

## Code Composer Studio™ 6.1 for MSP432

This manual describes the use of the TI Code Composer Studio™ IDE (CCS) version 6.1 with the MSP432™ low-power microcontrollers. This manual describes only CCS for Windows® operating systems. The versions of CCS for Linux® and Mac OS X® operating systems are similar and, therefore, are not described separately.

### Contents

1	Installing Code Composer Studio IDE .....	3
2	Updating Code Composer Studio IDE .....	3
3	Creating an MSP432 Project .....	5
4	Additional MSP432 Examples and Documentation .....	7
5	Debugging Your Application .....	8
6	Using Serial Wire Output (SWO) Hardware Trace Analyzer .....	12
7	EnergyTrace™ Technology .....	14
8	Device Security .....	25
9	Low-Power Debug .....	29
10	Frequently Asked Questions .....	32
11	Additional Code Composer Studio Information .....	33
12	References .....	33

### List of Figures

1	Check for Updates .....	3
2	Available Updates .....	4
3	Creating a New CCS Project .....	5
4	New Project Wizard .....	6
5	New Project Files .....	7
6	Project Properties .....	8
7	Choose Debugger Connection .....	9
8	Predefined Symbol for ROM Debugging .....	10
9	Launch Debug Session .....	10
10	Debug Session .....	11
11	Target Configuration .....	12
12	Connection Properties .....	12
13	Pulse Density and Current Flow .....	14
14	EnergyTrace™ Technology Preferences .....	15
15	EnergyTrace™ Technology Control Bar .....	16
16	Debug Session With EnergyTrace+ Graphs .....	17
17	Profile Window .....	17
18	States Window .....	18
19	Power Window .....	18

Code Composer Studio, MSP432, E2E, LaunchPad are trademarks of Texas Instruments.

CoreSight is a trademark of ARM Ltd.

ARM is a registered trademark of ARM Ltd.

OS X is a registered trademark of Apple Inc.

IAR Embedded Workbench is a trademark of IAR Systems.

Linux is a registered trademark of Linus Torvalds.

Windows is a registered trademark of Microsoft Corporation.

20	Energy Window .....	19
21	Debug Session With EnergyTrace Graphs .....	19
22	EnergyTrace Profile Window .....	20
23	Zoom Into Power Window.....	20
24	Current Profile (Blue) With Recorded Profile (Yellow).....	21
25	Energy Profile of the Same Program in Resume (Yellow Line) and Free Run (Green Line) .....	21
26	Comparing Profiles in EnergyTrace+ Mode .....	22
27	Comparing Profiles in EnergyTrace Mode .....	23
28	Show Target Configuration View .....	25
29	List of Target Configurations.....	26
30	Launch Selected Target Configuration .....	26
31	Debug View After Launching Target Configuration .....	26
32	Show All Cores .....	27
33	List of All Cores in the MSP432 .....	27
34	Manually Connecting to the DAP.....	28
35	DAP is Connected.....	28
36	Executing the Factory Reset Script .....	28
37	Mass Erase Script Console Output .....	29
38	Properties Menu .....	30
39	Enabling Low Power Run .....	31
40	CPU Core Status Display Indicating Deep Sleep Mode .....	31
41	Program Counter Located at WFI Instruction.....	32
42	Change Debugger Settings to SWD .....	32

## Read This First

### How to Use This Manual

This manual describes only those CCS features that are specific to the MSP432 low-power microcontrollers. It does not fully describe the MSP432 microcontrollers or the complete development software and hardware systems. For details on these items, see the appropriate TI documents listed in [Important MSP432 Documents on the Web](#).

### Important MSP432 Documents on the Web

The primary sources of MSP432 information are the device-specific data sheets and user's guides. The [MSP432 website](#) contains the most recent version of these documents.

Documents describing the Code Composer Studio tools (Code Composer Studio IDE, assembler, C compiler, linker, and librarian) can be found at [www.ti.com/tool/ccstudio](http://www.ti.com/tool/ccstudio). A CCS-specific Wiki page (FAQ) is available at [processors.wiki.ti.com/index.php/Category:CCS](http://processors.wiki.ti.com/index.php/Category:CCS), and the [TI E2E™ Community](#) support forums provide additional help.

Documentation for third party tools, such as IAR Embedded Workbench™ for ARM® or the Segger J-Link debug probe, can be found on the respective company's website.

### If You Need Assistance

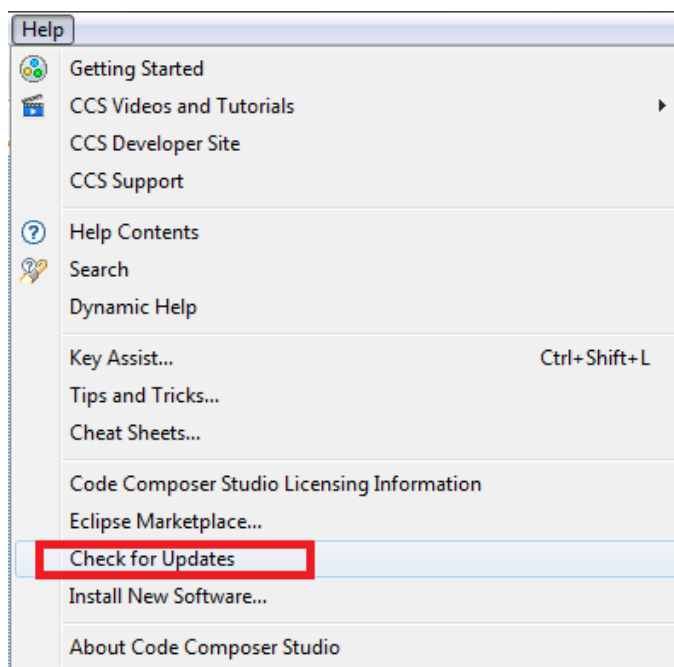
Support for the MSP432 devices and the hardware development tools is provided by the Texas Instruments Product Information Center (PIC). Contact information for the PIC can be found on the [TI website](#). The [TI E2E Community](#) support forums for the MSP432 provide open interaction with peer engineers, TI engineers, and other experts. Additional device-specific information can be found on the [MSP432 website](#).

## 1 Installing Code Composer Studio IDE

The Code Composer Studio IDE can be obtained from TI website under [www.ti.com/tool/ccstudio](http://www.ti.com/tool/ccstudio). Note that MSP432 low-power microcontrollers are supported by Code Composer Studio 6.1 and higher versions. Previous versions do not support MSP432.

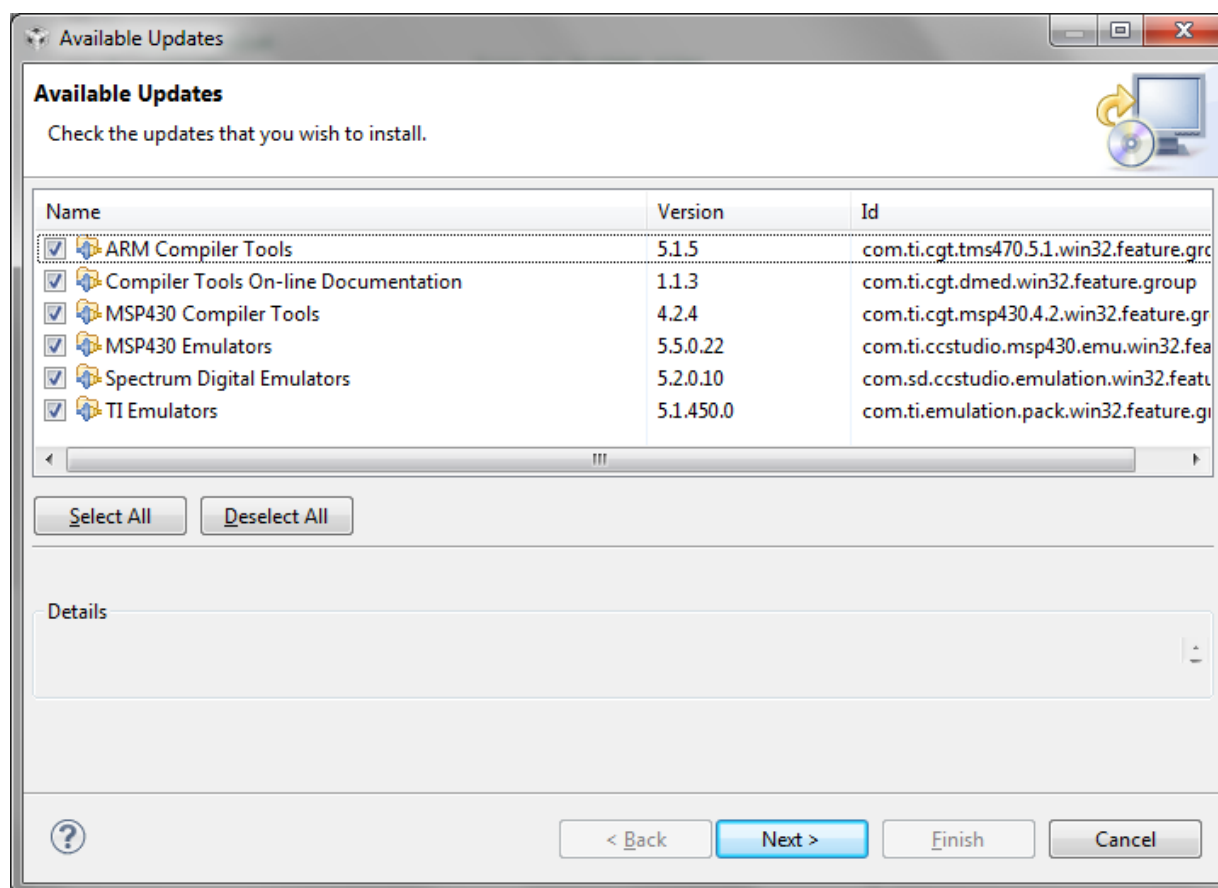
## 2 Updating Code Composer Studio IDE

Use the CCS update feature to update the installation before starting to work with MSP432 low-power microcontrollers. To check for updates, click **Help** → **Check for Updates**.



**Figure 1. Check for Updates**

CCS now automatically connects to the TI update server and retrieves information about relevant updates for the system. You can still select which update to install, but it is good practice to install all updates (see [Figure 2](#)).



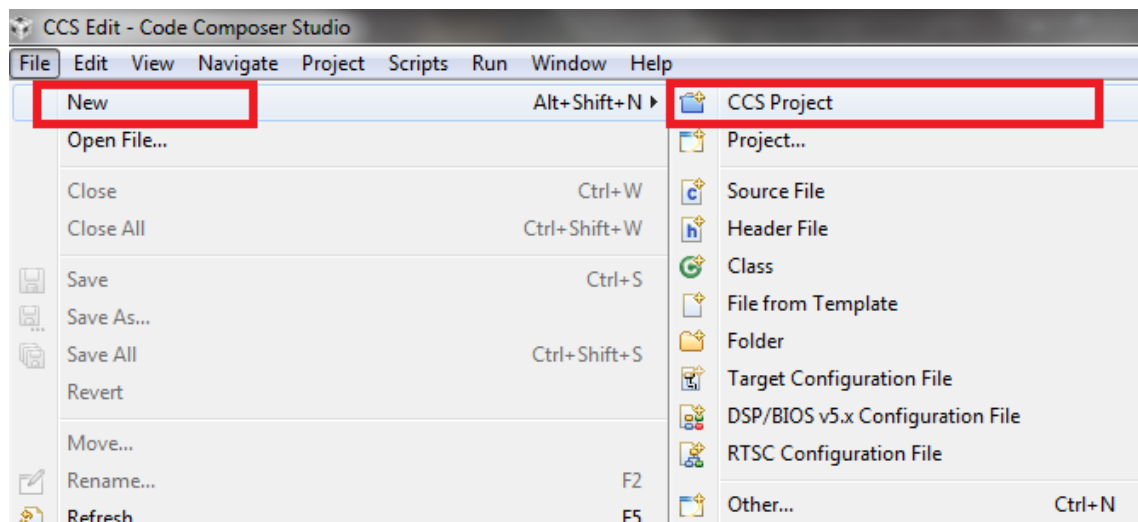
**Figure 2. Available Updates**

### 3 Creating an MSP432 Project

CCS organizes its projects in workspaces. A new workspace is generated automatically when you start CCS the first time. This workspace is blank.

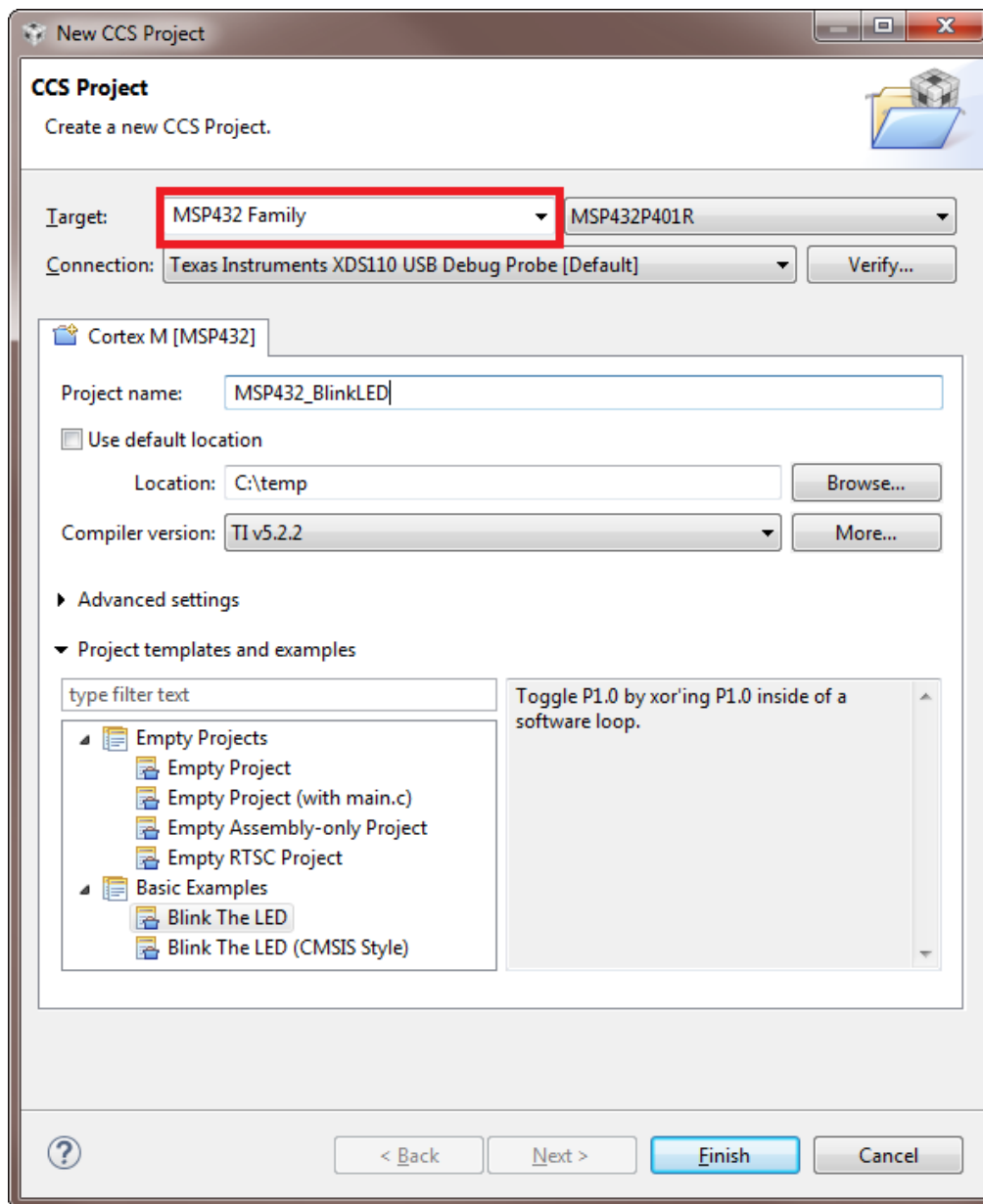
To create a new project in that workspace:

- Click **File** → **New** → **CCS Project**.



**Figure 3. Creating a New CCS Project**

- Select **MSP432** as the target family (see [Figure 4](#)), which limits the list of devices in the device pulldown list.
- Select the device being used, for example, MSP432P401R.
- Select the debug connection to use. In the following example, a **XDS110** debug probe. These settings can later be modified in the Project Properties.
- Type a unique project name.
- Click **Finish**. A new project is created and, along with it, a number of files are copied into the workspace.



**Figure 4. New Project Wizard**

The workspace now contains a newly created project (), including

- A basic main.c file
- The interrupt vector file **msp432\_startup\_ccs.c** where all interrupt handlers are predefined
- The linker command file **msp432p401r.cmd**
- The target configuration **MSP432P401R.ccxml** file including a link to the XDS-110 debug probe

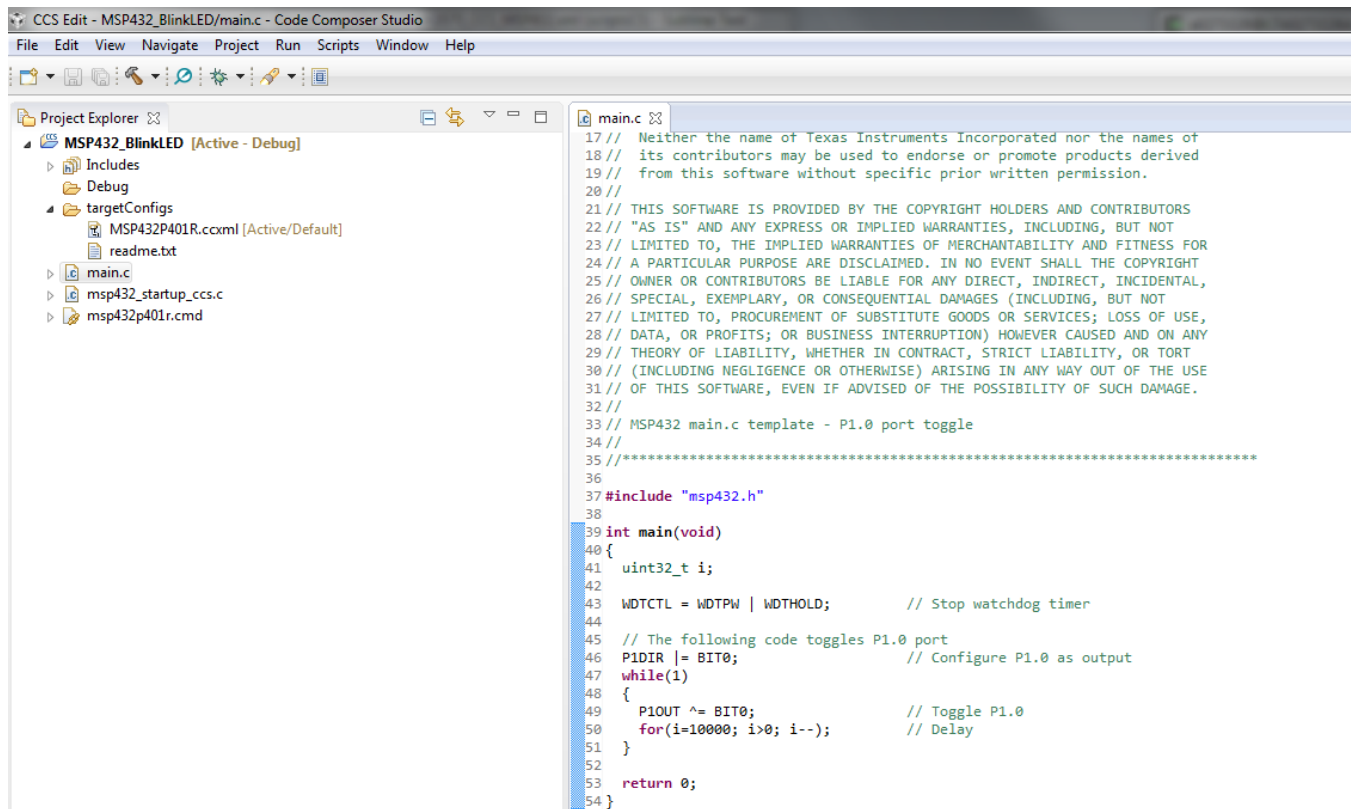


Figure 5. New Project Files

Now you can go on and start developing code.

#### 4 Additional MSP432 Examples and Documentation

The TI MSP team releases a software package called MSPWare that contains many software examples, projects, documentation as well as application notes and training for all MSP devices. This includes example projects for Code Composer Studio that work with MSP432. For more information, visit the [MSPWare website](#).

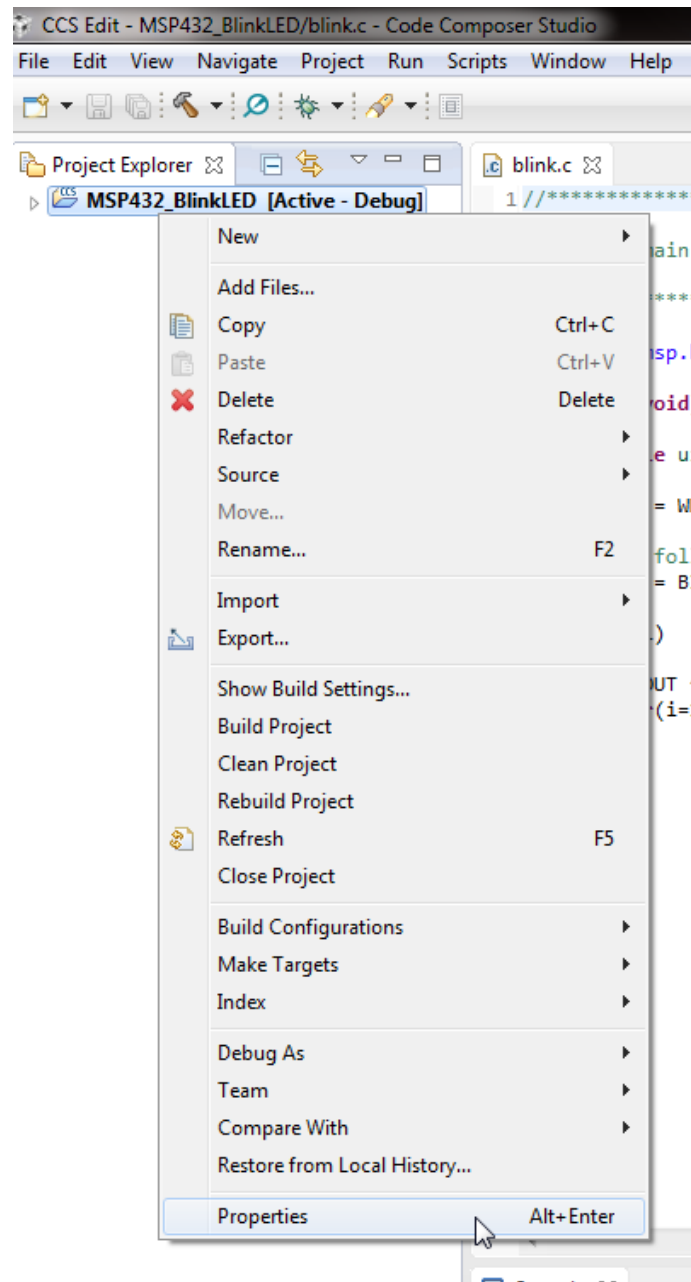
## 5 Debugging Your Application

The following debug probes have been tested successfully with CCS.

- Texas Instruments XDS100v2, XDS100v3, XDS200, XDS110
- Texas Instruments MSP-FET (for more information visit the [MSP-FET page](#))
- Segger J-Link (for more information, visit the [TI J-Link Support Emulator wiki page](#))

In CCS 6.1.3.x, the Segger installation instructions are directly linked in the CCS Apps center.

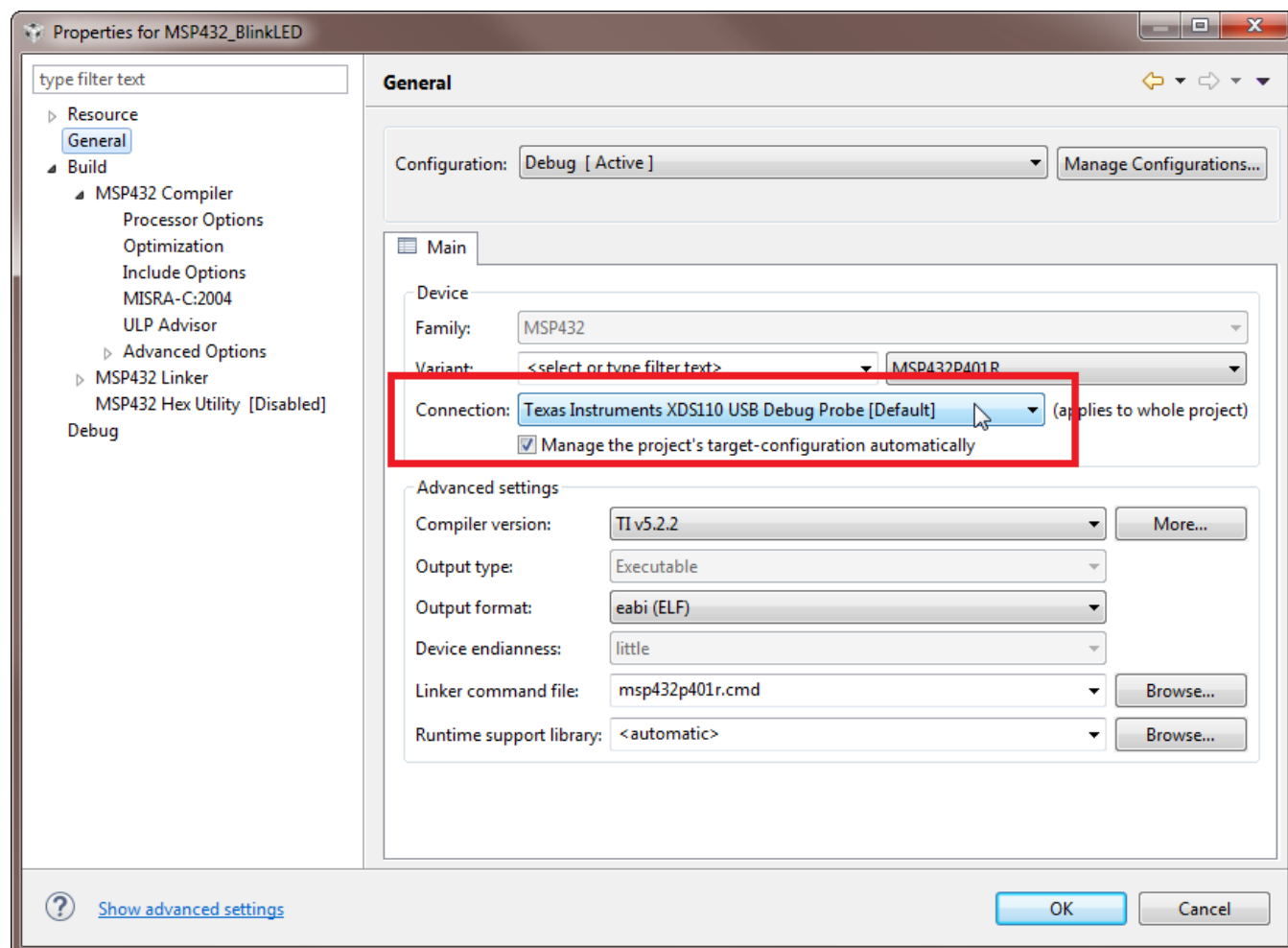
After you create a project, you can change the debugger used to debug the application. To change the debugger, **right click** on the project in the workspace and choose **Properties** (see [Figure 6](#)).



**Figure 6. Project Properties**

In the project properties window, make sure that you are in the **General** options pane. There you can see a drop-down list of target connections. Choose the debugger from this list and click **OK**.

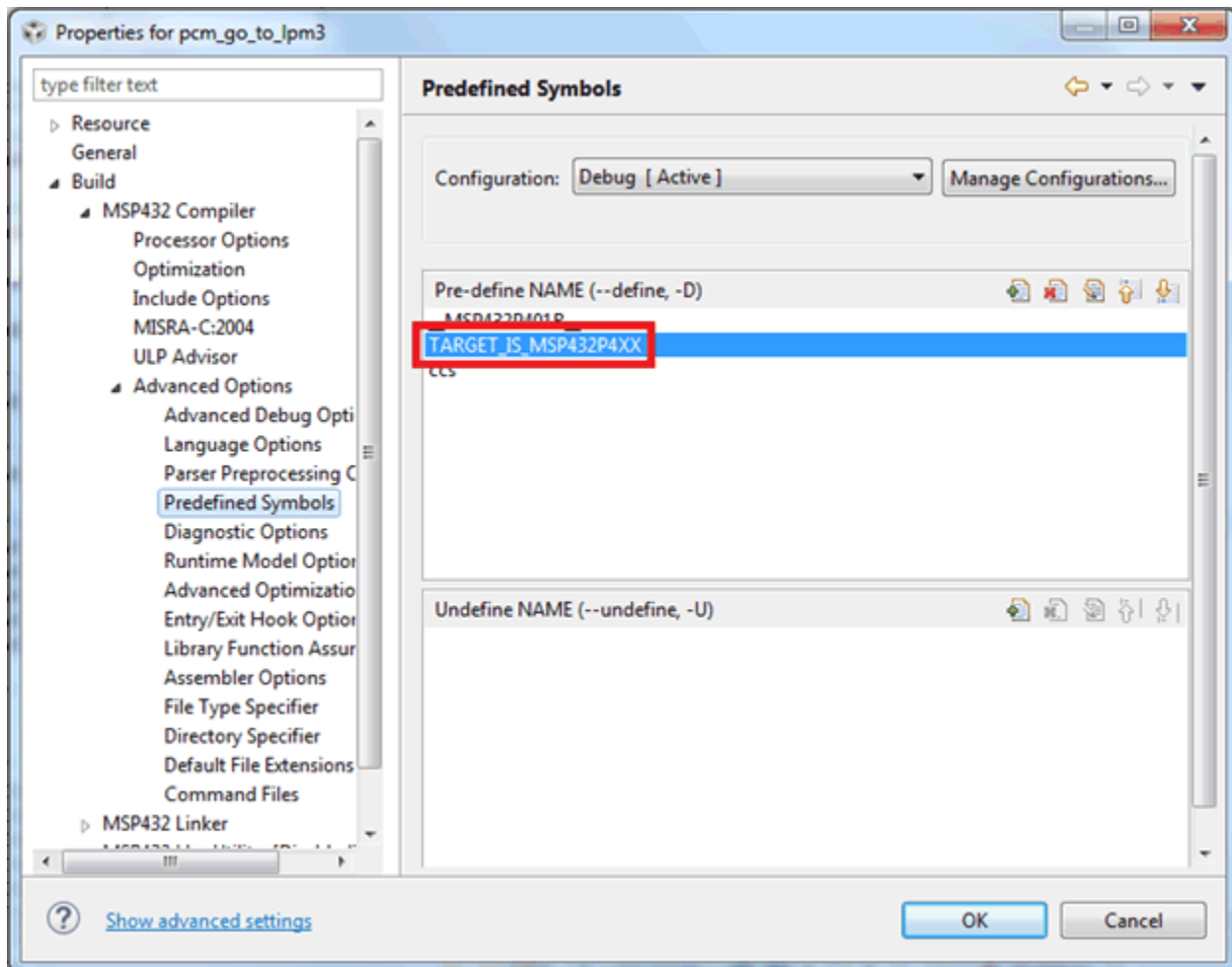





**Figure 7. Choose Debugger Connection**

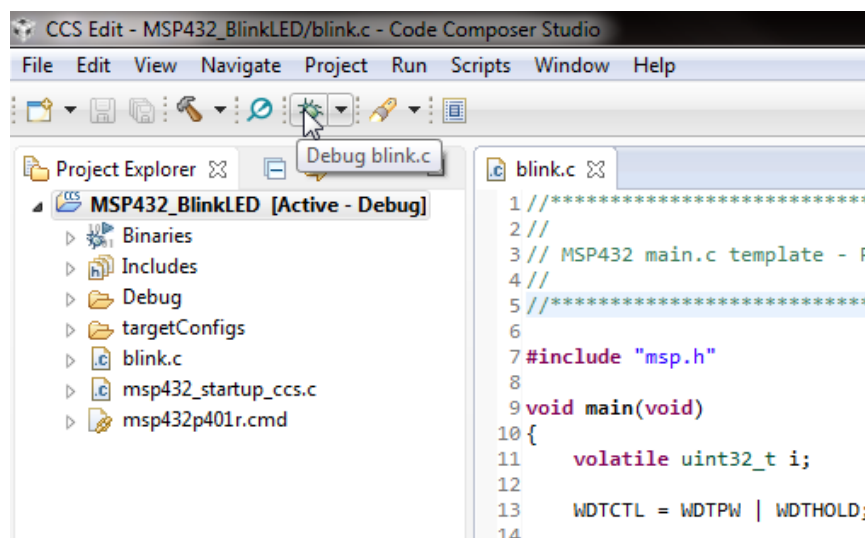
More debugger settings can be made in the target configuration settings. For example, switching from JTAG to the less pin-consuming serial wire debug (SWD). See [Section 6](#) for details on these settings.

To debug the ROM driver library [4], make sure that `TARGET_IS_MSP432P4XX` is listed in the Advanced Compiler → Predefined Symbols (see [Figure 8](#)).



**Figure 8. Predefined Symbol for ROM Debugging**

After the project has been set to the correct debugger, you can begin to debug the application. To launch a debug session, click the  icon in the top toolbar of the IDE (see [Figure 9](#)).



**Figure 9. Launch Debug Session**

After the IDE has finished compiling, linking, and downloading the code to the device, the debug session starts and you can begin to debug the application.

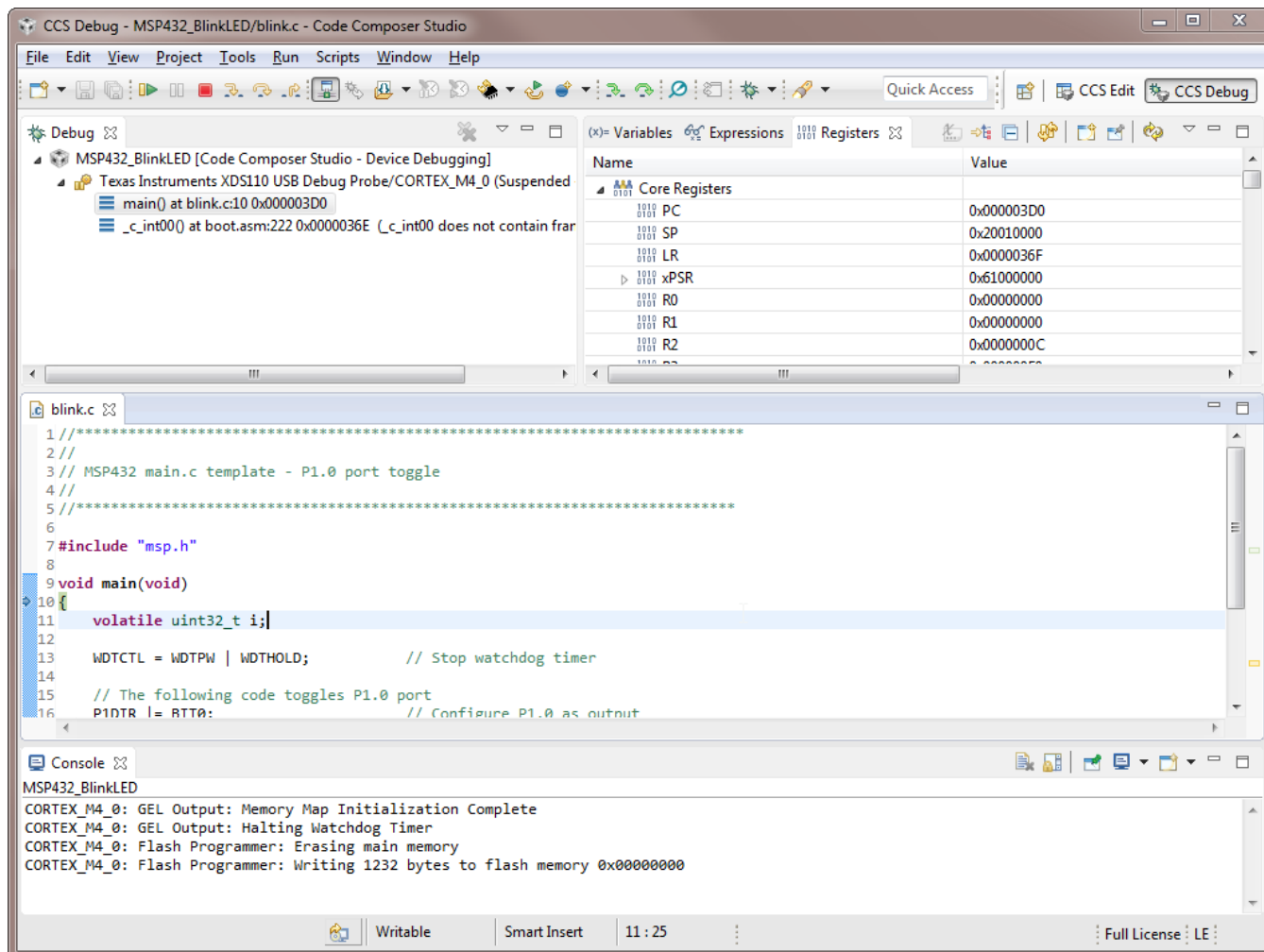


Figure 10. Debug Session

## 6 Using Serial Wire Output (SWO) Hardware Trace Analyzer

In CCS the SWO Trace tools for MSP432 MCUs are implemented using the features of the ARM® CoreSight™ components, especially the Instrumentation Trace Macrocell (ITM) and Data Watchpoint and Trace Unit (DWT) (ETM is not present in MSP432 MCUs).

This user's guide concentrates on enabling the SWO trace in CCS. A more profound description of the trace hardware and example use cases can be found in [Reference \[3\]](#).

### 6.1 Configure the Project for SWO Trace

To enable SWO trace:

1. Expand the project in Project Explorer and open the projects \*.ccxml file in the targetConfigs folder. Select the correct debug probe (see [Figure 11](#)).

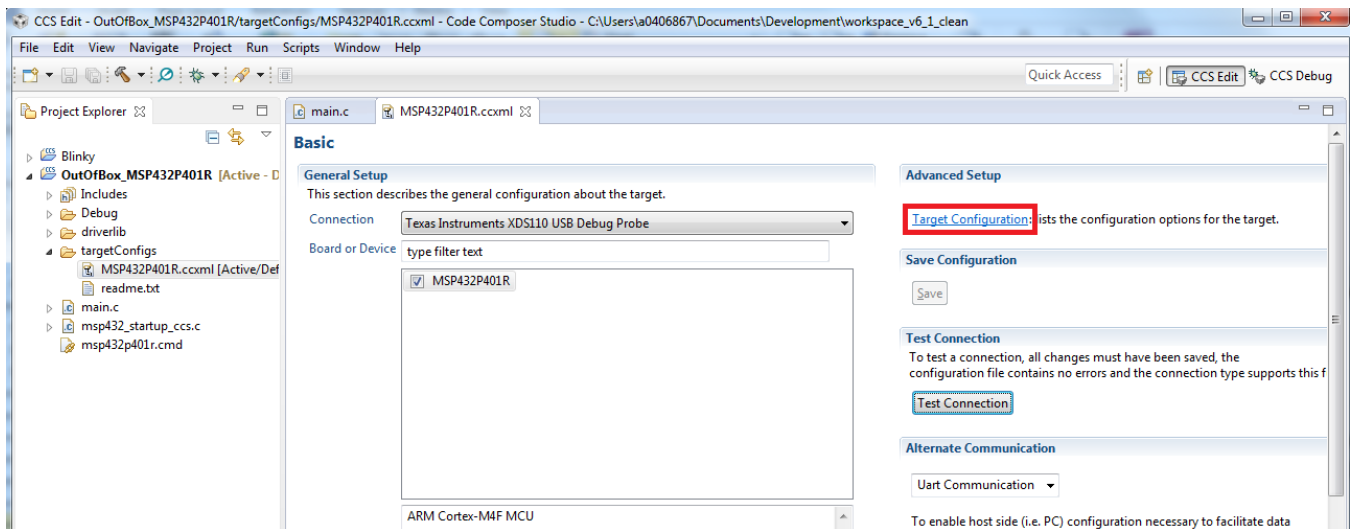


Figure 11. Target Configuration

2. Click Target Configuration in the Advanced Setup section then click on the selected debug probe.
3. [Figure 12](#) shows the Connection Properties are shown. In JTAG/SWD mode, select SWD Mode – Aux COM port is target TDO pin.

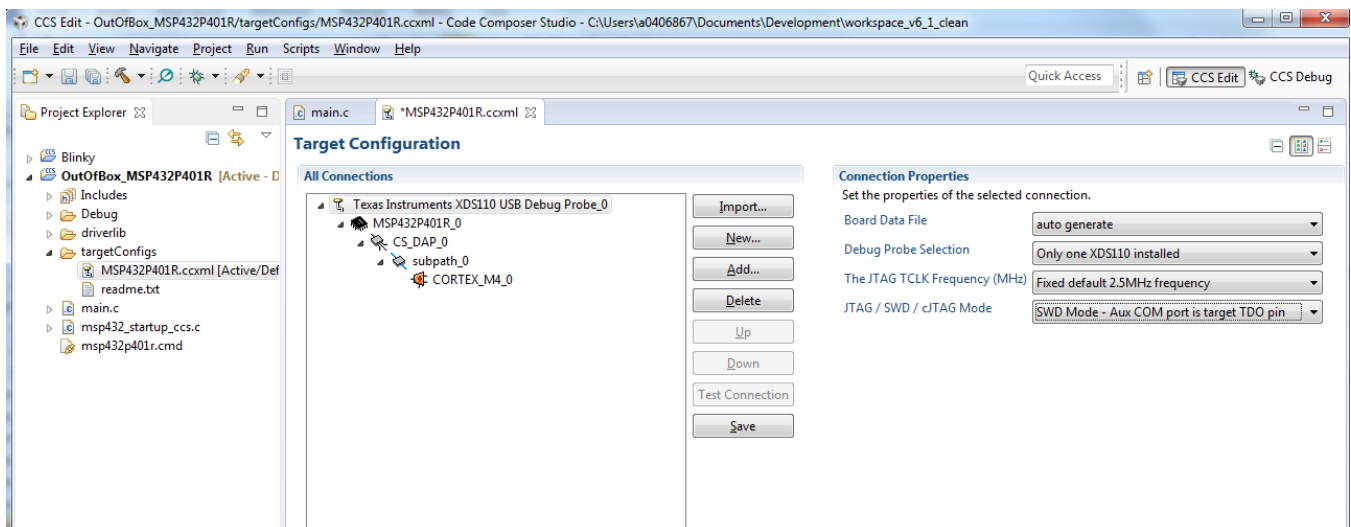


Figure 12. Connection Properties

## 6.2 Run Serial Wire Trace

After finishing SWO configuration, start a debug session to start tracing:

1. Build the project by either selecting Build Project from the projects context menu or by selecting the hammer icon.
2. Enter a debug session: Right click the project name, select Debug As → Code Composer Studio Debug Session. Alternatively, click the bug icon.
3. In debug mode, go to Tools → Hardware Trace Analyzer and choose one of the following options:
  - Statistical Function Profiling for analyzing how often each function is called and what percentage of PCU cycles is consumed
  - Data Variable Tracing for obtaining a graph on the value of the variable using the ITM
  - Interrupt Profiling for getting data on when interrupts occurred and how these interrupt each other.
  - Custom Core Trace for tracing user defined strings at specified code locations.

See [MSP432™ Debugging Tools: Using Serial Wire Output With CCS Trace Analyzer](#) for details.

## 7 EnergyTrace™ Technology

EnergyTrace™ Technology is an energy-based code analysis tool that measures and displays the application's energy profile and helps to optimize it for ultra-low power consumption.

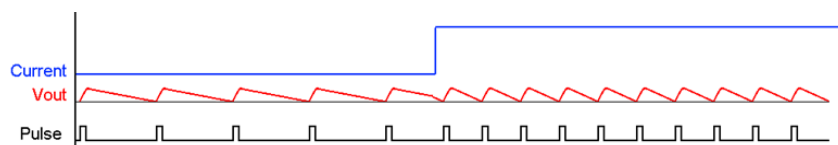
MSP432 devices with built-in **EnergyTrace+[CPU State]** (or in short **EnergyTrace+**) technology allow real-time monitoring of many internal device states while user program code executes. EnergyTrace+ technology is supported on selected MSP432 devices and debuggers.

**EnergyTrace** mode (without the "+") is the base of **EnergyTrace Technology** and enables analog energy measurement to determine the energy consumption of an application but does not correlate it to internal device information. The EnergyTrace mode is available for all MSP432 devices with selected debuggers, including CCS.

### 7.1 Energy Measurement

Debuggers with EnergyTrace Technology support include a new and unique way of continuously measuring the energy supplied to a target microcontroller that differs considerably from the well-known method of amplifying and sampling the voltage drop over a shunt resistor at discrete times. A software-controlled dc-dc converter is used to generate the target power supply. The time density of the dc-dc converter charge pulses equals the energy consumption of the target microcontroller. A built-in on-the-fly calibration circuit defines the energy equivalent of a single dc-dc charge pulse.

Figure 13 shows the energy measurement principle. Periods with a small number of charge pulses per time unit indicate low energy consumption and thus low current flow. Periods with a high number of charge pulses per time unit indicate high energy consumption and also a high current consumption. Each charge pulse leads to a rise of the output voltage  $V_{OUT}$ , which results in an unavoidable voltage ripple common to all dc-dc converters.



**Figure 13. Pulse Density and Current Flow**

The benefit of sampling continuously is evident: even the shortest device activity that consumes energy contributes to the overall recorded energy. No shunt-based measurement system can achieve this.

### 7.2 Code Composer Studio™ Integration

EnergyTrace Technology is available as part of Texas Instrument's Code Composer Studio IDE for MSP432 microcontrollers. During debugging of an application, additional windows are available if the hardware supports EnergyTrace Technology.

### 7.3 Enabling EnergyTrace Technology and Selecting the Default Mode

By default, the EnergyTrace Technology feature is disabled in the Code Composer Studio Preferences. To enable it, go to Window → Preferences → Code Composer Studio → Advanced Tools → EnergyTrace™ Technology (see Figure 14).

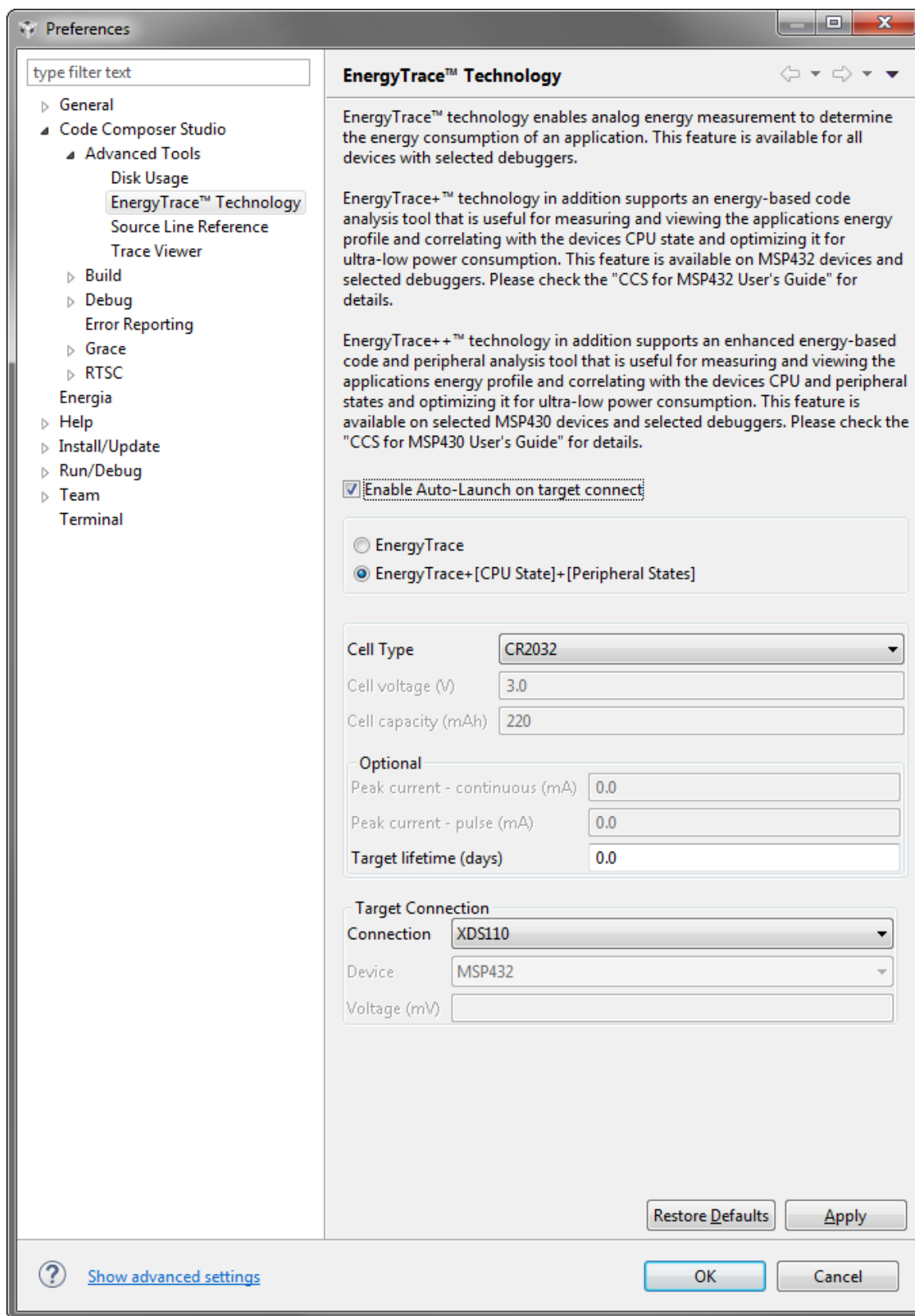



Figure 14. EnergyTrace™ Technology Preferences

Two capture modes are supported.

- The full-featured **EnergyTrace+[CPU State]** mode that delivers real-time device state information together with energy measurement data
- The **EnergyTrace** mode that delivers only energy measurement data

Use the radio button to select the mode to enable when a debug session is launched. If an MSP432 device does not support device state capturing, the selection is ignored and Code Composer Studio starts in **EnergyTrace** mode.

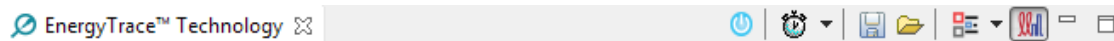
While a debug session is active, click the  icon in the **Profile** window to switch between the modes.

**NOTE:** If the EnergyTrace Technology windows are not opened when a debug session starts, verify the following items:

- Does the hardware (debugger and device) support EnergyTrace Technology? To determine if the selected device supports EnergyTrace, refer to the device-specific data sheet, the *MSP432 Hardware Tools User's Guide*, or the user guide that came with the evaluation board.
- Is EnergyTrace Technology globally enabled in Window → Preferences → Code Composer Studio → Advanced Tools → EnergyTrace™ Technology?







## 7.4 Controlling EnergyTrace Technology

EnergyTrace Technology can be controlled using the control bar icons in the **Profile** window (see [Figure 15](#)). [Table 1](#) describes the function of each of these buttons.



**Figure 15. EnergyTrace™ Technology Control Bar**

**Table 1. EnergyTrace™ Technology Control Bar Icons**

	Enable or disable EnergyTrace Technology. When disabled, icon turns gray.
	Set capture period: 5 sec, 10 sec, 30 sec, 1 min, or 5 min. Data collection stops after time has elapsed. However, the program continues to execute until the Pause button in the debug control window is clicked.
	Save profile to project directory. When saving an EnergyTrace+ profile, the default filename starts with "EnergyTrace_D" followed by a timestamp. When saving an EnergyTrace profile, the default filename starts with "EnergyTrace" followed by a timestamp.
	Load previously saved profile for comparison.
	Restore graphs or open Preferences window.
	Switch between <b>EnergyTrace+</b> mode and <b>EnergyTrace</b> mode

## 7.5 EnergyTrace+ Mode

When debugging devices with built-in EnergyTrace+ support, the **EnergyTrace+ mode** gives information about both energy consumption and the internal state of the target microcontroller. The following windows are opened during the startup of a debug session:

- Profile
- States
- Power
- Energy



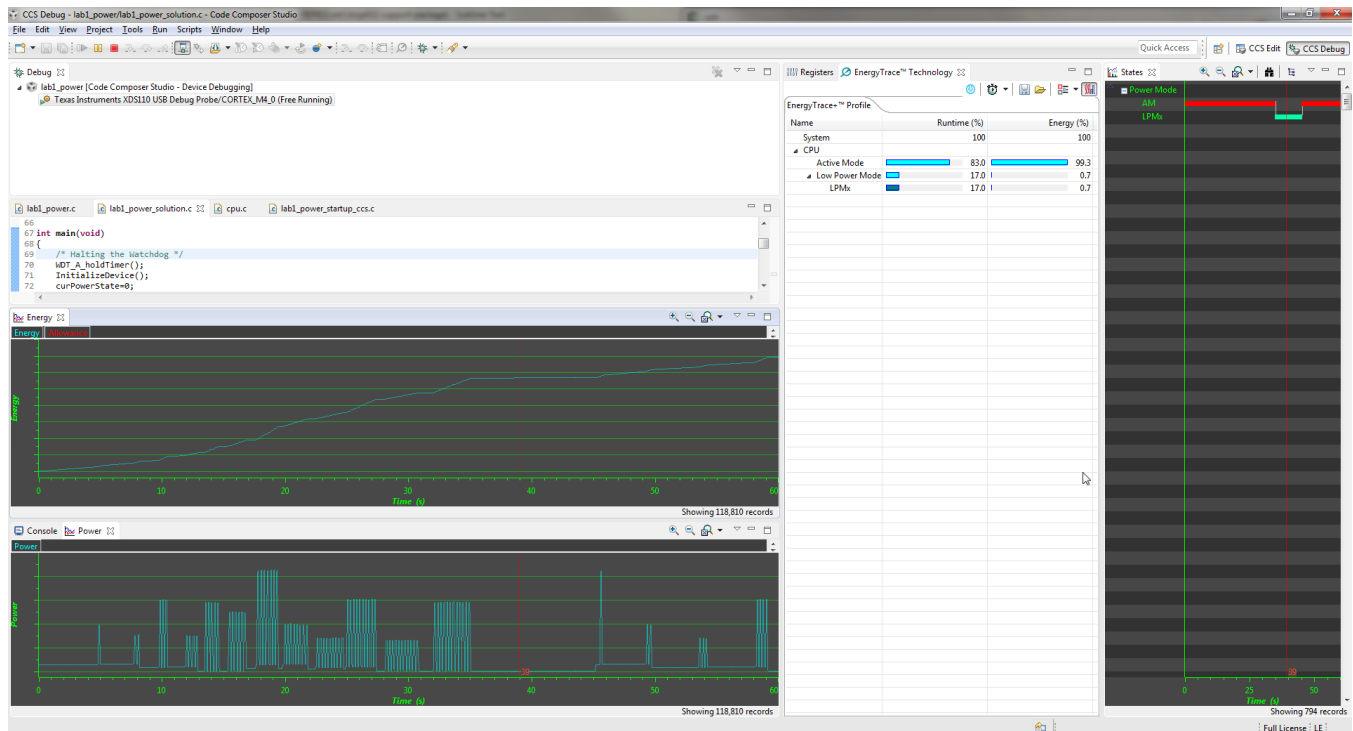


Figure 16. Debug Session With EnergyTrace+ Graphs

The **Profile** window (see Figure 17) is the control interface for EnergyTrace+. It can be used to set the capturing time or to save the captured data for later reference. The **Profile** window also displays a compressed view of the captured data and allows comparison with previous data.

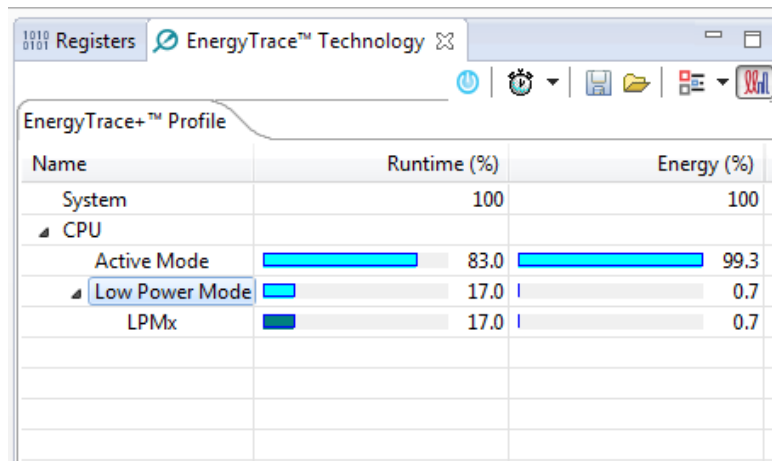


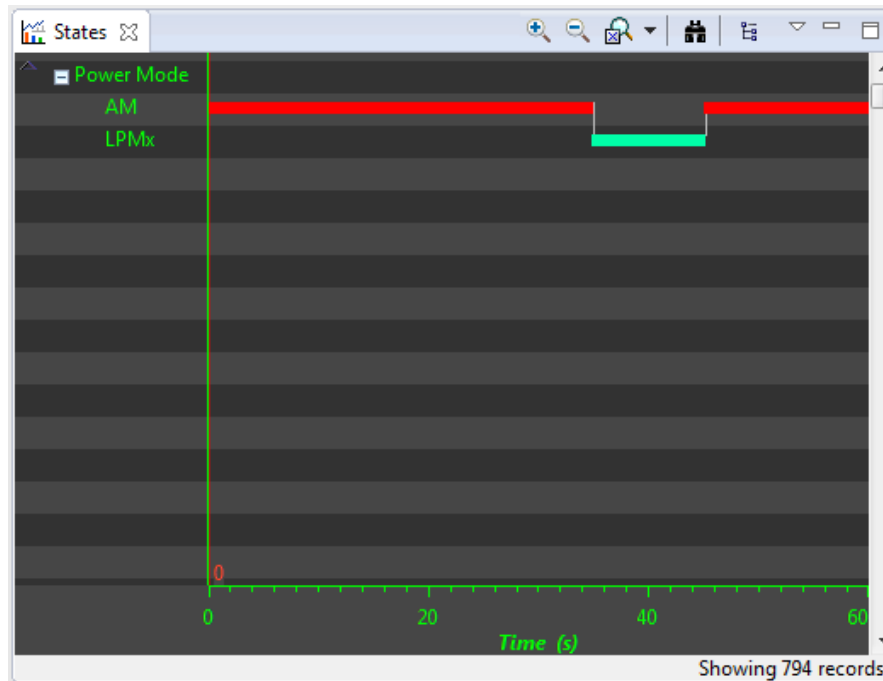
Figure 17. Profile Window

The **Profile** window enables a quick overview of the resource use of the profiled application.

- CPU: Shows information about program execution
  - Low Power Mode: Shows a summary of low-power mode use.
  - Active Mode: Shows which functions have been executed during active mode. Functions in the run-time library are listed separately under the `_RTS_` subcategory. If the device supports IP Encapsulation, a line labeled as `<Protected>` is displayed to indicate the time executing out of IP encapsulated memory.

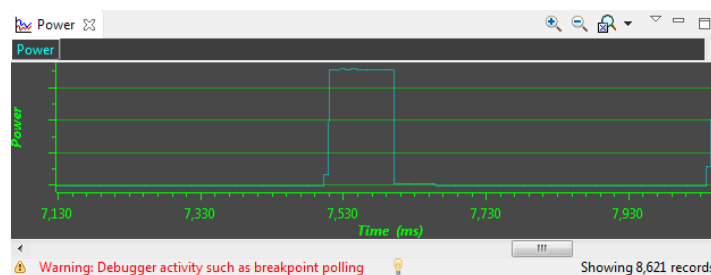
The **States** window (see Figure 18) shows the real-time trace of the target microcontroller's internal states during the captured session. State information includes the Power Modes, on and off state of peripheral modules and the state of the system clocks.

Figure 18 shows a device going into a Low Power mode then back to Active Mode. The **States** window allows a direct verification of whether or not the application exhibits the expected behavior; for example, that a peripheral is disabled after a certain activity.



**Figure 18. States Window**

The **Power** window (see Figure 19) shows the dynamic power consumption of the target over time. The current profile is plotted in light blue color, while a previously recorded profile that has been reloaded for comparison is plotted in yellow color.



**Figure 19. Power Window**

The **Energy** window (see Figure 20) shows the accumulated energy consumption of the target over time. The current profile is plotted in light blue color, while a previously recorded profile that has been reloaded for comparison is plotted in yellow color.

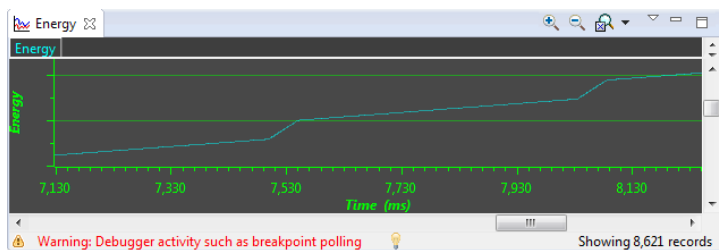


Figure 20. Energy Window

**NOTE:** During the capture of the internal states, the target microcontroller is constantly accessed by the JTAG or Spy-Bi-Wire debug logic. These debug accesses consume energy; therefore, no absolute power numbers are shown on the Power and Energy graph vertical axis. To see absolute power numbers of the application, it is recommended to use the **EnergyTrace mode** in combination with the **Free Run** option. In this mode, the debug logic of the target microcontroller is not accessed while measuring energy consumption.

Free Run mode is nonintrusive to the device but still uses the debug port, which affects energy measurement. For highest accuracy, run EnergyTrace without debugging or Free Run. Also note that the graphs for energy and power are printed in blue instead of green when no accurate measurement is possible.

## 7.6 EnergyTrace Mode

This mode allows a standalone use of the energy measurement feature with MSP432 microcontrollers that do not have built-in EnergyTrace+ support. It can also be used to verify the energy consumption of the application without debugger activity. If the **EnergyTrace** mode is selected in the Preferences window, the following windows open when a debug session starts (see Figure 21):

- Profile
- Power
- Energy

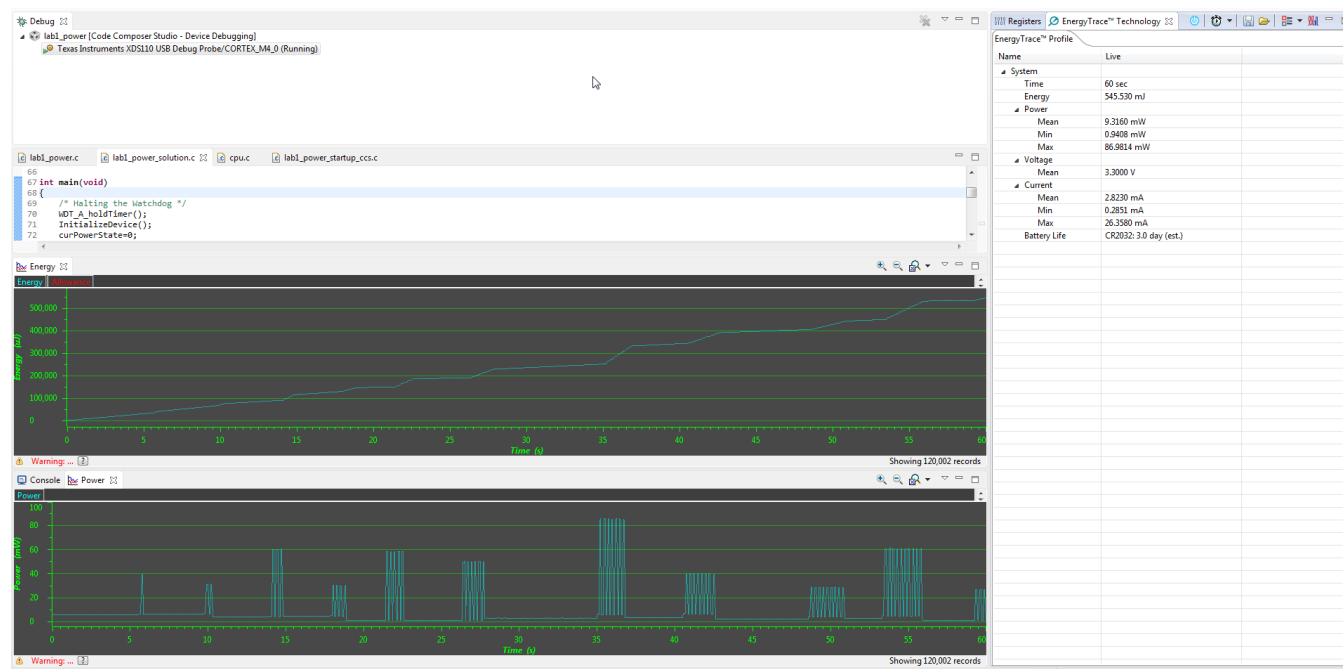


Figure 21. Debug Session With EnergyTrace Graphs

In the **EnergyTrace** mode, the **Profile** window shows statistical data about the application that has been profiled (see [Figure 22](#)). The following parameters are shown:

- Captured time
- Total energy consumed by the application (in mJ)
- Minimum, mean, and maximum power (in mW)
- Mean voltage (in V)
- Minimum, mean, and maximum current (in mA)
- Estimated life time of a CR2032 battery (in days) for the captured energy profile

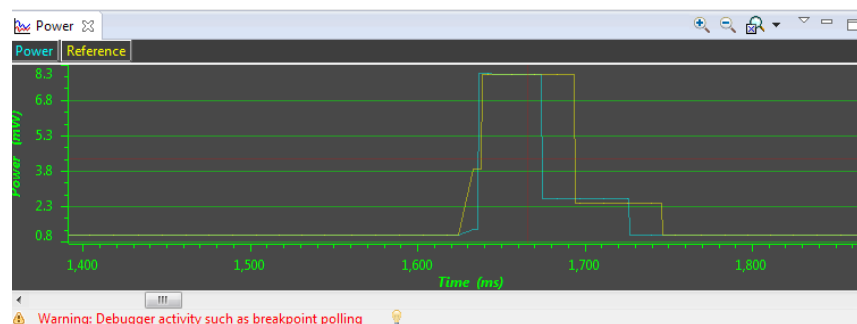
NOTE:

The formula to calculate the battery life time assumes an ideal 3-V battery and does not account for temperature, aging, peak current, and other factors that could negatively affect battery capacity. It should also be noted that changing the target voltage (for example, from 3.6 V to 3 V) might cause the analog circuitry to behave differently and operate in a more or less efficient state, hence reducing or increasing energy consumption. The value shown in the Profile window cannot substitute measurements on real hardware.

EnergyTrace™ Profile	
Name	Live
System	
Time	10 sec
Energy	14.61 mJ
Power	
Mean	1.75 mW
Min	1.068 mW
Max	7.943 mW
Voltage	
Mean	3.59 V
Current	
Mean	0.49 mA
Min	0.298 mA
Max	2.213 mA
Battery Life	CR2032: 18.8 day (est.)

**Figure 22. EnergyTrace Profile Window**

The **Power** window (see [Figure 23](#)) shows the dynamic power consumption of the target over time. The current profile is plotted in light blue color, while a previously recorded profile that has been reloaded for comparison is plotted in yellow color.



**Figure 23. Zoom Into Power Window**

The **Energy** window (see [Figure 24](#)) shows the accumulated energy consumption of the target over time. The current profile is plotted in light blue color, while a previously recorded profile that has been reloaded for comparison is plotted in yellow color.

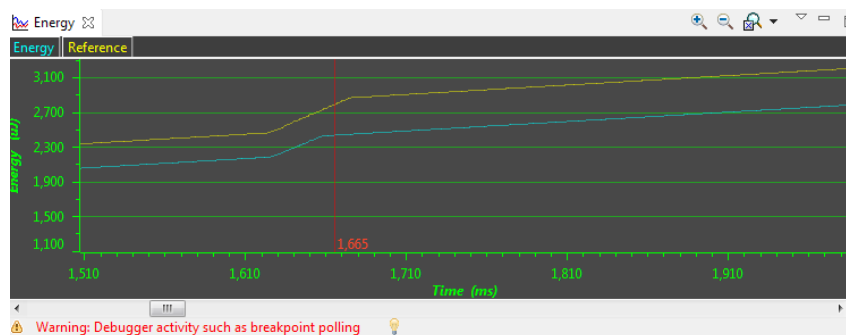


Figure 24. Current Profile (Blue) With Recorded Profile (Yellow)

**NOTE:** During program execution through the debugger's view **Resume** button, the target microcontroller is constantly accessed by the JTAG or Spy-Bi-Wire protocol to detect when a breakpoint has been hit. Inevitably, these debug accesses consume energy in the target domain and change the result shown in both Energy and Power graphs. To see the absolute power consumption of an application, it is recommended to use the **Free Run** mode. In Free Run mode, the debug logic of the target microcontroller is not accessed. See Figure 25 for an example of the effect of energy consumption coming from debug accesses. The yellow profile was recorded in **Resume** mode, and the green profile was recorded in **Free Run** mode.

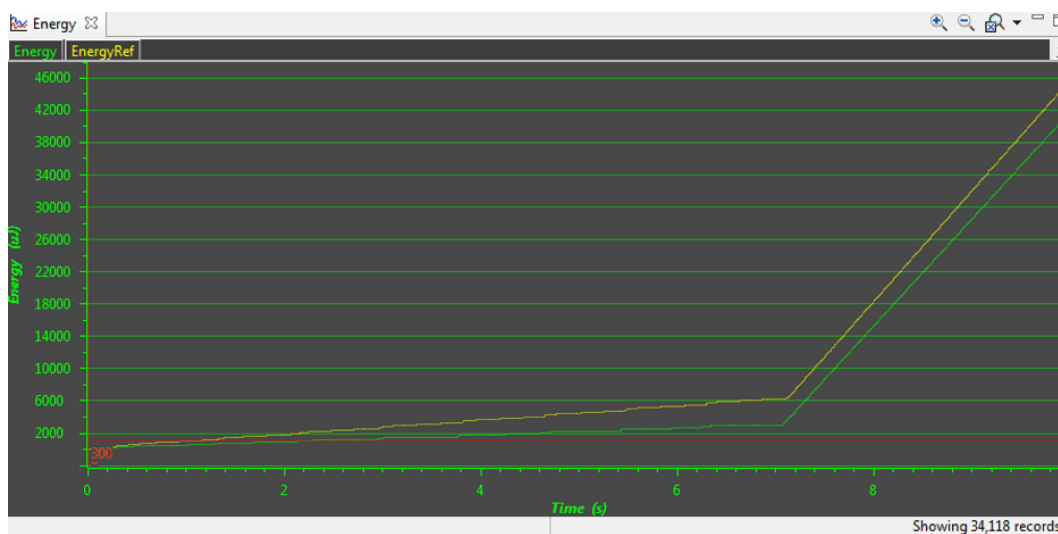


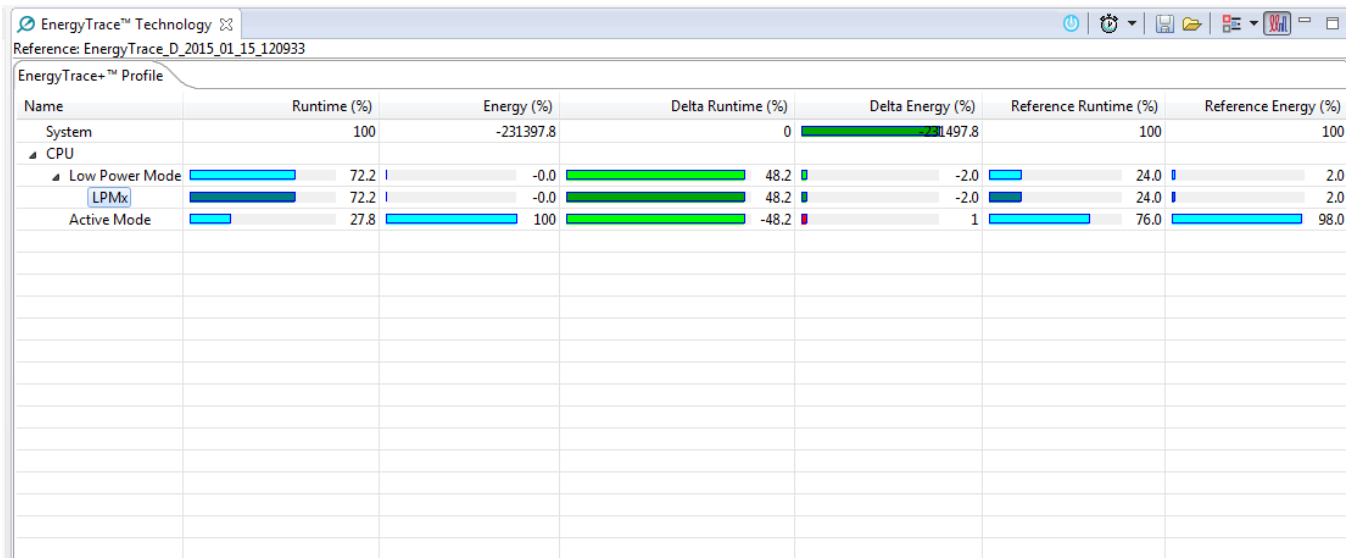
Figure 25. Energy Profile of the Same Program in Resume (Yellow Line) and Free Run (Green Line)

## 7.7 Comparing Captured Data With Reference Data

The EnergyTrace Technology can be used in various ways. One is to check the device's internal states over time against the expected behavior and correct any misbehavior; for example, due to a peripheral not being disabled after periodic usage. Another way is to compare the captured data against previously captured data. The previously captured data is called the *reference data* in the following discussion.

After the reference data has been loaded, a yellow reference graph is plotted in the Power and Energy windows. The Power window shows the power profiles of both data sets over time and is useful to determine any changes in static power consumption; for example, due to use of a deeper low-power mode or disabling of unused peripherals. It also shows how the dynamic power consumption has changed from one measurement to the other; for example, due to ULP Advisor hints being implemented. The Energy window shows the accumulated energy consumption over time and gives an indication which profile is more energy efficient.

In the **EnergyTrace+** mode, the condensed view of both captured and reference data is displayed in the Profile window (see [Figure 26](#)). You can quickly see how the overall energy consumption and use of power modes, peripherals, and clocks changed between both capture sessions. In general, parameters that have become better are shown with a green bar, and parameters that have become worse are shown with a red bar. For example, time spent in Active Mode is generally seen as negative. Hence, if a code change makes the application spend less time in active mode, the negative delta is shown as a green bar, and the additional time spent in a low-power mode is shown as a green bar.



Name	Runtime (%)	Energy (%)	Delta Runtime (%)	Delta Energy (%)	Reference Runtime (%)	Reference Energy (%)
System	100	-231397.8	0	-231497.8	100	100
CPU						
Low Power Mode	72.2	-0.0	48.2	-2.0	24.0	2.0
LPMx	72.2	-0.0	48.2	-2.0	24.0	2.0
Active Mode	27.8	100	-48.2	1	76.0	98.0

**Figure 26. Comparing Profiles in EnergyTrace+ Mode**

In the **EnergyTrace** mode, no States information is available to generate an exhaustive report. However, the overall energy consumed during the measurement is compared and, with it, the Min, Mean, and Max values of power and current. Parameters that have become better are shown with a green bar, and parameters that have become worse are shown with a red bar (see [Figure 27](#)).

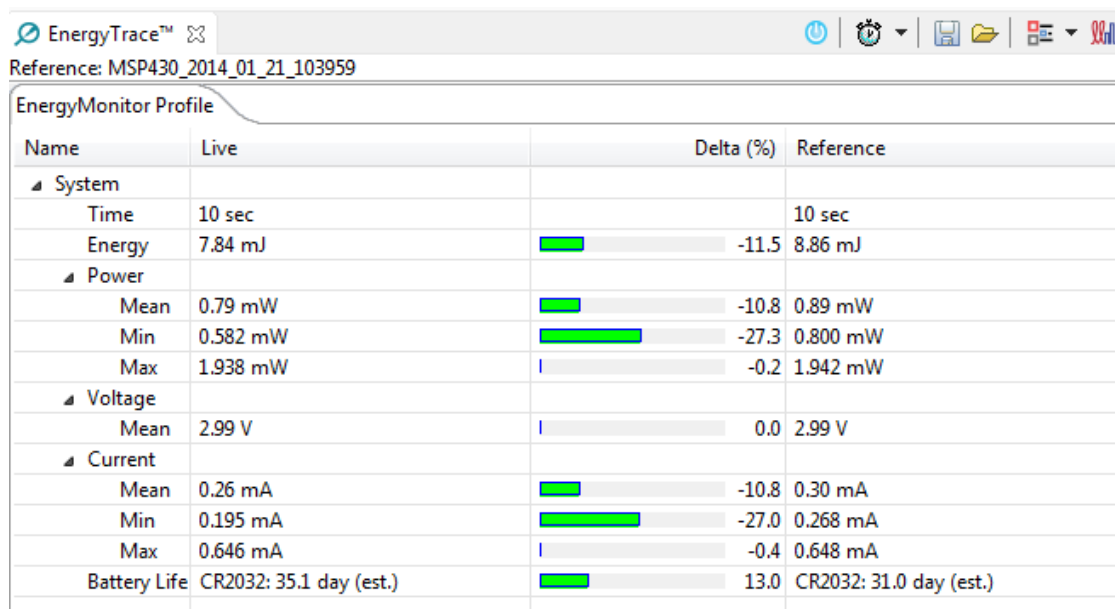


Figure 27. Comparing Profiles in EnergyTrace Mode

The delta bars are drawn linearly from 0% to 50%. Deltas larger than 50% do not result in a larger delta bar.

## 7.8 EnergyTrace Technology FAQs

*Q: What is the sampling frequency of EnergyTrace+ technology?*

A: The sampling frequency depends on the debugger and the selected debug protocol and its speed setting. It typically ranges from 1 kHz (for example, when using the Spy-Bi-Wire interface set to SLOW) up to 3.2 kHz (for example, when using the JTAG interface set to FAST). The debugger polls the state information of EnergyTrace+ from the device status information. Depending on the sampling frequency, a short or fast duty cycle active peripheral state may not be captured on the State graph. In addition, the higher sampling frequency affects the device energy consumption under EnergyTrace.

*Q: What is the sampling frequency of EnergyTrace technology?*

A: The sampling frequency to measure the energy consumption is the same independent of which debug protocol or speed and is approximately 4.2 kHz in Free Run mode.

*Q: My Power graph seems to include noise. Is my board defective?*

A: The power values shown in the Power graph are derived (that is, calculated) from the accumulated energy counted by the measurement system. When the target is consuming little energy, a small number of energy packets over time are supplied to the target, and the software needs to accumulate the dc-dc charge pulses over time before a new current value can be calculated. For currents under 1  $\mu$ A, this can take up to one second, while for currents in the milliamp range, a current can be calculated every millisecond. Additional filtering is not applied so that detail information is not lost. Another factor that affects the energy (and with it, the current) that is consumed by the target is periodic background debug access during normal code execution, either through capturing of States information or through breakpoint polling. Try recording in Free Run mode to see a much smoother Power graph.

*Q: I have a code that repeatedly calls functions that have the same size. I would expect the function profile to show an equal distribution of the run time. In reality, I see some functions having slightly more run time than expected, and some functions slightly less.*

A: During program counter trace, various factors affect the number of times a function is detected by the profiler over time. The microcontroller code could benefit from the internal cache, thus executing some functions faster than others. Another influencing factor is memory wait states and CPU pipeline stalls, which add time variance to the code execution. An outside factor is the sampling frequency of the debugger itself, which normally runs asynchronous to the microcontroller's code execution speed, but in some cases shows overlapping behavior, which also results in an unequal function run time distribution.

*Q: My profile sometimes includes an <Undetermined> low-power mode, and there are gaps in the States graph Power Mode section. Where does the <Undetermined> low-power mode originate from?*

A: During transitions from active mode to low-power mode, internal device clocks are switched off, and occasionally the state information is not updated completely. This state is displayed as <Undetermined> in the Profile window, and the States graph shows a gap during the time that the <Undetermined> low-power mode persists. The <Undetermined> state is an indication that the application has entered a low-power mode, but which mode cannot be accurately determined. If the application is frequently entering low-power modes, the <Undetermined> state will probably be shown more often than if the application only rarely uses low-power modes.

*Q: When capturing in EnergyTrace mode, the min and max values for power and current show deviation, even though my program is the same. I would expect absolutely the same values.*

A: The energy measurement method used on the hardware counts dc-dc charge pulses over time. Energy and power are calculated from the energy over time. Due to statistical sampling effects and charge and discharge effects of the output voltage buffer capacitors, it is possible that minimum and maximum values of currents vary by some percent, even though the program is identical. The captured energy, however, should be almost equal (in the given accuracy range).

*Q: What are the influencing factors for the accuracy of the energy measurement?*

A: The energy measurement circuit is directly supplied from the USB bus voltage, and thus it is sensitive to USB bus voltage variations. During calibration, the energy equivalent of a single dc-dc charge pulse is defined, and this energy equivalent depends on the USB voltage level. To ensure a good repeatability and accuracy, power the debugger directly from an active USB port, and avoid using bus-powered hubs and long USB cables that can lead to voltage drops, especially when other consumers are connected to the USB hub. Furthermore the LDO and resistors used for reference voltage generation and those in the calibration circuit come with a certain tolerance and ppm rate over temperature, which also influences accuracy of the energy measurement.

*Q: I am trying to capture in EnergyTrace+ mode or EnergyTrace mode with a MSP432 device that is externally powered, but there is no data shown in the Profile, Energy, Power and States window.*

A: Both EnergyTrace+ mode and EnergyTrace mode require the target to be supplied from the debugger. No data can be captured when the target microcontroller is externally powered.

*Q: I cannot measure LPM currents when I am capturing in EnergyTrace+ mode. I am expecting a few microamps but measure more than 150  $\mu$ A.*

A: Reading digital data from the target microcontroller consumes energy in the JTAG domain of the microcontroller. Hence, an average current of approximately 150  $\mu$ A is measured when connecting an ampere meter to the device power supply pins. If you want to eliminate energy consumption through debug communication, switch to EnergyTrace mode, and let the target microcontroller execute in Free Run mode.

*Q: My LPM currents seem to be wrong. I am expecting a few microamps, but measure more, even in Free Run mode or when letting the device execute without debug control from an independent power supply.*

A: The most likely cause of this extra current is improper GPIO termination, as floating pins can lead to extra current flow. Also check the JTAG pins again, especially when the debugger is still connected (but idle), as the debugger output signal levels in idle state might not match how the JTAG pins have been configured by the application code. This could also lead to extra current flow.



*Q: When I start the EnergyTrace+ windows through View → Other → EnergyTrace before launching the debug session, data capture sometimes does not start.*

*A: Enable EnergyTrace through Window → Preferences → Code Composer Studio → Advanced Tools → EnergyTrace™ Technology. When launching a debug session, the EnergyTrace+ windows automatically open, and data capture starts when the device executes. If you accidentally close all EnergyTrace+ windows during a debug session, you can reopen them through View → Other → EnergyTrace.*

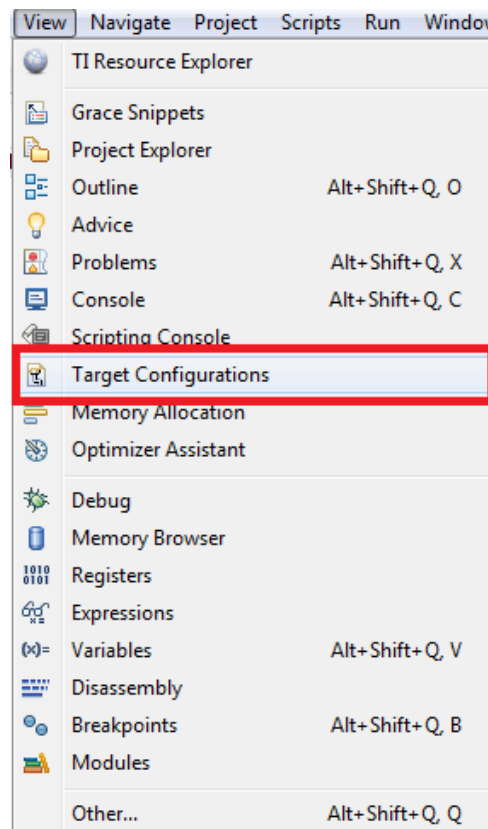
## 8 Device Security

If you have disabled JTAG access on the device or are working on an application where you need to unlock a secure IP zone, the lock can only be removed by erasing all Flash memory, including USER and INFO memory through a debugger invoked reboot cycle. To unlock a device, the following steps are required

- Select a Target Configuration that matches the current debugger type
- Execute a script that triggers a reboot erase

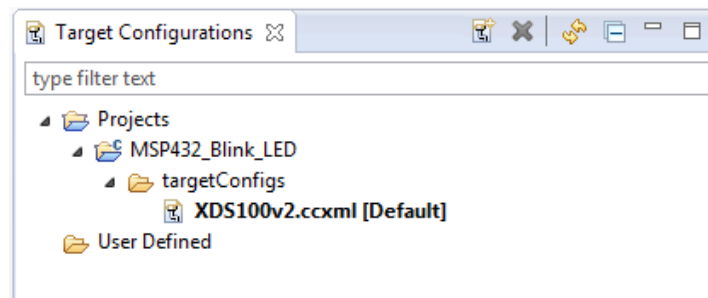
These steps are explained in detail next.

Go to **View → Target Configurations** to see the available debugger configurations.



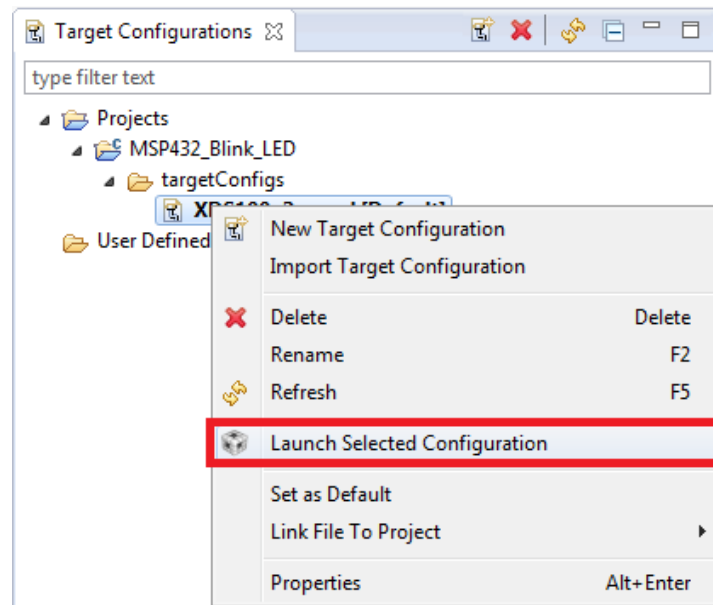
**Figure 28. Show Target Configuration View**

CCS opens a view that shows the target configurations it can identify in the current workspace. Pick the one that works for the device and debugger. This example uses the configuration file for an XDS100v2 debugger.



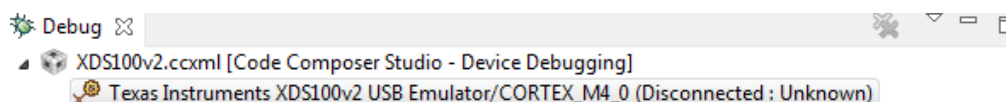
**Figure 29. List of Target Configurations**

Now right click on the target configuration and select **Launch Selected Configuration**.



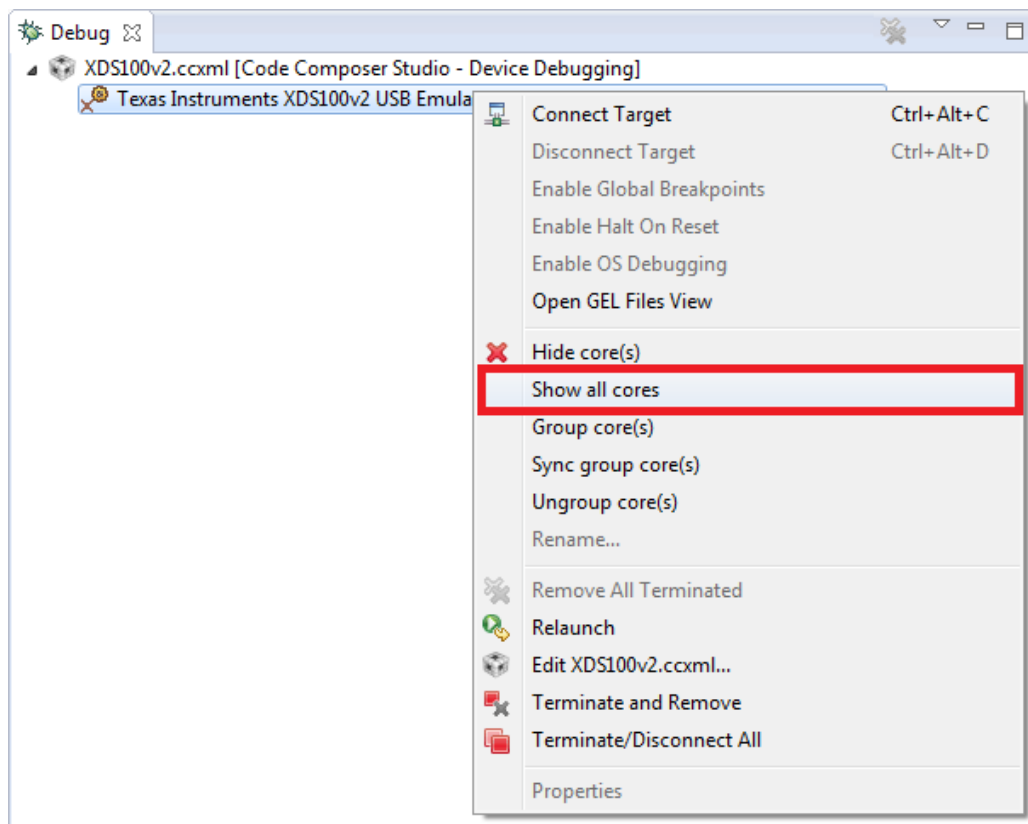
**Figure 30. Launch Selected Target Configuration**

The debugger now connects to the device (which is still possible), but does not try to halt the CPU, write to registers, or even download code (which would not be possible). The Debug view shows the CPU core, but marks it as disconnected.



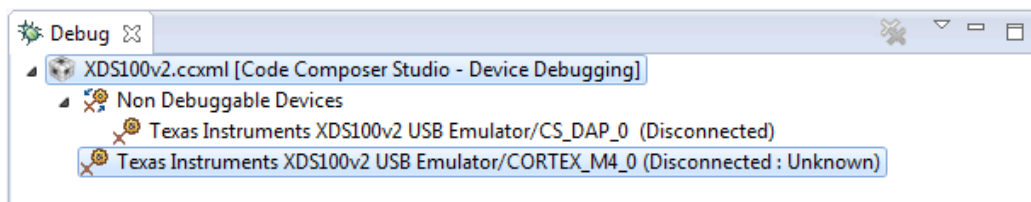
**Figure 31. Debug View After Launching Target Configuration**

To access the Debugger Access Port, or DAP, right-click on the highlighted line that shows the CPU core, and select **Show all cores** from the drop down menu.



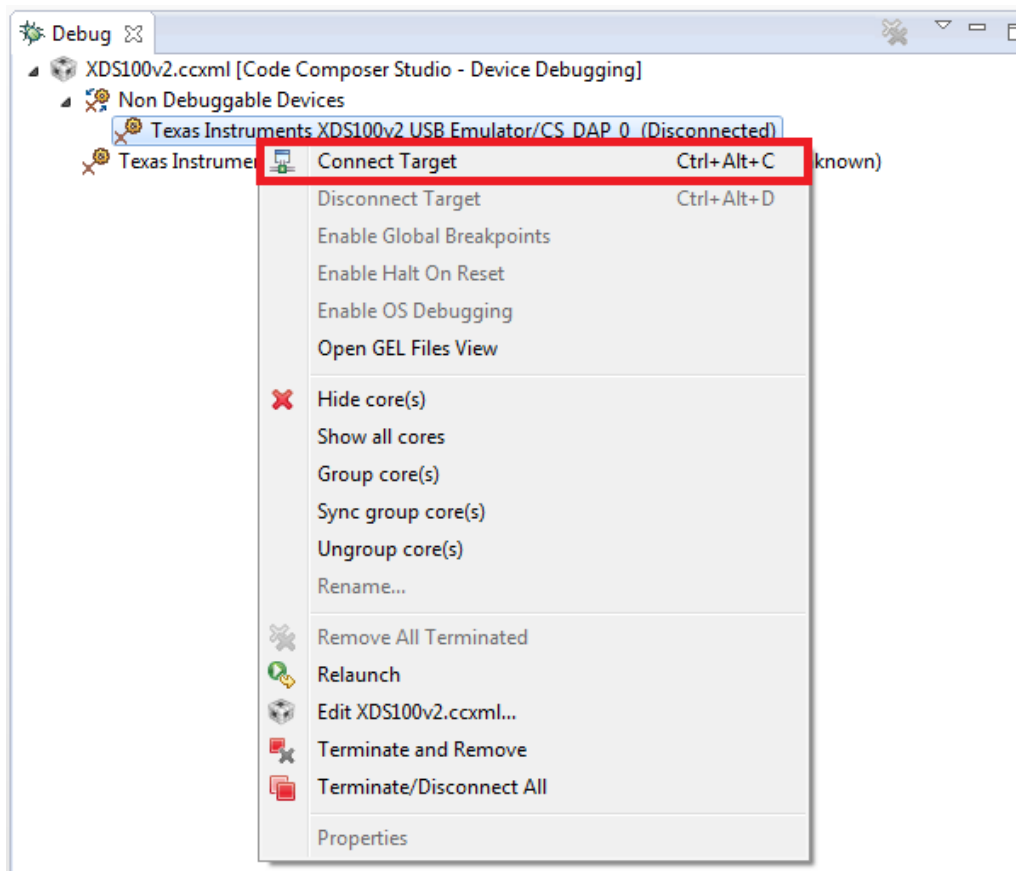
**Figure 32. Show All Cores**

The MSP432 Debug Access Port, or DAP, is now listed under **Non Debuggable Devices**.



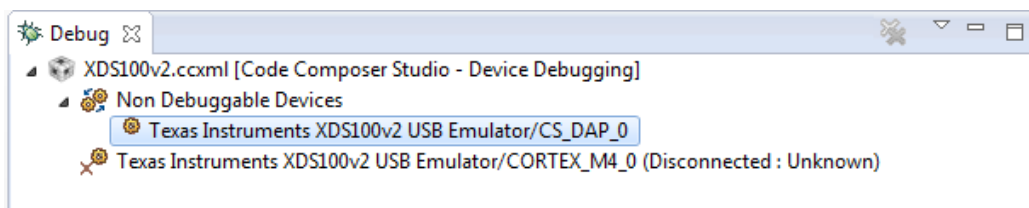
**Figure 33. List of All Cores in the MSP432**

Now right-click on the DAP and select **Connect Target**.



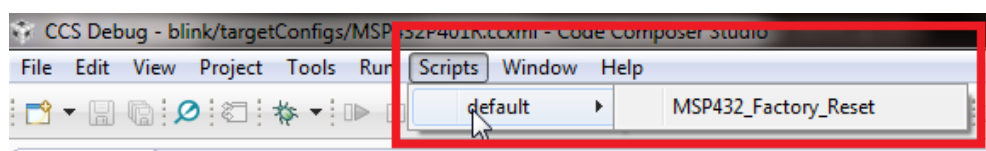
**Figure 34. Manually Connecting to the DAP**

When the debugger has connected to the DAP, the Debug view window changes, indicating a successfully connected DAP.



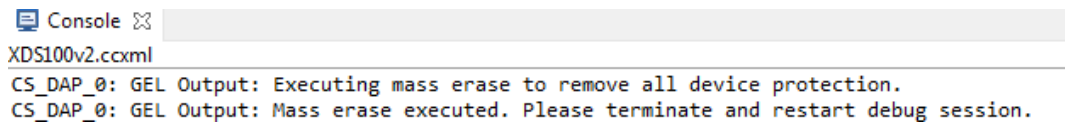
**Figure 35. DAP is Connected**

Now you need to execute a CCS script that performs triggers a reboot reset through the DAP. Click **Scripts** → **default** → **MSP432\_Factory\_Reset**.



**Figure 36. Executing the Factory Reset Script**

As the script is executed, the Console window shows that the mass erase has been executed. Now you can terminate the debug connection. After you have power cycled the device, it is accessible in a normal way again.



```

Console
XDS100v2.ccxml
CS_DAP_0: GEL Output: Executing mass erase to remove all device protection.
CS_DAP_0: GEL Output: Mass erase executed. Please terminate and restart debug session.

```

**Figure 37. Mass Erase Script Console Output**

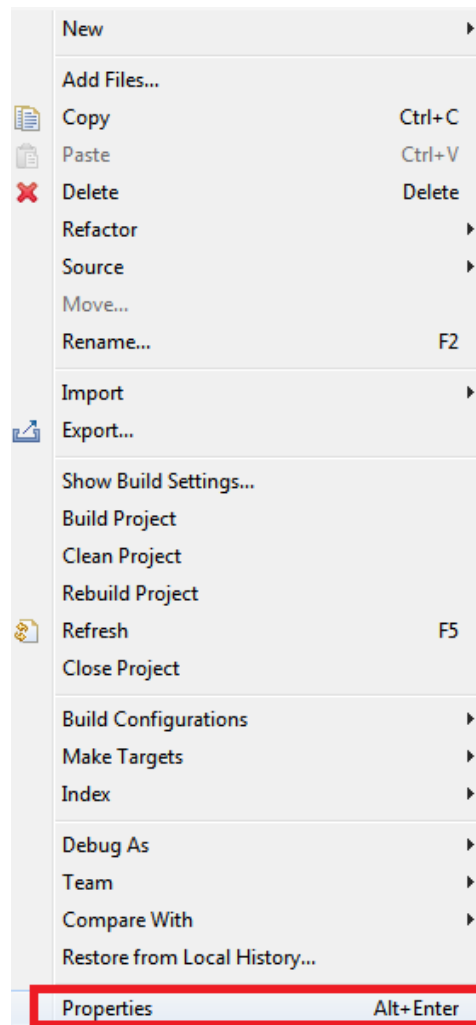
## 9 Low-Power Debug

Under normal debug control, MSP432 microcontrollers do not transition into low-power modes deeper than LPM0 mode; that is, into either LPM0\_VCORE0 or LPM0\_VCORE1 mode. This behavior is a consequence of the standardized Cortex-M debug architecture. Therefore, current consumption and IRQ wake-up timing are different from a free-running application.

To verify the current consumption and timing of an application while still under basic debug control, MSP432 offers the **Low Power Run** feature. When enabled, the MCU transitions into exactly the low-power mode that the application specifies, with internal clocks disabled and the power management module shutting down internal power domains. This has some implications:

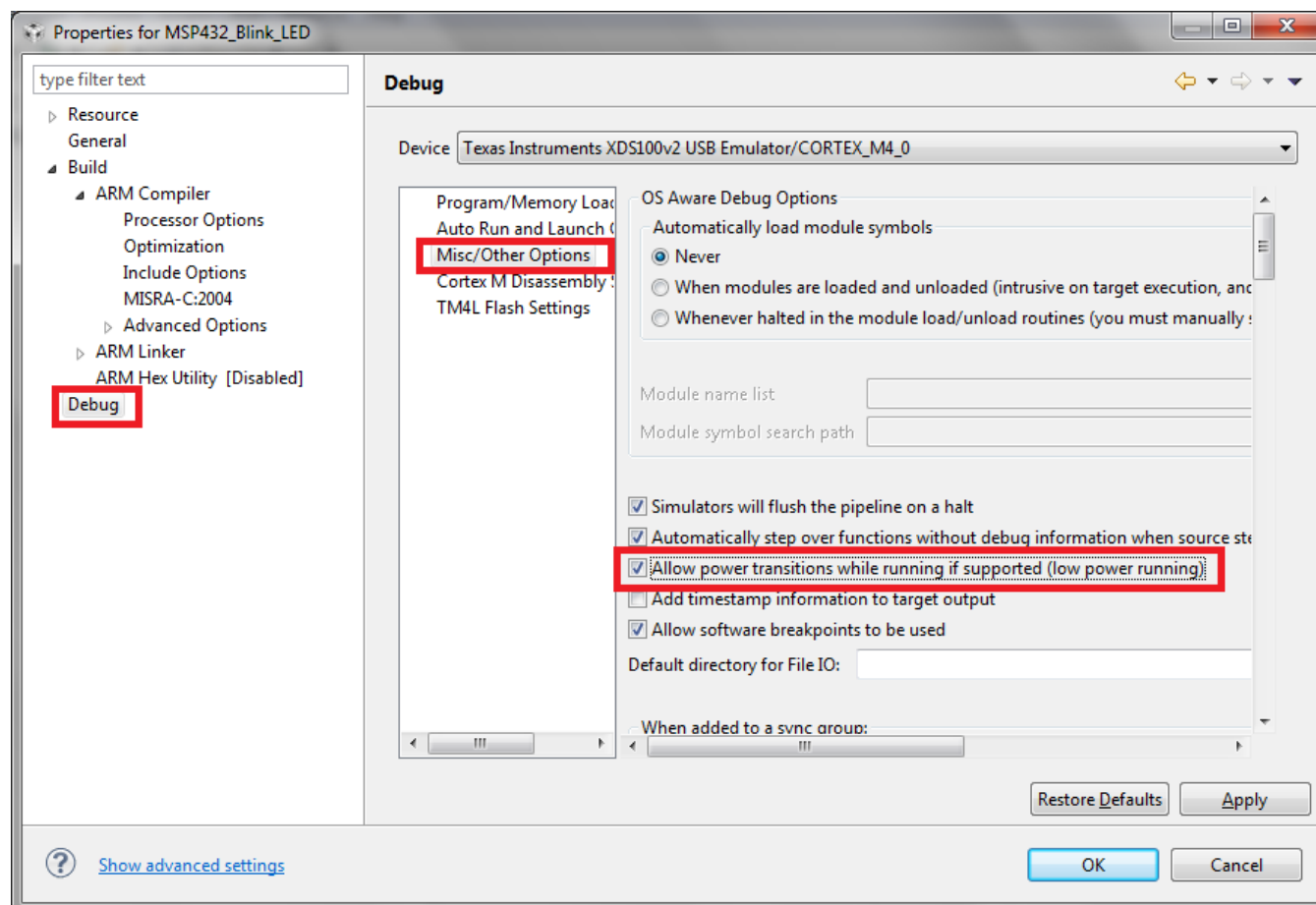
- Previously set breakpoints are reached, but the IDE does not automatically indicate that the device has halted. The user must click the halt icon to enable the IDE to reconnect and show where the program has halted.
- Auto Run after loading a program does not work: The breakpoint that is set automatically by the IDE [for example, at main()] is reached, but the IDE does not switch to halt automatically. When the user halts manually after program load, the program counter is at start of main().
- SWO trace does not work when transitioning into power modes lower than AM0\_SL or AM1\_SL mode

To enable the feature in CCS, right click on the active project in the Project Explorer and click on **Properties**.



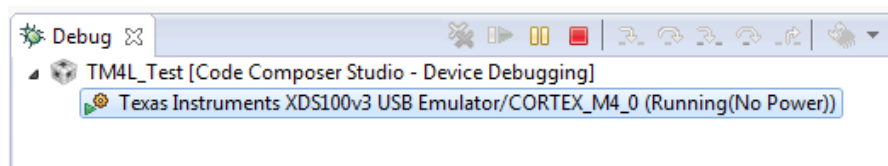
**Figure 38. Properties Menu**

In the Properties window, select **Debug**, then go to **Misc/Other Options**, and enable the **Allow power transitions while running if supported (low power running)** option.



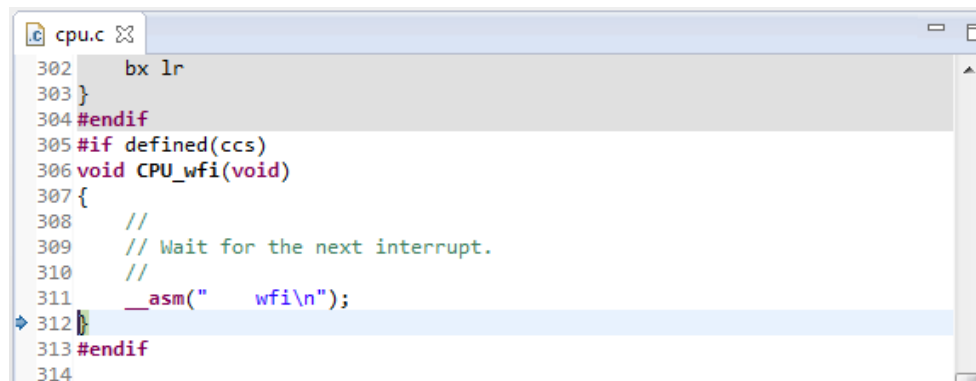
**Figure 39. Enabling Low Power Run**

When Low Power Run is enabled, the MCU will go to any low-power mode specified. You can verify the effect by watching the CPU core status display during a debug session. When the MCU goes to a low-power mode equal or deeper than DSL, the debugger will report a loss of connection, due to the MSP432 clocks all being disabled, including the clock that operates the Debug Access Port (DAP).



**Figure 40. CPU Core Status Display Indicating Deep Sleep Mode**

When halting the device, the debugger will reconnect, and the IDE will show the current program counter location, which in this case will be the WFI assembly instruction that sent the MSP432 to sleep.



**Figure 41. Program Counter Located at WFI Instruction**

## 10 Frequently Asked Questions

*Q: I cannot program my LaunchPad™ kit; the IDE cannot connect to target. What's wrong?*

A: Check the following:

- Is the JTAG switch (S101) in the correct orientation?

Switch to left for XDS110-ET onboard debugger

Switch to the right for external debugger connection

- Check the debugger settings: change to *SWD Mode – Aux COM port is target TDO pin*. When the settings of Port J (PJSEL0 and PJSEL1 bits) are changed, full JTAG access is prevented on these pins. Changing to use SWD allows access through the dedicated debug pins only. [Figure 42](#) shows how to configure the debugger to use SWD instead of JTAG by modifying the MSP432P401R.ccxml file.



**Figure 42. Change Debugger Settings to SWD**

- If even this cannot connect, reset the device to factory settings. Review [Section 8](#) for information on how to perform a factory reset on the device.



*Q: Why doesn't the backchannel UART on the MSP432 LaunchPad work with my serial terminal program at speeds faster than 56000 baud?*

A: Certain serial terminal programs such as HTerm or the CCS built-in terminal might not work with the MSP432 LaunchPad at specific baud rates, resulting in the software not being able to open the virtual COM port or in the baud rate getting configured incorrectly. An issue with the LaunchPad's emulator firmware has been identified and will be fixed in the next release. Until the update is available, use Tera Term, ClearConnex, or HyperTerminal instead or reduce the baud rate to speeds of 38400 baud or lower.

*Q: Problems plugging the MSP432 LaunchPad into a USB3.0 Port*

A: It has been observed that when the MSP432 LaunchPad is connected to USB3.0 ports provided by a certain combination of USB3.0 host controller hardware and associated device drivers that the IDE is unable to establish a debug session with the LaunchPad, resulting in an error message like "CS\_DAP\_0: Error connecting to the target: (Error -260 @ 0x0) An attempt to connect to the XDS110 failed" in the case of Code Composer Studio. In this case the CCS-provided low-level command line utility 'xdsdfu' will also not be able to establish a connection with the LaunchPad.

Specifically, this issue was observed on PCs running Windows 7 that show the "Renesas Electronics USB 3.0 Host Controller" and the associated "Renesas Electronics USB 3.0 Root Hub" in the device manager. After updating the associated Windows USB drivers to more recent versions obtained from the hardware vendor the issue went away. There might be other USB3.0 hardware and device driver combinations that will lead to the same issue. If you think you might be affected, contact the PC vendor or locate and install more recent versions of the USB3.0 device drivers. Alternatively, connect the LaunchPad to an USB2.0 port on the PC if available.

*Q: I cannot get the backchannel UART to connect. What's wrong?*

A: Check the following:

- Do the baud rate in the host's terminal application and the eUSCI settings match?
- Are the appropriate jumpers in place, on the isolation jumper block?
- Probe on RXD and send data from the host. If you don't see data, it might be a problem on the host side.
- Probe on TXD while sending data from the MSP432. If you don't see data, it might be a configuration problem with the eUSCI module.
- Consider the use of the hardware flow control lines (especially for higher baud rates).

*Q: How can I easily unlock the SYS\_CTL register block?*

A: For TI ARM compiler, there is a macro define available for doing so. See `#define UNLOCK_DEVICE` in the corresponding device header.

## 11 Additional Code Composer Studio Information

For more information about Code Composer Studio, refer to the following links:

- [Code Composer Studio Information](#)
- [Code Composer Studio v6 Training](#)
- [Code Composer Studio v6 Wiki](#)
- [Code Composer Studio Quick Start Guide](#)

## 12 References

1. [MSPWare](#)
2. [J-Link Emulator Support](#)
3. [MSP432™ Debugging Tools: Using Serial Wire Output With CCS Trace Analyzer](#)
4. [MSP Driver Library](#)
5. [MSP-FET for MSP432](#)

## Revision History

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

Changes from February 23, 2016 to May 13, 2016	Page
• Added the list item for MSP-FET in <a href="#">Section 5, Debugging Your Application</a> .....	8
• Added the sentence that starts "In CCS 6.1.3.x, the Segger installation..." in <a href="#">Section 5, Debugging Your Application</a> ....	8
• Added item 5 in <a href="#">Section 12, References</a> .....	33

## 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

Audio	<a href="http://www.ti.com/audio">www.ti.com/audio</a>
Amplifiers	<a href="http://amplifier.ti.com">amplifier.ti.com</a>
Data Converters	<a href="http://dataconverter.ti.com">dataconverter.ti.com</a>
DLP® Products	<a href="http://www.dlp.com">www.dlp.com</a>
DSP	<a href="http://dsp.ti.com">dsp.ti.com</a>
Clocks and Timers	<a href="http://www.ti.com/clocks">www.ti.com/clocks</a>
Interface	<a href="http://interface.ti.com">interface.ti.com</a>
Logic	<a href="http://logic.ti.com">logic.ti.com</a>
Power Mgmt	<a href="http://power.ti.com">power.ti.com</a>
Microcontrollers	<a href="http://microcontroller.ti.com">microcontroller.ti.com</a>
RFID	<a href="http://www.ti-rfid.com">www.ti-rfid.com</a>
OMAP Applications Processors	<a href="http://www.ti.com/omap">www.ti.com/omap</a>
Wireless Connectivity	<a href="http://www.ti.com/wirelessconnectivity">www.ti.com/wirelessconnectivity</a>

### Applications

Automotive and Transportation	<a href="http://www.ti.com/automotive">www.ti.com/automotive</a>
Communications and Telecom	<a href="http://www.ti.com/communications">www.ti.com/communications</a>
Computers and Peripherals	<a href="http://www.ti.com/computers">www.ti.com/computers</a>
Consumer Electronics	<a href="http://www.ti.com/consumer-apps">www.ti.com/consumer-apps</a>
Energy and Lighting	<a href="http://www.ti.com/energy">www.ti.com/energy</a>
Industrial	<a href="http://www.ti.com/industrial">www.ti.com/industrial</a>
Medical	<a href="http://www.ti.com/medical">www.ti.com/medical</a>
Security	<a href="http://www.ti.com/security">www.ti.com/security</a>
Space, Avionics and Defense	<a href="http://www.ti.com/space-avionics-defense">www.ti.com/space-avionics-defense</a>
Video and Imaging	<a href="http://www.ti.com/video">www.ti.com/video</a>

### TI E2E Community

[e2e.ti.com](http://e2e.ti.com)