Application Report
SLAA603–June 2014

MSP430™ Advanced Power Optimizations: ULP Advisor™ Software and EnergyTrace™ Technology

Brittany Finch and William Goh *MSP430 Applications*

ABSTRACT

MSP430 microcontrollers are designed specifically for ultra-low-power applications. Features such as multiple low-power modes, instant wakeup, intelligent autonomous peripherals, and much more to enable such ultra-low-power (ULP) capabilities. Texas Instruments provides valuable tools to help the programmer fully use these benefits and optimize power consumption of the target application. This application report details two of these tools, namely ULP Advisor software and EnergyTrace technology.

Following the explanation of ULP Advisor and EnergyTrace technology, a case study is included to demonstrate specifically how to use these tools. The MSP-EXP430FR5969 and MSP-EXP430G2 LaunchPad™ Evaluation Kits are the required hardware for the case study. Three code files are included in the project, all of which perform the same end application. The first version of the code has not been optimized for energy efficiency, the second is somewhat efficient, and the third is fully optimized. ULP Advisor and EnergyTrace technology are used to gather information about the efficiency of the application and point out potential areas of improvement after each revision of the code. By the end of this study the value of these energy optimization tools will have been clearly demonstrated.

Project collateral and source code discussed in this application report can be downloaded from the following URL: http://www.ti.com/lit/zip/slaa603.

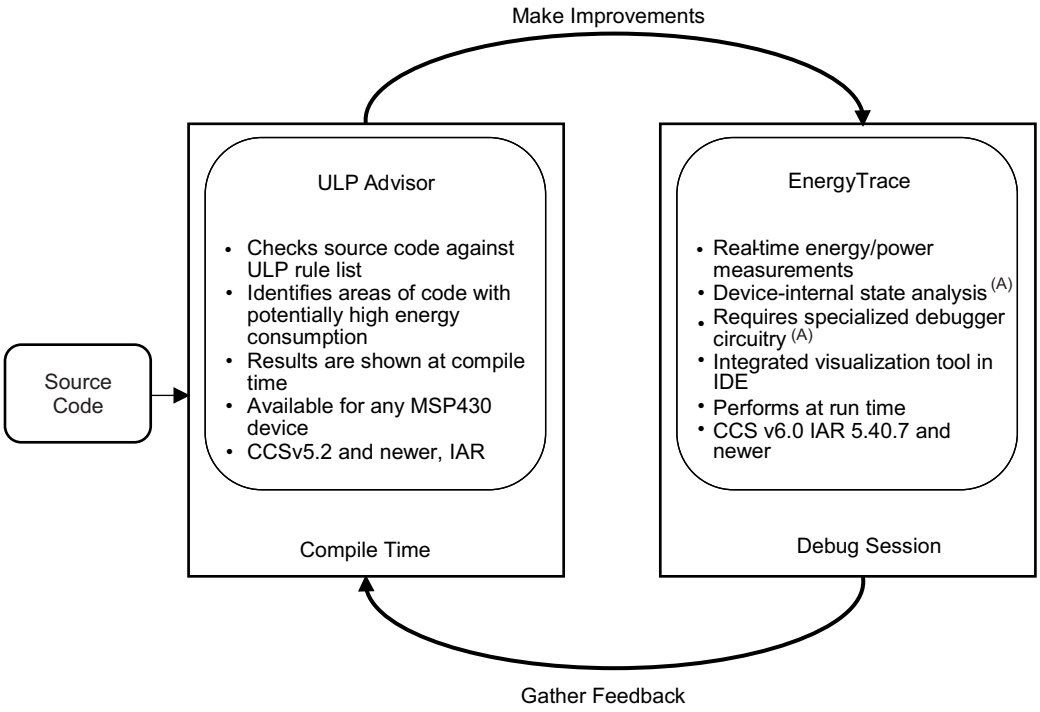
Contents

List of Figures

List of Tables

1 Development Tool Overview – ULP Advisor and EnergyTrace

Texas Instruments offers multiple tools for optimizing the energy consumption of your application. ULP Advisor provides advice on how to improve the energy efficiency of your application based on comparing your code with a list of ULP rules at compile time. EnergyTrace technology is a tool that enables power and energy based program code analysis during a debug session. This includes real-time monitoring of a multitude of internal device states during run time. Figure 1 demonstrates the integration of ULP Advisor and EnergyTrace Technology in the development process.



A For details on using EnergyTrace in IAR, see the *IAR Embedded Workbench™ Version 3+ for MSP430™ User's Guide* (SLAU138).

Figure 1. ULP Advisor and EnergyTrace Technology

2 ULP Advisor

2.1 Tool Overview

ULP Advisor software can be used at compile time to draw attention to inefficient code and help the developer to fully utilize the ULP capabilities of MSP430 microcontrollers. This tool works at build time to check your code against a list of ULP rules and identify possible areas where energy and power use is inefficient. A description of the ULP rule in violation, a link to the ULP Advisor wiki page, links to relevant documentation, code examples and forum posts are all included for each rule violation in the application code. Once the feedback has been presented, the developer can then learn more about the violation and go back to edit the code or continue to run the code as-is. For a full list of the ULP rules, see the ULP Advisor wiki page at http://www.ti.com/ulpadvisor.

2.2 Required Software

The ULP Advisor is built into Code Composer Studio™ IDE version 5.2 and newer, and it is included in the IAR v5.40.7 and newer. This tool is automatically installed and enabled by default. ULP Advisor supports every MSP430 device.

2.3 ULP Advisor in CCS

After compiling the code, a list of the ULP rule violations is provided in the *Advice* window. Unlike errors, these suggestions will not prevent your code from successfully compiling and downloading.

3 EnergyTrace Technology

3.1 Energy Measurement Method

Power is traditionally measured by amplifying the signal of interest and measuring the current consumption and voltage drop over a shunt resistor at discrete times. EnergyTrace technology implements a new method for measuring power. In debuggers that support EnergyTrace technology, a software-controlled DC-DC converter generates the target power supply (1.2 V-3.6 V). The time density of the DC-DC converter charge pulses equals the energy consumption of the target microcontroller. A built-in calibration circuit in the debug tool defines the energy equivalent for a single charge pulse. The width of each charge pulse remains constant. The debug tool counts every charge pulse and the sum of the charge pulses are used in combination with the time elapsed to calculate an average current.

Since this measurement technique continually samples the energy supplied to the microcontroller, even the shortest device activity that consumes energy contributes to the overall recorded energy. This is a clear benefit over shunt-based measurement systems, which cannot detect extremely short durations of energy consumption.

3.2 Required Hardware and Software

EnergyTrace technology is included in Code Composer Studio version 6.0 and newer. It requires specialized debugger circuitry, which is supported with the second-generation on-board eZ-FET flash emulation tool and second-generation standalone MSP-FET JTAG emulator. Target power must be supplied through the emulator when EnergyTrace is in use.

The *EnergyTrace*++ mode (see Section 3.3.2) is only supported with selected devices, such as the MSP430FR58xx and MSP430FR59xx, which have additional circuitry within the silicon itself.

The MSP-EXP430FR5969 LaunchPad (revision 2.0) is the first board equipped with EnergyTrace++ technology circuitry in the eZ-FET on-board emulation. Some examples of devices and emulators that do not support EnergyTrace technology are the MSP-EXP430G2 LaunchPad, the first-generation on-board eZ-FET emulator, and the first-generation standalone MSP-FET430UIF JTAG emulator.

3.3 Energy Capture Modes

There are two energy capture mode capabilities included: Energy Trace and EnergyTrace+[CPU State]+[Peripheral State].

3.3.1 EnergyTrace

This mode allows the standalone use of the energy measurement feature with all MSP430 microcontrollers, even unsupported devices (meaning there is no built-in EnergyTrace++ technology circuitry in the target microcontroller). The supply voltage on the target microcontroller is continuously sampled to provide you with energy and power data. This mode can be used to verify the energy consumption of the application without accessing the debugger. For more details on how to use this mode, see Section 4.10 and Section 4.11.

3.3.2 EnergyTrace+[CPU State]+[Peripheral State]

When debugging with devices that contain the built-in EnergyTrace++ technology support, the EnergyTrace++ mode yields information about energy consumption as well as the internal state of the microcontroller. These states include the ON/OFF status of the peripherals and all system clocks (regardless of the clock source) as well as the low power mode (LPM) currently in use (see Table 1).

Table 1. Specified States

Type	State
Power Mode	LPM0 to LMP4.5, Active Mode
Peripheral States	On or Off
System Clock States	On or Off

EnergyTrace++ constantly reads digital information from the target microcontroller to gather information about the internal states. The debugger can always read these states, even when the MCU sleeps in LPMx.5 mode. This tool provides a means of directly verifying whether an application is demonstrating the expected behavior at the correct points in the code, such as ensuring that a peripheral is turned off after a certain activity. For instance, the specified peripherals are FRAM, 32-bit hardware multiplier, watchdog timer, real-time clock (RTC), analog-to-digital converter, reference module, comparator, AES accelerator, enhanced universal serial communication interfaces, timers, and direct memory access. To see which peripherals can be monitored, see the device-specific data sheet.

3.4 EnergyTrace in CCS

To enable or disable EnergyTrace++, select or deselect *Enable* under *Window → Preferences → Code Composer Studio → Advanced Tools → EnergyTrace Technology* (see Figure 2). Also select the *EnergyTrace+[CPU State]+[Peripheral States]* mode. Additional windows become available during debug mode if EnergyTrace++ is enabled and the emulation in use supports EnergyTrace++. These windows include *EnergyTrace++*, *Power*, and *Energy*. The *States* window is only available in EnergyTrace++ mode (see Table 3). Control buttons are accessible in the *EnergyTrace++* window. Table 2 describes these buttons.

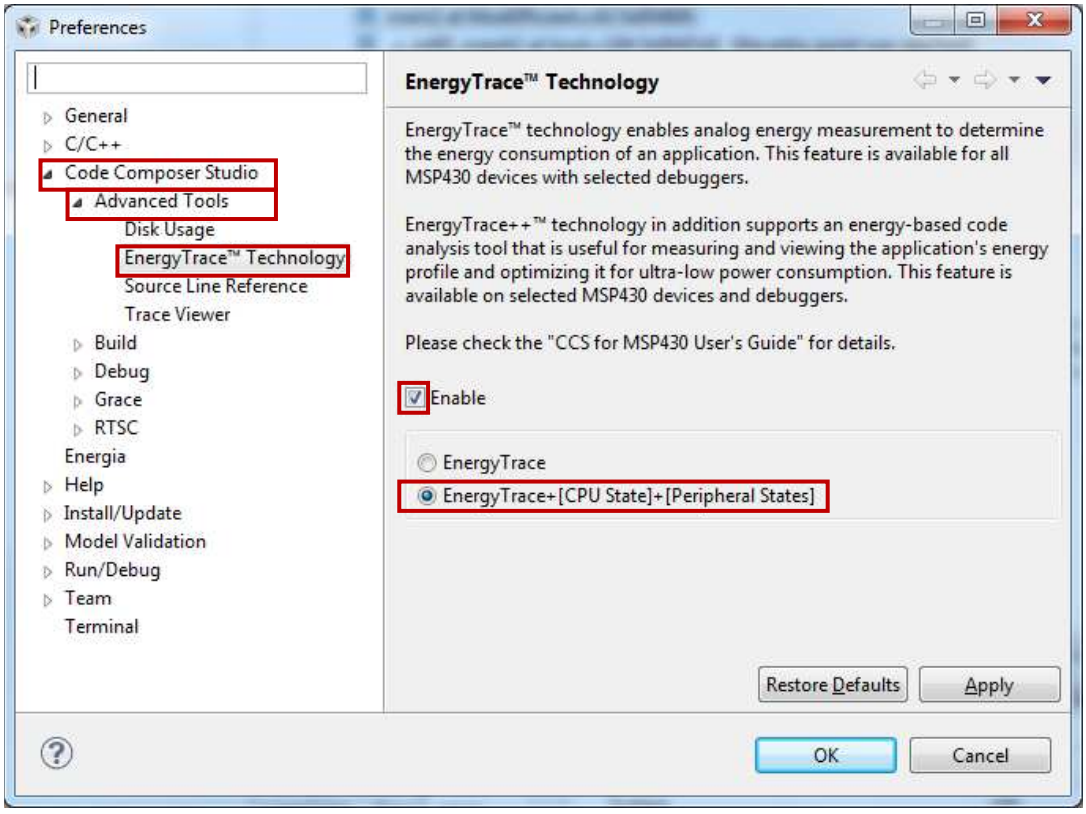


Figure 2. Enable or Disable EnergyTrace++

Table 2. EnergyTrace++ Window Control Buttons

Button	Description	Function
	Start and Stop EnergyTrace++	Toggles between Start and Stop EnergyTrace++. If started, all relevant EnergyTrace++ windows are available and the button is blue in color. If stopped, only the “EnergyTrace” window is open and a gray button is visible.
	Set Measurement Duration	Drop-down menu: select current profile time duration.
	Save Current Energy profile	Save current profile.
	Load Reference Energy Profile	Load a previously saved reference profile.
	Advanced Menu	Drop-down menu: edit the EnergyTrace++ preferences of the current project.
	Switch to EnergyTrace	Toggles between EnergyTrace and EnergyTrace++ modes. “States” window opens and closes appropriately.
	Minimize or maximize	Collectively minimize (left) or maximize (right) the EnergyTrace++ windows.

Table 3. Windows in CCS

Window	Contents	EnergyTrace (All MSP430 Devices)	EnergyTrace ++ (Selected Devices)
EnergyTrace Technology	Control interface for EnergyTrace using buttons to the right. Buttons only available when EnergyTrace is not actively sampling the target. Contains *Profile* tab.	√	√
Power	Plot of dynamic power consumption over time.	√	√
Energy	Plot of energy versus time.	√	√
States	Plot of internal state versus time. Only available in Energy and States capture mode.		√

3.5 Limitations

There are a few constraints concerning EnergyTrace++ technology. First off, the analog sampling of energy consumption and digital sampling of digital states is slower than the microcontrollers CPU speed. High-speed trace capability is not available. Specifically, when using the eZ-FET on-board emulator via Spy-bi-Wire the sampling frequency is 1 kHz, and the MSP-FET via JTAG runs at 10 kHz. This means that some information may be lost due to rapid code instruction execution and associated device state changes.

Next, there is no direct correlation between EnergyTrace++ data and a single line of code or executed instruction. A remedy for this mild inconvenience is to analyze statistical profiling over time. Also, EnergyTrace++ gathers relative energy and power consumption information (not absolute information) to avoid showing the impact of the debugger’s JTAG operation. For information on how to obtain absolute results, see the “Absolute Power Measurements” section of the case study.

For more details on EnergyTrace technology and a list of FAQs, see the *EnergyTrace Energy Aware Debugger* section in the *Code Composer Studio User’s Guide for MSP430* (SLAU157) or the integrated online-help in CCS via the Welcome View.

4 Case Study

This section explains in detail how to use ULP Advisor and EnergyTrace technology. Both the MSP-EXP430FR5969 and MSP-EXP430G2 are required for this case study. Three code files are included in this project (see Table 4), but only one file is included in the build at a time. Each file performs the same function: every second an Analog-to-Digital Converter (ADC) temperature measurement is taken, the degrees Celsius and Fahrenheit are calculated, and the results are sent through the backchannel Universal Asynchronous Receiver/Transmitter (UART) at 9600bps.

The first case study begins with an inefficient implementation of this application. ULP Advisor and EnergyTrace technology are used to observe the extreme inefficiency of this version. Based on the feedback from these two tools, improvements are made for the somewhat efficient second version of the code. The process is repeated, and ULP Advisor and EnergyTrace technology are used to compare this improved version with the original. Lastly, final steps are taken to improve the code even further in the very efficient third version of the code. Other tips and details are provided throughout.

The topics included in the case study are:

- Building the project
- ULP Advisor
- EnergyTrace Technology Integrated in CCS
- EnergyTrace Technology Reference Profiles
- LPM Considerations
- Hiding Unavoidable ULP Rule Violations
- Absolute Power Measurements
- EnergyTrace With Unsupported Devices

Table 4. Code Descriptions

File Name	Description
Inefficient.c	First draft of the application code. ULP Advisor and EnergyTrace++ identify an abundance of inefficiencies.
Efficient.c	Some improvements are implemented. However, ULP Advisor and EnergyTrace++ identify still more inefficiencies.
MostEfficient.c	Fully optimized version of the code.

4.1 Download and Import the Example Project 'ULP_Case_Study'

1. Download the CCS example project ULP_Case_Study from http://www.ti.com/lit/zip/slaa603 and extract the archive.
2. Import the project into your CCS workspace:
   (a) Click *Project > Import Existing CCS Eclipse Project*.
   (b) Browse for the directory where you downloaded the project by clicking the *Browse* button. Click *OK*.
   (c) Check the box next to the 'ULP_Case_Study' project in the *Discovered projects* window.
   (d) Un-check the box next to *Copy projects into workspace*.
   (e) Click *Finish* (see Figure 3). The project appears in your *Project Explorer* window.

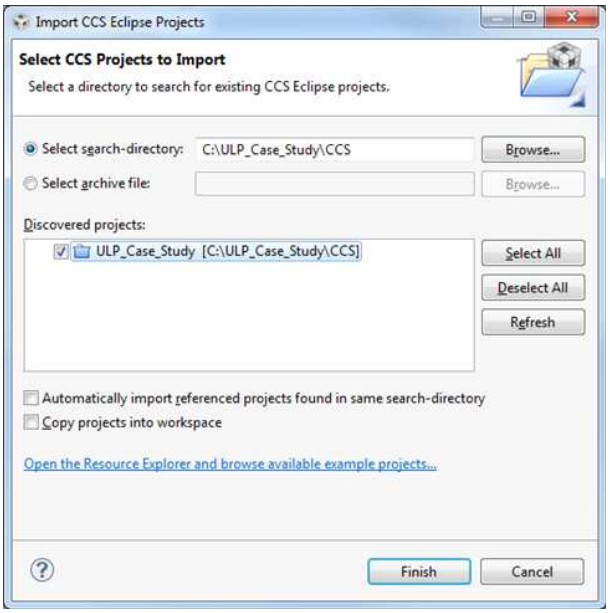


Figure 3. Import the Case Study Project

3. Set as active project by left-clicking on the project name. The active build configuration should be *Inefficient* (see Figure 4). If not, right click on the project name in the *Project Explorer*, then click *Build Configurations* → *Set Active* → *Inefficient* (see Figure 5). The file Inefficient.c is the only .c file included in this build.

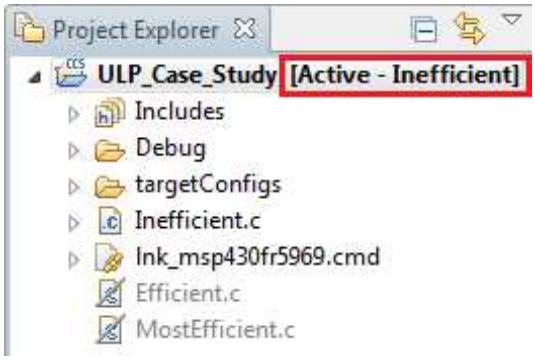


Figure 4. Active Debug Configuration: Inefficient

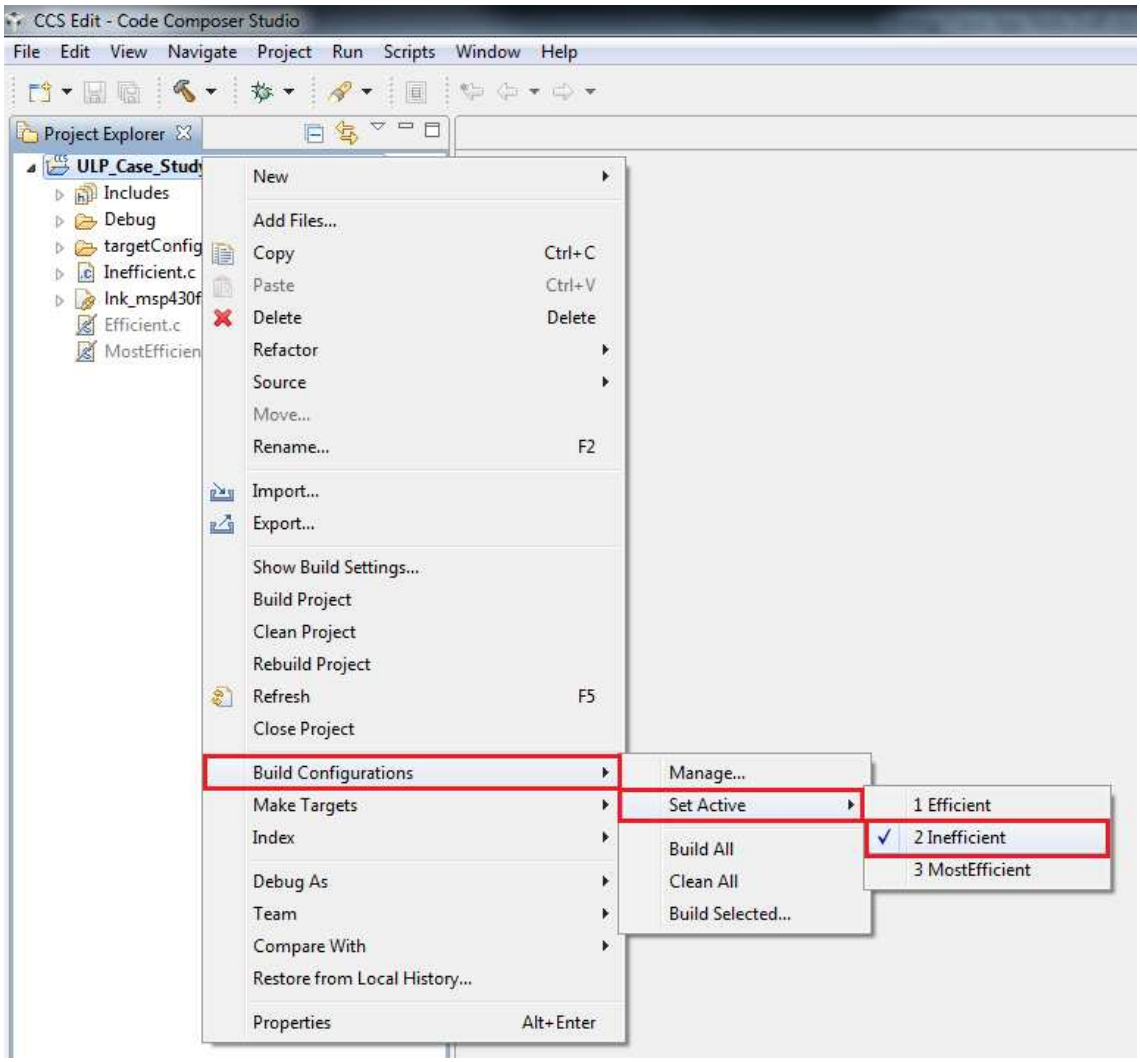


Figure 5. Set Active Debug Configuration

4.2 Build the Project

Right click the project name, then click *Build Project* in the provided menu. Alternatively, click the hammer icon. 

4.3 Obtain Feedback From ULP Advisor

1. There are no errors during compilation, so the *Problems* window is empty.
2. The *Advice* window shows suggestions from the ULP Advisor under the Power (ULP) Advice heading. If you do not see the *Advice* window, click *View* → *Advice* (see Figure 6). Figure 7 shows the comments for Inefficient.c.

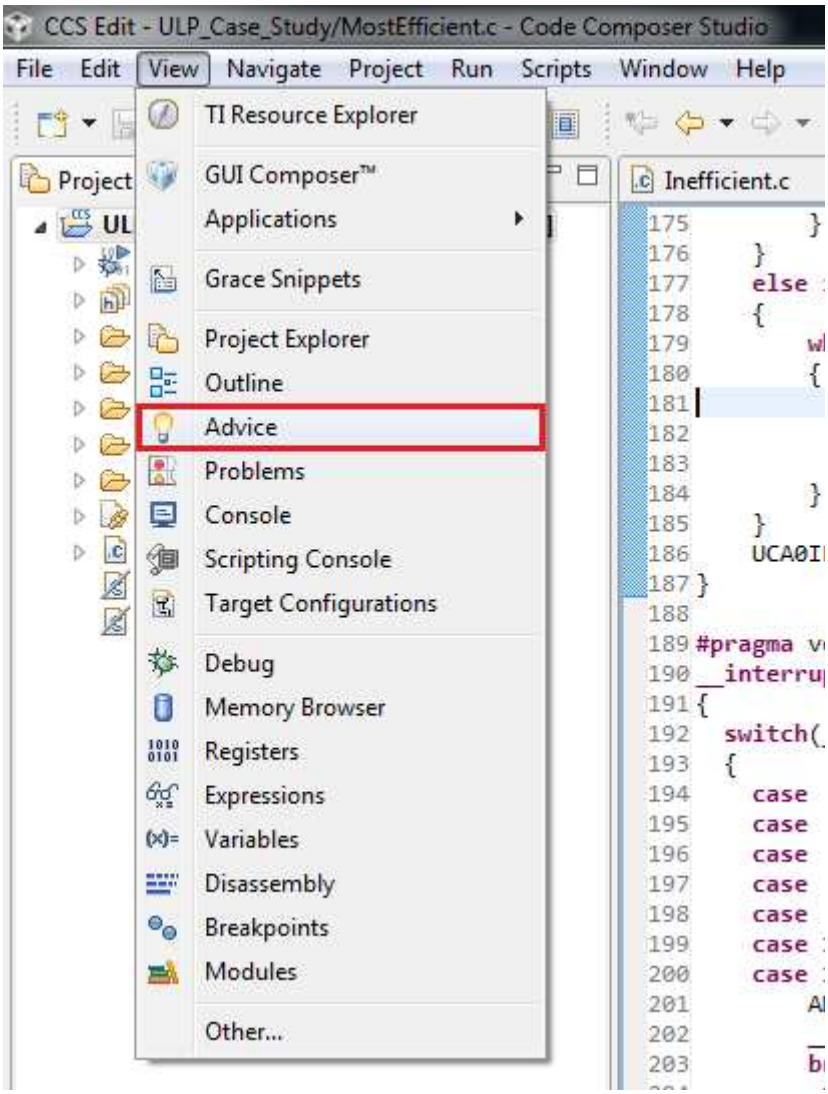


Figure 6. Show the Advice Window

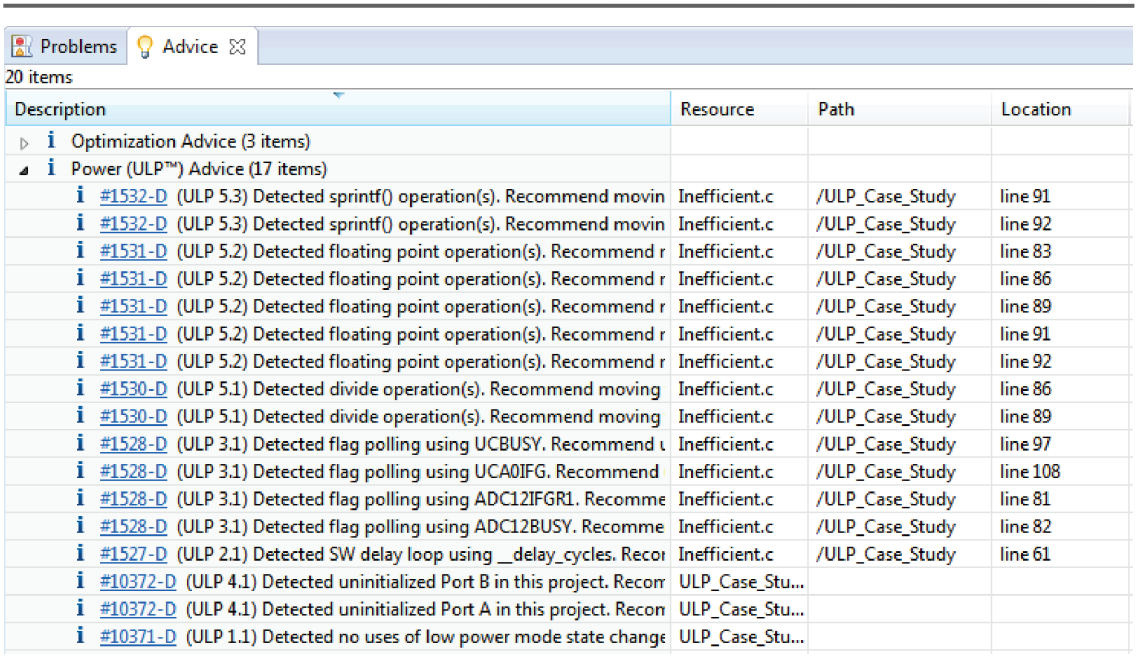


Figure 7. ULP Advisor Feedback for Inefficient.c

3. Expand the columns as needed to view more of the *Description* column.
4. Click the link #1532-D, which corresponds to the first ULP violation (for using *sprintf()*), to open the ULP Advisor wiki page in a second tab titled *Advice* (see Figure 8). All of the information for this particular rule violation is provided.
5. Scroll down to the *Remedy* section. The first suggestion is to avoid using *sprintf()* altogether and work with the raw data. The next version of the code, Efficient.c, implements this advice.

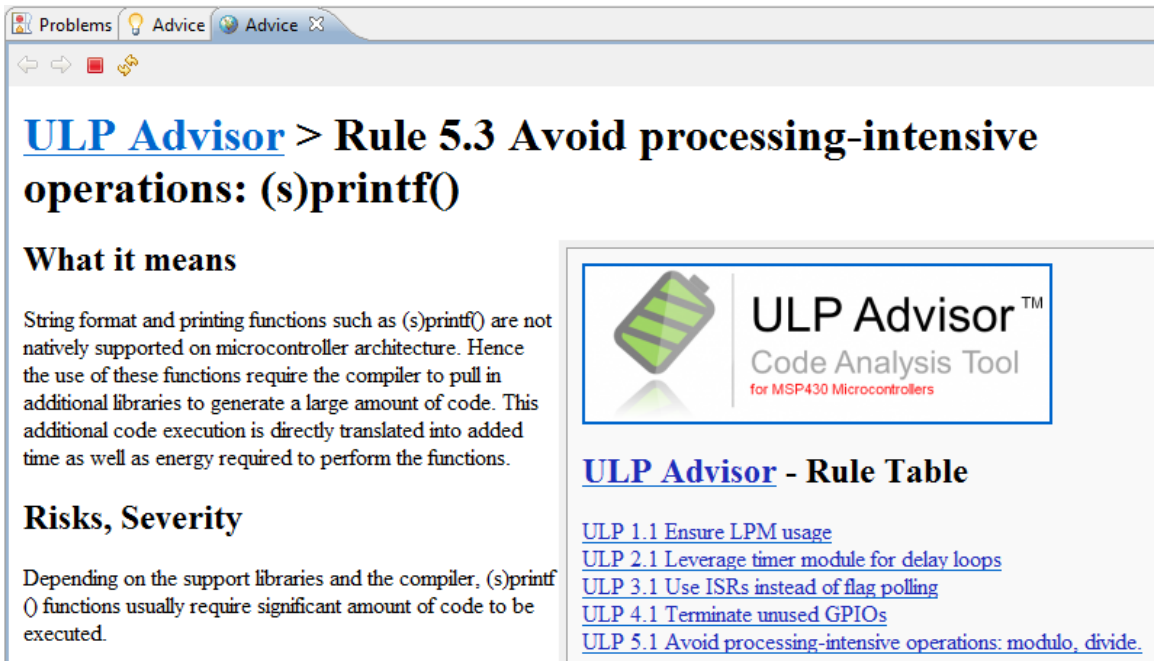
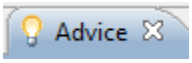


Figure 8. ULP Advisor Wiki Page

6. Return to the ULP Advice tab 

and investigate more advice as desired. Note that several of the recommendations are listed more than once, indicating that the corresponding rule was violated multiple times in the code. Observe the *Location* column for the exact details on where to find each issue in the code, and double-click the violation to jump to that line.
7. The following list shows all of the unique rule violations:
   - (ULP 5.3) Detected *sprintf()* operations.
   - (ULP5.2) Detected floating point operations.
   - (ULP 5.1) Detected divide operations.
   - (ULP 3.1) Detected flag polling.
   - (ULP 2.1) Detected software delay loop using __delay_cycles.
   - (ULP 4.1) Detected uninitialized port in this project.
   - (ULP 1.1) Detected no uses of low-power mode state changes using LPMx or *_bis_SR_register()* or *__low_power_mode_x()* in this project.

4.4 Create a Reference Profile Using EnergyTrace Technology

To evaluate improvements that are made in the code, make a reference of the starting point.

1. Make sure that EnergyTrace is enabled.
   (a) Click *Window* → *Preferences*, expand *Code Composer Studio* → expand *Advanced Tools* → *EnergyTrace Technology*.
   (b) Make sure that the box is checked next to *Enable.*
   (c) Select the EnergyTrace++ mode (see Figure 2).
   (d) Click *Apply* and *OK*.
2. Enable the debugger in LPMx.5 modes, if it is not enabled. This should be enabled by default.
   - Right click on the project in the project explorer and then click *Properties* → *Debug* → *MSP430 Properties*.
   - Scroll down and check the box for *Low Power Mode Settings* next to *Enable Ultra Low Power debug / LPMx.5 debug* (see Figure 9).
   - Click *OK*.

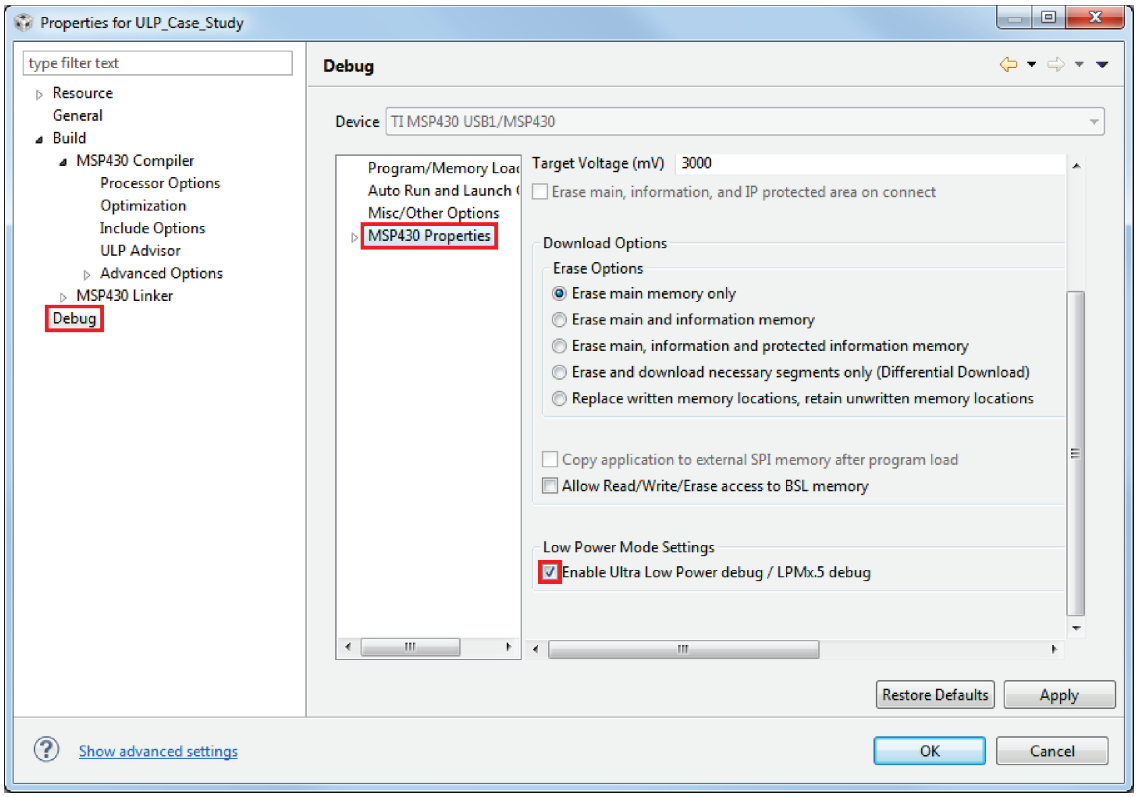


Figure 9. Enable ULP Debug

3. Remove the jumpers on the MSP-EXP430FR5969 for RTS and CTS in J13. This code does not use these signals, and keeping them connected draws slightly more power (see Figure 10).

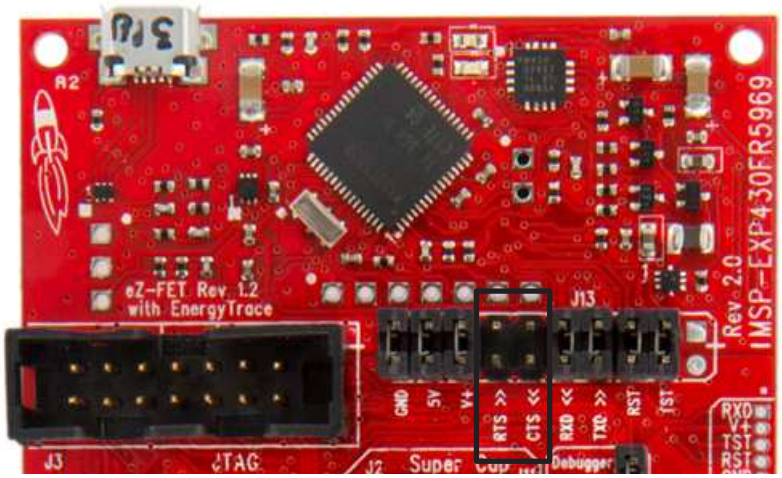


Figure 10. Remove RTS and CTS Jumpers From MSP-EXP430FR5969

4. Enter a debug session. Right click the project, select *Debug As, CCS Debug Session*. Alternatively, click the debug icon. 

   If the EnergyTrace window does not open by default, Click *View* → *Other* → expand *MSP430-EnergyTrace* → *EnergyTrace Technology*, and click *OK* (see Figure 11).

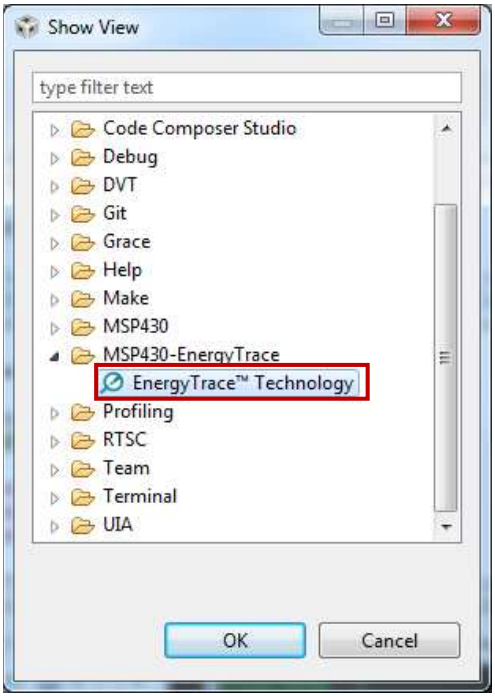


Figure 11. Show EnergyTrace Technology in Debug Mode

5. **Set a breakpoint at line 105**, which is after all the initializations and just before the main loop. Run the code to this point by clicking the yellow and green arrow  . This is where data collection should start (after skipping the initializations).
6. When the code is paused, go to the *EnergyTrace Technology* tab and adjust the capture duration time as desired by using the *Set measurement duration* drop-down menu button  . The default is 10 seconds.
7. Run the code and wait for the capture duration time to pass. The *EnergyTrace Technology* windows update with real-time data.
8. The *EnergyTrace Technology* window shows runtime and energy data for the low-power modes and each function used during Active Mode (see Figure 12). Runtime information is given for peripheral and clock states. The functions collected under *_RTS_* correspond to functions in the runtime system library.

EnergyTrace™ Technology | Power | Energy | States

EnergyTrace++™ Profile

Name	Runtime (%)	Energy (%)
System	100	100
CPU		
Active Mode	100	100
main	92.3	91.9
RTS	7.2	7.5
__mspabi_mpyl_f5hw	4.5	4.6
__TI_frcdivd	0.4	0.4
__mspabi_divul	0.3	0.3
__TI_frcmpyd	0.3	0.3
__mspabi_addd	0.2	0.2
__TI_printfi	0.2	0.2
__mspabi_mpyd	0.2	0.2
__mspabi_srll	0.1	0.1
__TI_frcaddd	0.1	0.1
__mspabi_sral_13	0.1	0.2
__mspabi_fixdu	0.1	0.1
__mspabi_cmpd	0.1	0.1
__mspabi_srll_1	0.1	0.1
__mspabi_sral_1	0.1	0.1
__TI_renormd	0.0	0.1
__mspabi_fixdli	0.0	0.0
__mspabi_slll_11	0.0	0.0
__mspabi_divli	0.0	0.0
__mspabi_slll_1	0.0	0.0
mspabi_fltlid	0.0	0.0

Figure 12. Inefficient.c Energy Profile

9. Click the *Power* tab (see Figure 13). Notice that the power consumption spikes up about every second, which corresponds to taking and processing the ADC temperature measurement. Be aware that the measurement taken by EnergyTrace is an energy measurement, and the power results are calculated from the accumulated energy value (why the power graph is not smooth).

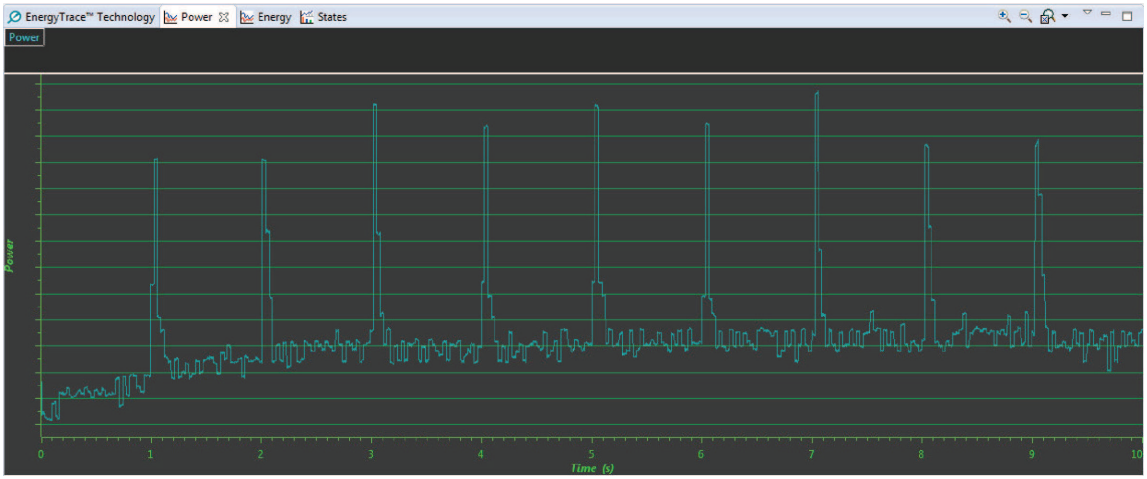


Figure 13. Inefficient.c Power

10. Click the *Energy* tab (see Figure 14). To zoom in, click and drag across the horizontal (time) axis or the vertical (energy) axis. To reset to the original view, click the *Reset Zoom* or *Select Zoom Options* button  .

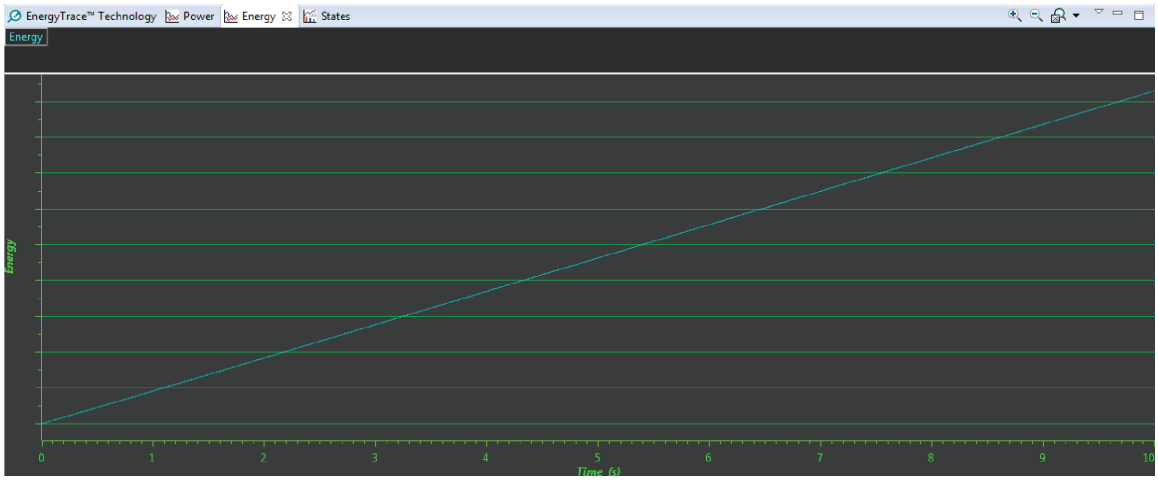


Figure 14. Inefficient.c Energy

11. Notice that the *Power and Energy* graphs do not have units on the y-axis. During *Energy and States* capture mode, the debugger is constantly accessed, which continuously draws power. As a result, absolute power and energy readings would be misleading. Instead, the plots are shown by reference.
12. Click the *States* tab (see Figure 15).
13. Scroll up and down to observe the states of the power modes, peripherals, and system clocks.

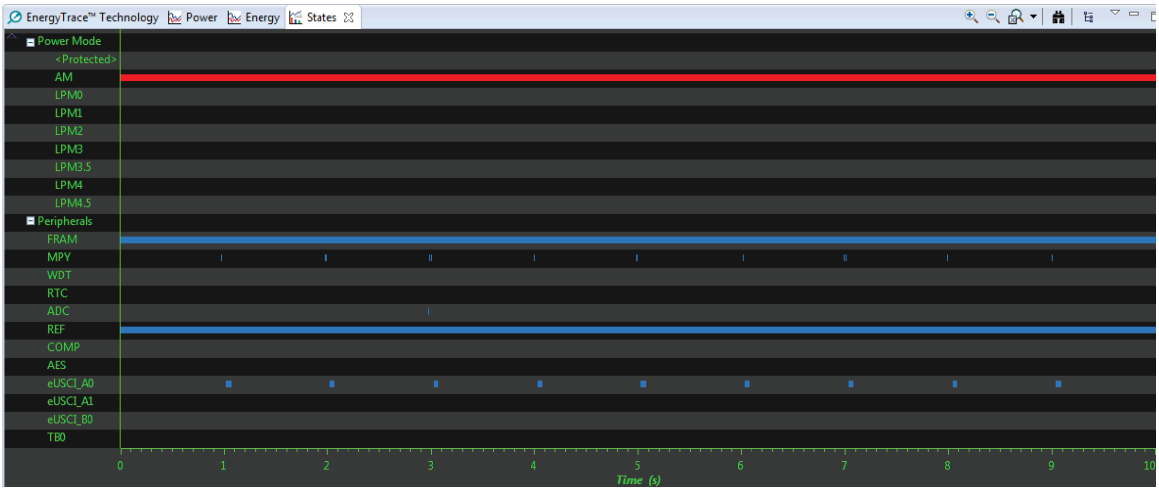


Figure 15. Inefficient.c States

14. In the *EnergyTrace Technology* tab, click the *Save* button  . Rename the file Inefficient.profxml. This is used as a reference profile.
15. Exit the debug session by clicking the red square icon. 

4.5 Improve the Code and Compare Profiles

The following steps are very similar to the previous section, with a few differences that are in **bold** font.

1. Change the active build configuration to *Efficient* (see Figure 5). Efficient.c is the only .c file included in this build.
2. View the comments at the top of Efficient.c and throughout the code for details on the improvements made from Inefficient.c.
3. Build the project. 
4. Look at the ULP *Advice* window. Notice that only a subset of the recommendations remain after making some improvements in the code. The following list includes the remaining unique advice:
   - (ULP 13.1) Detected loop counting up.
   - (ULP 5.1) Detected divide operation(s).
   - (ULP 3.1) Detected flag polling.
5. Enter a debug session.
6. **Set a breakpoint at line 116**, which is after all the initializations and before the main loop. Run the code to this point.
7. **Once the code is paused, go to the *EnergyTrace Technology* tab and adjust the capture duration time to match the reference profile. If the time durations do not match, EnergyTrace will display power and energy plots according to the current energy profile capture time (not in congruence with the reference profile). In contrast, the *EnergyTrace Technology* tab provides percentage data according to each profile individually.**
8. Run the code and wait for the capture time duration to pass.
9. **Open the reference profile**:
   - Click the *Open* button in the *EnergyTrace* tab.
   - Select the file Inefficient.profxml and click *Open*.
   - Now you can directly compare the original code with the improved code (see Figure 16). 
10. Notice in the *Power* window that the current profile power consumption (shown in blue) is consistently less than the reference (shown in yellow) (see Figure 17). Also note in the *Energy* window that the energy consumption now increases at a slower rate than the reference profile (see Figure 18). This is a good sign, showing that the code adjustments have made a positive impact on the energy efficiency of the system. The *States* window does not show a comparison of the current states to a reference profile; only the states corresponding to the current profile are shown.

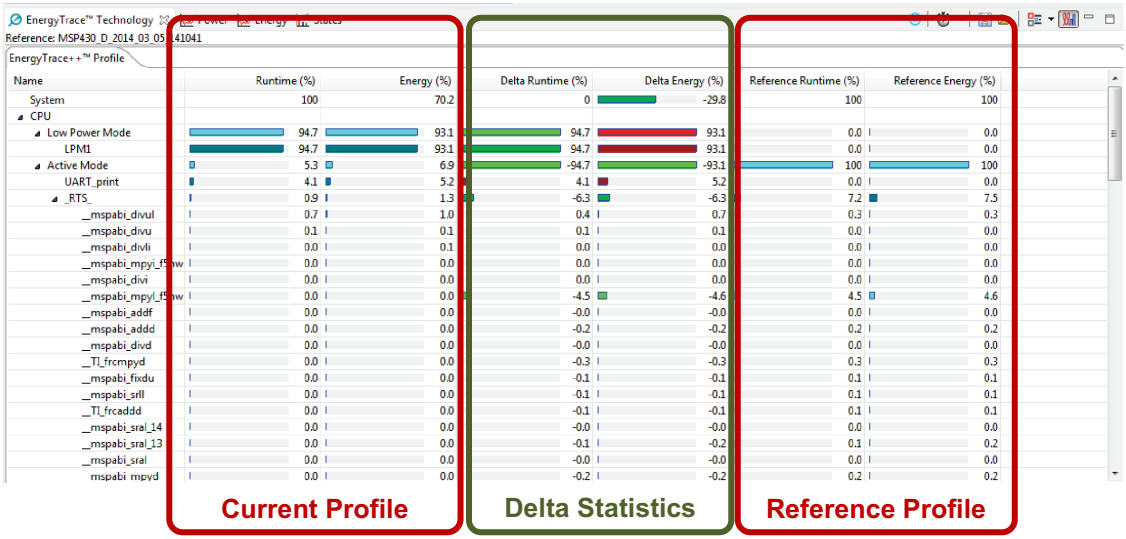


Figure 16. "Efficient.c" Referenced With "Inefficient.c"

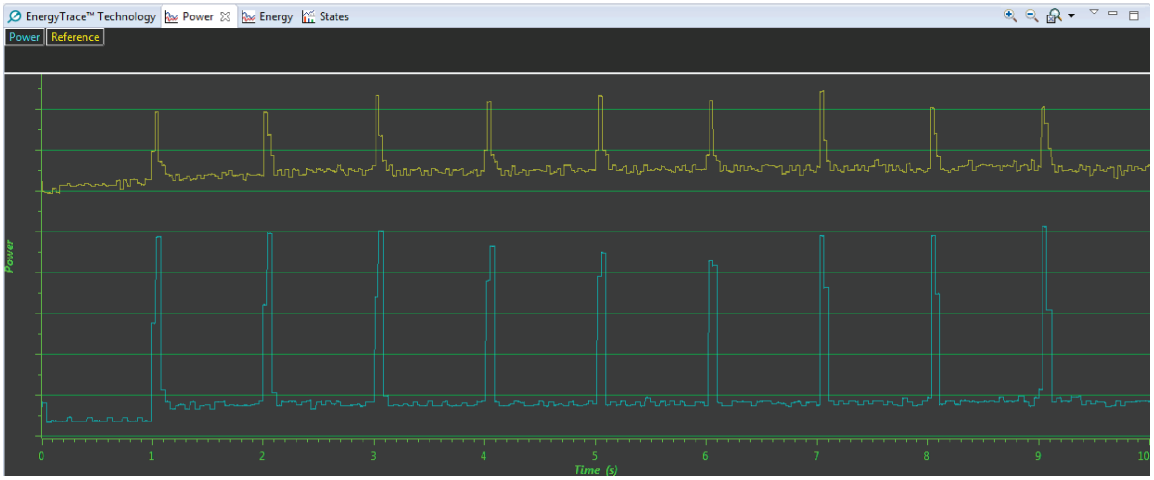


Figure 17. Efficient.c Power (Blue) Referenced With Inefficient.c Power (yellow)

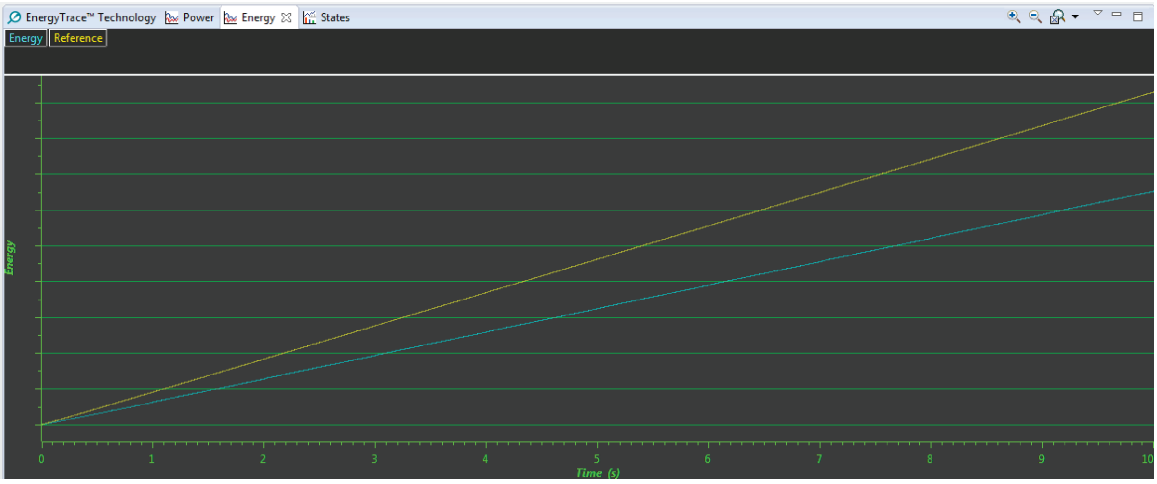


Figure 18. "Efficient.c" Energy (blue) Referenced With "Inefficient.c" Energy (yellow)

4.6 Pay Special Attention to Low-Power Mode Use

1. Look at the *States* tab. Notice anything interesting? According to the code (lines 89 and 92), the MCU is entering LPM3, but the states view only shows LPM1 (see Figure 19).

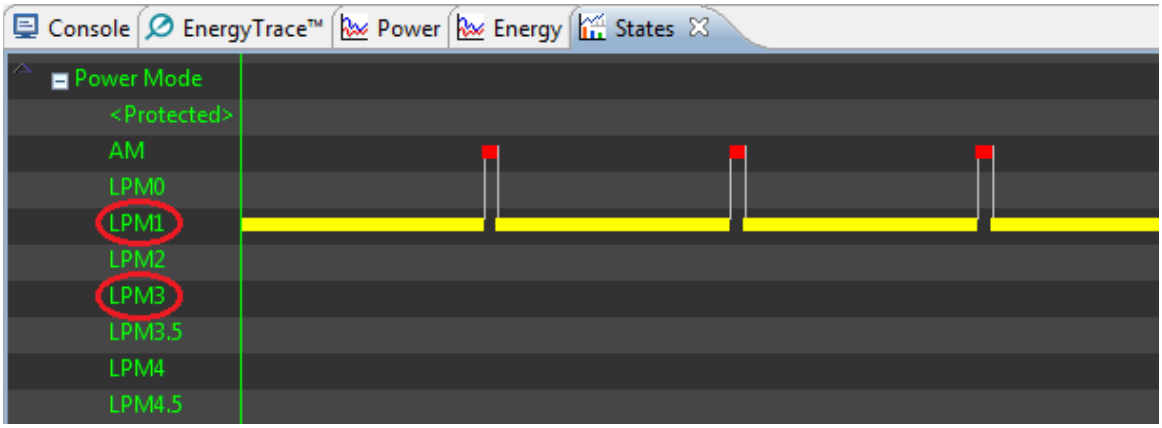


Figure 19. Efficient.c States

This may appear to be an error, but it is actually an interesting technicality that must be addressed.

The *Operation Modes* table in the *MSP430FR58xx, MSP430FR59xx, MSP430FR68xx, and MSP430FR69xx Family User's Guide* (SLAU367) shows the status of all the clocks during each low-power mode. MCLK and SMCLK are both disabled in LPM3. Looking back to the code, notice that the ADC, Timer, and UART all use SMCLK (lines 61, 72, and 80, respectively). Therefore, a request is made to keep SMCLK running, which affects the low-power mode.

According to the *MSP430FR58xx, MSP430FR59xx, MSP430FR68xx, and MSP430FR69xx Family User's Guide* (SLAU367), the deepest low-power mode that still allows SMCLK to remain on is LPM1. Consequently, the application as-is only reaches LPM1.

This LPM1 status can be verified by setting a breakpoint at the beginning of an ISR and observing the status register (SR) in the *Register* window. To view this window, click *View* → *Registers*. The SR is nested under the *Core Registers*.

Thanks to EnergyTrace, a weakness in the code has successfully been identified: the MCU does not enter the expected low-power mode. According to the user's guide, to make the MCU enter LPM3, it must use ACLK for all peripherals. This improvement is made in the next revision of the case study code.

For more information on peripheral clock requests, how to enable or disable them, and the effect on low-power modes, see the *Operation From Low-Power Modes, Requested by Peripheral Modules* section in the *MSP430FR58xx, MSP430FR59xx, MSP430FR68xx, and MSP430FR69xx Family User's Guide* (SLAU367).

2. In the *States* tab, scroll down to the system clocks. Notice that the MODOSC is running throughout the entire code (see Figure 20), yet it is never used. Therefore, this clock should be turned off in the next revision of the code.

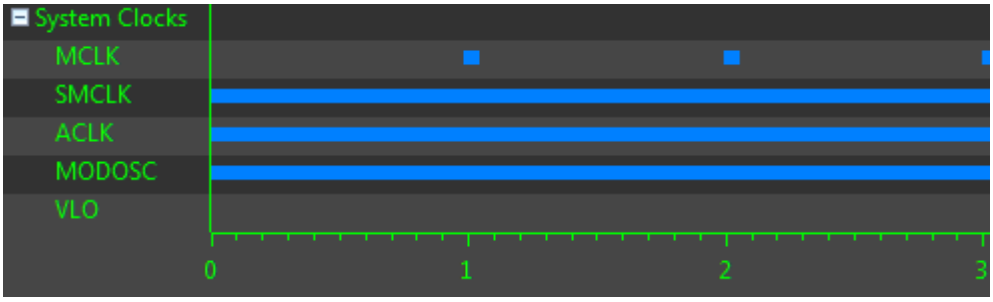


Figure 20. Efficient.c States

3. Save this energy profile using the same method as the inefficient profile. Save the file as Efficient.profxml. 
4. Exit debug mode. 

4.7 Make Final Code Improvements

The following steps are again very similar to the previous iterations.

1. Change the active build configuration to *MostEfficient* (see Figure 5). MostEfficient.c is the only .c file included in this build.
2. View the comments at the top of MostEfficient.c and throughout the code for details on the improvements made from Efficient.c.
3. Build the project. 
4. Notice that only one unique ULP suggestion remains in the *Advice* window:
   - Detected divide operation(s)

     The divide operations used in the function rawToAsciiString(int16_t input) have been replaced with shifts and addition. However, two divisions are still required for converting the raw ADC values to degrees Celsius and Fahrenheit. There is no straightforward way to avoid this division, so according to the ULP suggestion the best option would be to move the calculations to RAM during run time. Another option would be to defer the calculations to the host computer (because it has more processing power) by sending the raw ADC values to the PC via the UART and completing the conversion at that time. These topics are out of the scope of this application report, but for more details and instructions simply follow the link in the *Advice* window corresponding to the division rule violation.

4.8 Hide Unavoidable ULP Rule Violations (Optional)

You must have full understanding of how the code works interpret ULP advice with inquisition and care. At this point, the remaining divide operations are essentially unavoidable and the ULP advice for a subset of the calculations is being overlooked. It is sometimes desired to hide a rule that you know you cannot further optimize. This keeps the ULP *Advice* window less cluttered and more focused on the remaining issues to address. To disable the division rule, follow these steps:

1. Right click the project name (EnergyTrace_Case_Study) → left click *Properties* →> expand *Build* → expand *MSP430 Compiler* → click *ULP Advisor*.
2. Expand number 5 in the *Enable checking of ULP power rules* window.
3. Uncheck rule 5.1 (Detected division or modulus operations) (see Figure 21).

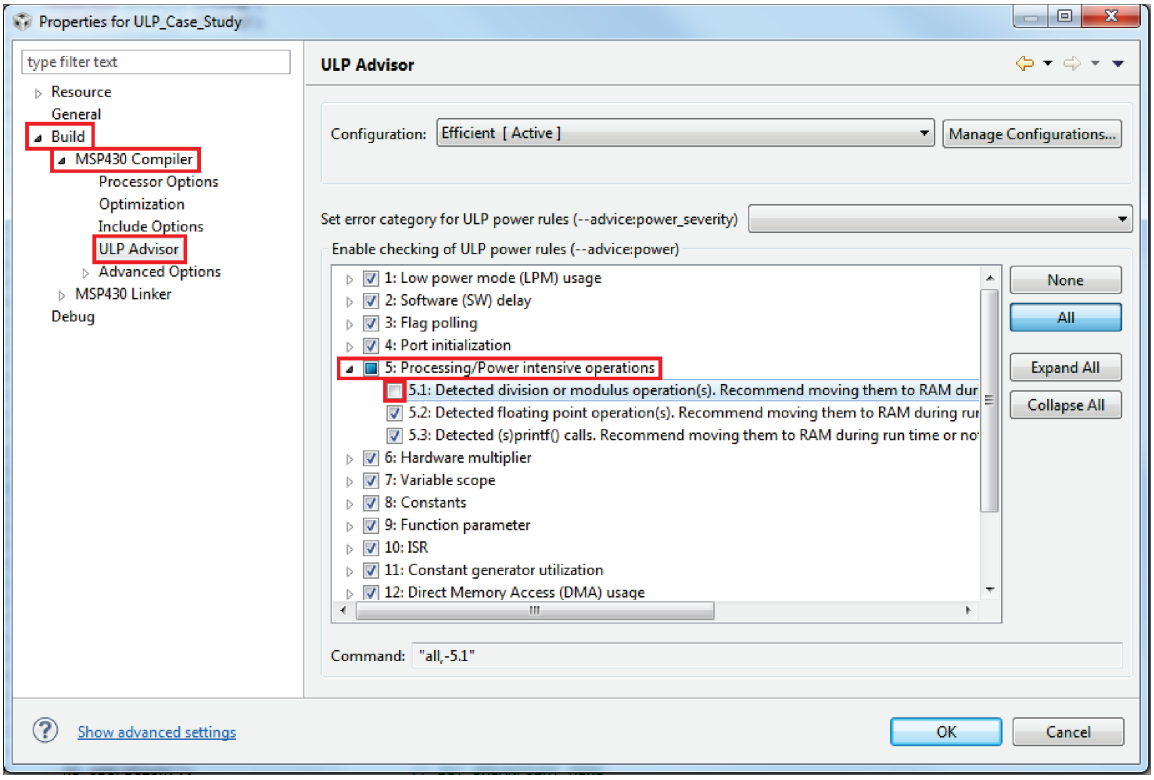


Figure 21. Hide ULP Advisor Rule 5.1

4. Click *OK*.
5. Rebuild the project.  The corresponding ULP advice no longer appears in the *Advice* window.

This process can be repeated for any other ULP rule that you choose to ignore, depending on the specific application.

4.9 Compare All Profiles

1. Enter a debug session.
2. **Set a breakpoint at line 130**, which is after all the initializations and just before the main loop. Run the code to this point.
3. When the code is paused, go to the *EnergyTrace Technology* tab and adjust the capture duration time to match the reference profiles.
4. Run the code and wait for the capture time duration to pass.
5. Pause the debugger. Save this profile as MostEfficient.profxml. 
6. Go to the *States* window. Notice that LPM3 is now correctly implemented (see Figure 22). Also, SMCLK and MODOSC no longer run.

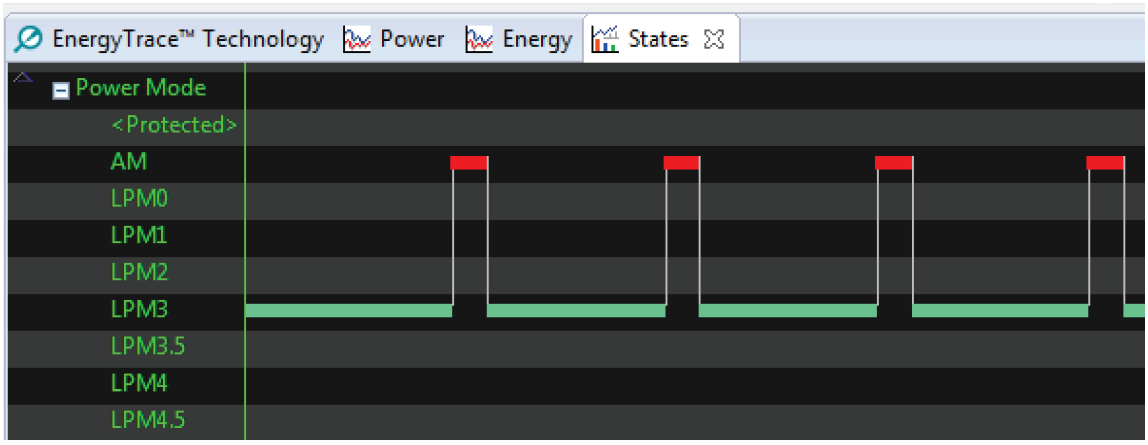


Figure 22. MostEfficient.c States

7. Return to the *EnergyTrace Technology* window and open the reference profile Efficient.profxml. 
8. Notice the improvements by exploring the *EnergyTrace* tabs. First, the energy consumption decreased significantly with the most efficient code (see Figure 23 through Figure 25).

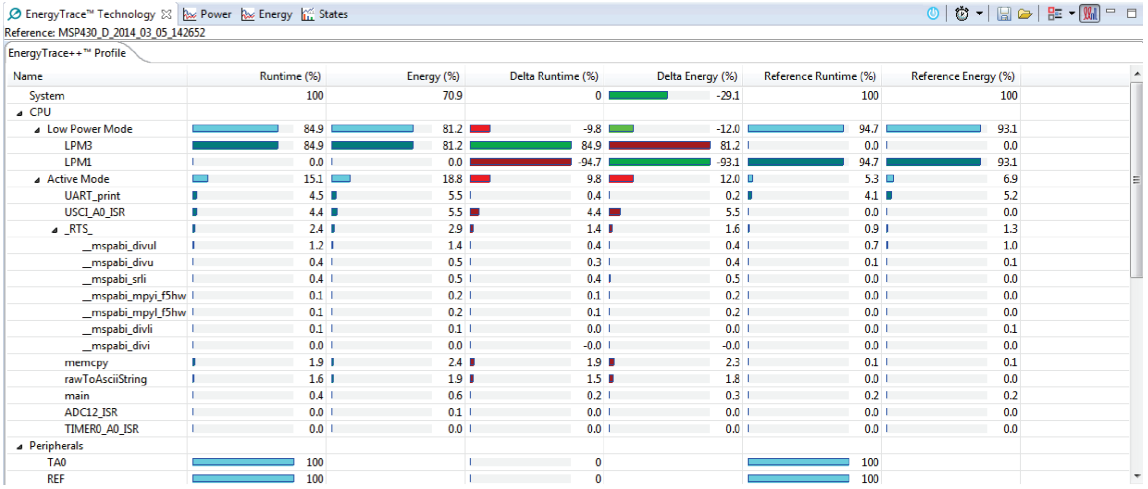


Figure 23. MostEfficient.c Referenced With Efficient.c

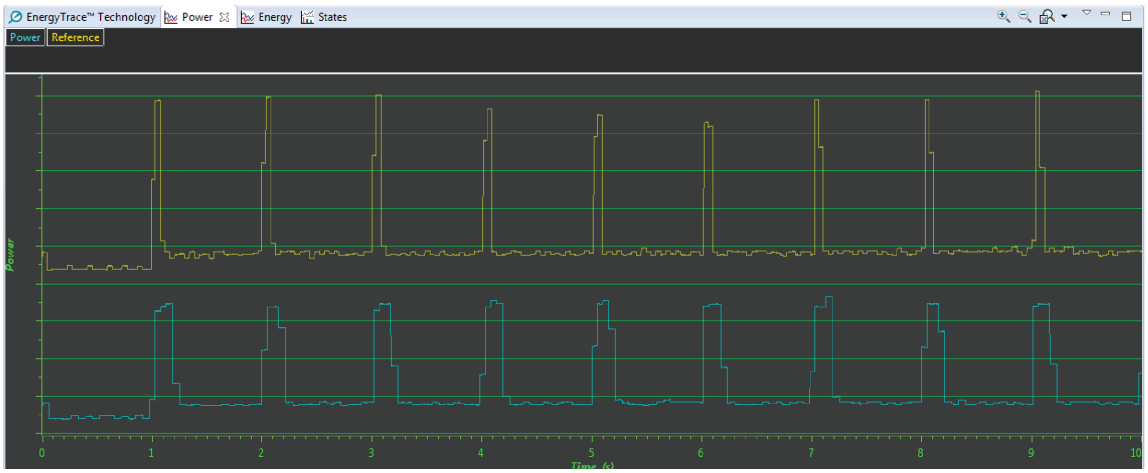


Figure 24. MostEfficient.c Power (Blue) Referenced With Efficient.c Power (Yellow)

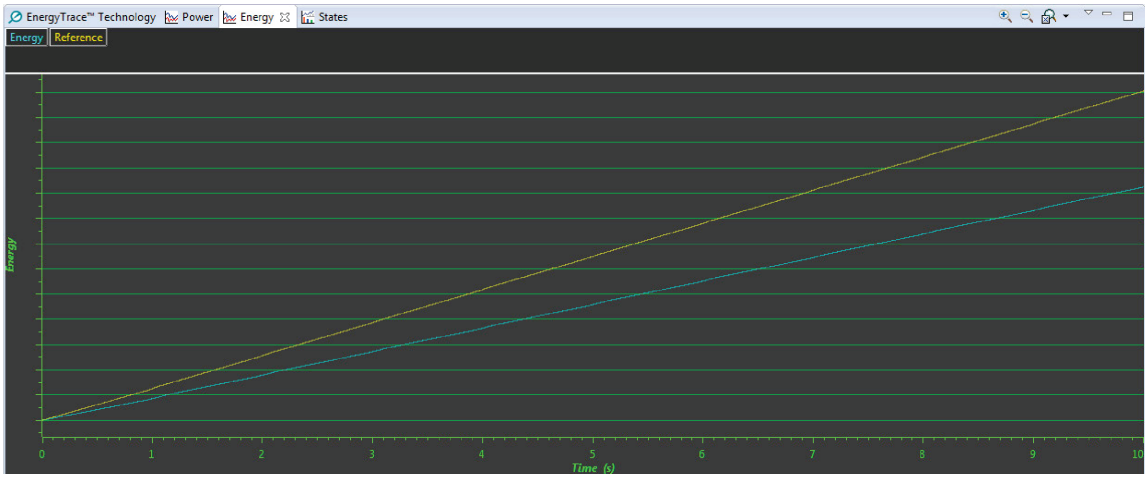


Figure 25. MostEfficient.c Energy (Blue) Referenced With Efficient.c Energy (Yellow)

9. Return to the *EnergyTrace Technology* window and open the reference profile Inefficient.profxml. **Take a moment to observe and appreciate how much the energy efficiency of this application has improved with the help of ULP Advisor and EnergyTrace**. Notice that the best case power consumption for the inefficient code is still much less efficient than the worst case power consumption of the most efficient code. These are incredible results (see Figure 26 through Figure 28).
10. Exit debug mode. 

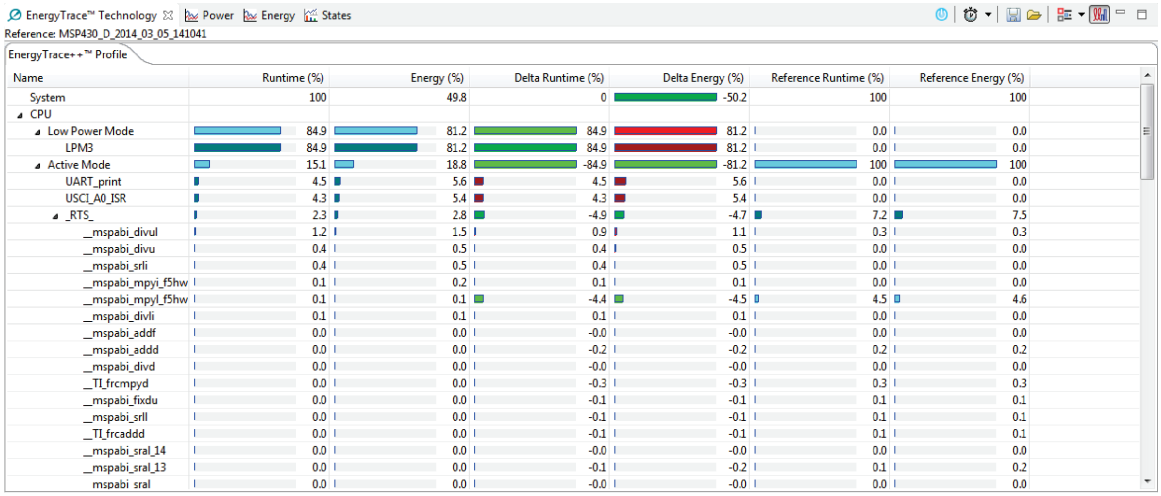


Figure 26. MostEfficient.c Referenced With Inefficient.c

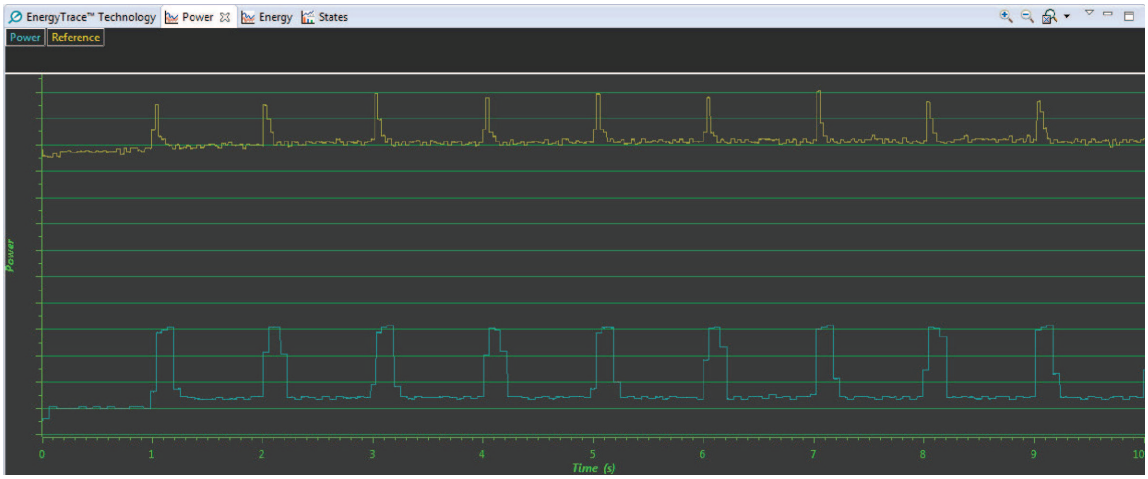


Figure 27. MostEfficient.c Power Referenced With Inefficient.c Power

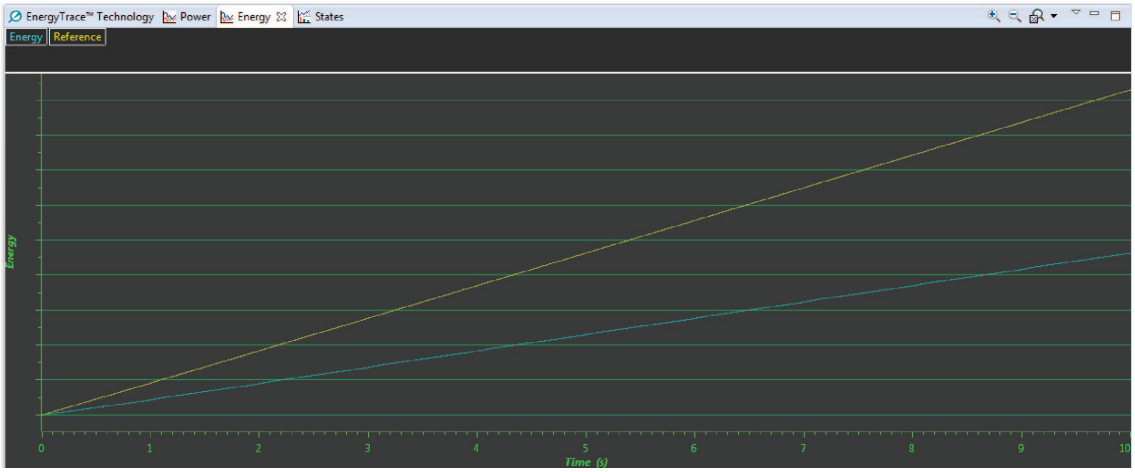


Figure 28. MostEfficient.c Energy Referenced With Inefficient.c Energy

4.10 Absolute Power Measurements

Now that you have the final version of the code, compare the absolute power consumption between the original code and the optimized code. Note that during the capture of internal states in EnergyTrace++ mode, the target microcontroller is constantly being accessed via 4-wire JTAG or Spy-bi-Wire. These processes consume energy, and you cannot separate energy consumption due to the debugger from the application energy. As a result there are no absolute power numbers shown on the y-axis of the power graphs because the energy consumption is misleading when there is debugger activity. In order to see more accurate values for absolute power, use the EnergyTrace mode in combination with the “Free Run” debug option. This is the only setting that captures energy consumption without accessing the debug logic of the target microcontroller.

MostEfficient.c is currently included in the build, so start with that.

1. Change the *EnergyTrace++* mode to *EnergyTrace* mode:
   - Go to *Window* → *Preferences* → expand *Code Composer Studio* → expand *Advanced Tools* → click *EnergyTrace Technology*.
   - Select the *EnergyTrace* mode (see Figure 2).
   - Click *OK*.
2. Enter a debug session.

3. **Set a breakpoint at line 99**, which is after all the initializations and before the main loop. Run the code to this point.
4. Now instead of pressing the resume button, go to *Run → Free Run* (see Figure 29). This removes any interference from the debugger.

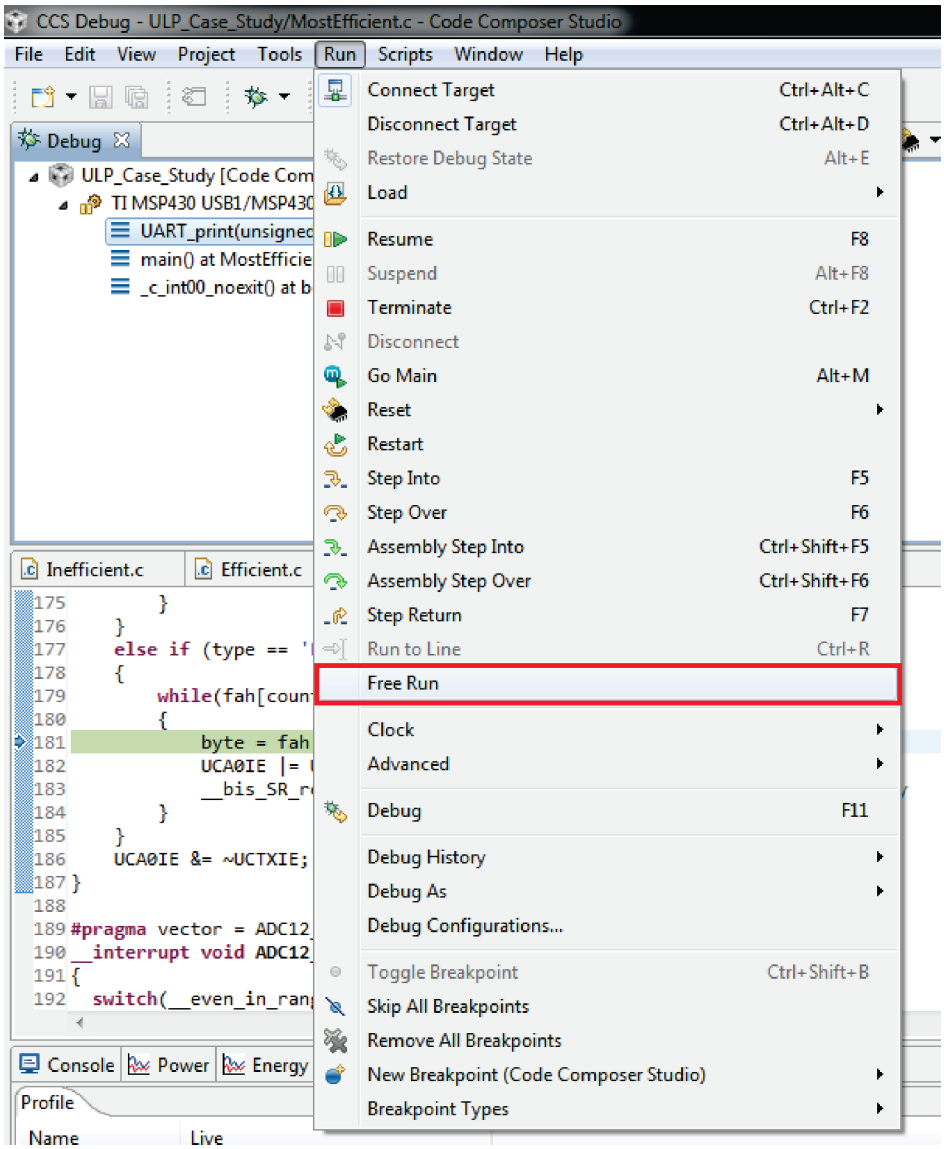


Figure 29. Free Run

5. Let the code run until the EnergyTrace capture time duration has passed.
6. Now the energy and power plots have units on the y-axis. These measurements are absolute (see Figure 30 and Figure 31).

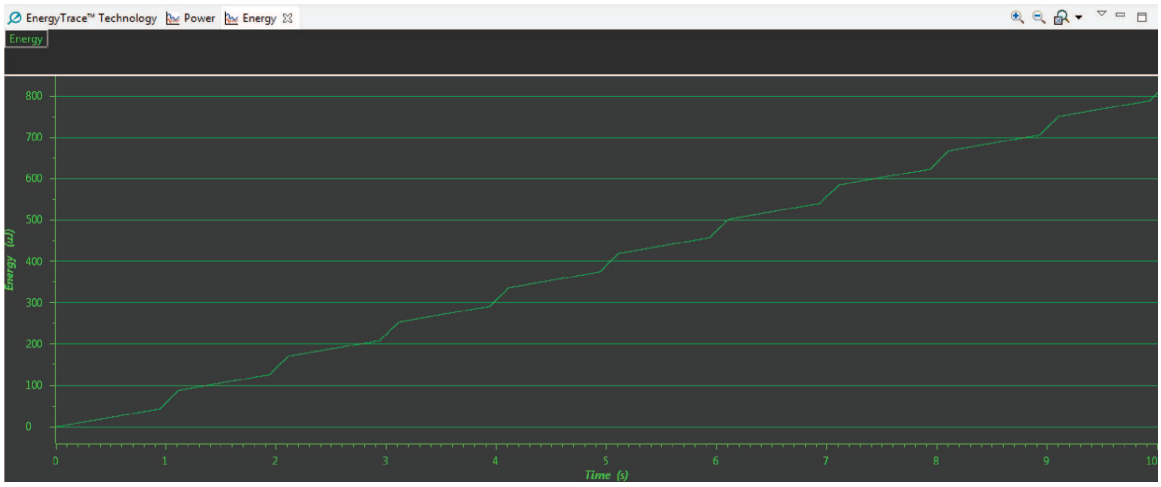


Figure 30. MostEfficient.c Absolute Energy

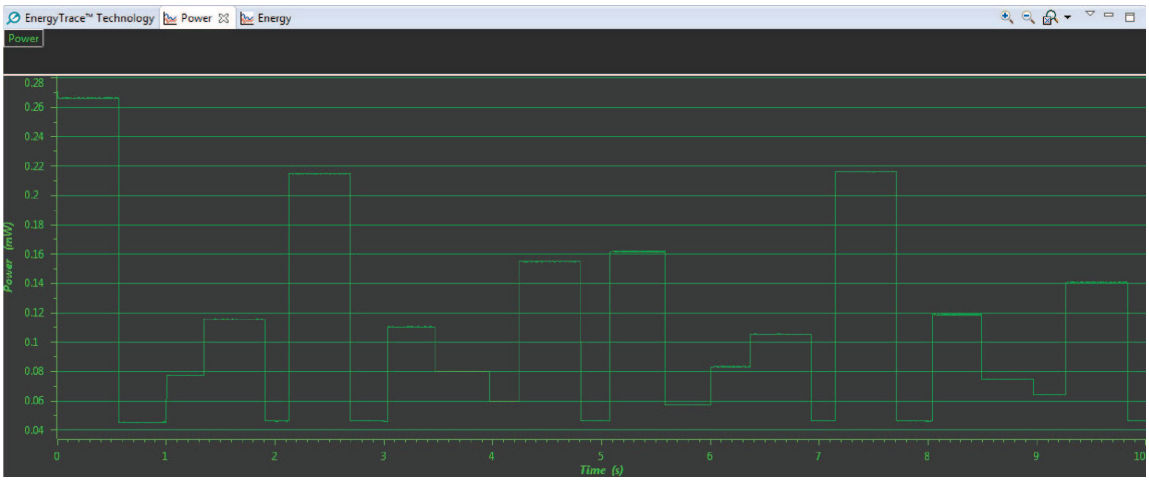


Figure 31. MostEfficient.c Absolute Power

7. To toggle between *EnergyTrace* and *EnergyTrace++* modes, press the *Switch to EneryTrace or Switch to EnergyTrace++* button in the *EnergyTrace Technology* window. The *States* window opens and closes accordingly.

4.11 Use EnergyTrace Technology With All MSP430 Devices

Any MSP430 evaluation module (EVM), experimenter's board (EXP), target socket module (TS), or application board can utilize the EnergyTrace component despite the fact that the EnergyTrace++ onboard circuitry is not present. To do this, the on-board emulation of a LaunchPad that contains the EnergyTrace technology circuitry is used to program and debug the intended target. In this example, the MSP-EXP430G2 is used. Note that this configuration functions properly only if the target device is supported by the eZ-FET. For details, see the eZ-FET lite revision 1.10 wiki.

1. Remove the RST, TST, V+, and GND jumpers from J13 on the MSP-EXP430FR5969 LaunchPad (see Figure 32). These jumpers connect the on-board emulation to the FR5969 target. Removing these jumpers allows you to connect the emulation to another target board.

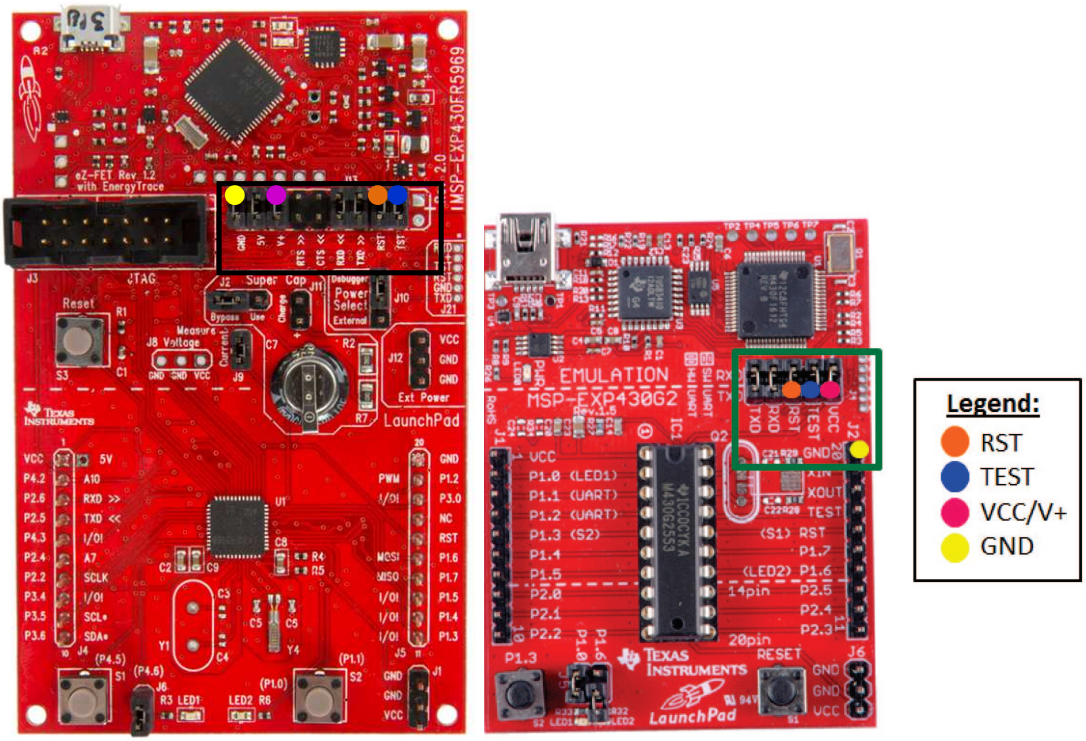


Figure 32. Connections Required to Monitor an Unsupported Device: MSP-EXP430FR5969 (left), MSP-EXP430G2 (right)

2. Remove the RST, TEST, and VCC jumpers from J3 on the MSP-EXP430G2 LaunchPad (see Figure 32). This effectively disconnects the G2553 device from its on-board emulation. The eZ-FET emulation from MSP-EXP430FR5969 board is used instead to program and debug this device.
3. Use jumper wires to connect the signals on the emulation side of the MSP-EXP430FR5969 board to the corresponding signal on the device side of the MSP-EXP430G2 LaunchPad. RST, TST, VCC, and GND on the MSP-EXP430FR5969 must be connected to each corresponding signal on the MSP-EXP430G2. The connections are shown as small color-coded circles in Figure 32.
4. Target power must be supplied through the eZ-FET of the board that has the EnergyTrace technology included in the on-board emulation (in this case, the MSP-EXP430FR5969 board). Plug-in the micro-USB to the MSP-EXP430FR5969 board.
5. After these connections have been made and power is supplied, a debug session can be initialized for a project set up for the target microcontroller (in this case, the MSP430G2553).The EnergyTrace mode will be available.

5 Summary

The use of ULP Advisor and EnergyTrace technology during the development process helps to identify energy inefficiencies in the code. If appropriate action is taken in response to the results of these two tools, then the energy efficiency of the target application can be dramatically improved.

Table 5 shows the energy consumption for each code file in the case study using the procedure that is described in the *Absolute power measurements* section of the case study for 10 seconds.

Table 5. Absolute Energy Comparison

Code File	Energy (mJ)
Inefficient.c	10.03
Efficient.c	4.48
MostEfficient.c	0.82

In the final version of the code, the energy consumption has been decreased to 8.1% of its original value. This is an incredible result that was made possible by ULP Advisor and EnergyTrace. These tools provide an easy way to save power (and money) for your target application and enable you to do more with less.

6 References

- *EnergyTrace Energy Aware Debugger* section in the *Code Composer Studio User's Guide for MSP430* (SLAU157)
- *MSP430FR58xx, MSP430FR59xx, MSP430FR68xx, and MSP430FR69xx Family User's Guide* (SLAU367)

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