Application Report
SLAA698B–March 2016–Revised November 2016

Benchmarking the Signal Processing Capabilities of the Low-Energy Accelerator on MSP430™ MCUs

Evan Wakefield *MSP Applications*

ABSTRACT

This application report covers the performance of the low-energy accelerator (LEA) in advanced signal processing while maintaining the ultra-low-power consumption on a 16-bit MSP430™ FRAM microcontroller (MCU). The LEA module is compared against the performance of optimized software-enabled signal processing code running on both the 16-bit MSP430 MCUs and a competitor ARM® Cortex®-M0+ based MCU. This application report describes how the LEA module can perform signal processing algorithms such as Fast Fourier Transform (FFT) and Finite Impulse Response (FIR) efficiently, allowing for new possibilities in real-time signal processing for 16-bit MCUs used in a variety of fields such as metering, process control, sensing, and monitoring. Project collateral discussed in this application report can be downloaded from the following URL: www.ti.com/lit/zip/SLAA698.

Contents

List of Figures

List of Tables

(1) MSP430, Code Composer Studio, E2E are trademarks of Texas Instruments.
(2) ARM, Cortex are registered trademarks of ARM Ltd.
(3) IAR Embedded Workbench is a registered trademark of IAR Systems.
(4) All other trademarks are the property of their respective owners.

1 Introduction

The MSP430FR5994 microcontroller features the low-energy accelerator (LEA) for signal conditioning that performs vector math operations efficiently and with minimal energy. This low-energy accelerator (LEA) performs complex operations such as FFT, FIR, and matrix multiplication thus enabling more signal processing capabilities to the MCU market. This extends the amount of time applications spend in low-power modes, conserving energy at an application level, while improving signal processing performance. This application report describes the capabilities of the LEA module and benchmarks the 16-bit MSP430FR5994 MCU against a 32-bit ARM Cortex-M0+ MCU.

2 Fast Fourier Transform (FFT) and Finite Impulse Response (FIR)

FFTs compute the discrete Fourier transform (DFT) of a vector. Fourier analysis is frequently used in digital signal processing to convert a signal from the time domain to the frequency domain or the other direction. FFT is often used in various applications such as metering, process control, sensing, and audio processing to analyze events happening in the time domain and react accordingly in real time to an event.

FIR is a type of filter used in signal processing whose impulse response settles to zero in a finite amount of time, having no feedback. FIR filters can be set up to condition signals in a variety of ways by creating low-pass, band-pass, and high-pass type filters with a variety of variables to adjust in each.

3 Low-Energy Accelerator (LEA) for Signal Processing

The LEA is a 32-bit hardware engine designed for operations that involve vector-based signal processing, such as FIR, IIR, and FFT, without CPU intervention and triggers an interrupt when the operation is completed. The LEA supports multiple commands, which are issued by CPU. The LEA offers incomparable performance and energy consumption when performing vector-based digital signal processing computations. The LEA begins executing the selected operation when the CPU writes a LEA command to the LEA command register through the peripheral bus when the LEA is in idle mode. Before writing the command, the CPU must configure the LEA argument registers with pointers to the parameter blocks for the designated operation. The LEA performs the operation without any CPU intervention and triggers an interrupt when the operation is complete. This application report focuses on how the LEA module can perform FFT and FIR functions efficiently. The MSP430FR5994 MCU has a total of 8KB of SRAM, and 4KB is shared with the LEA module for all data input, output, and parameters. For more information on the LEA, see the *MSP430FR58xx, MSP430FR59xx, MSP430FR68xx, and MSP430FR69xx Family User's Guide*.

It is not necessary to understand how the LEA works or the details of the LEA registers. The Digital Signal Processing (DSP) Library for MSP Microcontrollers offers easy-to-use APIs that use the functions of the LEA and provide a high-level environment to use the LEA in various applications. The DSP Library is a set of highly optimized API functions to perform many common signal processing operations on fixed-point numbers for MSP430 microcontrollers. The APIs automatically enable and use the LEA module if the LEA is available in the target device, and apply the optimal configurations to the LEA registers with the correct sequence. If the LEA is not available, the CPU is selected to perform the operations.

Before using the DSP Library for MSP MCUs APIs, the application must first specify the input and output memory locations by allocating the array that must reside in the shared 4KB of LEA SRAM memory. For example, to perform 256-point complex FFT with the LEA, the data input array consists of 256-word real and 256-word complex values which totals 512 words (1024 bytes).

The full LEA commands are supported by the DSP Library APIs, which are listed in the *DSP Library API Guide*, *Low-Energy Accelerator (LEA) Frequently Asked Questions (FAQ)*, and *Benchmarking the Signal Processing Capabilities of the Low-Energy Accelerator on MSP430™ MCUs*.

The following sequence of operations is an example of performing a vector-based algorithm using the LEA, DMA, ADC, and SPI.

1. The CPU sets up the DMA controller, ADC, and SPI.
2. A DMA channel collects samples from the ADC converter at a defined sampling rate and transfers data to the LEA memory.
3. After a block of data has been collected, the CPU enables one or a series of operations of the LEA using the APIs in the Digital Signal Processing (DSP) Library for MSP Microcontrollers to perform the required algorithm (for example, FIR, IIR, correlation, or FFT).
4. When the algorithm is complete, another DMA channel transfers the result of that algorithm to the SPI.
5. The SPI transfers the data to an external device.

In this application, the LEA module is used on the MSP430FR5994 MCU. See Section 5.4 for more information on the MSP430FR5994 MCU.

4 Hardware

This application report compares the performance of two devices that are computing both complex FFT and basic FIR filtering calculations. The comparison includes the number of clock cycles used and energy consumption while doing the FFT and FIR filtering by using a device variant of the MSP430FR5994 MCU without the LEA module, and then the LEA module is enabled by using the MSP430FR5994 MCU to show the performance improvement. Then there is an investigation into the performance advantages of MSP430FR5994 MCU with the LEA compared to a 32-bit ARM Cortex-M0+ based MCU while doing the same FFT and FIR operations.

4.1 MSP430FR5994 Microcontroller

The MSP430FR5994 MCU features 256KB of embedded ferroelectric random access memory (FRAM), a nonvolatile memory known for its ultra-low power, high endurance, and high-speed write access. The MCU also features 8KB of SRAM, supports CPU speeds up to 16 MHz and integrated ADC, timers, AES encryption, LEA, and more.

For this benchmark, an MSP-EXP430FR5994 LaunchPad™ development kit was used to test the performance advantages of the LEA module while computing a variety of complex FFTs and FIR filtering. For more information on the development kit, visit the MSP-EXP430FR5994 LaunchPad development kit tool page.

4.2 ARM® Cortex®-M0+ Based Microcontroller

The ARM Cortex M0+ is a 32-bit RISC-style MCU that is an optimized subset of the Cortex-M0 architecture. The Cortex-M0+ architecture is one of the primary ARM cores being used for low-power operations. This application report describes benchmarking with a Cortex-M0+ based MCU run at 12 MHz with a 3.0-V supply. If there are specific questions regarding the set up used for the ARM Cortex-M0+ based MCU that are not covered in this application report, contact your local TI sales team.

4.3 Power Profiling

To gather accurate DC power use of each device during operation, a Keysight N6705B DC Power Analyzer was used. The MSP-EXP430FR5994 LaunchPad development kit and the ARM Cortex-M0+ based MCU were both powered at 3.0 V.

5 Benchmark Test Setup

Although the source files can be viewed with any text editor, more can be done with the projects when opened with a development environment like TI Code Composer Studio™ integrated development environment (IDE), IAR Embedded Workbench® IDE, or others. The benchmark for the LEA module on the MSP430FR5994 MCU is implemented in Code Composer Studio IDE. The benchmark for the ARM devices is implemented in IAR Embedded Workbench.

All projects and documents that are covered in this document can be found at http://www.ti.com/lit/zip/slaa698.

5.1 Code Composer Studio™ IDE Setup for MSP430FR5994 MCU

Code Composer Studio Desktop is a professional integrated development environment that supports TI's microcontroller and embedded processors portfolio. Code Composer Studio IDE comprises a suite of tools used to develop and debug embedded applications. It includes an optimizing C/C++ compiler, source code editor, project build environment, debugger, profiler, and many other features.

You can learn more about Code Composer Studio IDE and download it at http://www.ti.com/tool/ccstudio. For the MSP430FR5994 device, Code Composer Studio IDE v6.1.3 or higher is required. This application report was written and tested using Code Composer Studio IDE v6.2. When Code Composer Studio IDE has been launched, and a workspace directory chosen, use Project→Import Existing Code Composer Studio IDE Eclipse Project. Direct it to [Download Location]\SLAA698\SLAA698\ and click OK.

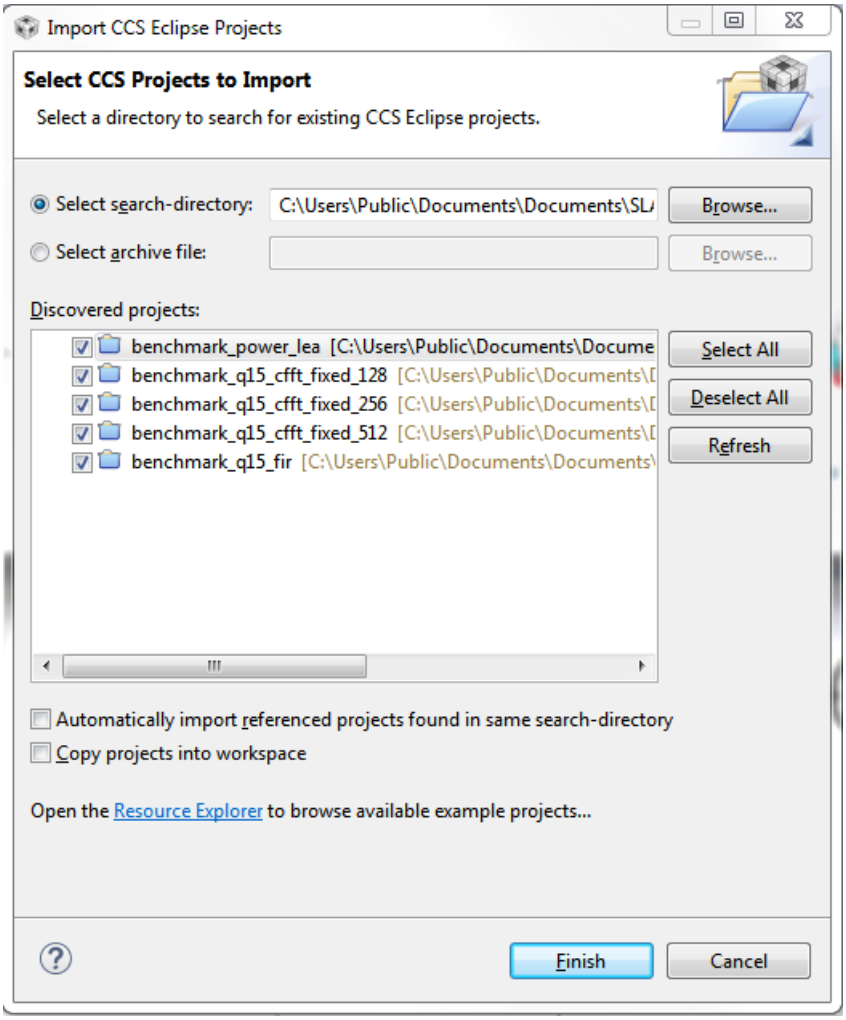

Figure 1. Directing the Project→Import Function to the Demo Project

When you click OK, Code Composer Studio IDE should recognize the project and allow you to import it. The indication that Code Composer Studio IDE has found it is that the project appears in the box shown in Figure 2, and it has a checkmark to the left of it.

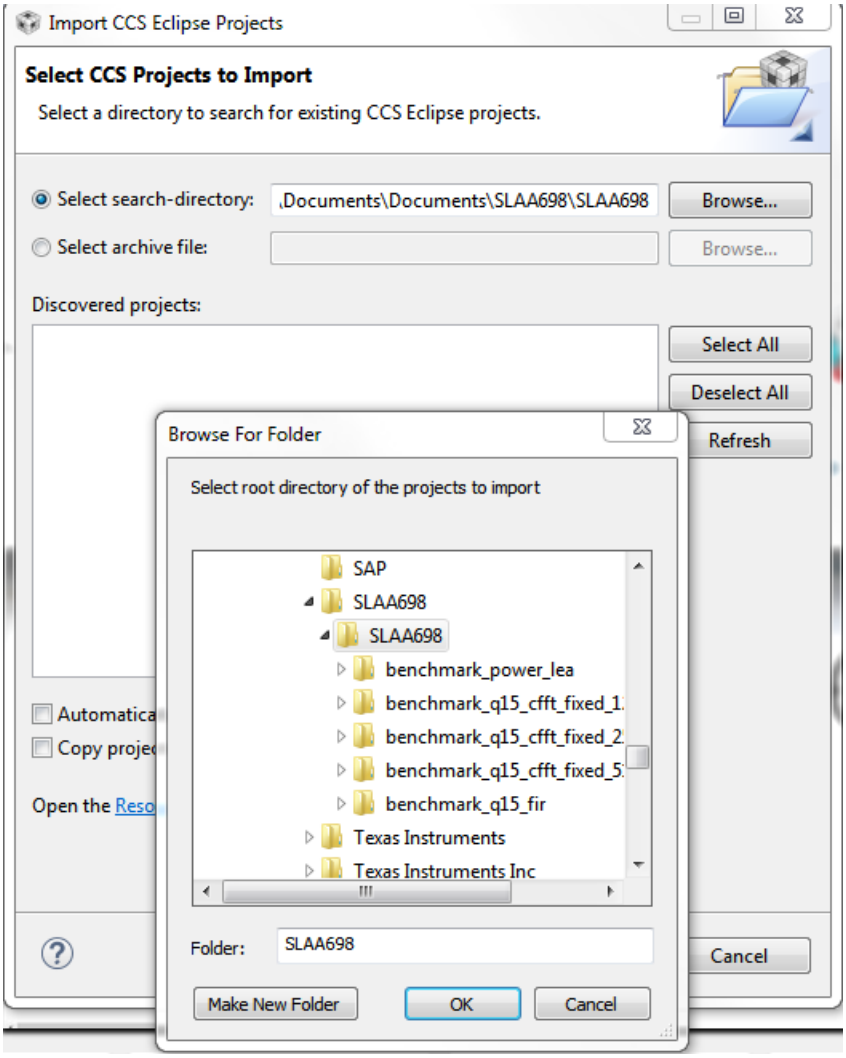

Figure 2. Code Composer Studio IDE Project Import

Sometimes Code Composer Studio IDE finds the project but does not show a checkmark; this might mean that the workspace already has a project by that name. Resolve this by renaming or deleting that project. (Even if you do not see it in the Code Composer Studio IDE workspace, be sure to check the workspace directory on the file system.)

To build the desired project, first click the project to build, then click Project→Build project.

Right click a project and go to Properties, and look for Build>MSP430 Compiler>Preprocessor to see how to alter the data and code memory model options for the purpose of this application report. These settings are investigated to understand how they impact the cycle count results of the LEA module.

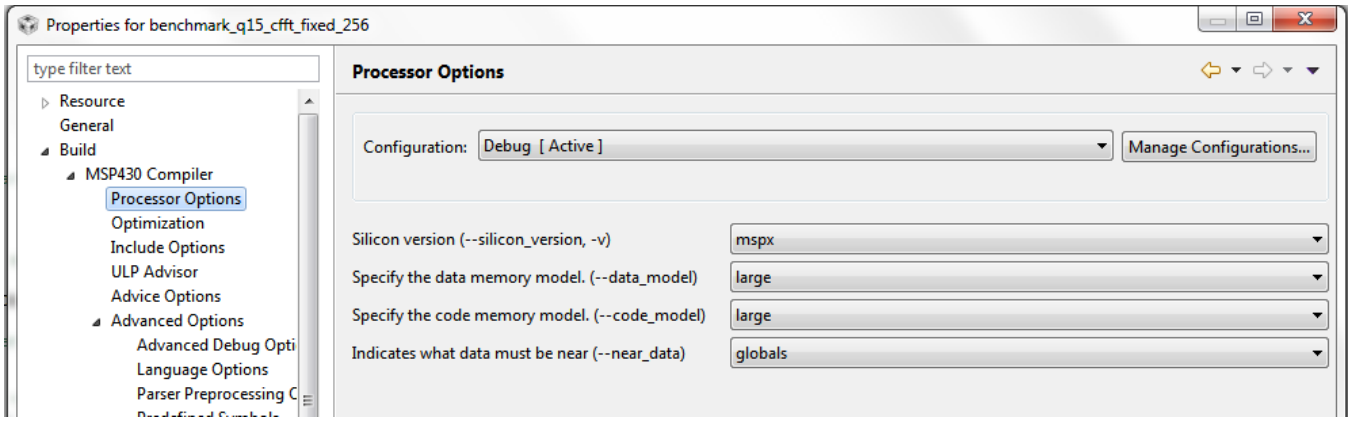

Figure 3. Processor Options

Right click a project and go to Properties, and look for Build>MSP430 Compiler>Optimization to enable "Whole Program Optimizations" for the optimization level and select option 5 to focus towards speed.

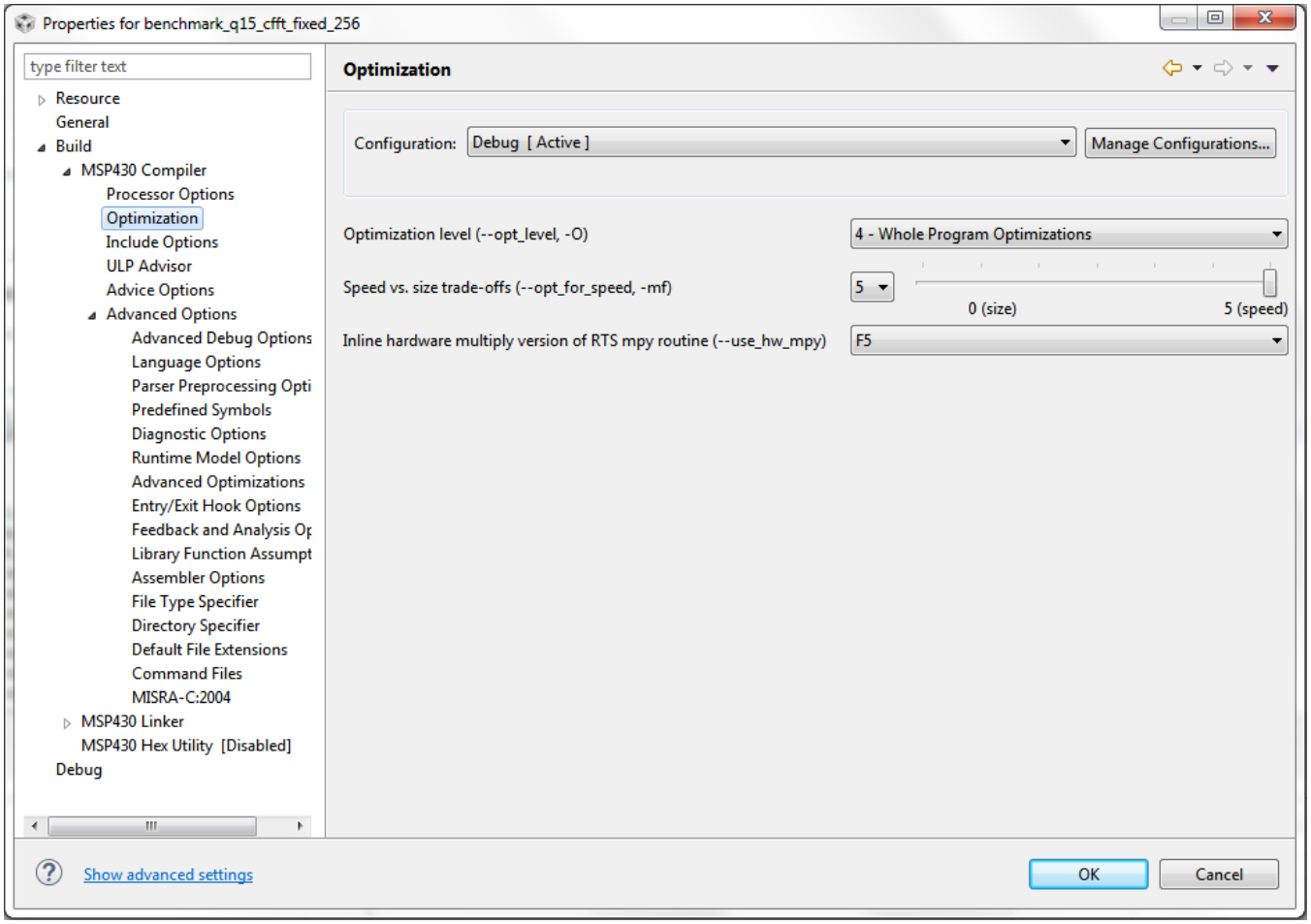

Figure 4. Optimization

To set up the predefined symbols that are used for further optimizing the MSP DSP Library and enabling a few benchmarking options, go to Build>MSP430 Compiler>Advanced Options>Predefined Symbols and use the button with a green plus symbol under "Pre-define NAME" to add other symbols (see Figure 5).

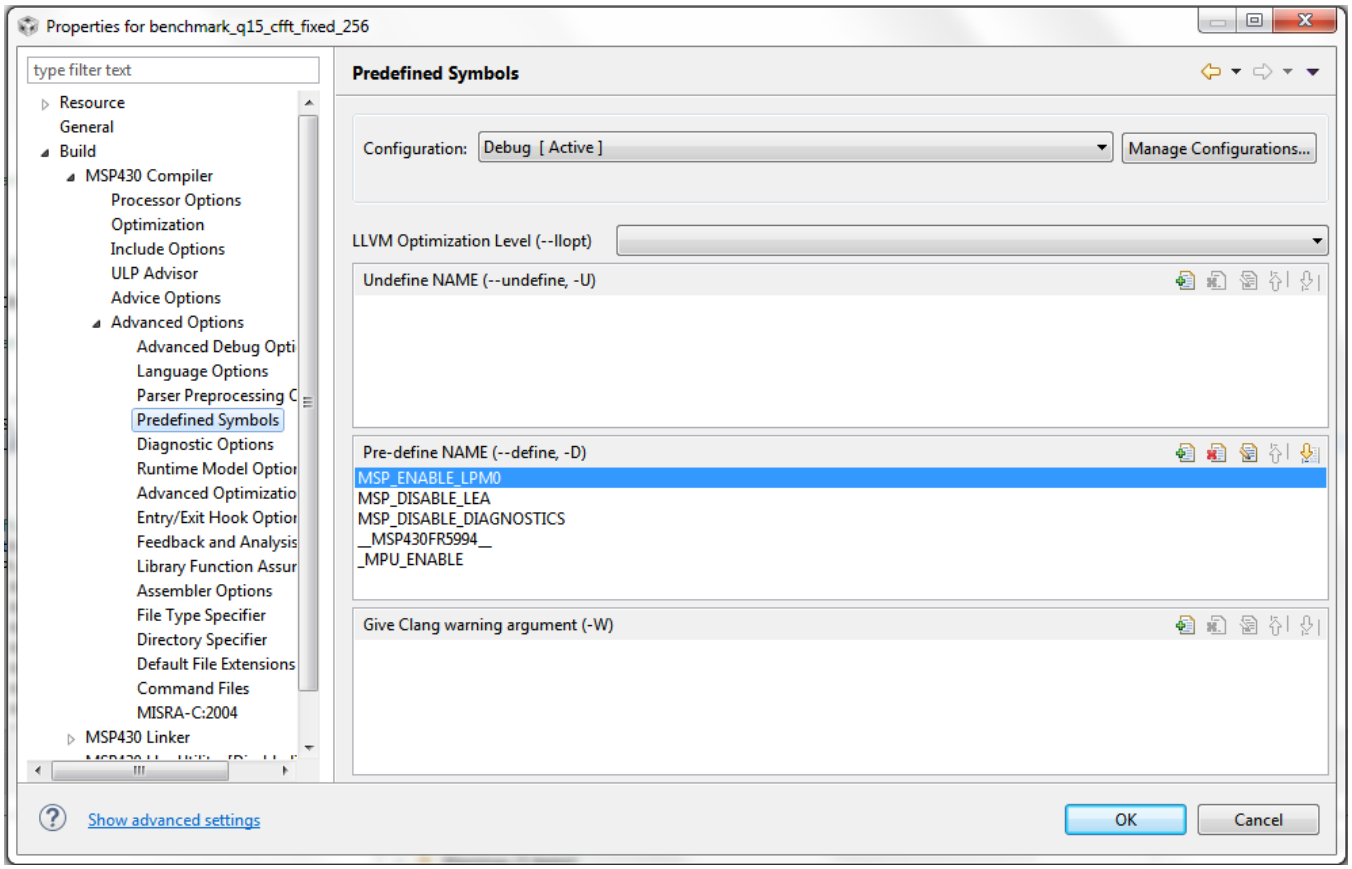

Figure 5. Predefined Symbols

These options are used throughout to enable various features and to help optimize the MSP DSP Library for benchmarking purposes, but also to optimize end applications for engineers.

5.2 IAR Setup for ARM Cortex-M0+ Based MCU

For the purposes of this application report, the IAR Embedded Workbench for ARM was used to enable the CMSIS DSP Library for the ARM Cortex-M0+ based MCU.

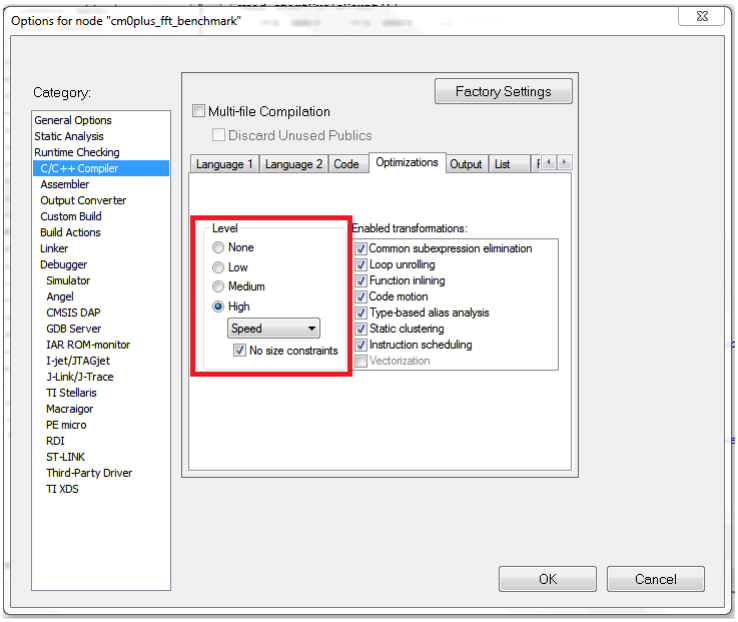

Figure 6. IAR Compiler Optimizations

This benchmark is implemented on IAR Embedded Workbench for ARM version 7.50.3.10751. The compiler was set up to provide the highest optimizations, specifically for speed. Learn more about IAR Embedded Workbench for ARM and download it at https://www.iar.com/.

5.3 IAR Setup for MSP430FR599x MCU

The IAR Embedded Workbench for MSP MCUs was used to develop the IAR benchmarks for the LEA module in this application report. When IAR IDE has been launched, and a workspace directory chosen, use Project→Add Existing Project and look for the .eww file in the directory "[Download Location]\SLAA698\SLAA698\" and open it.

In the project settings, under General Options, make sure the Target Device is the MSP430FR5994. This application report also investigates how compiling with large and small code and data memory models affects the cycle count results of the LEA module.

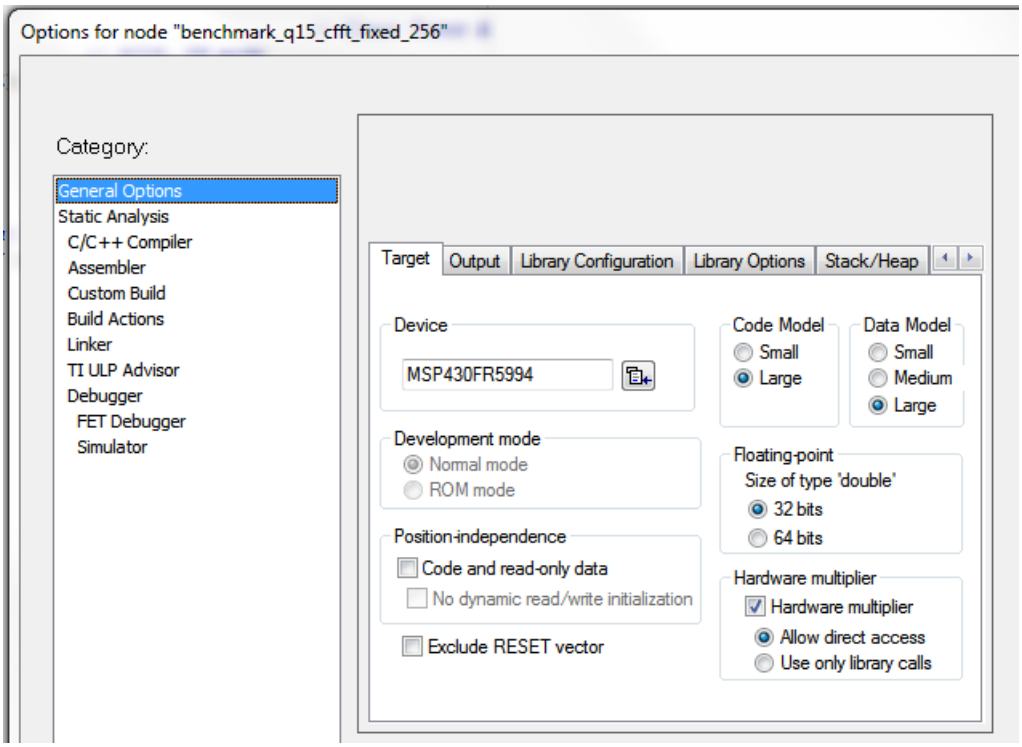

Figure 7. IAR Target Options

In the project settings, under C/C++ Compiler, the compiler is set for optimization to be on "High" and optimized for speed. Under the tab for Preprocessor, there is an area for preprocessor defined symbols to enable various options in the MSP DSP Library (see Section 5.6.3).

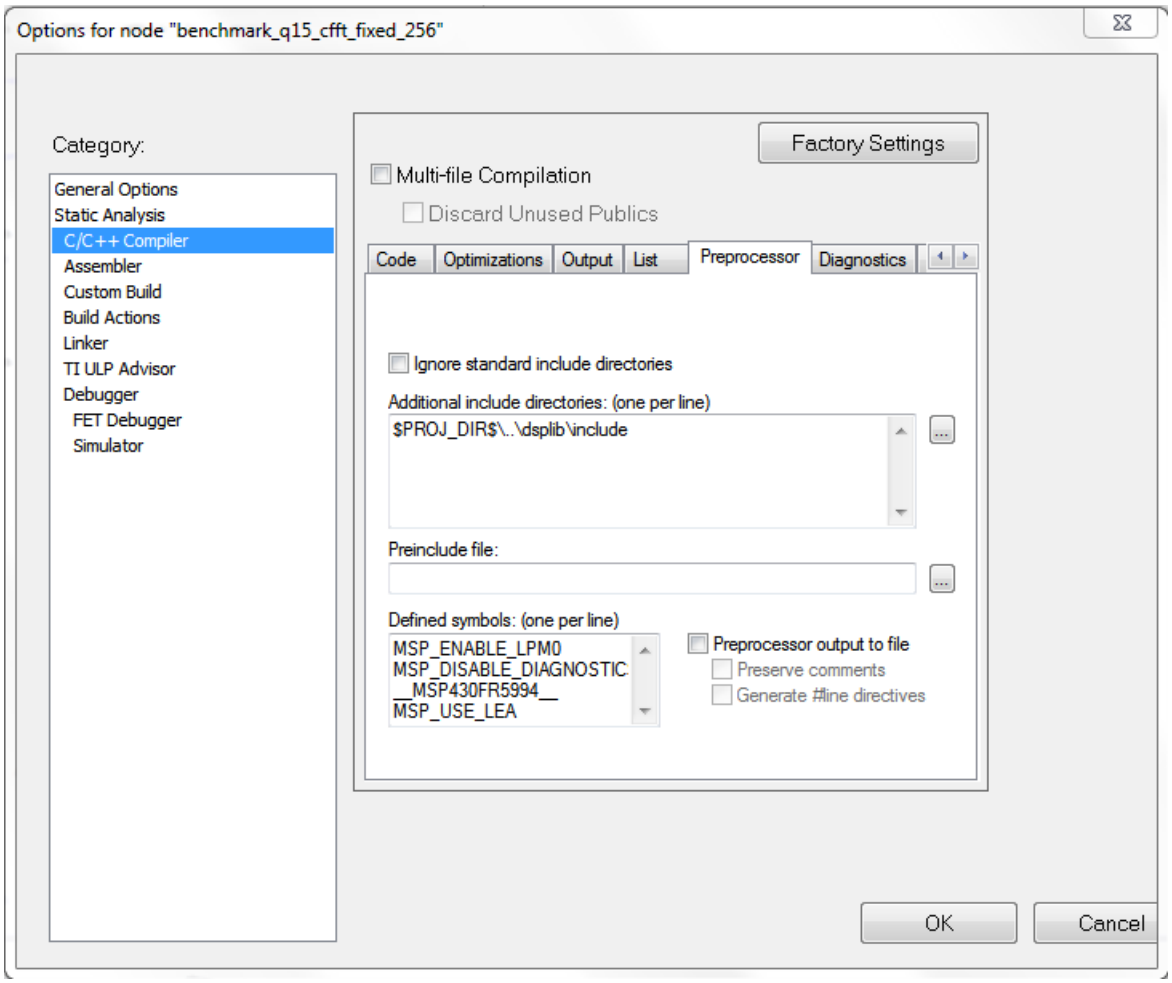

Figure 8. IAR Preprocessor Options

5.4 Digital Signal Processing (DSP) Library for MSP MCUs

The TI DSP Library for MSP MCUs is a highly optimized set of functions to perform many common signal processing operations on fixed-point numbers for MSP430 MCUs. The function set is typically used for applications where processing-intensive transforms are done in real time for minimal energy and very high accuracy. Learn more about the DSP Library for MSP MCUs and download it at www.ti.com/tool/msp-dsplib.

5.5 CMSIS DSP Library

The CMSIS DSP software library (version 1.4.7) was used to perform FFT and FIR functions on the ARM Cortex-M0+ based MCU tested. The software library is a suite of common signal processing functions that are used on Cortex-M devices. The 16-bit q15 data-type complex FFT and real FIR functions from the CMSIS DSP Library are used to compare the ARM Cortex-M0+ based MCU with the MSP430FR5994 MCU in this application report. For more information on the CMSIS DSP Library, visit http://www.keil.com/pack/doc/CMSIS/DSP/html/index.html.

The FIR function for the ARM CM0+ device was set up to be the exact same filter used in the benchmark_fir_q15 example that is provided for the MSP430FR5994 MCU. The example exhibits a 50-tap filter with the same coefficients and a vector length of 200.

When implementing and IAR version, make sure to verify the correct CMSIS DSP Library, right click the project name→Options→General Options→Library Configuration→CMSIS→ and make sure that *Use CMSIS* and *DSP library* are both checked (see Figure 9).

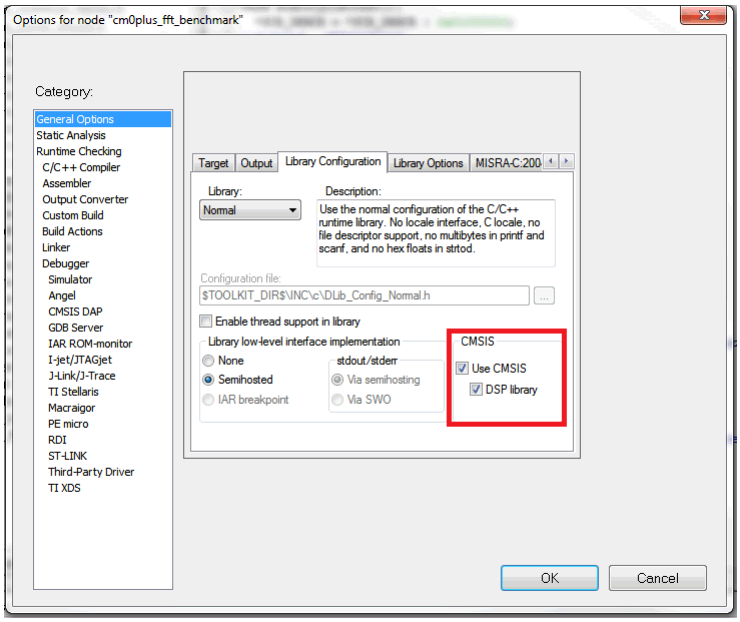

Figure 9. CMSIS DSP Library

5.6 Software Architecture

5.6.1 LEA Module Power Benchmark Example

Linked in the software project folder in Section 5, there is an example that shows how to replicate the data sheet parameter for power consumption on the LEA module. The example is called "benchmark_power_lea". First, import the project into either Code Composers Studio IDE or IAR IDE.

To accurately reproduce the specification in the data sheet of 67 µA/MHz, there are two measurements that need to be taken; power consumed while continuously computing a 32-bit (iq31) 256-sample complex FFT and power consumed while in active mode. Therefore the program is set up to have two different functions dependent on the value of port P1.0 being high or low at program start.

If P1.0 is high at program start, the runLEA variable is set, and the LEA module is enabled in active mode. LPM0 is disabled using the MSP_DISABLE_LPM0 predefine. This test is given the variable x.

If P1.0 is low at program start, the runLEA variable is not set, defaulting the LEA module to not execute, and the program polls the msp_lea_ifg flag. This test is given the variable y.

Average current consumption measurements over the course 4 seconds are taken for both cases running at 16 MHz. To get the power consumption of the LEA module, subtract test case x from test case y. This subtracts the average current of the device running in active mode and leave only the power consumption of the LEA module.

At the top of this program, there is an option to set the input of the FFT as pure zeros, a sine wave input, or 0xA5 repeated. Select the appropriate #define to test how this affects the power consumption of LEA. The data sheet parameter is taken when the input is all zeros; however, the input to the LEA module can cause higher average current consumption, because gates flip more often instead of being primarily static. Despite this, the influence of the various inputs is not more than approximately 20 µA/MHz.

The 32-bit 256-sample complex FFT is one of the more math intensive functions supported on the LEA module. Other functions such as vector add, matrix multiplication, and scaling result in even less power consumed.

While this is a µA/MHz number, it is better to look at the pure energy consumption of the device while using LEA. This is because the energy consumption is not time dependent and allows for a more accurate representation of the benefits that LEA gives the user.

5.6.2 FFT and FIR Benchmark Examples

The MSP430FR5994 MCU is set up for ultra-low-power operation by configuring the GPIO to ensure low leakage currents, setting up the DCO to source the SMCLK and MCLK, and using low-power modes. In the software examples, there are five benchmark examples; one benchmark for recreating the data sheet parameter for the LEA module to operate at 67 µA/MHz, and four others that are used for calculating the cycle count for the three different FFTs and the FIR function. In the cycle count benchmark projects, the code provided enables the device to be setup to operate at 1 MHz, 8 MHz, or 16 MHz. It also gives the programmer the option to measure the cycle counts or measure for power consumption by allowing them to alter a #define.

When measuring the number of cycles needed to compute a function, the #define MEASURE_CYCLE_COUNT is set. This enables a timer clocked by the SMCLK to count the number of cycles. When measuring energy consumption of the MSP430FR5994, the #define MEASURE_POWER was set to ensure that the timer does not consume any unnecessary energy.

Because the MSP430FR599x MCU device family has both 128KB and 256KB, the addressable memory range for anything greater than 64KB must be assigned an address 20 bits in length. When compiling for a large memory model, the user is intending to use the full 20-bit addressable range of memory. When compiling for a small memory model, the user is intending to use any addressable memory only below 64KB. For the various results that are taken, each example is executed in both the IAR Embedded Workbench for MSP430 IDE v6.50.1 and Code Composer Studio IDE v6.2 IDE using the TI v15.12.3.LTS compiler for both large and small code and data memory models. When compiled for a large memory model, extra cycles may need to be taken due to the 16-bit architecture of the device. See Section 5.1 and Section 5.3 for instructions on how to adjust this parameter.

Each example is also set up to be compiled at the highest optimization for speed. See Section 5.1 and Section 5.3 to understand how to achieve this.

5.6.3 MSP Predefines For Software Examples Explained

MSP_DISABLE_DIAGNOSTICS – Disables diagnostic error checking for each DSP Lib function call. If there is high confidence that the program will not fail, this can be used to optimize the program a little further.

MSP_ENABLE_LPM0 – Enables the device to run in LPM0.

MSP_DISABLE_LPM0 – Enables the device to run in active mode for benchmarking purposes.

MSP_USE_LEA – Enables the LEA accelerator for DSPLib functions. The default is to use the LEA module when it is available. This is used in the purposes of this benchmark to clearly indicate when the accelerator is being used.

MSP_DISABLE_LEA – Disables the LEA module. It is always recommended to use the LEA module for lowest power and best performance. Disabling the LEA module is only intended to be used for benchmarking the LEA module performance against the CPU and hardware multiplier or by users who would like to dedicate the LEA peripheral to a specific function and run the remainder on the CPU.

For more information on each of these predefines, see the *MSP DSP Library API Guide*.

5.6.4 ARM Cortex-M0+ Software Examples

The ARM Cortex-M0+ based MCU is set up to run at 12 MHz to enable the DC/DC converter that is on the chip, allowing for efficient and low-power operation. All of the GPIOs that were not used were configured to ensure that leakage current was minimized and that the lowest-power setting observed matched with what was expected from the data sheet.

6 Results

6.1 Cycle Count

Table 1 summarizes results from the cycle count calculations for the MSP430FR5994 MCU when it is using the LEA module and when it is not. The table also shows the difference between the IAR Embedded Workbench for MSP430 v6.50.1 and the Code Composer Studio IDE v6.2 using the TI v15.12.3.LTS compiler as well as the difference between large and small data and code memory models.

Table 1. Cycle Count Comparison

8 MHz				16 MHz			
Function	**Cycle Count LEA**	**Cycle Count CPU**	**Improvement**	**Function**	**Cycle Count LEA**	**Cycle Count CPU**	**Improvement**
Code Composer Studio IDE, Data Memory Model = Small, Code Memory Model = Small							
CFFT 128	2672	46864	17.54x	CFFT 128	2736	55520	20.29x
CFFT 256	5360	106016	19.78x	CFFT 256	5424	125392	23.12x
CFFT 512	11136	236240	21.21x	CFFT 512	11632	278896	23.98x
FIR	11440	268480	23.47x	FIR	11488	310496	27.03x
IAR Embedded Workbench IDE, Data Memory Model = Small, Code Memory Model = Small							
CFFT 128	2720	39872	14.66x	CFFT 128	2800	48016	17.15x
CFFT 256	5408	91024	16.83x	CFFT 256	5488	109344	19.92x
CFFT 512	11200	204400	18.25x	CFFT 512	11280	245248	21.74x
FIR	11536	212960	18.46x	FIR	11488	246560	21.46x
Code Composer Studio IDE, Data Memory Model = Large, Code Memory Model = Large							
CFFT 128	2832	85808	30.30x	CFFT 128	2928	98992	33.81x
CFFT 256	5520	197008	35.69x	CFFT 256	5616	226944	40.41x
CFFT 512	11312	444208	39.27x	CFFT 512	11408	511360	44.82x
FIR	11600	318592	27.46x	FIR	11648	379424	32.57x
IAR Embedded Workbench IDE, Data Memory Model = Large, Code Memory Model = Large							
CFFT 128	2816	44336	15.74x	CFFT 128	2912	53440	18.35x
CFFT 256	5504	101056	18.36x	CFFT 256	5600	121568	21.71x
CFFT 512	11296	226704	20.07x	CFFT 512	11376	272416	23.95x
FIR	11552	213792	18.51x	FIR	11616	247424	21.30x

With the power analyzer, it is easy to identify when the MSP430FR5994 MCU is computing the FFT. In Figure 10, a 256-point complex FFT is being computed at 16-MHz operation without the LEA module. The program first calculates the vector, then enters into LPM1 to isolate the FFT calculation. The program exits LPM1 through way of a Timer_A0 interrupt, calculates the FFT, and then goes back into LPM1 and eventually LPM4. The FFT calculation takes approximately 24.584 ms to complete.

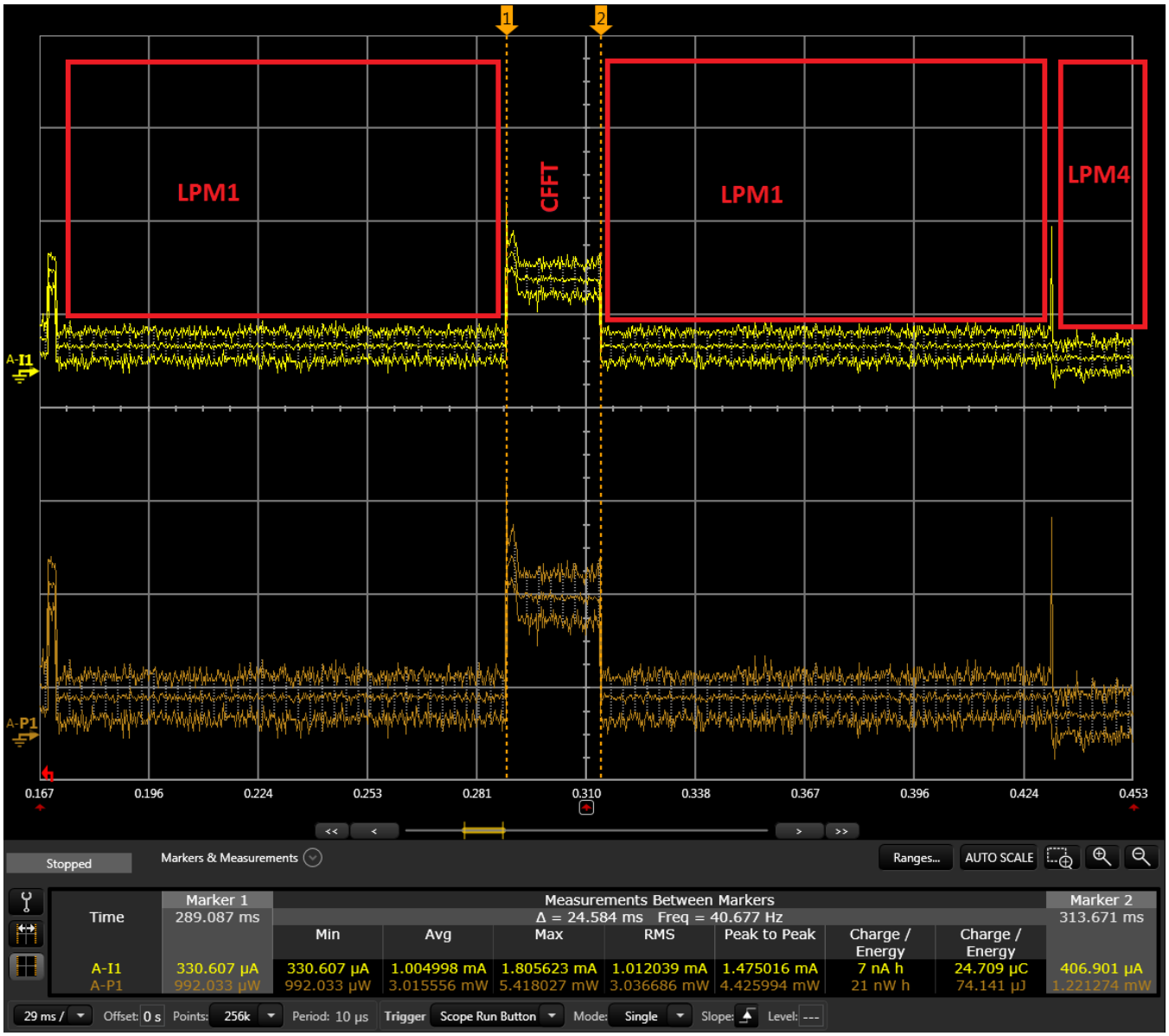

Figure 10. 256-Point Complex FFT With the LEA Module Disabled

However, when the MSP430FR5994 MCU leverages the LEA module, the same FFT takes only approximately 715 µs to complete (see Figure 11).

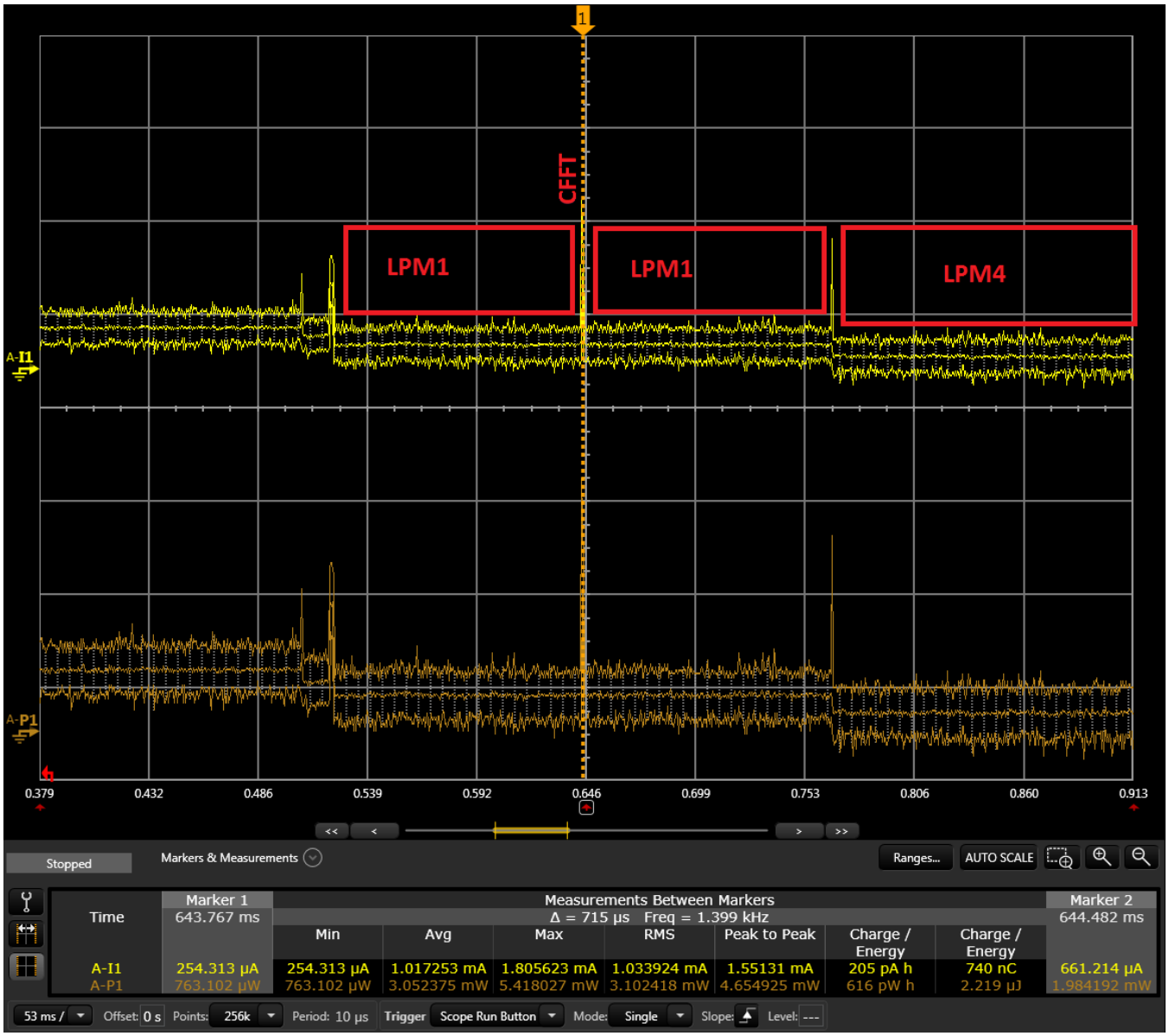

Figure 11. 256-Point Complex FFT With the LEA Module Enabled

Table 2 lists the performance advantage of the LEA module by comparing the MSP430FR5994 MCU using the LEA peripheral clocked at 8 MHz, with a competitor ARM Cortex-M0+ based MCU clocked at 12 MHz with a DC/DC converter enabled to operate as efficiently as possible.

Table 2. Clock Cycle Count for DSP Function Call

DSP Function	Competitor A (CM0+) at 12 MHz	MSP430 With LEA at 8 MHz	Performance Improvement
128-Point Complex FFT	94805	2672	35.5x
256-Point Complex FFT	223360	5360	41.7x
512-Point Complex FFT	475920	11136	42.7x
50-Tap FIR	179378	11440	15.7x

When performing a 16-bit complex FFT ranging in precision, MSP MCUs can outperform the competitor ARM Cortex-M0+ by up to 42 times. Improvements are not limited to only FFT calculations. The advantages of the LEA module can also be seen when computing a 50-tap 16-bit real FIR calculation of a vector with 200 input samples.

Table 3 lists the clock cycles needed to compute the various signal processing functions. Because the DSP Library for MSP MCUs automatically uses the LEA module if it is available on the target device, the MSP430FR5964 MCU (a variant of the MSP430FR5994 MCU without the LEA coprocessor) was used for calculations of clock cycles using DSP Library for MSP MCUs. This MCU should give the best baseline comparison of the impact of the LEA module on MSP MCUs.

Another advantage to the LEA module is its ability to operate out of SRAM. As mentioned in Section 3, the LEA module has 4KB of shared space in SRAM to execute from. Because of this, the LEA can operate with zero wait states when running at CPU clock frequencies greater than 8 MHz. Table 3 lists the cycle counts of a 16-bit 256-Point Complex FFT.

Table 3. Clock Cycles for 16-Bit 256-Point Complex FFT

16-Bit 128-Point Complex FFT	CPU Frequency (MHz)	Cycle Count	Elapsed Time
MSP430FR5994 with LEA enabled	8	5360	670 µs
MSP430FR5994 with LEA disabled		106016	13252 µs
MSP430FR5994 with LEA enabled	16	5424	339 µs
MSP430FR5994 with LEA disabled		125392	7837 µs

In the case where the LEA is enabled, the advantage of the LEA module running out of SRAM, where FRAM wait states do not apply, is shown. Running at 16 MHz with the LEA module enabled, there is only approximately 100 cycles added to the calculation. These extra cycles can be accounted for in waking up the device and moving the resultant vector into FRAM from SRAM. This is a very small margin in comparison to the clock cycles needed when using a device that does not have LEA module present. With the improvement that the LEA coprocessor brings to real-time complex vector math operations, end applications need to spend less time performing calculations, which allows those applications to increase performance or spend more time in lower-power modes where total application energy consumption can be saved.

6.2 Energy Profile

Each measurement was taken using the Keysight power analyzer by executing the program and measuring the isolated function call to the LEA module (see Figure 12).

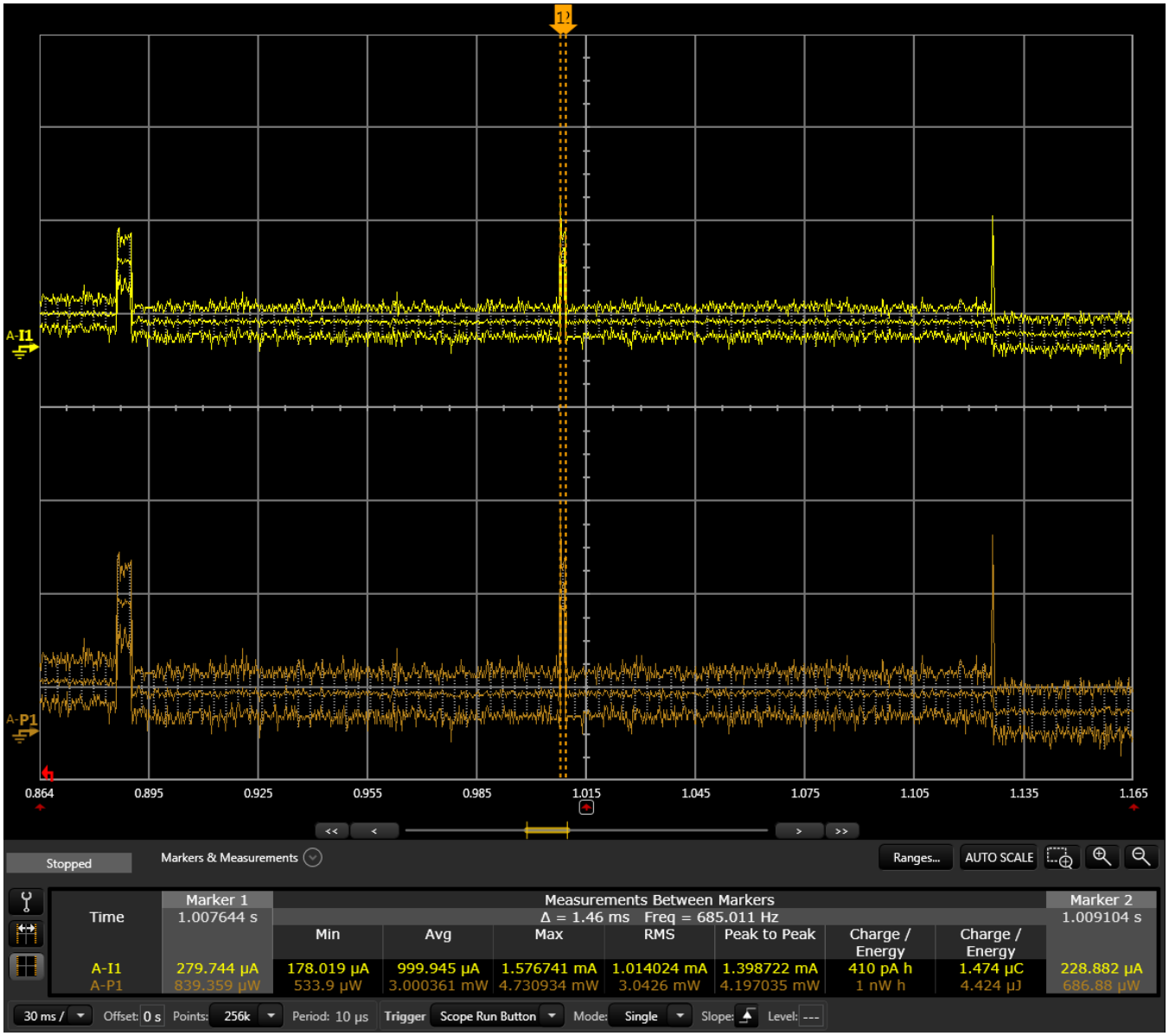

Figure 12. 512-Point Complex FFT Running at 8 MHz With the LEA Module Enabled

Table 4 summarizes the extremely low-power and efficient vector math operation capabilities of the LEA module. A variety of complex FFTs and an FIR filter are benchmarked using the power analyzer.

Table 4. Energy Use Comparison

Processor	Clock Frequency	Energy (µJ)			
		128-Point Complex FFT	256-Point Complex FFT	512-Point Complex FFT	FIR
MSP430FR5994 with LEA enabled	8 MHz	1.228	2.219	4.424	4.378
MSP430FR5994 with LEA enabled	16 MHz	1.182	2.092	4.184	4.065
ARM Cortex-M0+ MCU	12 MHz with DC/DC	10.722	24.777	52.806	32.295
Performance improvement of MSP at 16 MHz with LEA enabled		**9.07x**	**11.84x**	**12.6x**	**7.94x**

The advantage of having the LEA module enabled on the MSP430FR5994 MCU is apparent, at times being approximately 12.5 times more energy efficient than the ARM Cortex-M0+ based MCU. As noted in Section 4.3, both devices are set up for low-power operation and both MCUs are run at 3.0 V. Even with the MSP430FR5994 MCU running at 16 MHz, the energy consumption stays the same as operation at 8 MHz.

Table 5. Energy Use Comparison

Processor	Clock Frequency	Energy (µJ)			
		128-Point Complex FFT	256-Point Complex FFT	512-Point Complex FFT	FIR
MSP430FR5994 with LEA enabled	16 MHz	1.182	2.092	4.184	4.065
MSP430FR5994 with LEA disabled	16 MHz	30.33	69.54	152.584	72.305
Performance improvement with LEA enabled		**25.6x**	**33.2x**	**36.4x**	**17.8x**

Even when the MSP430FR5994 MCU is using optimized C-code and is compared to itself using the LEA module, there is a significant performance increase of energy consumption of up to 36.4 times.

6.3 LEA Module Cycle Count Estimation

The LEA module overall is more efficient and faster at computing various signal processing algorithms than software enabled algorithms on both 16-bit and some 32-bit platforms. Section 6.3.2 lists the APIs along with a formula for each API. Each formula is deduced from hand-optimized assembly coding and is cycle accurate for the LEA module itself. With each call to the LEA module, there is a little overhead that must be added to account for setting up various parameters and vectors needed prior to calling the LEA module.

6.3.1 Example Calculations

To break down some of the cycle estimates that were calculated earlier for the various functions that were benchmarked, the formulas in Section 6.3.2 can be used. Each time the LEA module is called for the first time in any program, DSPLib automatically checks to see if the LEA module has been initialized. If it has not been initialized, then DSPLib automatically initializes the LEA module and then executes the command. Due to the way the benchmarks are set up, this minimal overhead is added into the cycle count results.

6.3.1.1 FIR Calculation

For example, for the FIR calculation, the msp_fir_q15 function in the MSP DSP Library is used. Looking for the function, Equation 1 is found:

$$17 + \frac{N}{2} \times (12 + 4 \times \frac{tapSize}{2}) \quad (1)$$

N is the FIR block or vector size and tapSize is the FIR filter tap. In the case of the benchmark examples, the block size is 200 and the tap size is 50. Therefore, the equation is:

$$17 + \frac{200}{2} \times \left(12 + 4 \times \frac{50}{2}\right) = 11217 \text{ cycles} \quad (2)$$

Earlier, the FIR took 11440 cycles when compiled in Code Composer Studio IDE using a small code and data memory model. That is only 223 cycles of overhead, some of which can be attributed to the initialization overhead and the rest for setting up parameters and loading of the vector.

6.3.1.2 FFT Calculation

The MSP430FR599x MCU with the LEA module was able to calculate a 512-point complex FFT in 11136 cycles using the provided benchmark example. In the benchmark example, the msp_cmplx_fft_fixed_q15 function was used. The formula sheet shows that the msp_cmplx_fft_fixed_q15 function cycle count can be computed by Equation 3.

$$\text{msp\_cmplx\_bitrev\_q15} + 17 + \log_2(N) \times \left(22 + 4 \times \frac{N}{2}\right)$$

where

- N = FFT block size (3)

This is a case where the msp_cmplx_fft_fixed_q15 function actually uses another DSPLib function called msp_cmplx_bitrev_q15. Therefore, the cycle count for the msp_cmplx_bitrev_q15 function must be taken into account. Again, see Section 6.3.2 for the formula for the msp_cmplx_bitrev_q15.

For the msp_cmplx_bitrev_q15, there are two equations, depending on the value of R. R can be calculated by Equation 4.

$$N = 2^R \quad (4)$$

which can be simplified to

$$\log_2(N) = R \quad (5)$$

Earlier it was noted that N = 512; therefore, R is calculated as:

$$\log_2(512) = R = 9 \quad (6)$$

Because R is odd, the odd version of the cycle count equation for the complex bit reversal is used:

$$\text{Cycle count} = 22 + 11 \times (S - 1) + 10 \times S \times \frac{S-1}{2}$$

where

- $S(\text{odd}) = \sqrt{\frac{N}{2}}$
- N = FFT block size (7)

The equation for the full complex FFT using the LEA module then becomes Equation 8:

$$22 + 11 \times \left(\sqrt{\frac{N}{2}} - 1\right) + 10\sqrt{\frac{N}{2}} \times \frac{\left(\sqrt{\frac{N}{2}} - 1\right)}{2} + 17 + \log_2(N) \times (21 + 4 \times \frac{N}{2})$$

where

- N = FFT block size (8)

In the case of a 512 point complex FFT, N is equal to 512. When this is taken into consideration, the formula produces 10818 cycles. This is only 318 cycles off of the actual measured cycle count and again includes the LEA module initialization routine.

When using N = 256, R is then even as $\log_2(256) = 8$. Therefore the even equation is used. See Table 6 for the rest of the benchmark results and how they relate to the number of clock cycles calculated that LEA requires to compute.

Table 6. Comparison of Calculated and Actual Clock Cycles

Function	Calculated	Actual	Difference
128-Point Complex FFT	2342	2672	330
256-Point Complex FFT	5028	5360	332
512-Point Complex FFT	10818	11136	318
FIR	11217	11440	223

Minimal overhead is added to each function. Given that these are some of the more math intensive functions supported in the MSP DSP Library, it can be assumed that there are not many scenarios where the worst case overhead is much more than a few hundred clock cycles per function call to the LEA module.

6.3.2 LEA Cycle Count Formulas

Table 7 lists the cycle count formula for each function.

Table 7. LEA Cycle Count Formulas

Function	Description	LEA Support	Clock Cycle Equation	Variable Description[1]
msp_add_q15	Addition of two real source vectors	yes	$14 + 3 \times \frac{N}{2}$	N is the block size
msp_add_iq31	Addition of two real source vectors	yes	$11 + 3 \times N$	N is the block size
msp_sub_q15	Subtraction of two real source vectors	yes	$14 + 3 \times \frac{N}{2}$	N is the block size
msp_sub_iq31	Subtraction of two real source vectors	yes	$11 + 3 \times N$	N is the block size
msp_mpy_q15	Multiplication of two real source vectors	yes	$16 + 3 \times \frac{N}{2}$	N is the block size
msp_mpy_iq31	Multiplication of two real source vectors	yes	$13 + 4 \times N$	N is the block size
msp_mac_q15	Multiply and accumulate of real source vectors	yes	$16 + 3 \times \frac{N}{2}$	N is the block size
msp_mac_iq31	Multiply and accumulate of real source vectors	yes	$15 + 4 \times N$	N is the block size
msp_neg_q15	Negation of a source vector	yes	Done through msp_mpy_q15	N/A
msp_neg_iq31	Negation of a source vector	yes	Done through msp_mpy_iq31	N/A
msp_abs_q15	Absolute value of a real source vector	no	C-Code	N/A
msp_abs_iq31	Absolute value of a real source vector	no	C-Code	N/A
msp_offset_q15	Constant offset of a real source vector	yes	Done through msp_add_q15	N/A
msp_offset_iq31	Constant offset of a real source vector	yes	Done through msp_add_iq31	N/A

(1) N/A = Not applicable

Table 7. LEA Cycle Count Formulas (continued)

Function	Description	LEA Support	Clock Cycle Equation	Variable Description[1]
msp_scale_q15	Scale a real source vector	no	Uses MPY32	N/A
msp_scale_iq31	Scale a real source vector	no	Uses MPY32	N/A
msp_shift_q15	Bitwise shift of a real source vector	yes	Done through msp_mpy_q15	N/A
msp_shift_iq31	Bitwise shift of a real source vector	yes	Done through msp_mpy_iq31	N/A
msp_max_q15	Signed maximum of a source vector	yes	$18 + 6 \times \frac{N}{2}$	N is the number of input samples in an input vector
msp_max_iq31	Signed maximum of a source vector	yes	$17 + 3 \times N$	N is the number of input samples in an input vector
msp_max_uq15	Unsigned maximum of a source vector	yes	$18 + 6 \times \frac{N}{2}$	N is the number of input samples in an input vector
msp_max_uq31	Unsigned maximum of a source vector	yes	$17 + 3 \times N$	N is the number of input samples in an input vector
msp_min_q15	Signed minimum of a source vector	yes	$18 + 6 \times \frac{N}{2}$	N is the number of input samples in an input vector
msp_min_iq31	Signed minimum of a source vector	yes	$17 + 3 \times N$	N is the number of input samples in an input vector
msp_min_uq15	Unsigned minimum of a source vector	yes	$18 + 6 \times \frac{N}{2}$	N is the number of input samples in an input vector
msp_min_uq31	Unsigned minimum of a source vector	yes	$17 + 3 \times N$	N is the number of input samples in an input vector
msp_cmplx_add_q15	Addition of two complex source vectors	yes	Done using msp_add_q15	N/A
msp_cmplx_add_iq31	Addition of two complex source vectors	yes	Done using msp_add_iq31	N/A
msp_cmplx_sub_q15	Subtraction of two complex source vectors	yes	Done using msp_sub_q15	N/A
msp_cmplx_sub_iq31	Subtraction of two complex source vectors	yes	Done using msp_sub_iq31	N/A
msp_cmplx_mpy_q15	Multiplication of two complex source vectors	yes	$13 + 5 \times N$	N is the block size
msp_cmplx_mpy_iq31	Multiplication of complex source vectors	no	Uses MPY32	N/A
msp_cmplx_mpy_real_q15	Multiplication of complex source vector by real source vector	no	C-Code	N/A
msp_cmplx_mpy_real_iq31	Multiplication of complex source vector by real source vector	yes	Done through msp_mpy_iq31	N/A
msp_cmplx_mac_q15	Multiply and accumulate of complex source vectors	yes	$14 + 5 \times N$	N is the block size
msp_cmplx_mac_iq31	Multiply and accumulate of complex source vectors	no	Uses MPY32	N/A
msp_cmplx_conj_q15	Conjugation of a source vector	no	C-Code	N/A

Table 7. LEA Cycle Count Formulas (continued)

Function	Description	LEA Support	Clock Cycle Equation	Variable Description[1]
msp_cmplx_conj_iq31	Conjugation of a source vector	yes	$\text{msp\_mpy\_iq31} \times \frac{N}{2} + \text{msp\_add\_iq31} \times \frac{N}{2}$ (add constants + 3.5 × blockSize)	N is the block size
msp_cmplx_scale_q15	Scale a complex source vector	no	Uses msp_scale_q15, which uses MPY32	N/A
msp_cmplx_scale_iq31	Scale a complex source vector	no	Uses msp_scale_iq31, which uses MPY32	N/A
msp_cmplx_shift_q15	Bitwise shift of a complex source vector	yes	Done with msp_mpy_q15	N/A
msp_cmplx_shift_iq31	Bitwise shift of a complex source vector	yes	Done with msp_mpy_iq31	N/A
msp_matrix_add_q15	Addition of two real source matrices	yes	$14 + 3 \times \frac{N}{2}$	N is the block size
msp_matrix_add_iq31	Addition of two real source matrices	yes	$11 + 3 \times N$	N is the block size
msp_matrix_sub_q15	Subtraction of two real source matrices	yes	$14 + 3 \times \frac{N}{2}$	N is the block size
msp_matrix_sub_iq31	Subtraction of two real source matrices	yes	$11 + 3 \times N$	N is the block size
msp_matrix_mpy_q15	Multiplication of two real source matrices	yes	$16 + 3 \times \frac{N}{2}$	N is the block size
msp_matrix_mpy_iq31	Multiplication of two real source matrices	no	$13 + 4 \times N$	N is the block size
msp_matrix_trans_q15	Transposition of a source matrix	no	C-Code	N/A
msp_matrix_trans_iq31	Transposition of a source matrix	no	C-Code	N/A
msp_matrix_neg_q15	Negation of a source matrix	yes	Done through msp_neg_q15	N/A
msp_matrix_neg_iq31	Negation of a source matrix	yes	Done through msp_neg_iq31	N/A
msp_matrix_abs_q15	Absolute value of a real source matrix	no	C-Code	N/A
msp_matrix_abs_iq31	Absolute value of a real source matrix	no	C-Code	N/A
msp_matrix_offset_q15	Constant offset of a real source matrix	yes	Done through msp_offset_q15	N/A
msp_matrix_offset_iq31	Constant offset of a real source matrix	yes	Done using msp_offset_iq31	N/A
msp_matrix_scale_q15	Scale a real source matrix	no	Done using msp_scale_q15	N/A
msp_matrix_scale_iq31	Scale a real source matrix	no	Done using msp_scale_iq31	N/A
msp_matrix_shift_q15	Bitwise shift of a real source matrix	yes	Done using msp_shift_q15	N/A
msp_matrix_shift_iq31	Bitwise shift of a real source matrix	yes	Done using msp_shift_iq31	N/A
msp_fir_q15	Discrete-time convolution of a source vector with real coefficients to apply an FIR filter	yes	$17 + \frac{N}{2} \times \left(12 + 4 \times \frac{\text{tapSize}}{2}\right)$	N is the FIR block size and tapSize is the FIR Filter TAP

Table 7. LEA Cycle Count Formulas (continued)

Function	Description	LEA Support	Clock Cycle Equation	Variable Description[1]
msp_fir_iq31	Discrete-time convolution of a source vector with real coefficients to apply an FIR filter	yes	17 + N × (9 + 4 × tapSize)	N is the FIR block size and tapSize is the FIR Filter TAP
msp_cmplx_fir_q15	Discrete-time convolution of a complex source vector with complex coefficients to apply an FIR filter	yes	15 + N × (10 + 5 × tapSize)	N is the FIR block size and tapSize is the FIR Filter TAP
msp_cmplx_fir_iq31	Discrete-time convolution of a complex source vector with complex coefficients to apply an FIR filter	yes	15 + N × (14 + 14 × tapSize)	N is the FIR block size and tapSize is the FIR Filter TAP
msp_biquad_df1_q15	Second-order direct form 1 biquad filter	yes	$16 + 15 \times \frac{N}{2}$	N is the block size
msp_biquad_df2_q15	Second-order direct form 2 biquad filter	yes	$14 + 18 \times \frac{N}{2}$	N is the block size
msp_biquad_df2_ext_q15	Second-order direct form 2 biquad filter extended with DC bias and minimum and maximum tracking	yes	$25 + 22 \times \frac{N}{2}$	N is the block size
msp_biquad_cascade_df1_q15	Cascaded direct form 1 biquad filter	yes	Uses msp_biquad_df1_q15 × numberOfStages	numberOfStages is the number of IIR stages
msp_biquad_cascade_df2_q15	Cascaded direct form 2 biquad filter	yes	Uses msp_biquad_df2_q15 × numberOfStages	numberOfStages is the number of IIR stages
msp_biquad_cascade_df2_ext_q15	Cascaded direct form 2 biquad filter extended with DC bias and minimum and maximum tracking	yes	Uses msp_biquad_df2_ext_q15 × numberOfStages	numberOfStages is the number of IIR stages
msp_fft_auto_q15	Real forward FFT function with auto-scaling	yes	Uses N/2 for msp_cmplx_fft_auto_q15 + msp_split_q15	N is the block size
msp_fft_fixed_q15	Real forward FFT function with fixed scaling by two at each stage	yes	Uses N/2 for msp_cmplx_fft_fixed_q15 + msp_split_q15	N is the block size
msp_fft_iq31	Real forward FFT function without scaling	yes	Uses N/2 for msp_cmplx_fft_iq31 + msp_split_iq31	N is the block size
msp_ifft_auto_q15	Real result inverse FFT function with auto-scaling	yes	Uses N/2 for msp_cmplx_conj_q15 + msp_cmplx_fft_auto_q15 + msp_cmplx_shift_q15 + msp_split_q15	N is the block size
msp_ifft_fixed_q15	Real result inverse FFT function with fixed scaling by two at each stage	yes	Uses N/2 for msp_cmplx_conj_q15 + msp_split_q15 + msp_cmplx_fft_auto_q15 + msp_cmplx_shift_q15	N is the block size
msp_ifft_iq31	Real result inverse FFT function without scaling	yes	Uses N/2 for msp_max_iq31 + msp_min_iq31 + msp_cmplx_shift_iq31 + msp_cmplx_fft_iq31 + msp_cmplx_shift_iq31	N is the block size
msp_cmplx_fft_auto_q15	Complex forward FFT function with auto-scaling	yes	$\text{msp\_cmplx\_bitrev\_q15} + 29 + 4 \times \frac{N}{2} + \log_2(N) \times \left(21 + 4 \times \frac{N}{2}\right)$	N is the FFT block size

Table 7. LEA Cycle Count Formulas (continued)

Function	Description	LEA Support	Clock Cycle Equation	Variable Description[1]
msp_cmplx_fft_fixed_q15	Complex forward FFT function with fixed scaling by two at each stage	yes	msp_cmplx_bitrev_q15 + 17 + $\log_2(N) \times \left(22 + 4 \times \frac{N}{2}\right)$	N is the FFT block size
msp_cmplx_fft_iq31	Complex forward FFT function without scaling	yes	msp_cmplx_bitrev_iq31 + 21 + $\log_2(N) \times \left(18 + 13 \times \frac{N}{2}\right)$	N is the FFT block size
msp_cmplx_ifft_auto_q15	Complex inverse FFT function with auto-scaling	yes	msp_cmplx_conj_q15 + msp_cmplx_fft_auto_q15 + msp_cmplx_shift_q15	N/A
msp_cmplx_ifft_fixed_q15	Complex inverse FFT function with fixed scaling by two at each stage	yes	msp_cmplx_conj_q15 + msp_cmplx_fft_auto_q15 + msp_cmplx_shift_q15	N/A
msp_cmplx_ifft_iq31	Complex inverse FFT function without scaling	yes	msp_max_iq31 + msp_min_iq31 + msp_cmplx_shift_iq31 + msp_cmplx_fft_iq31 + msp_cmplx_shift_iq31	N/A
msp_copy_q15	Real Q15 vector copy	yes	Done through msp_add_q15	N/A
msp_copy_iq31	Real IQ31 vector copy	yes	Done through msp_add_iq31	N/A
msp_fill_q15	Real Q15 vector fill with constant	yes	Done through msp_add_q15	N/A
msp_fill_iq31	Real IQ31 vector fill with constant	yes	Done through msp_add_iq31	N/A
msp_cmplx_fill_q15	Complex Q15 vector fill with constant	yes	Done through msp_add_q15 × 2	N/A
msp_cmplx_fill_iq31	Complex IQ31 vector fill with constant	yes	Done through msp_add_iq31 × 2	N/A
msp_deinterleave_q15	Extract a single channel from multiple-channel source	yes	$16 + 3 \times \frac{N}{2}$	N is the block size
msp_deinterleave_iq31	Extract a single channel from multiple-channel source	yes	$9 + 2 \times N$	N is the block size
msp_cmplx_bitrev_q15	Complex bit-reversal function	yes	If $N = 2^R$ (R is even), cycle count = $19 + 8 \times (S - 1) + 5 \times S \times \frac{S-1}{2}$ If $N = 2^R$ (R is odd), cycle count = $22 + 11 \times (S - 1) + 10 \times S \times \frac{S-1}{2}$	Where S(even) = $\sqrt{N}$ and S(odd) = $\sqrt{\frac{N}{2}}$ and N is the block size
msp_cmplx_bitrev_iq31	Complex bit-reversal function	yes	If $N = 2^R$ (R is even), cycle count = $17 + 17 \times (S - 1) + 16 \times S \times \frac{S-1}{2}$ If $N = 2^R$ (R is odd), cycle count = $23 + 10 \times (S - 1) + 27 \times S \times \frac{S-1}{2}$	Where S(even) = $\sqrt{N}$ and S(odd) = $\sqrt{\frac{N}{2}}$ and N is the block size
msp_cmplx_q15	Converts a q15 real vector to have a complex component	Yes	Uses msp_interleave_q15 twice	N/A
msp_cmplx_iq31	Converts a iq31 real vector to have a complex component	Yes	Uses msp_interleave_iq31 twice	N/A
msp_interleave_q15	Insert a single channel into a multiple-channel destination	Yes	$17 + 5 \times \frac{N}{2}$	N is the block size
msp_interleave_iq31	Insert a single channel into a multiple-channel destination	Yes	Uses msp_add_iq31	N/A

Table 7. LEA Cycle Count Formulas (continued)

Function	Description	LEA Support	Clock Cycle Equation	Variable Description[1]
msp_iq31_to_q15	Convert IQ31 vector to Q15 format	Yes	msp_deinterleave_q15	N/A
msp_q15_to_iq31	Convert Q15 vector to IQ31 format	Yes	msp_fill_iq31 + msp_interleave_q15	N/A
msp_sinusoid_q15	Generate a sinusoid with specified amplitude and frequency	Yes	Uses msp_biquad_df1_q15	N/A
msp_split_q15		Yes	$46 + \log_2(N) \times 2 + 9 \times N$	N is the vector size
msp_split_iq31		Yes	$46 + \log_2(N) \times 2 - 1 + 17 \times \left(\frac{N}{2} - 1\right)$	N is the vector size

6.4 EnergyTrace++ Technology

EnergyTrace++ technology allows a user to quickly and easily analyze the energy consumption and the internal state of the MCU for debugging applications. These states include the status (on or off) of the peripherals and all system clocks (regardless of the clock source), and the low-power mode (LPM) currently in use. Visit the EnergyTrace™ technology page for more information.

For this application, Figure 13 shows the profile for FFT with the LEA module where the application remains in LPM1 before the FFT is triggered. When the FFT is triggered, the LEA module executes and then the device goes back into LPM1 before going to LPM4 at approximately 250 ms. This trace might not provide the full picture of the state as EnergyTrace++ technology reads data at a maximum rate of 4 kHz.

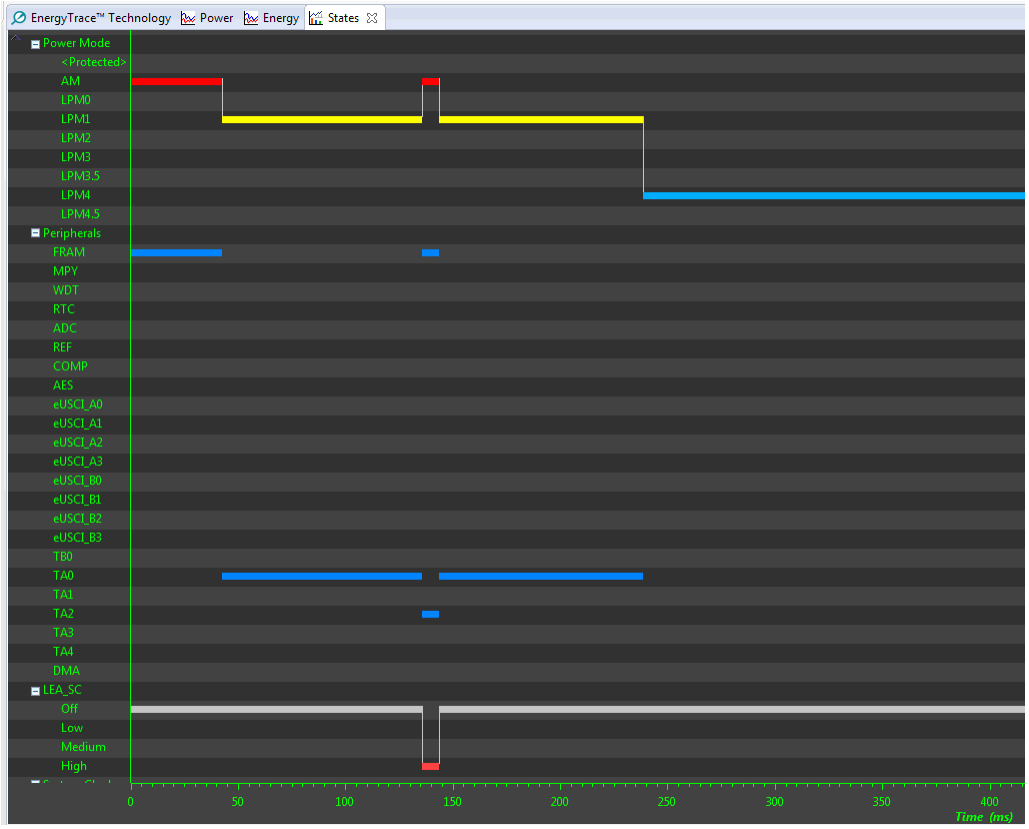

Figure 13. EnergyTrace™ Technology

7 Conclusion

The LEA module on MSP MCUs brings a new standard for signal processing capabilities for low-cost 16-bit MCUs. The LEA coprocessor allows engineers to no longer sacrifice speed for efficiency when designing systems with MCUs and enables signal processing with more efficiency and speed than 32-bit ARM Cortex M0+ MCUs.

8 Software Files

Download the software files for this application report from http://www.ti.com/lit/zip/slaa698.

9 References

1. TI E2E™ Community
2. *MSP430FR58xx, MSP430FR59xx, MSP430FR68xx, and MSP430FR69xx Family User's Guide*
3. *MSP430FR599x, MSP430FR596x Mixed-Signal Microcontrollers*
4. Filtering and Signal Processing Reference Design Using MSP430 FRAM Microcontroller
5. *Setting a new standard for MCU performance while minimizing energy consumption*

Revision History

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

Changes from April 7, 2016 to November 1, 2016 **Page**

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale (www.ti.com/legal/termsofsale.html) or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

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