aligned by simply inserting a "nop" (no operation instruction) inside the code, the current consumption is identical for both delay loops (see Figure 7).

```
            ORG     0100AEh
test_loop
            PUSHM.A     #2,R14
            MOV.W       #0x108c,R13
            MOV.W       #0x0003,R14

loop1s
            DEC.W       R13
            SBC.W       R14
            JNE         loop1s
            TST.W       R13
            JNE         loop1s
            POPM.A      #2,R14
            CLR.W       R15
            CMP.W       #4,R15
            JGE         end_loop

for_loop
            PUSHM.A     #2,R14
            MOV.W       #0x2844,R13
            MOV.W       #1,R14
            nop                     ; align code to 32 bit

loop0_5s
            DEC.W       R13
            SBC.W       R14
            JNE         loop0_5s
            TST.W       R13
            JNE         loop0_5s
            POPM.A      #2,R14
            NOP
            XOR.B       #1,&P1OUT
            INC.W       R15
            CMP.W       #4,R15
            JL          for_loop

            PUSHM.A     #2,R14
            MOV.W       #0x108c,R13
            MOV.W       #0x0003,R14
```

```
@100A0
xx xx xx xx xx xx xx xx xx xx xx xx xx xx 1E 14
3D 40 8C 10 3E 40 03 00 1D 83 0E 73 FD 23 0D 93
FB 23 1D 16 0F 43 2F 92 10 34 1E 14 3D 40 44 28
1E 43 03 43 1D 83 0E 73 FD 23 0D 93 FB 23 1D 16
03 43 D2 E3 02 02 1F 53 2F 92 EF 3B 1E 14 3D 40
8C 10 3E 40 03 00 1D 83 0E 73 FD 23 xx xx xx xx
q
```

Figure 7. Manual Code Alignment to 32 Bit

This results in a similar current consumption of the 1-s delay loop and the 0.5-s delay loop (see Figure 8).

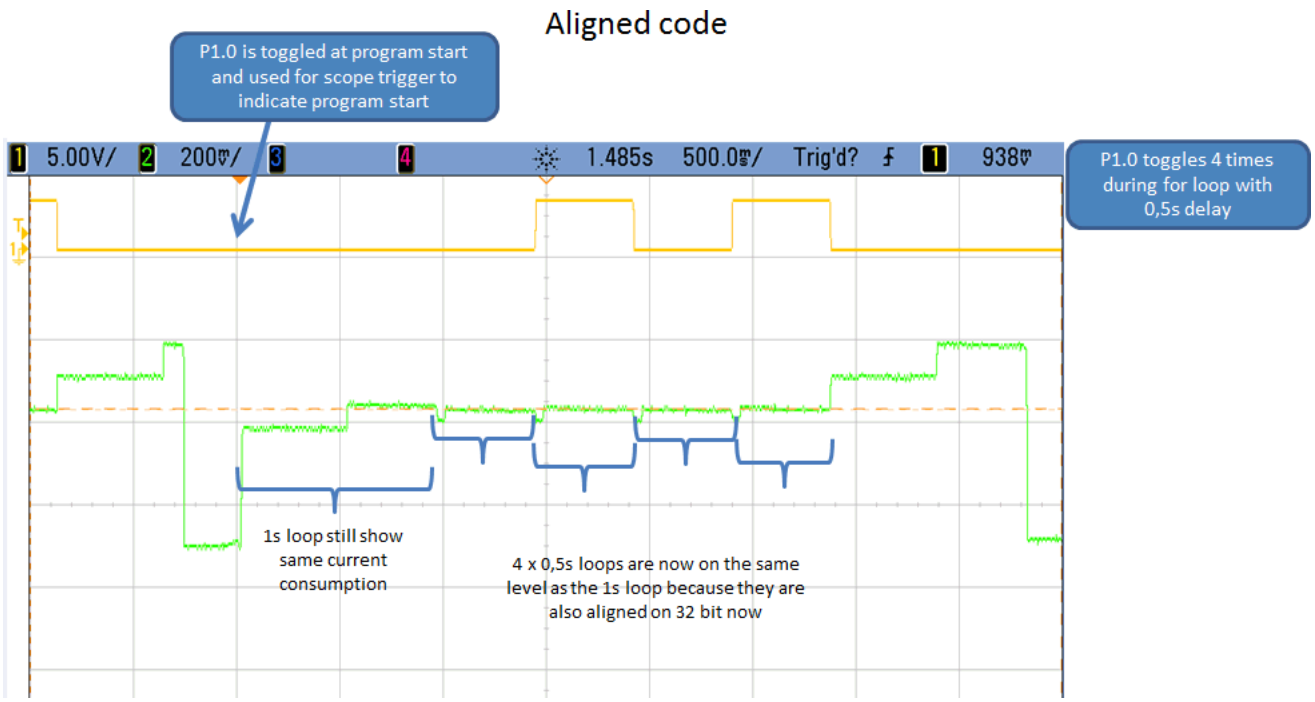

Figure 8. Manual Code Alignment to 32 Bit

1.3 FR2xx, FR4xx, FR5xx, and FR6xx Families

The FRxx family extends the read buffer concept already used on the F5xx and F6xx family and introduces 4 cache lines of 64-bit size, which act as an interface between FRAM memory and CPU. The cache is implemented in the FRAM controller which contains two cache sets. Each of these cache sets contains two lines that are preloaded with four words (64 bits) during one access cycle. An intelligent logic selects one of the cache lines to preload FRAM data and preserves recently accessed data in the other cache. If one of the four words stored in one of the cache lines is requested (a cache hit), no FRAM access occurs; instead, a cache request occurs. No wait state is needed for a cache request, and the data is accessed with full system speed. However, if none of the words that are available in the cache are requested (a cache miss), the wait state controls the CPU to ensure proper FRAM. This implemented in case the system clock frequency for the CPU or DMA may exceed the FRAM access and cycle time requirements.

This architecture also has a significant impact on current consumption. If the cache hit rate is high, the current consumption is lower, because the memory itself does not need to be accessed. Current consumption numbers are specified in the data sheet. Figure 9 shows an extract of the MSP430FR5969 data sheet. If the deprecated method of testing an active mode current ("while (1)" or "jmp $") is applied on this architecture, the principles described in the previous chapters do not apply to 100% accurate anymore. If the "jmp $" instruction is located at the end of a cache buffer line, the current consumption is higher only the first time it is executed, because the content is loaded to a second cache line. Afterward the "while (1)" loop is completely executed out of cache, saving power. Therefore the "while (1)" or "jmp $" instruction is not used to characterize and specify the current consumption in the MSP430 data sheets.

5.4 Active Mode Supply Current Into V_{CC} Excluding External Current

over recommended operating free-air temperature (unless otherwise noted)(1) (2)

PARAMETER	EXECUTION MEMORY	V_{CC}	FREQUENCY ($f_{MCLK} = f_{SMCLK}$)										UNIT
			1 MHz 0 wait states (NWAITSx = 0)		4 MHz 0 wait states (NWAITSx = 0)		8 MHz 0 wait states (NWAITSx = 0)		12 MHz 1 wait states (NWAITSx = 1)		16 MHz 1 wait states (NWAITSx = 1)		
			TYP	MAX	TYP	MAX	TYP	MAX	TYP	MAX	TYP	MAX	
$I_{AM,\ FRAM\_UNI}$ (Unified memory)(3)	FRAM	3.0 V	210		640		1220		1475		1845		µA
$I_{AM,\ FRAM}$(0%) (4) (5)	FRAM 0% cache hit ratio	3.0 V	370		1280		2510		2080		2650		µA
$I_{AM,\ FRAM}$(50%) (4) (5)	FRAM 50% cache hit ratio	3.0 V	240		745		1440		1575		1990		µA
$I_{AM,\ FRAM}$(66%) (4) (5)	FRAM 66% cache hit ratio	3.0 V	200		560		1070		1300		1620		µA
$I_{AM,\ FRAM}$(75%) (4) (5)	FRAM 75% cache hit ratio	3.0 V	170	255	480		890	1085	1155	1310	1420	1620	µA
$I_{AM,\ FRAM}$(100% (4) (5)	FRAM 100% cache hit ratio	3.0 V	110		235		420		640		730		µA
$I_{AM,\ RAM}$ (6)	RAM	3.0 V	130		320		585		890		1070		µA
$I_{AM,\ RAMonly}$ (7) (5)	RAM	3.0 V	100	180	290		555		860		1040	1300	µA

(1) All inputs are tied to 0 V or to V_{CC}. Outputs do not source or sink any current.
(2) Characterized with program executing typical data processing.
f_{ACLK} = 32768 Hz, f_{MCLK} = f_{SMCLK} = f_{DCO} at specified frequency, except for 12 MHz. For 12 MHz, f_{DCO} = 24 MHz and f_{MCLK} = f_{SMCLK} = f_{DCO}/2.
At MCLK frequencies above 8 MHz, the FRAM requires wait states. When wait states are required, the effective MCLK frequency ($f_{MCLK,eff}$) decreases. The effective MCLK frequency also depends on the cache hit ratio. SMCLK is not affected by the number of wait states or the cache hit ratio.
The following equation can be used to compute $f_{MCLK,eff}$:
$f_{MCLK,eff} = f_{MCLK} / [\text{wait states} \times (1 - \text{cache hit ratio}) + 1]$
For example, with 1 wait state and 75% cache hit ratio $f_{MCLK,eff} = f_{MCLK} / [1 \times (1 - 0.75) + 1] = f_{MCLK} / 1.25$.
(3) Represents typical program execution. Program and data reside entirely in FRAM. All execution is from FRAM.
(4) Program resides in FRAM. Data resides in SRAM. Average current dissipation varies with cache hit-to-miss ratio as specified. Cache hit ratio represents number cache accesses divided by the total number of FRAM accesses. For example, a 75% ratio implies three of every four accesses is from cache, and the remaining are FRAM accesses.
(5) See Figure 5-1 for typical curves. Each characteristic equation shown in the graph is computed using the least squares method for best linear fit using the typical data shown in Section 5.4.
(6) Program and data reside entirely in RAM. All execution is from RAM.
(7) Program and data reside entirely in RAM. All execution is from RAM. FRAM is off.

Figure 9. Current Consumption Specification MSP430FR5969

So the implementation of the cache architecture of 4 × 64 bit would also help to decrease power consumption of larger loops like the delay loops described in Section 1.2. Nevertheless, it is recommended to consider the code alignment with respect to the cache architecture to achieve the highest cache hit ratio. By doing this you get advantages not only in power consumption but also in speed, because the device can operate without wait states at higher speed. This becomes true if the code is written in a size matching the cache size; for example, small delay loops that can be completely executed from the cache.

1.4 P4xx (MSP432™) Family

The MSP432 family is based on a ARM® Cortex®-M4F CPU which is one of the multiple main differentiators compared to the 16 bit MSP430 families mentioned before. The devices of the P4xx family have a read buffering concept implemented as well. This read buffer has a width of 128 bits, which are read once from the flash regardless of the access size of 8, 16, or 32 bits. The 128-bit data and its associated address is internally buffered by the flash controller. Therefore subsequent accesses (expected to be contiguous in nature) within the same 128-bit address boundary are serviced by the buffer, which is configurable from user's perspective (used or not used).

Therefore the real flash hit ratio plays a significant role as well, and loops contained within the 128-bit read buffer should be aligned accordingly to achieve lowest power.

Because the flash-to-SRAM hit ratio still has an influence, the specification of the active mode current was divided into pure flash or SRAM execution. This gives the customer better data on how to design the application. Figure 10 shows an example how the differences for a simple while(1) program looks like if the device is operated with the LDO out of flash or SRAM.

5.16 Typical Characteristics of Active Mode Currents for While(1) Program

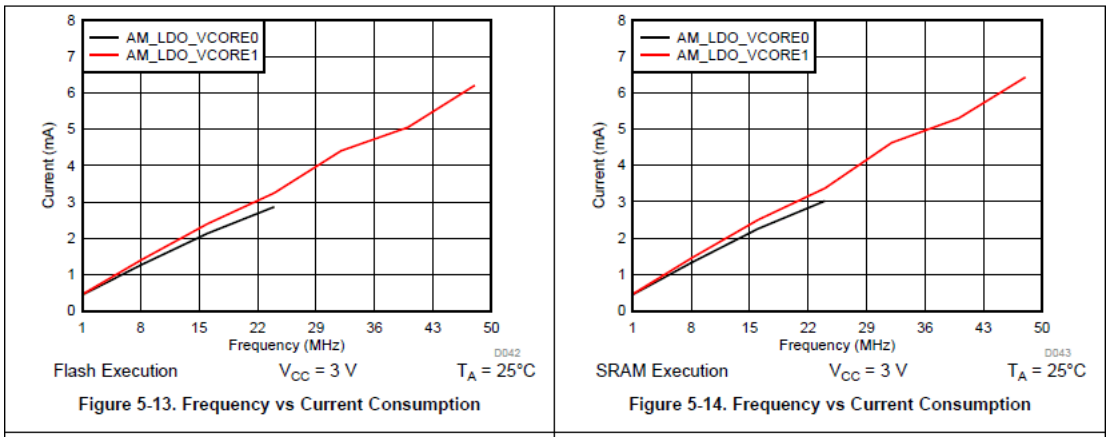

Figure 10. Current Consumption Specification MSP432P401

In addition, more focus was put on providing active mode current consumption for different typical and comparable code execution profiles like CoreMark®, prime number, or Fibonacci test programs.

2 MSP430 Compiler --align_for_power Option

An alternative approach is to utilize the compiler to let it take care about the code alignment also with respect to the placement of a standard "while (1)" loop. The TI MSP430 compiler v16.9.0.LTC delivered with the Code Composer Studio IDE provides an option called "--align_for_power". This new --align_for_power compiler option enables power savings by aligning all functions and loops to 4-byte boundaries. When this option is specified, power savings are achieved if a small function or loop aligns to the 32-bit buffer for fetching code from flash. However, there is less benefit for larger functions and loops. See the *MSP430 Optimizing C/C++ Compiler User's Guide* for more details.

For the F5xx and F6xx family, the option --align_for_power used in the compiler make sure that "while(1)" loops are always placed in the first word of a 32-bit read buffer. This ensures that the execution of the "while(1)" is always done from the 32-bit read buffer without additional flash access cycles. Figure 11 shows the differences in the machine code of specific bench test code and how the alignment works and that the "while (1)" loop (machine code = 0xFF3F) is always placed at the first word of the 32-bit read buffer, because it is aligned to 4 bytes.

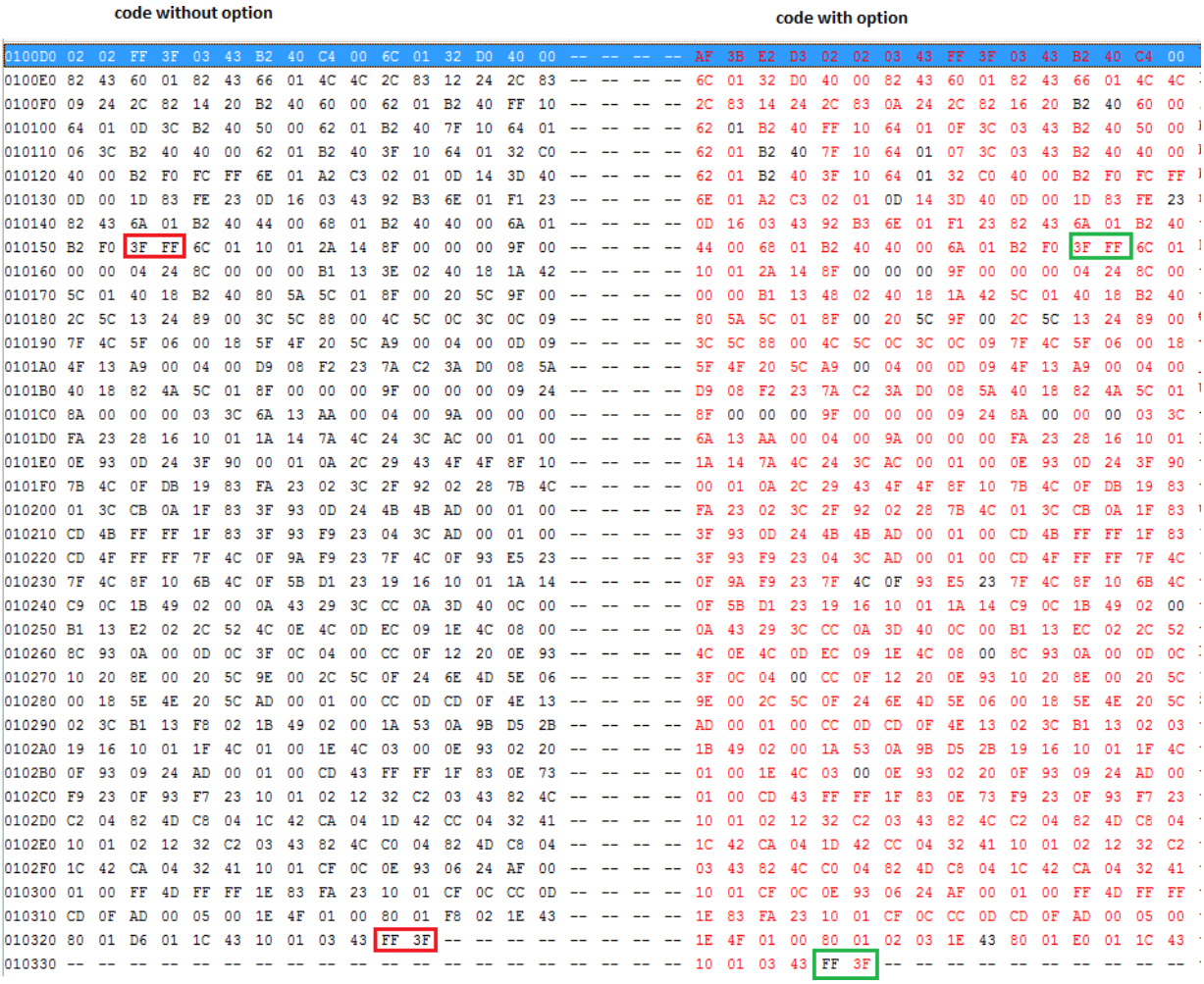

Figure 11. Code Alignment Using TI Compiler Option ""--align_for_power--"

3 Conclusion

This paper proves that thanks to progress in ultra-low power microcontroller technology, such as different memory (flash or FRAM) access mechanisms, a comparison of low-power using yesterday's techniques – like the "while(1)" loop or by comparing low-power mode numbers – might be misleading. The ultra-low power challenge must be answered from a system-level perspective, and tested with use cases that leverage the capabilities of the technology in question. This application report provides a technical analysis of the MSP430 and MSP432 current consumption messaging to provide the appropriate framework for the interpretation and recreation of the data sheet specifications. It provides an overview over the different implementations across all MSP families and clearly shows that more advanced techniques are implemented on chip level to achieve lowest power consumption. It also clarifies that current consumption values in the data sheet represent typical values for common application scenarios.

4 References

1. *MSP430FR58xx, MSP430FR59xx, MSP430FR68xx, and MSP430FR69xx Family User's Guide*
2. *MSP430FR59xx Mixed-Signal Microcontrollers*
3. *MSP432P4xx Family Technical Reference Manual*
4. *MSP432P401R, MSP432P401M Mixed-Signal Microcontrollers*
5. TI E2E™ Community thread: MSP430 Power Consumption

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products (also referred to herein as “components”) are sold subject to TI’s terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its components to the specifications applicable at the time of sale, in accordance with the warranty in TI’s terms and conditions of sale of semiconductor products. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

TI assumes no liability for applications assistance or the design of Buyers’ products. Buyers are responsible for their products and applications using TI components. To minimize the risks associated with Buyers’ products and applications, Buyers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI components 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 such information 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.

Reproduction of significant portions of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI components or services with statements different from or beyond the parameters stated by TI for that component or service voids all express and any implied warranties for the associated TI component or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards which anticipate dangerous consequences of failures, monitor failures and their consequences, lessen the likelihood of failures that might cause harm and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI’s goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed a special agreement specifically governing such use.

Only those TI components which TI has specifically designated as military grade or “enhanced plastic” are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components which have ***not*** been so designated is solely at the Buyer's risk, and that Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

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		

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