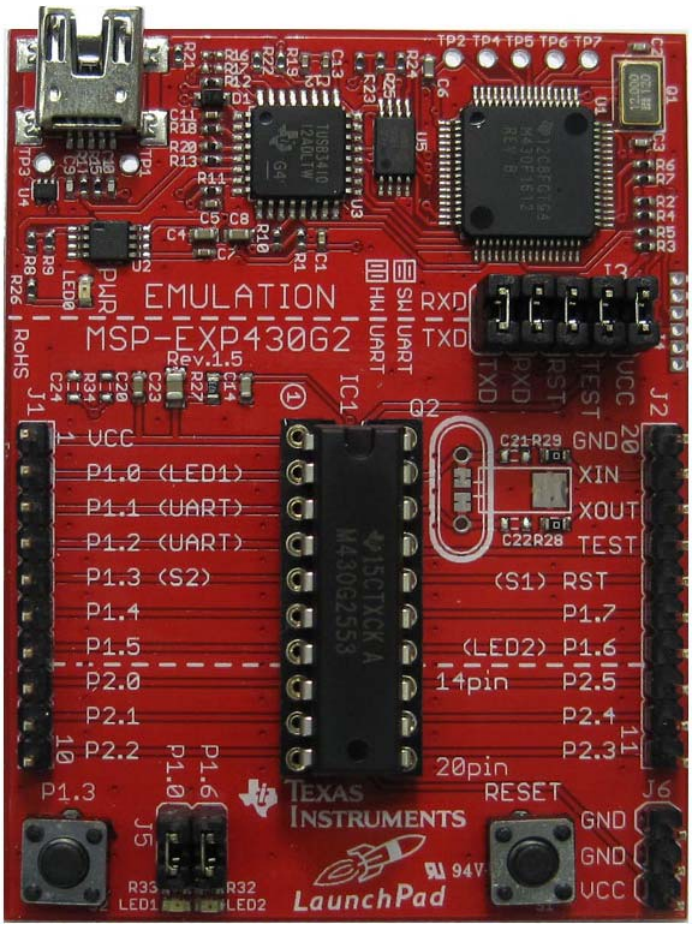
MSP430 LaunchPad Lab Manual



TEXAS INSTRUMENTS

Contents

List of Figures

List of Tables

Preface

The MSP430 family of microcontrollers from Texas Instruments are ultra-low-power, mixed-signal processors that are ideal for industrial and consumer application. These devices feature a powerful 16-bit, RISC (Reduced Instruction Set) CPU, 16-bit registers, and constant generators which results in maximum code efficiency. The MSP430 family also consists of several devices featuring different sets of peripherals targeted for various applications. The MSP430 architecture combined with low-power modes is optimized to achieve an extended battery life in portable measurement applications. The Digitally Controlled Oscillator (DCO) of the MSP430 allows wake-up from a low-power mode to an active mode in less than 1 microsecond. In short, the MSP430 family of microcontrollers offer the best balance between energy efficiency and performance.

In this manual, MSP430G2553, MSP430F5529 and MSP430FR5969 platforms are used. The experiments take advantage of a wide range of parts and tools supported by Texas Instruments for design with the MSP430 series of microcontrollers and wireless communication products. These easily-available evaluation kits assist the users to get started with designing applications using MSP430 series microcontroller.

Each experiment in this manual includes:

- An introduction to the features of the microcontroller that is used
- A design overview
- The software flowchart and the C program for the application
- The procedure to run the application and document the observations
- An exercise to further enhance learning

Experiments 1 to 8 are implemented using MSP430G2 Launchpad which includes typical applications like GPIO, PWM, ADC, Interrupt, Low Power Modes, UART. Experiments 9 and 10 are implemented using MSP430F5529 Launchpad which includes SPI and WIFI applications. Experiment 11 is implemented using MSPEXP430FR5969 Launchpad which has an on-board Energy Trace Technology Circuitry as the experimenter board and the MSP430G2 Launchpad as a target module to achieve Energy Trace Technology. Experiment 12 is implemented using MSP430G2 Launchpad to test battery life.

On completion of all the experiments in this manual, the user will be able to build a variety of production-ready applications with the MSP430 Series microcontrollers.

Introduction

Topics **Page**

Overview

Introduction

The MSP microcontrollers (MCUs) from Texas Instruments (TI) are 16-bit and 32-bit RISC-based, mixed-signal processors designed for ultra-low power applications. The MSP430 family of microcontrollers consists of several devices featuring different sets of peripherals targeted for various applications. The MSP430 device features a powerful 16-bit RISC CPU, 16-bit registers, and constant generators that contribute to maximum code efficiency. The architecture, which includes five low-power modes, is optimized to achieve extended battery life in portable measurement applications. The digitally controlled oscillator (DCO) allows wake-up from low-power modes to active mode in less than 1second. The MSP430 instruction set consists of 27 core instructions that make it easy to program in assembler or in C, and provide exceptional flexibility and functionality.

MSP low-power + performance microcontrollers from TI provide designers with enhanced performance and more peripherals on chip while using less power. MSP ultra-low-power microcontrollers are ideal for applications where the majority of the microcontroller life is spent in standby. Typical applications of these processors include low-cost sensor systems that capture analog signals, convert them to digital values, and then process the data for display or for transmission to a host system.

MSP430 Family

In this manual, MSP430G2553, MSP430F5529, MSP430FR5969 platforms are used. The MSP430G2553 series are ultra-low-power mixed signal microcontrollers with built-in, 16-bit timers; up to 24 I/O capacitive-touch enabled pins; a versatile analog comparator; and built-in communication capability using the universal serial communication interface. In addition, the MSP430G2553 family members have a 10-bit Analog-to-Digital Converter (ADC).

The features of MSP430G2553 and MSP430F5529 is shown in **Table 1**.

Table 1: Features of MSPEXP430G2553 and MSP430x5xx Series Processors

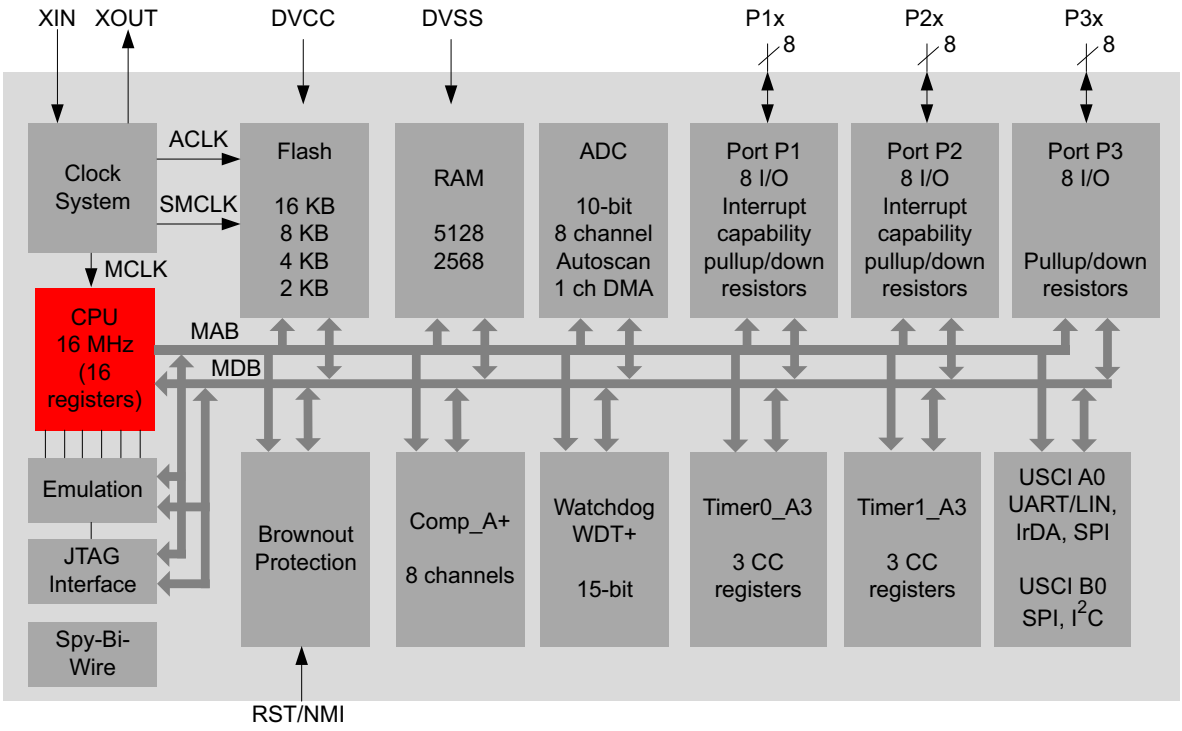
MSP430 G2 Series	MSP430x5xx series
Low supply voltage range: 1.8 V to 3.6 V	**Programmable core voltage with integrated PWM**
Low-power consumption • 220 µA / MHz active mode • 0.7 µA real-time clock mode • 0.1 µA RAM retention	**Ultra low power consumption** • 195 µA / MHz active • 2.5 µA real-time clock mode • 0.1 µA RAM retention • Fast wake-up from Standby Mode in <5 µs
16-bit RISC Architecture	**16-bit RISC Architecture**
Basic clock module configurations Internal frequencies up to 16 MHz with four calibrated frequency	**Unified clock system** Internal frequencies up to 25 MHz with six independent clock sources
Serial communication modules • USART • USI	**Serial communication modules** • USCI module • Supports up to two I^2C, four SPI and two UART interfaces

Table 1: Features of MSPEXP430G2553 and MSP430x5xx Series Processors

MSP430 G2 Series	MSP430x5xx series
Memory • Up to 56 KB Flash • Up to 4 KB SRAM	**Memory** • Up to 512 KB Flash • Up to 66 KB RAM

For reference, the functional block diagram, pin diagram, description of MSP430G2553 are discussed in the sections that follow. For other references like MSP430F5529 and MSP430FR5969, refer the datasheet link in the reference section.

Functional Block Diagram of MSP430G2553



Note: Port P3 is available on 28-pin and 32-pin devices only

Figure 1 Functional Block Diagram of MSP430G2553

The functional block diagram for the MSP430G2553 is as shown in **Figure 1**. The architecture of the MSP430 family is based on a memory-to-memory architecture, a common address space for all functional blocks, and a reduced instruction set applicable for all functional blocks.

The MSP430G2553 **CPU** has a 16-bit RISC architecture integrated with 16 registers that provide reduced instruction execution time. All operations, other than program-flow instructions, are performed as register operations.

Memory of the MSP430G2553 is divided to program memory (ROM) and data memory (RAM). The program memory is accessed by the 16-bit Memory Data bus. The data memory is accessed by the Memory Address Bus (MAB) and the Memory Data Bus (MDB).

Peripheral modules such as Digital I/O, Analog-to-Digital Converter (ADC), Timers, Universal Asynchronous Receiver Transmitter (UART), Universal Serial Communications Interface (USCI), Comparator_A+, etc., are connected to the CPU via MAB, MDB and interrupt service and request lines. The operations of the various peripherals are controlled by the contents of the Special Function Registers (SFRs).

The **clock system** is supported by the basic clock module that includes support for a 32768-Hz watch crystal oscillator, an internal very-low-power, low-frequency oscillator and an internal Digitally Controlled Oscillator (DCO).

The **brownout** circuit is implemented to provide the proper internal reset signal to the device during power on and power off.

The **Watchdog Timer** (WDT+) module restarts the system on occurrence of a software problem or if a selected time interval expires. The WDT can also be used to generate timely interrupts for an application.

The pin diagram for the MSP430G2553 is given in **Figure 2** below.

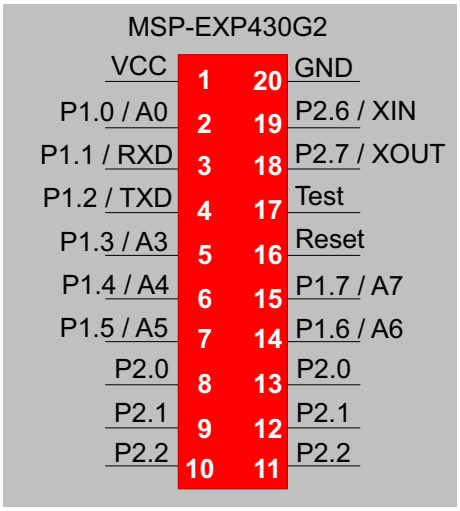


Figure 2 Pin Diagram of MSP430G2553

MSP430G2553 Address Space

The MSP430G2553 von-Neumann architecture has one address space shared with Special Function Registers (SFRs), peripherals, RAM, and Flash/ROM memory as shown in **Figure 3**.

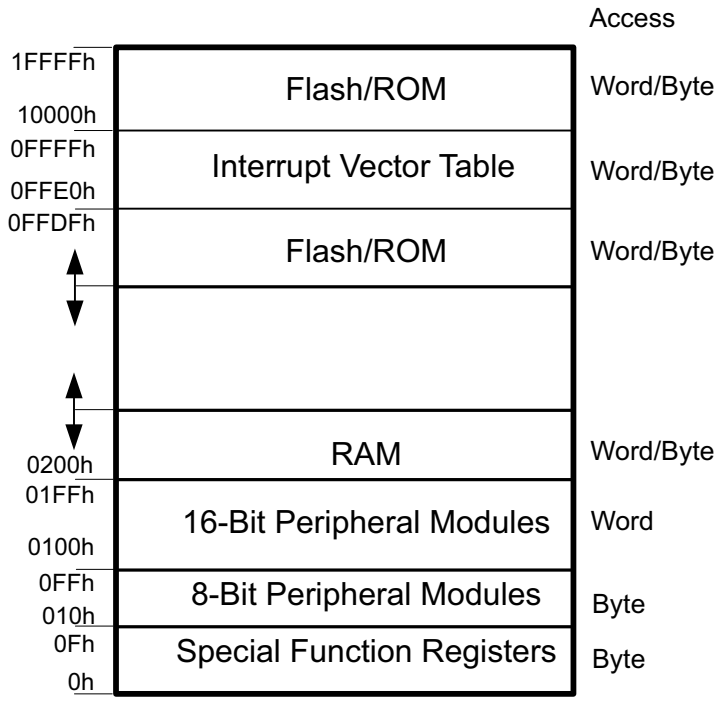


Figure 3 MSP-EXP430G2 von-Neumann Architecture

Data in memory can be accessed as either bytes (8-bit) or words (16-bit). The Flash (ROM) and RAM memories can be used for both code and data. The peripheral modules of the processor are mapped into the address space for read/write access. Hence, the address space from 0100 to 01FFh is reserved for 16-bit peripheral modules that can be accessed with word instructions. The address space from 010h to 0FFh is reserved for 8-bit peripheral modules that are accessed with byte instructions. SFRs control the operation of the various modules in the processor. Some peripheral functions are also configured in the SFRs.

MSP-EXP430G2 LaunchPad

Introduction

The MSP-EXP430G2 is a low-cost experimenter board, also known as LaunchPad. It provides a complete development environment that features integrated USB-based emulation and all of the hardware and software necessary to develop applications for the MSP430G2xx Value Line series devices.

The MSP430 LaunchPad kit box includes:

- LaunchPad emulator socket board (MSP-EXP430G2)
- Mini USB-B cable
- MSP430G2553 (pre-installed and pre-loaded with demo program)
- MSP430G2452 processor
- 10-pin PCB connectors are soldered to the board; two female are also included
- 32.768 kHz micro crystal (not soldered to the board)
- Quick start guide and two LaunchPad stickers

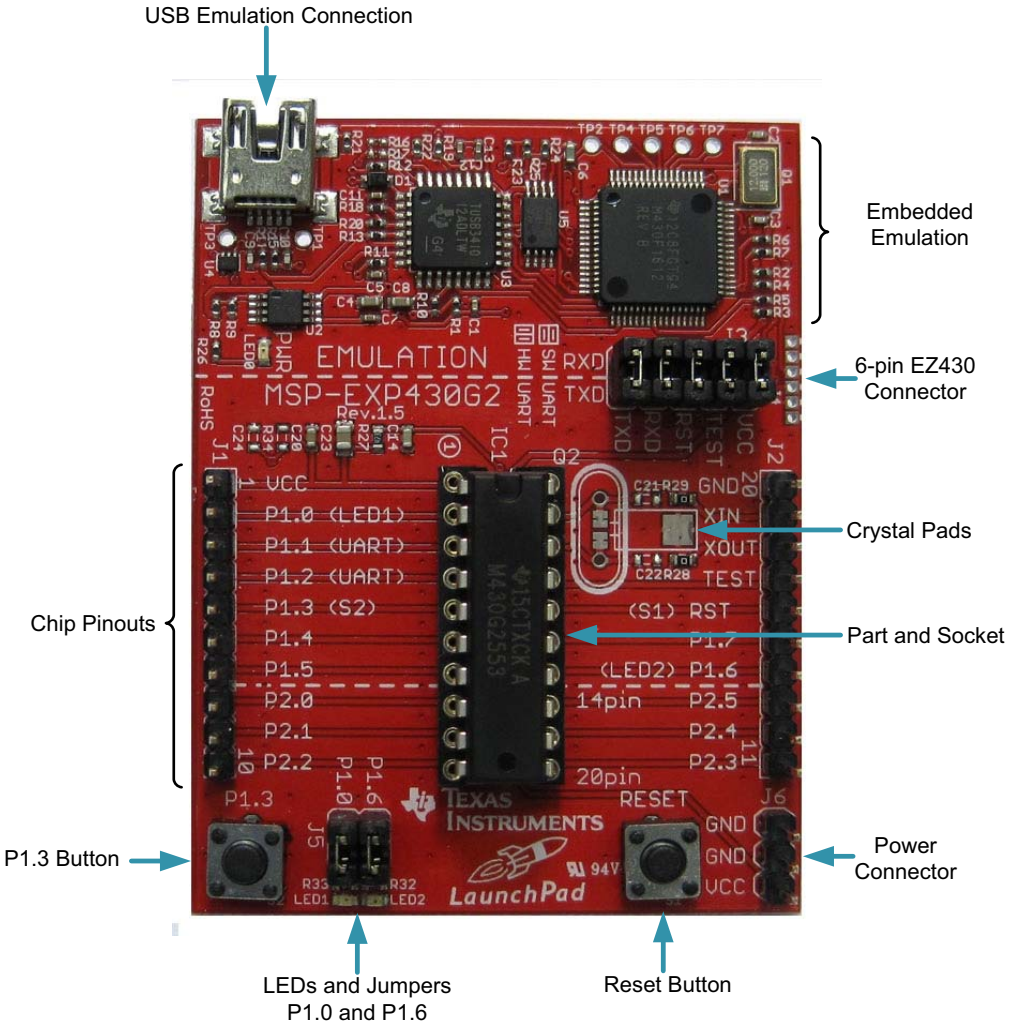


Figure 4 MSP-EXP430G2 LaunchPad

Hardware Setup

The LaunchPad experimenter board includes a pre-programmed MSP430 device which is already located in the target socket. When the LaunchPad is connected to your PC via USB, the demo starts with an LED toggle sequence. The on-board emulator generates the supply voltage and all of the signals necessary to start the demo.

1. Connect the MSP430 LaunchPad to your PC using the included USB cable.

2. The driver installation starts automatically. If prompted for software, allow Windows to install the software automatically.

At this point, the on-board red and green LEDs should toggle back and forth. This confirms that the hardware is working and has been set up correctly.

The Application Demo Program

The pre-programmed application demo takes temperature measurements using the internal temperature sensor. This demo exercises the various on-chip peripherals of the MSP430 device and can transmit the temperature via UART to the PC for display.

Press button P1.3 (lower-left) to switch the application to the temperature measurement mode.

A temperature reference is taken at the beginning of this mode and the LEDs on the LaunchPad signal a rise or fall in temperature by varying the brightness of the on-board red LED for warmer or green LED for cooler temperatures.

Rub your fingertip on your clothing to warm it up and place it on the top of the MSP430 device on the LaunchPad board. After a few seconds the red LED should start to light, indicating a temperature rise. When the red LED is solidly lit, remove your finger and press button P1.3 again. This will set the temperature reference at the higher temperature. As the part cools, the green LED will light, indicating decreasing temperature. Also remember that ambient temperatures will affect this exercise.

Documents for Reference

- MSP430G2553 Datasheet
- MSP430x2xx User's Guide
- C Compiler User's Guide
- MSP430F5529 Datasheet
- MSP430F5529 User's Guide
- MSP430FR5969 Datasheet
- MSP430FR5969 User's Guide

Code Composer Studio

Overview

Code Composer Studio (CCS) is the integrated development environment for the whole span of TI Microcontrollers, DSPs and application processors. Code Composer Studio includes a suite of tools used to develop and debug embedded applications. For all device families, CCS includes compilers, source code editor, project build environment, debugger, profiler, simulators and many other features.

Code Composer Studio v6 is the latest version of the IDE used currently and is based on Eclipse 4.3, whereas the earlier version CCSv5 was based on the Eclipse 3.x stream. The CCS App center, introduced in CCSv6, lists all the relevant packages for the chosen devices during the installation.

The Getting Started View, which is open by default when CCS is first launched, provides fast access to many of the common tasks the users would prefer to experiment when starting to use a new environment, such as creating a new project, browsing examples and visiting the App Center. There are also links to the support forums, YouTube videos, training material and the wiki. There is also a video that is prominently displayed in the center of the screen that walks users through the steps to use the CCS environment. It also provides the user the option of switching to "Simple Mode", which strips down the CCS user interface to a more basic set of functionalities with a reduced number of buttons, menu items and views that are open at a given point in time. Simple mode is also recommended for those picking up a LaunchPad for the first time and staring to work with it.

CCS Functional Overview

The CCS supports all the phases of the development cycle: design, code and build, debug. The CCS includes all the development tools - compilers, assembler, linker, debugger, BIOS and one target - the Simulator as shown in **Figure 5**.

- The **C compiler** converts C source code into assembly language source code.
- The **assembler** translates assembly language source files into machine language object files.
- The **standard run-time libraries** contain ANSI standard run-time-support functions, compiler-utility functions, floating-point arithmetic functions and I/O functions that are supported by the C compiler.
- The **linker** combines object files, library files and system or BIOS configuration files into a single executable object module. The .out file is the executable program for the target device.
- The **debugger** supports debug activities such as watching variables, graphing signals on the target, viewing the memory and call stack, etc. The .gel files initialize the debugger with address of memory, peripherals and other hardware setup details.
- The program code can be debugged on the **simulator** or **emulator** or the **target device** based on the target config file (.ccxml) that specifies the connection to the target.

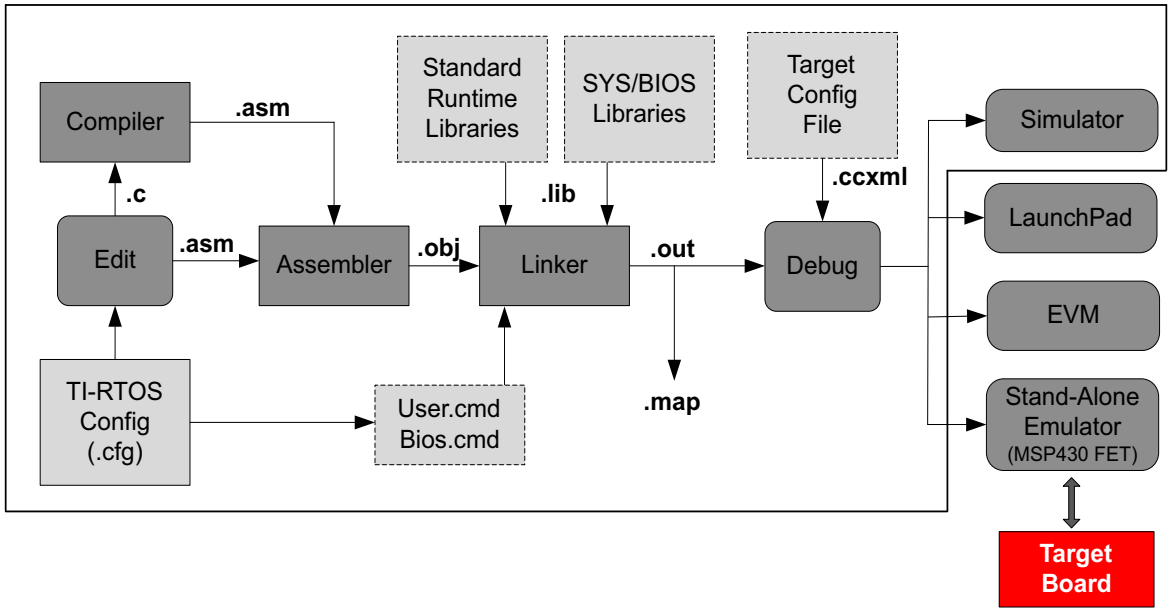


Figure 5 CCS Functional Overview

CCS Perspectives

The Code Composer GUI has two perspectives:

- **THE EDIT PERSPECTIVE**
- **THE DEBUG PERSPECTIVE**

The **edit perspective** has menus, buttons and editor window to edit the source code as shown in **Figure 6**.

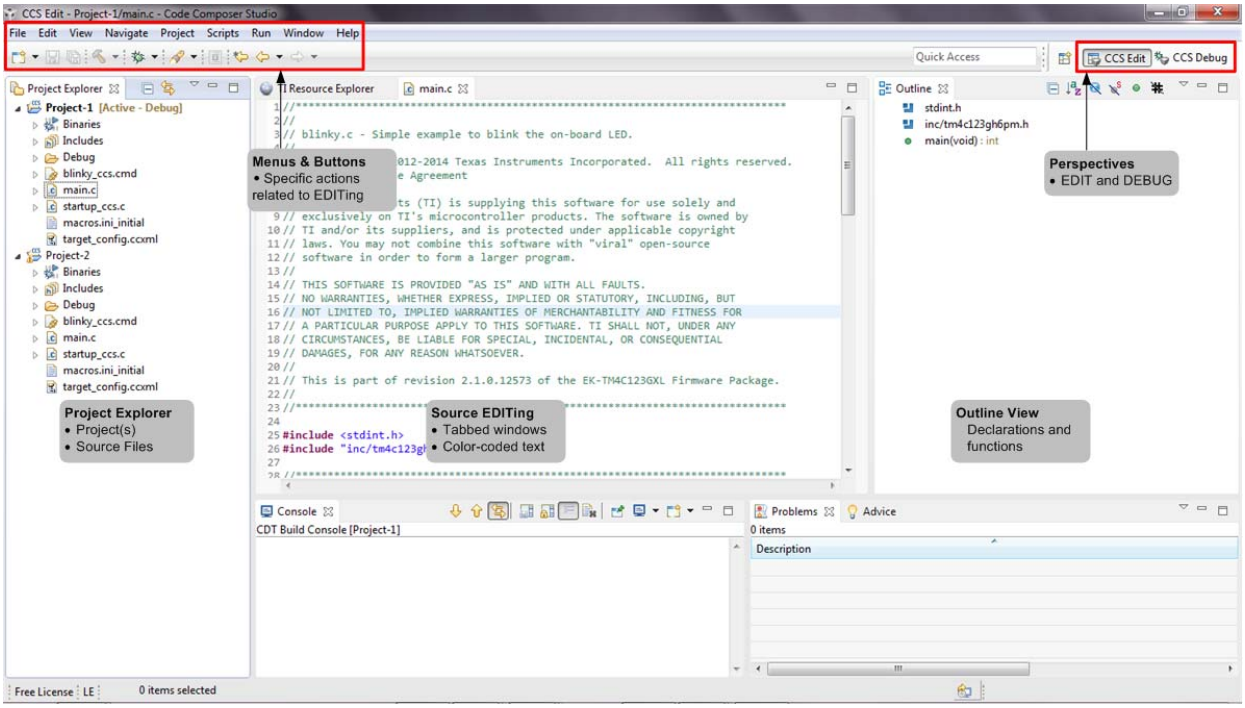


Figure 6 CCS Edit Perspective

The **debug perspective** has menus, buttons and debug window related to debugging as shown in **Figure 7**.

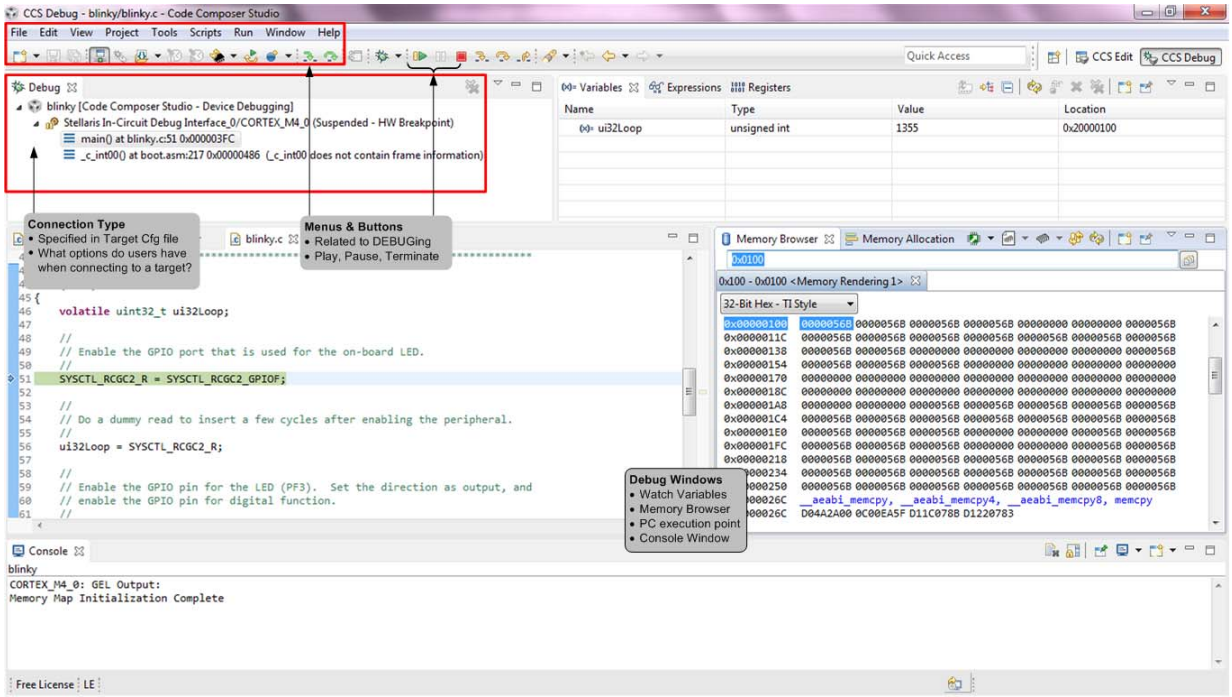


Figure 7 CCS Debug Perspective

Workspaces and Projects

A workspace contains settings and preferences, as well as links to the projects as shown in **Figure 8**.

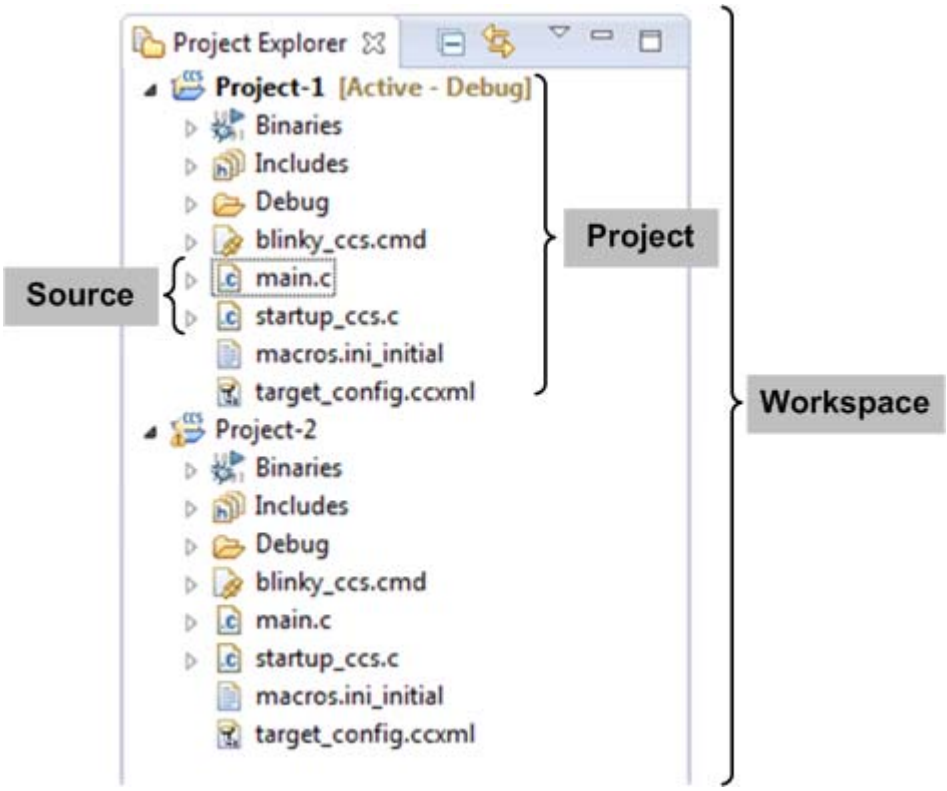


Figure 8 Workspace and Projects (as viewed in CCS)

A project contains build and tool settings, as well as links to the input files - source files, header files and library files as shown in **Figure 9**.

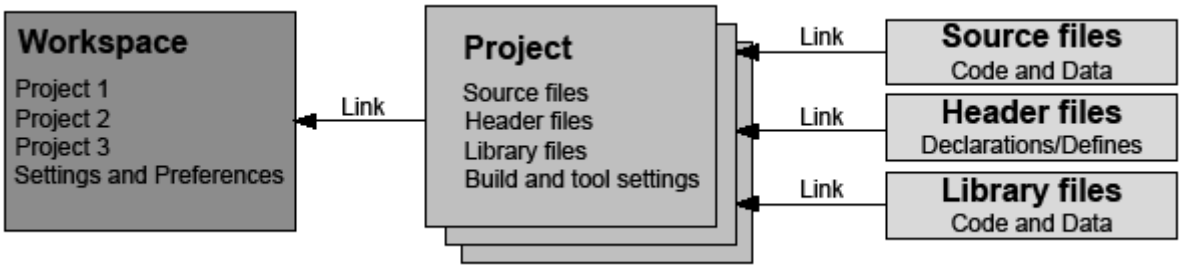


Figure 9 Workspace and Projects

Hence, deleting projects from the workspace deletes the links, not the projects. Similarly, deleting files from the project deletes the links, not the files.

Project Creation and Build

A project contains all the files you will need to develop an executable output file (.out) which can be run on the MSP430 hardware. To create a new project:

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

2. Type in your Project name and make the selections shown below (your dialog may look slightly different than this one).

3. If you are using the MSP430G2553, make the appropriate choices for that part. Make sure to click **Empty Project** (with main.c) and then click **Finish**.

4. If you are writing an Assembly Code, select Empty Assembly-only Project.

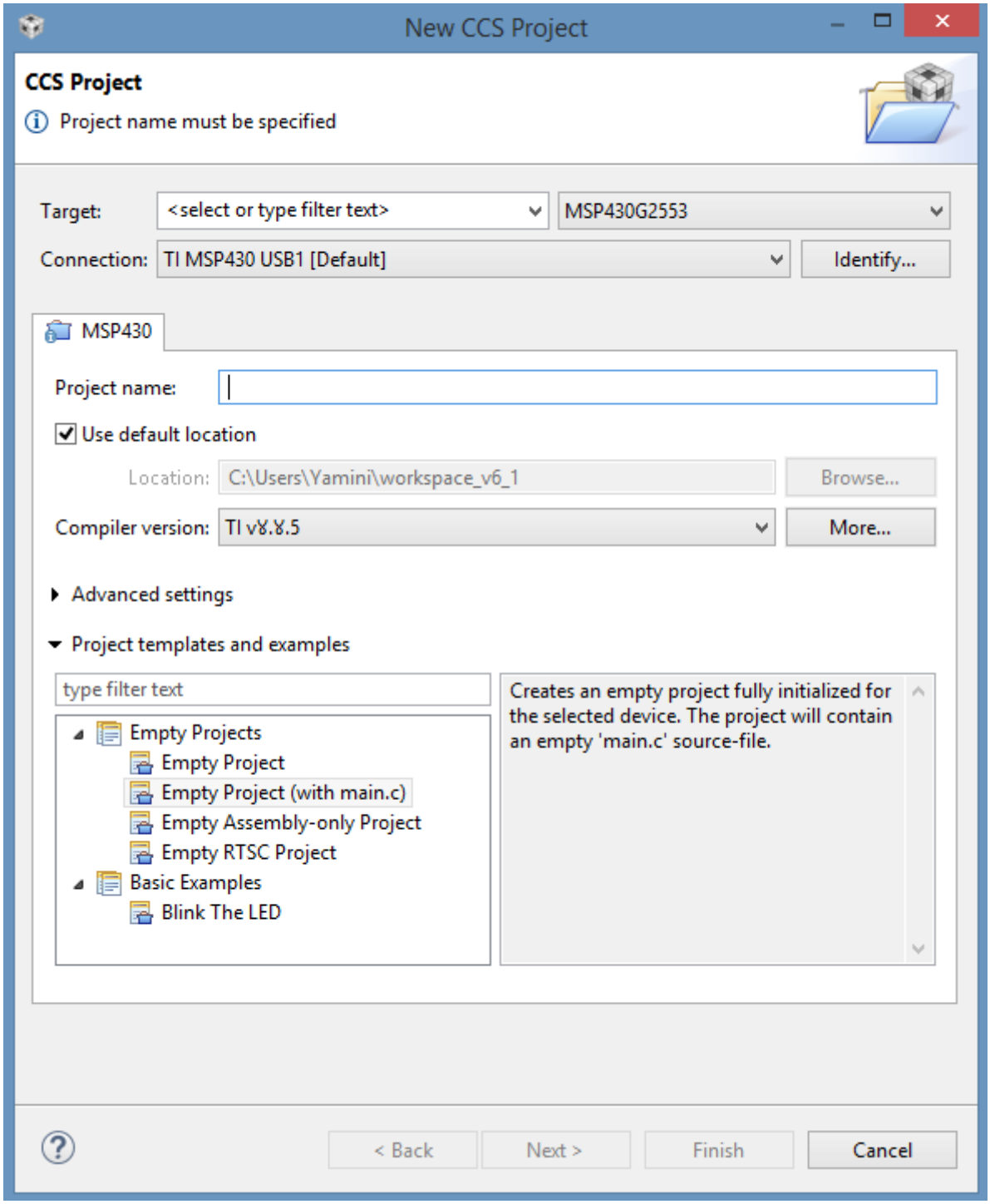


Figure 10 New CCS Project Creation

5. Code Composer will add the named project to your workspace and display it in the Project Explorer pane. Based on your template selection, it will also add a file called main.c/ main.asm and open it for editing.

6. Change the include header file statement from **#include** <msp430.h> to **#include** <msp430g2553.h>.

7. Type in your program code in the main.c / main.asm file and save it.

8. CCS can automatically save modified source files, build the program, open the debug perspective view, connect and download it to the target (flash device), and then run the program to the beginning of the main function. To do this, click on the "Debug" button.  .

9. When the Ultra-Low-Power Advisor (ULP Advisor) appears, click the Proceed button.

10. When the download completes, CCS is in the Debug perspective. Notice the Debug tab in the upper right-hand corner indicating that we are now in the "CCS Debug" view.

11. Click and drag the perspective tabs to the left until you can see all of both tabs. You can observe that the program has run through the C-environment initialization routine in the runtime support library and stopped at main() in main.c.

Debug Environment

After completion of target download, CCS enters the Debug perspective. Notice the Debug tab in the upper right-hand corner indicating that we are now in the "CCS Debug" view (See **Figure 7**). Click and drag the perspective tabs to the left until you can completely see both tabs.

The basic buttons that control the debug environment are located at the top of the Debug pane. If the pane is closed accidentally, the Debug controls will disappear. To bring back the debug controls, click **View** → **Debug** on the menu bar.

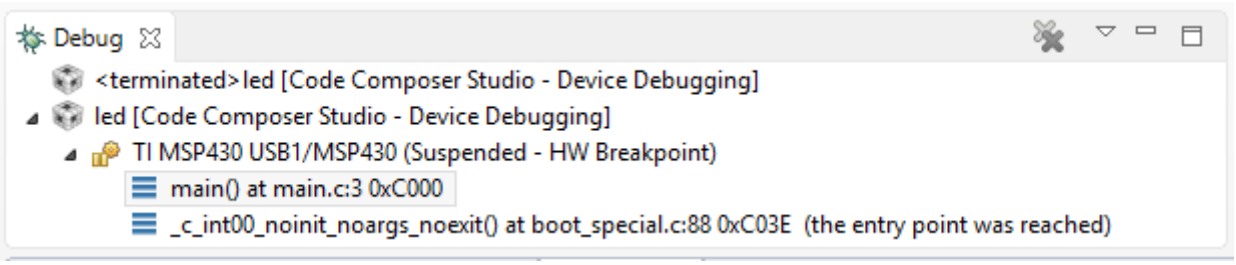


Figure 11 Debug Environment

You can explore each button by clicking on it to see its function. At this point, your code should be at the beginning of main (). Look for a small blue arrow left of the opening brace of main() in the middle window as shown in **Figure 11**. The blue arrow indicates where the Program Counter (PC) is pointing to.

Click the **Resume**  button to run the code.

Click **Suspend**  to stop code execution in the middle of the program.

To single-step into the code, click **Step Into**  to help in debugging the program and check if each line of code is producing the desired result.

Click **Reset CPU**  and you should be back at the beginning of main().

The **Terminate**  button will terminate the active debug session, close the debugger and return CCS to the "CCS Edit" perspective. It also sends a reset to the LaunchPad board.

Important Notes for CCS

1. Make sure your LaunchPad USB DEBUG port is connected to your PC.

2. Check if the project selected for the experiment is active.

3. Build the project by clicking the Debug button on the menu bar.

4. If the Launch pad has gone to Hibernate mode, it can be awakened by pressing SW2.

5. Any warning that the project was created with an earlier compiler version for experiments which are taken from the library can be safely ignored.

6. When you import the project, it will be automatically copied into your workspace, preserving the original files.

7. All the modifications done are in the local workspace and do not get reflected onto the TivaWare folder. So, you can safely modify, save and always retrieve the demo code from TivaWare.

8. If you delete the project accidentally or intentionally in CCS, only the project in the workspace is deleted, while the imported project stays intact in your workspace.

9. In the dialog box, it is also possible to delete files from the disk which will erase the files from the library but that is a general OS file delete operation and should not be attempted.

Using Tera Term

Tera Term is open-source, freely-downloadable, terminal emulator software used to test serial communication. The software can be easily downloaded from the internet. The procedure to use Tera Term as a serial terminal is given below:

1. Open Tera Term program and Select Serial Port. The corresponding port to which MSP430G2 Launchpad is connected in Tera Term is displayed in the **New Connection** Window as shown in **Figure 12**. The serial port option is enabled only if the MSP430G2 Launchpad Drivers are properly installed, else it stays at TCP/IP. The Serial Port number will differ based on the USB to Serial Emulation and the connectivity.

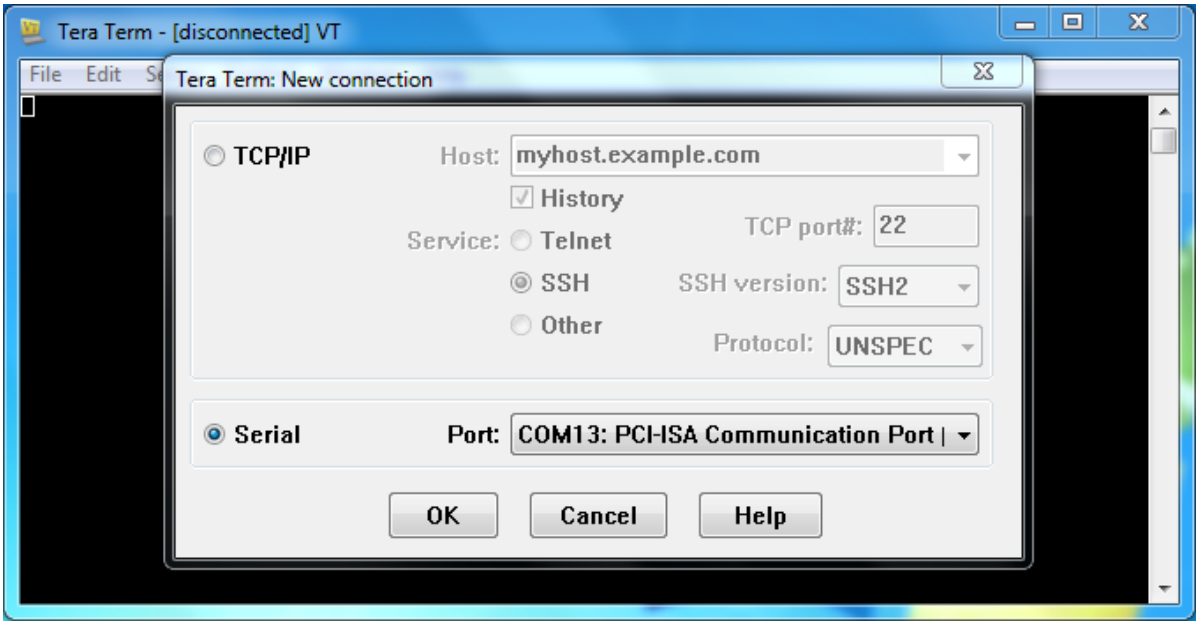


Figure 12 Tera Term New Connection Window

2. The port is opened with a default Baud Rate of 9600. Change the Baud Rate by selecting **Setup → Serial Port**. Change the required Serial Port Setup parameters like Baud Rate, Data Bits, Parity, Stop Bit as shown in **Figure 13**.

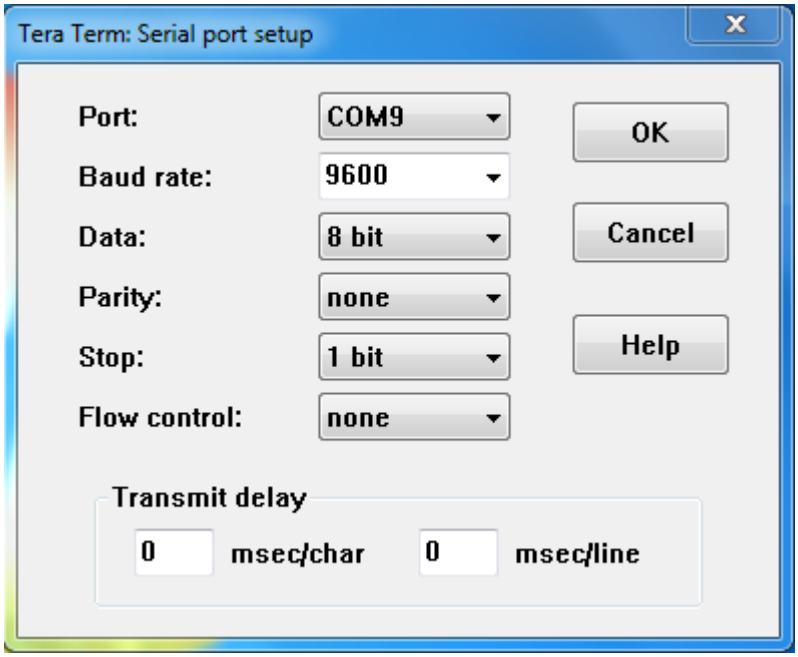


Figure 13 Tera Term Serial Port Setup Window

3. Select **Setup** → **Terminal** → **New Line** → **Receive** → **Auto** to configure the new line character as shown in **Figure 14**.

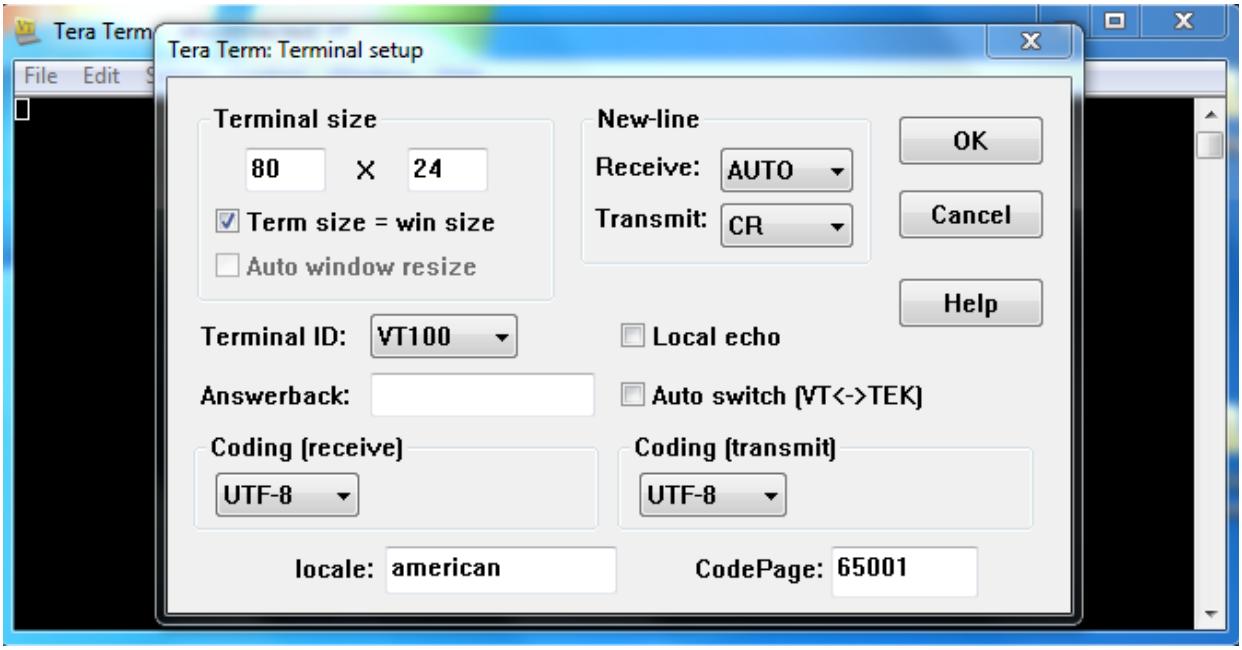


Figure 14 Tera Term Terminal Setup Window

Experiment 1

To Blink an LED with GPIO

Topics **Page**

1.1 Objective

The main objective of this experiment is to blink the on-board, red LED (connected to P1.0) using GPIO. This experiment will help you to learn and understand the procedure for programming the MSP-EXP430G2 LaunchPad digital I/O pins.

1.2 Introduction

1.2.1 Digital I/O

MSP430G2553 devices have up to eight digital I/O ports (P1 to P8). Each port has up to eight I/O pins that are configured with user software. Every I/O pin is individually configurable as input or output, and can be individually read or written to.

The features of the digital I/O are:

- Independently programmable individual I/Os
- Any combination of input or output
- Individually configurable P1 and P2 interrupts
- Independent input and output data registers
- Individually configurable pull-up or pull-down resistors
- Individually configurable pin-oscillator function in some MSP430 devices

The MSP430G2553 has two general-purpose digital I/O pins connected to green LED (P1.6) and red LED (P1.0) on the MSP-EXP430G2 LaunchPad for visual feedback. In this experiment, the code programmed into the MSP430G2553 processor toggles the output on Port P1.0 at fixed time intervals computed within the code. A HIGH on P1.0 turns the LED ON, while a LOW on P1.0 turns the LED OFF. Thus, the toggling output blinks the red LED connected to it. The functional block diagram shown in **Figure 1-1** illustrates the working principle of the experiment.

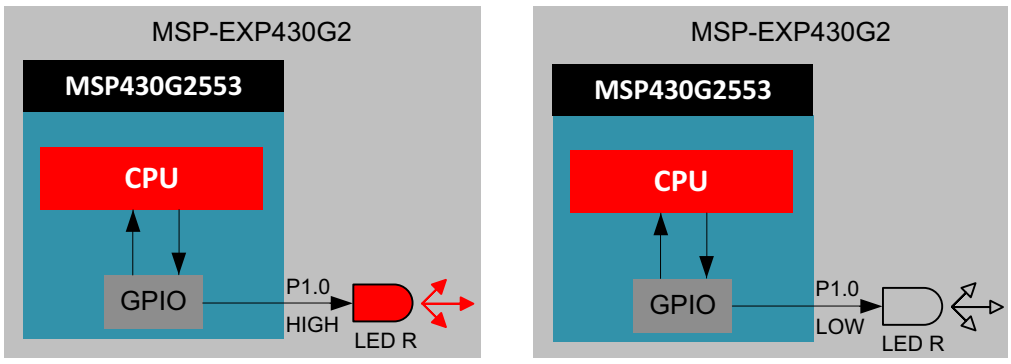


Figure 1-1 Functional Block Diagram

Figure 1-2 shows the Launchpad schematics for the connection of Digital I/O pins to the on-board LEDs in the LaunchPad.

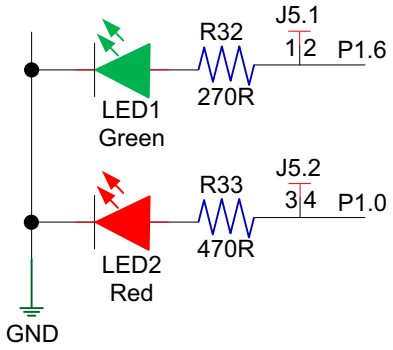


Figure 1-2 LaunchPad Schematics

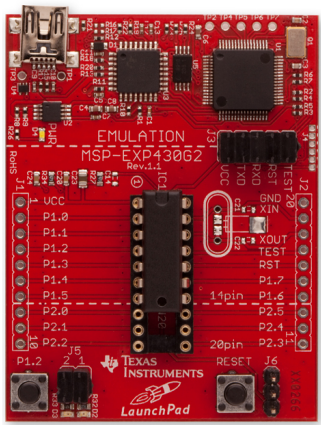
1.3 Component Requirements

1.3.1 Software Requirement

Code Composer Studio

1.3.2 Hardware Requirement

Table 1-1: Components Required for the Experiment

S.No	Components	Specifications	Images
1.	MSP430G2 Launch-Pad	**MSP-EXP430G2XL**	
2.	USB cable		

1.4 Software

The software for the experiment is written in C and developed using the CCS Integrated Development Environment (IDE). Refer to "**Project Creation and Build**" in the Getting Started section of this manual for project build and debug using CCS. The software is programmed into the target device MSP430G2553 on the MSP-EXP430G2 LaunchPad using the USB interface.

1.4.1 Flowchart

The flowchart for the code is shown in **Figure 1-3**. The program code first disables the watchdog timer to prevent a processor restart on expiry of the WDT count. The port pin P1.0 connected to

the red LED is configured as output. A HIGH on the pin turns the red LED on, while a LOW turns the LED off. The P1.0 output is toggled using bit XOR function at regular intervals determined by the software delay set in the program using variable 'i'. Thus, the red LED on the MSP-EXP430G2 LaunchPad blinks at regular intervals.

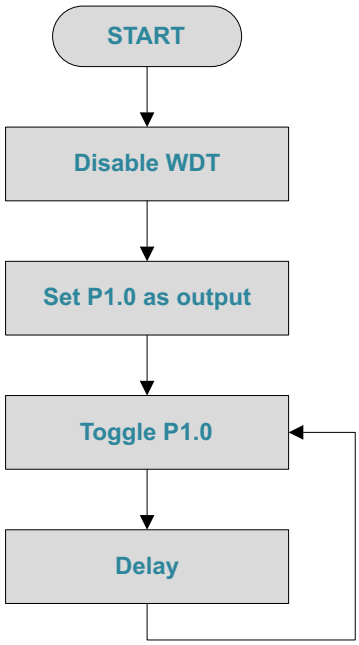


Figure 1-3 Flowchart to Blink the LED Using GPIO

1.4.2 C Program Code to Blink the LED with GPIO

```
#include<msp430.h>

int main(void) {
WDTCTL = WDTPW | WDTHOLD;       // Stop watchdog timer
P1DIR |= 0x01;                  // Set P1.0 to output direction

while(1) {
volatile unsigned long i;       // Volatile to prevent
                                //optimization
P1OUT ^= 0x01;                  // Toggle P1.0 using XOR
i = 50000;                      // SW Delay
do i--;
while(i != 0);
}
}
```

1.4.3 Registers Used

The registers used in this program are:

- **WDTCTL: 16-bit Watchdog Timer Control Register** - The Watchdog Timer (WDT) is typically used to trigger a system reset after a certain amount of time. In most examples, the timer is stopped during the first line of code to prevent a continuous restart loop and for ease of debugging. In our program, the instruction WDTCTL = WDTPW | WDTHOLD sets the password (WDTPW) and WDTHOLD bit to stop the timer.

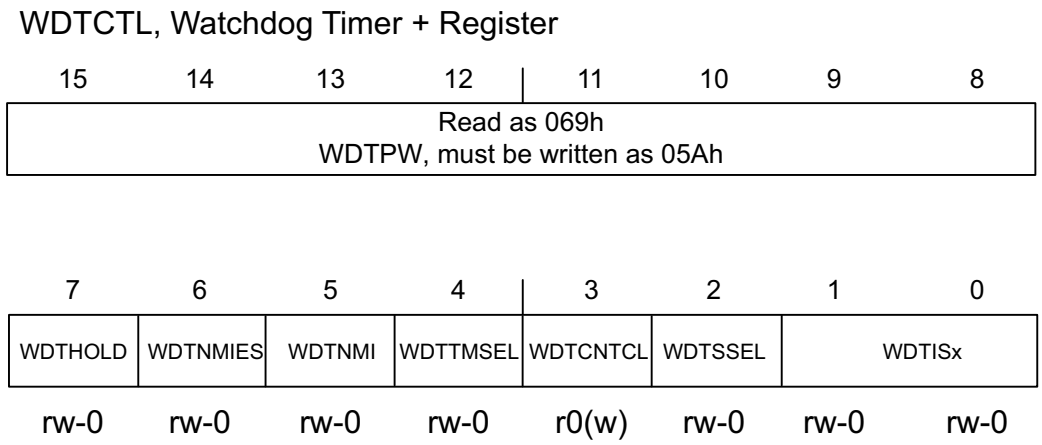


Figure 1-4 Watchdog Timer Control Register

- **P1DIR: 8-bit Direction Register** - Each bit in the direction register selects the direction of the corresponding pin as an input or an output. Here, bit 0 is set to "1" directing the Port Pin 1.0 as an output.
- **P1OUT: 8-bit Output Register** - Each bit in the output register is the value to be output on the corresponding I/O pin when the pin is configured as an output. A bit 1 sets the pin HIGH, while a 0 resets the pin to LOW.

1.5 Procedure

1. Connect the MSP-EXP430G2 LaunchPad to the PC using the USB cable supplied.

2. Build, program and debug the code into the LaunchPad using CCS to view the status of the red LED.

3. In the CCS debug perspective, select **View** → **Registers**.

1.6 Observation

While the code is running, pause it. In the **Registers** tab of CCS, expand and view the registers P1OUT, P1DIR and WDTCTL. Compare the values with your code. The value of the registers when the LED is ON appears in the CCS watch window as shown in **Figure 1-5**.

Figurep 1-5 Register values in CCS Watch Window for LED On

The value of the registers when the LED is OFF appears in the CCS watch window as shown in **Figure 1-6**.

Figure 1-6 Register values in CCS Watch Window for LED Off

1.7 Summary

In this experiment, we have learnt to configure and program the digital I/O pins of MSP430G2553 and have successfully programmed the port P1.0 to blink the on-board green LED of MSP430G2 LaunchPad.

1.8 Exercise

1. Alter the delay with which the LED blinks.
2. Alter the code to make the green LED blink.
3. Alter the code to make the green and red LEDs blink:
   i. Together
   ii. Alternately

Experiment 2

LED Control Using a Switch

Topics **Page**

2.1 Objective

The main objective of this experiment is to control the on-board, green LED of MSP430G2 Launchpad by an input from the on-board switch of MSP430G2 Launchpad. This experiment will help you to learn and understand the procedure for programming the MSP430G2553 GPIO pins as input and output.

2.2 Introduction

2.2.1 Digital I/O

MSP430 devices have up to eight digital I/O ports (P1 to P8). Each port has up to eight I/O pins that are configured with user software. Every I/O pin is individually configurable as input or output, and can be individually read or written to.

The features of the digital I/O are:

- Independently programmable individual I/Os
- Any combination of input or output
- Individually configurable P1 and P2 interrupts
- Independent input and output data registers
- Individually configurable pull-up or pull-down resistors
- Individually configurable pin-oscillator function in some MSP430 devices

The MSP-EXP430G2 LaunchPad has two general-purpose digital I/O pins connected to green and red LEDs for visual feedback. It also has two push buttons for user feedback and device reset. In this experiment, the input from the left push button switch (S2) connected to Port P1.3 is read by the processor for LED control. If the switch is pressed, the green LED is turned OFF by output at port P1.6 reset to '0'. Else, the output at port P1.6 is set HIGH and the green LED is turned ON. The LED is therefore controlled by the switch S2.

The functional block diagram shown in **Figure 2-1** illustrates the working principle of the experiment.

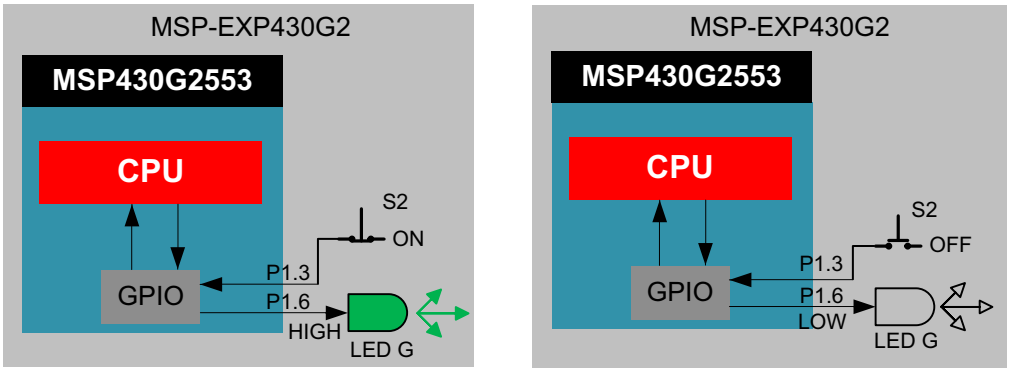


Figure 2-1 Functional Block Diagram

Figure 2-2 shows the Launchpad schematics for the connection of Digital I/O pins to the on-board LEDs in the LaunchPad.

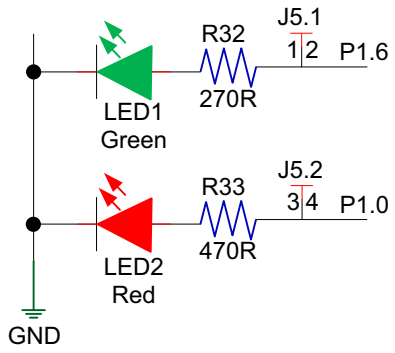


Figure 2-2 Launchpad Schematics

The input pins for the MSP430G2553 need to be either pulled-up or pulled-down with a resistor to avoid floating inputs. The selection of a pull-up or pull down resistor for the inputs is performed in software. A pull-up resistor enables a default hardware connection for the pin to V_{CC} as shown in **Figure 2-3**. Hence, when the switch is open, the V_{CC} (logic HIGH) is available on the input pin of the MCU.

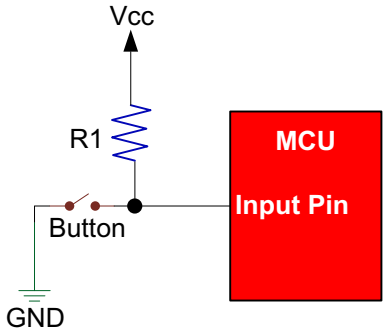


Figure 2-3 Pull-up Resistor for Input Pin

Similarly, a pull-down resistor defaults a hardware connection to ground, 0V as shown in **Figure 2-4**. When the switch is open, 0V GND (logic LOW) is available on the input pin of the MCU.

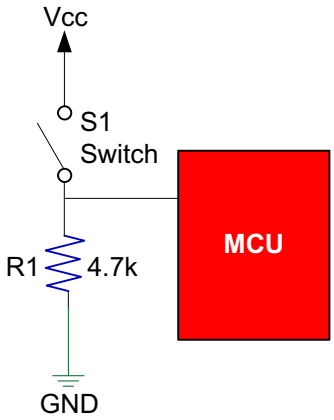


Figure 2-4 Pull-down Resistor for Input Pin

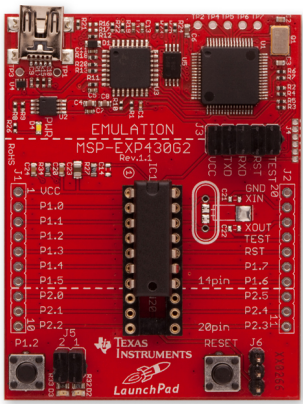
2.3 Component Requirements

2.3.1 Software Requirement

Code Composer Studio

2.3.2 Hardware Requirement

Table 2-1: Components Required for the Experiment

S.No	Components	Specifications	Images
1.	MSP430G2 LaunchPad	**MSP-EXP430G2XL**	
2.	USB cable		

2.4 Software

The software for the experiment is written in C and developed using the CCS Integrated Development Environment (IDE). Refer to "**Project Creation and Build**" in the Getting Started section of this manual for project build and debug using CCS. The software is programmed into the target device MSP430G2553 on the MSP-EXP430G2 LaunchPad using the USB interface.

2.4.1 Flowchart

The flowchart for the code is shown in **Figure 2-5**. The program code first disables the watchdog timer to prevent a restart on expiry of the WDT count. The port pin P1.6 connected to the green LED is configured as output and port pin P1.3 connected to the left push button switch (SW2) is configured as input with pull up resistor.

The push button input at P1.3 is read continuously using a while(1) infinite loop. When the push button switch is open, input at P1.3 will be HIGH because of the pull up resistor, and when the button is pressed, the switch is closed and the input at pin P1.3 is LOW.

If the push button switch is open, the code turns the green LED ON with a HIGH on output pin P1.6. Else, if the push button switch is closed, the code turns the green LED OFF with a LOW on output pin P1.6.

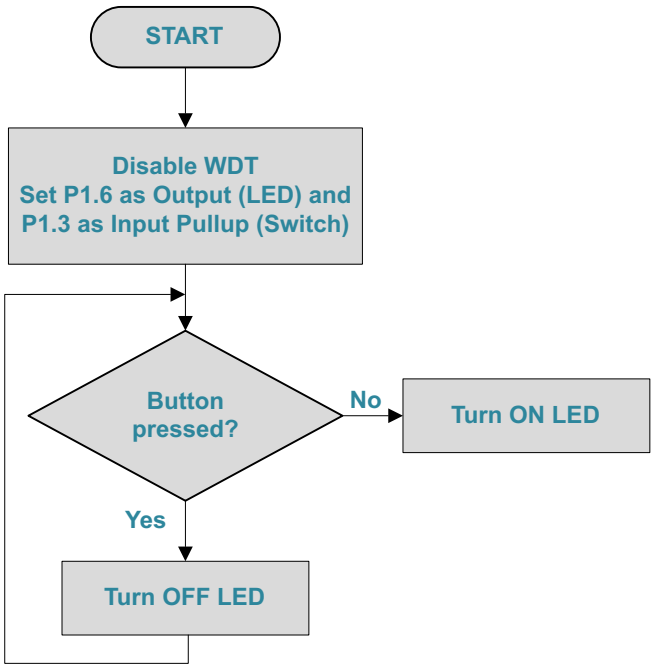


Figure 2-5 Flowchart for Controlling LED with a Switch

2.4.2 C Program Code for Controlling the LED with a Switch

```
#include<msp430.h>

int main(void) {
WDTCTL = WDTPW | WDTHOLD;           // Stop watchdog timer
P1DIR |= 0x40;                      // Set P1.6 to output direction
P1REN |= 0x08;
P1OUT |= 0X08;
while(1) {
if ((P1IN & BIT3)) {                // If button is open(P1.3 HIGH)
   P1OUT = P1OUT | BIT6;            // ... turn on LED
}                                   // or P1OUT |= BIT0;
else {
   P1OUT = P1OUT & ~BIT6;           // ... else turn it off.
                                    // or P1OUT &= ~BIT0
      }
    }
}
```

2.4.3 Registers Used

The registers used in this program are:

- **WDTCTL: 16-bit Watchdog Timer Control Register** - The Watchdog Timer (WDT) is typically used to trigger a system reset after a certain amount of time. In most examples, the timer is stopped during the first line of code to prevent a continuous restart loop.
- **P1DIR: 8-bit Direction Register** - Each bit in the direction register selects the direction of the corresponding pin as an input or an output. Here, bit 6 is set to 1 directing the Port Pin 1.6 as an output.
- **P1REN: Resistor Enable Register** - It is used to specify configuration of pull-up or pull-down to the input pin. Here, bit 3 is set to enable a pull-up resistor on switch input pin P1.3.
- **P1OUT: 8-bit Output Register** - Each bit in the P1OUT register is the value to be output on the corresponding I/O pin when the pin is configured as an output, and the pull-up/down resistor is disabled. A bit 1 sets the pin HIGH, while a 0 resets the pin to LOW. In our program, since the pull-up resistor is enabled on pin P1.3, the corresponding input pin on the P1OUT register is set.

2.5 Procedure

1. Connect the MSP-EXP430G2 LaunchPad to the PC using the USB cable supplied.

2. Build, program and debug the code into the LaunchPad using CCS.

2.6 Observation

Initially, the green LED is ON as shown in **Figure 2-6**. When the left push button switch is pressed, the green LED turns OFF.

Figure 2-6 Green LED Turned On in MSP-EXP430G2 LaunchPad

2.7 Summary

In this experiment, we have learnt to configure and program the GPIO pins of MSP430G2553 as input and output.We have successfully programmed to read input from the on-board switch of MSP430G2 Launchpad connected to pin P1.3 to blink the on-board green led of MSP430G2 Launchpad connected to pin P1.6.

2.8 Exercise

1. Alter the code to turn the LED ON when the button is pressed and OFF when it is released.

2. Alter the code to make the green LED stay ON for around 1 second every time the button is pressed.

3. Alter the code to turn the red LED ON when the button is pressed and the green LED ON when the button is released.

Experiment 3

Low Power Modes and Current Measurement

Topics **Page**

3.1 Objective

The main objective of this experiment is to configure the MSP-EXP430G2 LaunchPad for Low Power Mode (LPM3) and measure current consumption both in active and low power modes. This experiment will help in learning the various low power modes of the MSP430G2553.

3.2 Introduction

The MSP430 family is designed for ultra low-power applications. The MSP430G2553 supports one active mode and five software selectable low-power modes of operation. The operating modes take into account three different needs:

- Ultra low-power
- Speed and data throughput
- Minimization of individual peripheral current consumption

The low power modes for the MSP430G2553 are configured in the status register. An interrupt event can wake up the device from any of the low-power modes. It services the request and restores back to the same low-power mode if the status register is not altered within the interrupt service routine.

Power consumption of target device is directly related to the clock speed. Hence, the power consumption is reduced by disabling the clock for a particular peripheral as per the application requirement. The state of the CPU and clocks for the different low power modes of the MSP430G2553 is given in **Table 3-1**.

Table 3-1: CPU and Clock States in Various Operating Modes

Mode	CPU and Clocks
Active	CPU active. All enabled clocks active
LPM0	CPU. MCLK disabled. SMCLK. ACLK active
LPM1	CPU. MCLK disabled. DCO disabled if not used for SMCLK. ACLK active
LPM2	CPU. MCLK. SMCLK. DCO disabled. ACLK active
LPM3	CPU. MCLK. SMCLK. DCO disabled. ACLK active
LPM4	CPU and all clocks disabled

The power consumption of MSP430G2553 is the maximum in active mode, where CPU clock and related peripheral clocks are enabled. Stand by current consumption of the MSP430G2553 device depends upon the particular low power mode as shown in the **Figure 3-1**.

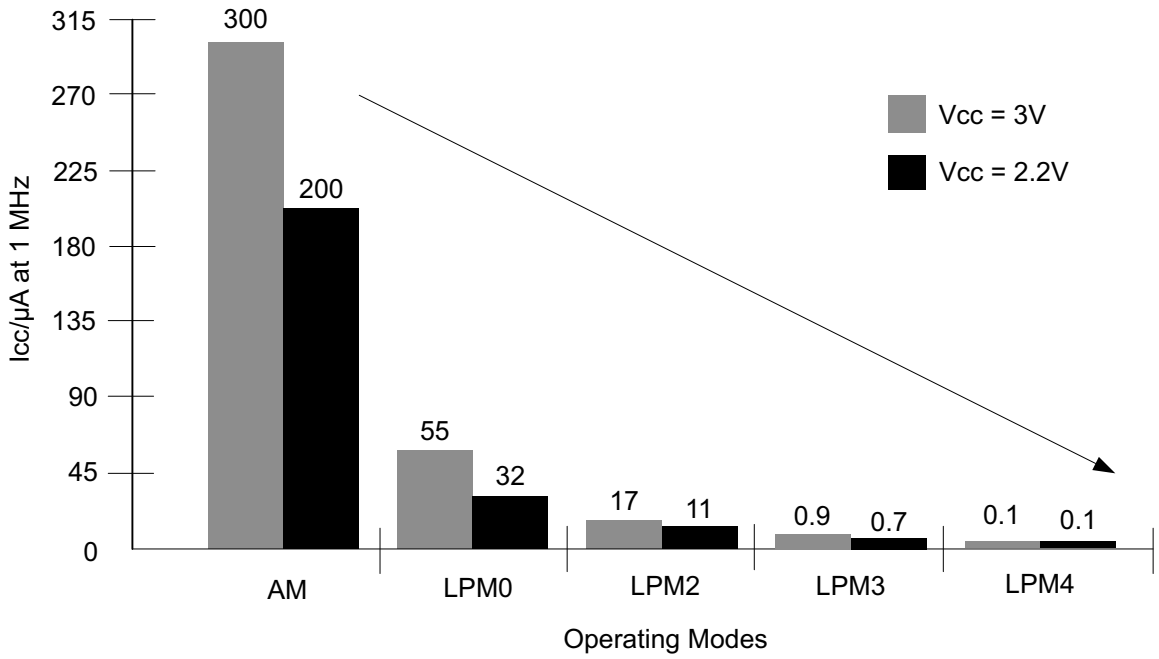


Figure 3-1 Typical Current Consumption for Various Operating Modes

In the current experiment, we are measuring the active current and standby current of the MSP430G2553 for a given application.

The majority of the power used by the application is spent in the **while (1)** loop, waiting for an interrupt. We can place the device in a low-power mode during that time and save a considerable amount of power.

In this experiment, we will turn on interrupts and put the device in LPM3 mode. It is to be noted that this mode will place restrictions on the resources available during the low-power mode. The CPU, MCLK, SMCLK and DCO are off in LPM3 mode. Only the ACLK (sourced by the VLO in our code) is still running. In MSP430G2553, all device pins must be configured to draw the lowest current to demonstrate low power modes. MSP430G2553 port 1 defaults to GPIO, among them only P1.3 is configured as an input to support push button switch SW2, and the rest are configured as outputs. P2.6 and P2.7 default to crystal inputs, so in the current example program, they are configured as GPIO.

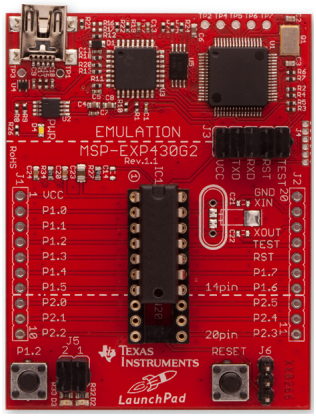
3.3 Component Requirements

3.3.1 Software Requirement

Code Composer Studio

3.3.2 Hardware Requirement

Table 3-2: Components Required for the Experiment

S.No	Components	Specifications	Images
1.	MSP430G2 LaunchPad	MSP-EXP430G2XL	
2.	USB cable		
3.	Multimeter		

3.4 Software

The software for the experiment is written in C and developed using the CCS Integrated Development Environment (IDE). Refer to "**Project Creation and Build**" in the Getting Started section of this manual for project build and debug using CCS. The software is programmed into the target device MSP430G2553 on the MSP-EXP430G2 LaunchPad using the USB interface.

3.4.1 C Program Code for Active and Low Power Operating Modes

Main_active.c:

```
#include <msp430g2553.h>

#ifndef TIMER0_A1_VECTOR
#define TIMER0_A1_VECTOR    TIMERA1_VECTOR
#define TIMER0_A0_VECTOR    TIMERA0_VECTOR
#endif

volatile long tempRaw;
volatile unsigned int i;

void FaultRoutine(void);
void ConfigWDT(void);
void ConfigClocks(void);
void ConfigPins(void);
```

```
void ConfigADC10(void);
void ConfigTimerA2(void);

void main(void)
{
  ConfigWDT();
  ConfigClocks();
  ConfigPins();
  ConfigADC10();
  ConfigTimerA2();

 _BIS_SR(GIE);

  while(1)
  {
for (i = 100; i > 0; i--);
 for (i = 5000; i > 0; i--);  }
}

void ConfigWDT(void)
 {
 WDTCTL = WDTPW + WDTHOLD;                    // Stop watchdog timer
 }

void ConfigClocks(void)
 {
 if (CALBC1_1MHZ ==0xFF || CALDCO_1MHZ == 0xFF)
   FaultRoutine();                          // If calibration data is erased
                                            // run FaultRoutine()
  BCSCTL1 = CALBC1_1MHZ;                    // Set range
  DCOCTL = CALDCO_1MHZ;                     // Set DCO step + modulation
  BCSCTL3 |= LFXT1S_2;                      // LFXT1 = VLO
  IFG1 &= ~OFIFG;                           // Clear OSCFault flag
  BCSCTL2 |= SELM_0 + DIVM_3 + DIVS_3;      // MCLK = DCO/8, SMCLK = DCO/8
 }
```

```
void FaultRoutine(void)
 {
   P1OUT = BIT0;                             // P1.0 on (red LED)
   while(1);                                 // TRAP
 }

void ConfigPins(void)
 {
  P1DIR = ~BIT3;                             // P1.6 and P1.0 outputs
  P1OUT = 0;
  P2SEL = ~(BIT6 + BIT7);
  P2DIR |= BIT6 + BIT7;
  P2OUT = 0;                                 // LEDs off
 }

void ConfigADC10(void)
 {
  ADC10CTL1 = INCH_10 + ADC10DIV_0;          // Temp Sensor ADC10CLK
 }

void ConfigTimerA2(void)
  {
   CCTL0 = CCIE;
   CCR0 = 36000;
   TACTL = TASSEL_1 + MC_2;
  }

#pragma vector=TIMER0_A0_VECTOR
__interrupt void Timer_A (void)
{
  ADC10CTL0 = SREF_1 + ADC10SHT_3 + REFON + ADC10ON;
  _delay_cycles(4);                         // Wait for ADC Ref to settle
  ADC10CTL0 |= ENC + ADC10SC;               // Sampling and conversion start
  P1OUT |= BIT6;                            // P1.6 on (green LED)
  _delay_cycles(100);
  ADC10CTL0 &= ~ENC;                        // Disable ADC conversion
```

```
  ADC10CTL0 &= ~(REFON + ADC10ON);      // Ref and ADC10 off
  tempRaw = ADC10MEM;                   // Read conversion value
  P1OUT &= ~BIT6;                       // green LED off
  CCR0 +=36000;                         // add 3 second to the timer

}
```

Main_standby.c:

```
#include <msp430g2553.h>

#ifndef TIMER0_A1_VECTOR
#define TIMER0_A1_VECTOR    TIMERA1_VECTOR
#define TIMER0_A0_VECTOR    TIMERA0_VECTOR
#endif

volatile long tempRaw;
//volatile unsigned int i;

void FaultRoutine(void);
void ConfigWDT(void);
void ConfigClocks(void);
void ConfigPins(void);
void ConfigADC10(void);
void ConfigTimerA2(void);

void main(void)
{
  ConfigWDT();
  ConfigClocks();
  ConfigPins();
  ConfigADC10();
  ConfigTimerA2();

 // _BIS_SR(GIE);

  while(1)
```

```
  {
  _bis_SR_register(LPM3_bits + GIE);    // Enter LPM3 with interrupts
  }
}

void ConfigWDT(void)
 {
 WDTCTL = WDTPW + WDTHOLD;              // Stop watchdog timer
 }

void ConfigClocks(void)
 {
 if (CALBC1_1MHZ ==0xFF || CALDCO_1MHZ == 0xFF)
   FaultRoutine();                      // If calibration data is erased
                                        // run FaultRoutine()
  BCSCTL1 = CALBC1_1MHZ;                // Set range
  DCOCTL = CALDCO_1MHZ;                 // Set DCO step + modulation
  BCSCTL3 |= LFXT1S_2;                  // LFXT1 = VLO
  IFG1 &= ~OFIFG;                       // Clear OSCFault flag
  BCSCTL2 |= SELM_0 + DIVM_3 + DIVS_3;  // MCLK = DCO/8, SMCLK = DCO/8
 }

void FaultRoutine(void)
 {
   P1OUT = BIT0;                        // P1.0 on (red LED)
   while(1);                            // TRAP
 }

void ConfigPins(void)
 {
  P1DIR = ~BIT3;                        // P1.6 and P1.0 outputs
  P1OUT = 0;
  P2SEL = ~(BIT6 + BIT7);
  P2DIR |= BIT6 + BIT7;
  P2OUT = 0;                            // LEDs off
 }
```

```
void ConfigADC10(void)
 {
  ADC10CTL1 = INCH_10 + ADC10DIV_0;      // Temp Sensor ADC10CLK
 }

void ConfigTimerA2(void)
  {
   CCTL0 = CCIE;
   CCR0 = 36000;
   TACTL = TASSEL_1 + MC_2;
  }

#pragma vector=TIMER0_A0_VECTOR
__interrupt void Timer_A (void)
{
  ADC10CTL0 = SREF_1 + ADC10SHT_3 + REFON + ADC10ON;
  _delay_cycles(4);                       // Wait for ADC Ref to settle
  ADC10CTL0 |= ENC + ADC10SC;             // Sampling and conversion start
  P1OUT |= BIT6;                          // P1.6 on (green LED)
  _delay_cycles(100);
  ADC10CTL0 &= ~ENC;                      // Disable ADC conversion
  ADC10CTL0 &= ~(REFON + ADC10ON);        // Ref and ADC10 off
  tempRaw = ADC10MEM;                     // Read conversion value
  P1OUT &= ~BIT6;                         // green LED off
  CCR0 +=36000;                           // add 1 second to the timer
  _bic_SR_register_on_exit(LPM3_bits);    // Clr LPM3 bits from SR on exit
}
```

3.5 Procedure

The experiment involves two phases:

- Measurement of current in Active Mode
- Measurement of current in Standby (LPM3) Mode

Initially, connect the MSP-EXP430G2 LaunchPad to the PC using the USB cable supplied.

3.5.1 Measurement of Current in Active Mode

1. Copy the code main_active.c in the CCS new project.

2. Build, load, and run the code. The green LED will blink once every three or four seconds.

3. When done, halt the code and click the Terminate button to return to the "CCS Edit".

4. Remove all five jumpers on header J3.

5. The red lead of the multimeter should connect to the top (emulation side) V_{cc} pin on header J3 and the black lead of the multimeter should connect to the bottom (target side) V_{cc} pin on header J3.

6. Press the Reset button on the LaunchPad board.

7. Measure the current drawn by the MSP430G2553.

3.5.2 Measurement of Current in Standby (LPM3) Mode

1. Copy the code main_standby.c in the CCS new project.

2. Build, load, and run the code. The green LED will blink once every three or four seconds.

3. When done, halt the code and click the Terminate button to return to the "CCS Edit".

4. Remove all five jumpers on header J3.

5. The red lead of the multimeter should connect to the top (emulation side) V_{cc} pin on header J3 and the black lead of the multimeter should connect to the bottom (target side) V_{cc} pin on header J3.

6. Press the Reset button on the LaunchPad board.

7. Measure the current drawn by the MSP430G2553.

3.6 Observation

The current consumption in both the active and standby modes is measured while running the same application code. In Main_standby.c, the processor enters into low power mode LPM3 and waits for an ADC interrupt. The reading for both the cases for MSP430G2553 are tabulated in **Table 3-3**.

Table 3-3: Current Consumption for MSP430G2553

Platform	Active mode	Standby mode
MSP430G2553	123µA	1.2µA

3.7 Summary

In this experiment, we have learnt to configure the MSP430G2553 in low power mode (LPM3). We have found that the current consumption of the processor is reduced in low power modes.

3.8 Exercise

1. How many Low power modes are supported by the MSP430G2553 platform?
2. Measure the Active and Standby Current consumption in LPM3 mode for the same application using MSP430F5529 LaunchPad.

Experiment 4

Interrupt Programming Through GPIO

Topics **Page**

4.1 Objective

The main objective of this experiment is to configure GPIO and interrupts for the MSP430G2553. This experiment will help to learn and understand the GPIO and interrupt peripherals and their operation.

4.2 Introduction

The MSP430G2553 has one active mode and six software selectable low-power modes of operation. An interrupt event can wake up the device from any of the low-power modes, service the request, and restore back to the low power mode on return from the interrupt program. And these Interrupts may be generated from the GPIO of MSP430G2553.

Some GPIOs in the MSP430G2553 have the capability to generate an interrupt and inform the CPU when a transition has occurred. The MSP430G2553 allows flexibility in configuring which GPIO will generate the interrupt, and on what edge (rising or falling). MSP430G2553 devices typically have interrupt capability on Ports 1 and 2.

The registers controlling these options are as follows:

- PxIE - Each bit enables (1) or disables (0) the interrupt for the particular I/O pin.
- PxIES - Selects whether a pin will generate an interrupt on the rising-edge (0) or the falling-edge (1) of input.
- PxIFG - Interrupt flag register indicating whether an interrupt has occurred on a particular pin (a transition has occurred on the pin).

4.2.1 Digital I/O

MSP430 devices have up to eight digital I/O ports (P1 to P8). Each port has eight I/O pins that can be configured with user software. Every I/O pin is individually configurable as input or output, and can be individually read or written to.

The features of the digital I/O are:

- Independently programmable individual I/Os
- Any combination of input or output
- Individually configurable P1 and P2 interrupts
- Independent input and output data registers
- Individually configurable pull-up or pull-down resistors
- Individually configurable pin-oscillator function in some MSP430 devices

4.2.2 Interrupts

There are three types of interrupts:

- System Reset
- Non-Maskable Interrupt (NMI): NMIs are not masked by the General Interrupt Enable bit (GIE) in the status register but are enabled by individual interrupt enable bits.
- Maskable Interrupt: Maskable interrupts are caused by peripherals with interrupt capability. Each maskable interrupt can be disabled individually by an interrupt enable bit or all maskable interrupts can be disabled by the GIE in the status register.

When an interrupt is requested from a peripheral and the peripheral interrupt enable bit and GIE bit are set, the interrupt service routine is requested. The interrupt service routine involves the following execution steps:

1. Any currently executing instruction is completed.

2. The Program Counter (PC), which points to the next instruction, is pushed onto the stack.

3. The Status Register (SR) is pushed onto the stack.

4. If multiple interrupts occurred during the last instruction and are pending for service, the interrupt with the highest priority is selected.

5. The interrupt request flag resets automatically on single-source flags. Multiple source flags remain set for servicing by the software.

6. The SR is cleared. This terminates any low-power mode. Because the GIE bit is cleared, further interrupts are disabled.

7. The content of the interrupt vector is loaded onto the PC. The program continues with the interrupt service routine at that address.

The interrupt handling routine terminates with the instruction RETI (return from an interrupt service routine)

On return from the interrupt:

1. The SR with all previous settings pops from the stack. All previous settings of GIE, CPUOFF, etc., are now in effect, regardless of the settings used during the interrupt service routine.

2. The PC pops from the stack and begins execution at the point where it was interrupted.

The current experiment makes complete use of the GPIO interrupts. In this experiment P1.3 is configured as an input pin with a falling edge interrupt. The general interrupts are enabled so that the pin will make the CPU to call the Interrupt Service routine. This routine sets a flag that causes toggling of the green LED which is connected to P1.6 of the MSP_EXP430G2XL Launchpad.The functional block diagram shown in **Figure 4-1** illustrates the working principle of the experiment.

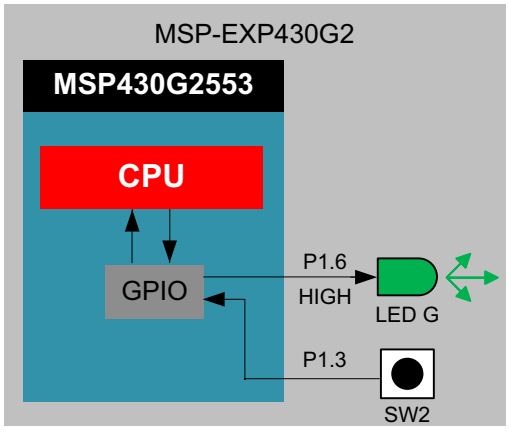


Figure 4-1 Functional Block Diagram

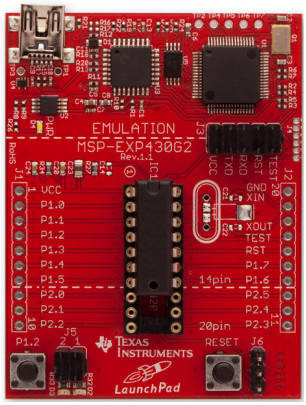
4.3 Component Requirements

4.3.1 Software Requirement

Code Composer Studio

4.3.2 Hardware Requirement

Table 4-1: Components Required for the Experiment

S.No	Components	Specifications	Image
1.	MSP430G2 LaunchPad	**MSP-EXP430G2XL**	
2.	USB cable		

4.4 Software

The software for the experiment is written in C and developed using the CCS Integrated Development Environment (IDE). Refer to "**Project Creation and Build**" in the Getting Started section of this manual for project build and debug using CCS. The software is programmed into the target device MSP430G2553 on the MSP-EXP430G2 LaunchPad using the USB interface.

4.4.1 Flowchart

The flowchart for the code is as shown in **Figure 4-2**. The program code first disables the watchdog timer to prevent a restart on expiry of the WDT count. The port pin P1.6 connected to the green LED on the MSP-EXP430G2 LaunchPad is configured as output. The port pin P1.3 connected to switch S2 on the LaunchPad is configured as GPIO input with falling edge interrupt. The global interrupt is enabled with low power mode 4 enabled during interrupt. On each interrupt, the green LED connected to P1.6 is toggled.

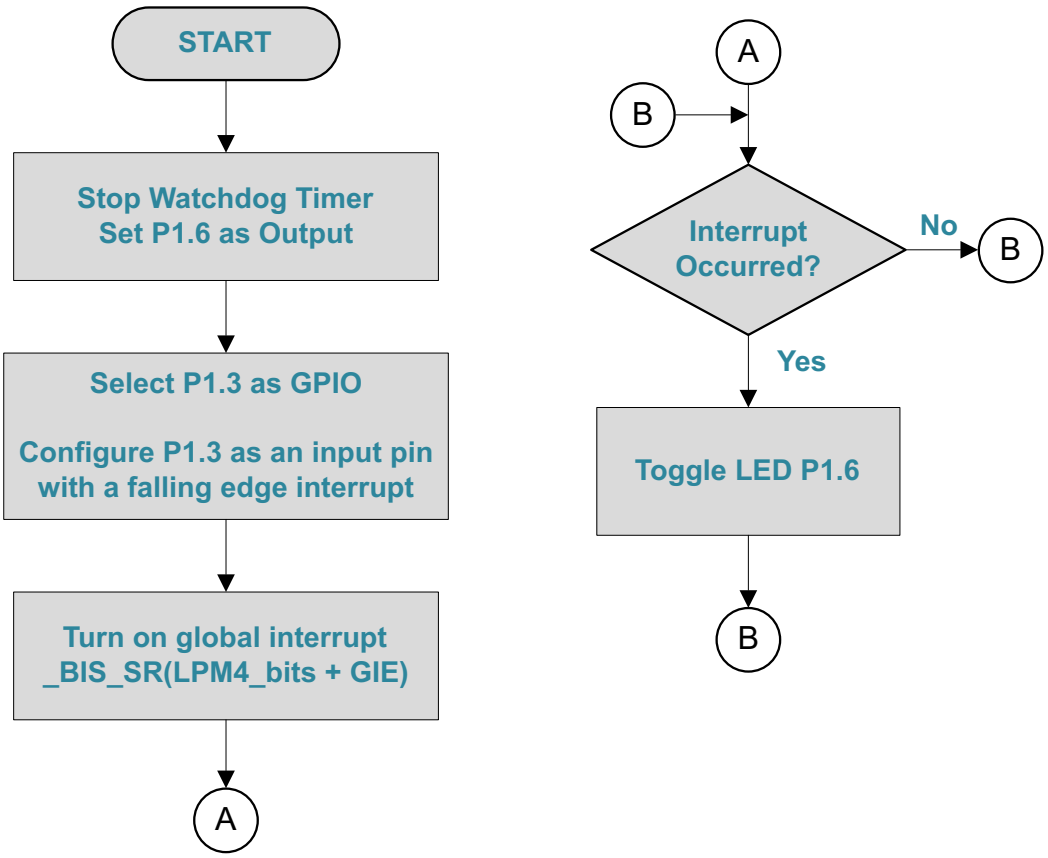


Figure 4-2 Flowchart for Interrupt Programming with GPIO

4.4.2 C Program Code for Interrupt Programming with GPIO

```
#include <msp430.h>
int main(void)
{
WDTCTL = WDTPW + WDTHOLD;       // Stop WDT
P1DIR |= BIT6;                  // Set P1.6 to output direction
P1REN |= BIT3;                  // Enable P1.3 internal resistance
P1OUT |= BIT3;                  // Set P1.3 as pull up resistance
P1IES |= BIT3;                  // P1.3 High/Low Edge
P1IFG &= ~BIT3;                 // P1.3 IFG Cleared
P1IE |= BIT3;                   // P1.3 Interrupt Enabled

_bis_SR_register(LPM4_bits + GIE);       // Enter LPM4 w/ interrupt
_no_operation();                // For debugger
```

```
}
#pragma vector=PORT1_VECTOR
__interrupt void Port_1 (void)
{
P1OUT ^= BIT6;                    // Toggle P1.6
P1IFG &= ~BIT3;                  // P1.3 IFG Cleared
}
```

4.4.3 Registers Used

The registers used in this program are:

- **WDTCTL: 16-bit Watchdog Timer Control Register** - The Watchdog Timer (WDT) is typically used to trigger a system reset after a certain amount of time. In most examples, the timer is stopped during the first line of code to prevent a continuous restart loop.
- **P1DIR: 8-bit Direction Register** - Each bit in the direction register selects the direction of the corresponding pin as an input or an output. Here, bit 6 is set to 1 directing the Port Pin 1.6 as an output.
- **P1OUT: 8-bit Output Register** - Each bit in the P1OUT register is the value to be output on the corresponding I/O pin when the pin is configured as an output, and the pull-up/down resistor is disabled. A bit 1 sets the pin HIGH, while a 0 resets the pin to LOW.
- **P1REN: Resistor Enable Register** - It is used to specify configuration of pull-up or pull-down to the input pin. Here, bit 3 is set to enable a pull-up resistor on switch input pin P1.3.
- **P1IES: Interrupt Edge Select Register -** Each bit of the register selects the interrupt edge for the corresponding I/O pin. When the bit is set, the PxIFGx flag is set with a high-to-low transition. Here, the bit is set to interrupt on high to low transition.
- **P1IFG: Interrupt Flag Register -** Each PxIFGx bit is the interrupt flag for its corresponding I/O pin and is set when the selected input signal edge transition occurs at the pin.
- **P1IE: Interrupt Enable -** Each PxIE bit enables the associated PxIFG interrupt flag. Here P1.3 is interrupt enabled.

4.5 Procedure

1. Connect the MSP-EXP430G2 LaunchPad to the PC using the USB cable supplied.

2. Build, program and debug the code into the LaunchPad using CCS.

4.6 Observation

The port pin P1.3 of MSP430G2 Launchpad is configured as an input pin with a falling edge interrupt. On pressing on-board switch S2 of MSP430G2 Launchpad, an interrupt occurs and Interrupt Service Routine (ISR) blinks the on-board green led of MSP430G2 Launchpad which is connected to P1.6.

4.7 Summary

In this experiment, we have learnt to configure and program the GPIO pins of MSP430G2553 as interrupt and have successfully programmed the GPIO port P1.3 as falling edge interrupt to blink the on-board green led of MSP430G2 Launchpad.

4.8 Exercise

1. Write the code to enable a Timer interrupt for the pin P1.1.
2. Write the code to turn on interrupts globally.

Experiment 5

Pulse Width Modulation

Topics **Page**

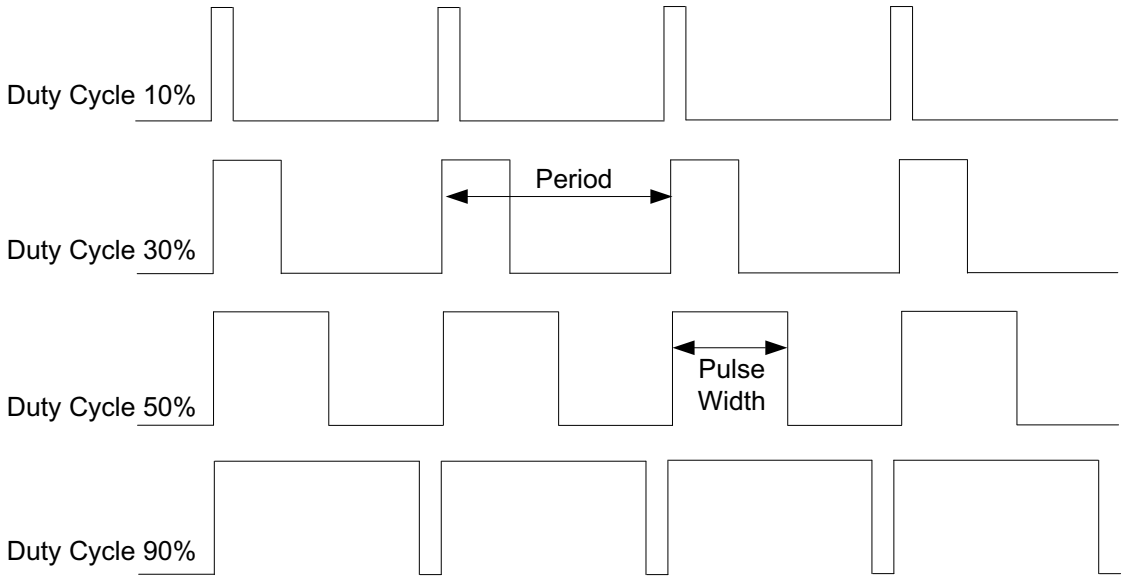
5.1 Objective

The main objective of this experiment is to implement Pulse Width Modulation to control the brightness of the on-board, green LED. This experiment will help you to learn and understand the configuration of PWM and Timer peripherals of the MSP430G2553.

5.2 Introduction

5.2.1 PWM

Pulse Width Modulation (PWM) is a method of digitally encoding analog signal levels. High-resolution digital counters are used to generate a square wave of a given frequency, and the duty cycle of the square wave is modulated to encode the analog signal. The duty cycle determines the time during which the signal pulse is HIGH. For example, **Figure 5-1** shows the different waveforms for varying duty cycles of a signal.



Duty Cycle = Pulse Width x 100 / Period

Figure 5-1 PWM Signal for Varying Duty Cycle

In PWM, a large analog voltage results in increased pulse width and vice versa. Hence, the duty cycle of the wave changes with the analog voltage that is encoded as shown in **Figure 5-2**.

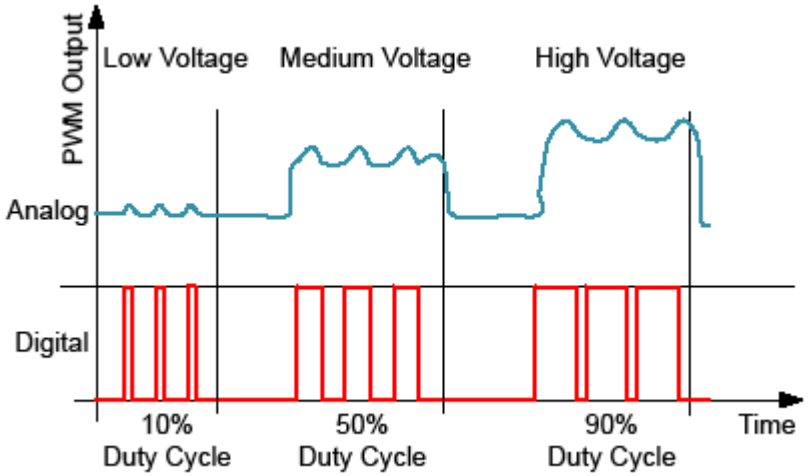


Figure 5-2 PWM Output for Analog Signal

The Timer A and Timer B of the MSP430G2553 are capable of PWM outputs in compare mode.

In this experiment, the brightness of the on-board, green LED is altered by varying the voltage on pin P1.6 which is connected to the green LED. The MSP430G2553 does not have a digital-to-analog (DAC) module; therefore, Pulse Width Modulation is used to achieve this.

The device is kept in low power mode zero till it is woken up by the Timer interrupt. In the Timer ISR, the brightness of the LED is varied by varying the duty cycle of the signal supplied to the output from the predefined array. The brightness of the LED will be gradually increased as per the array declared.

The functional block diagram shown in **Figure 5-3** illustrates the working principle of the experiment.

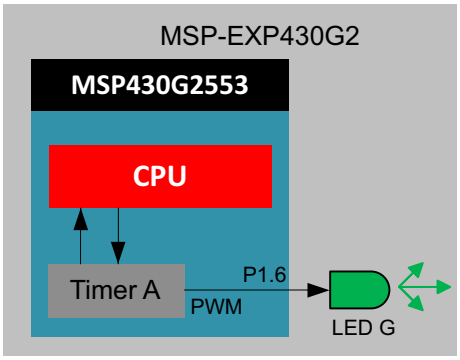


Figure 5-3 Functional Block Diagram

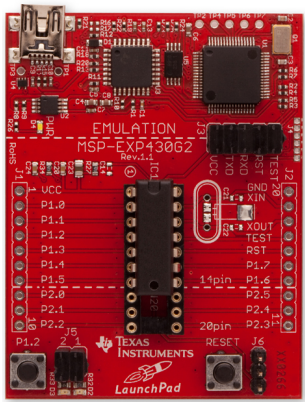
5.3 Component Requirements

5.3.1 Software Requirement

Code Composer Studio

5.3.2 Hardware Requirement

Table 5-1: Components Required for the Experiment

S.No	Components	Specifications	Image
1.	MSP430G2 LaunchPad	MSP-EXP430G2XL	
2.	USB cable		

5.4 Software

The software for the experiment is written in C and developed using the CCS Integrated Development Environment (IDE). Refer to "**Project Creation and Build**" in the Getting Started section of this manual for project build and debug using CCS. The software is programmed into the target device MSP430G2553 on the MSP-EXP430G2 LaunchPad using the USB interface.

5.4.1 Flowchart

The flowchart for the code is shown in **Figure 5-4**.The program code first disables the watchdog timer to prevent a restart on expiry of the WDT count. The port pin P1.6 connected to the green LED is configured as PWM output of Timer A. The Timer A is configured with the required period (TA0CCR0) and the PWM duty cycle (TA0CCR1) is set to first value in the predefined array. The Timer is configured in Set/Reset output mode 7. The Timer A control register is set to use SMCLK as clock source for up counting. The global interrupt is enabled, and the processor is switched to low power mode 0 (LPM0).

On interrupt by the Timer, the PWM duty cycle (TA0CCR1) is set to the next value from the predefined array. An increase in PWM duty cycle increases the brightness of the LED connected to its output. On reaching the last element of the array, maximum brightness is reached and the duty cycle is reset to the first value in the array.

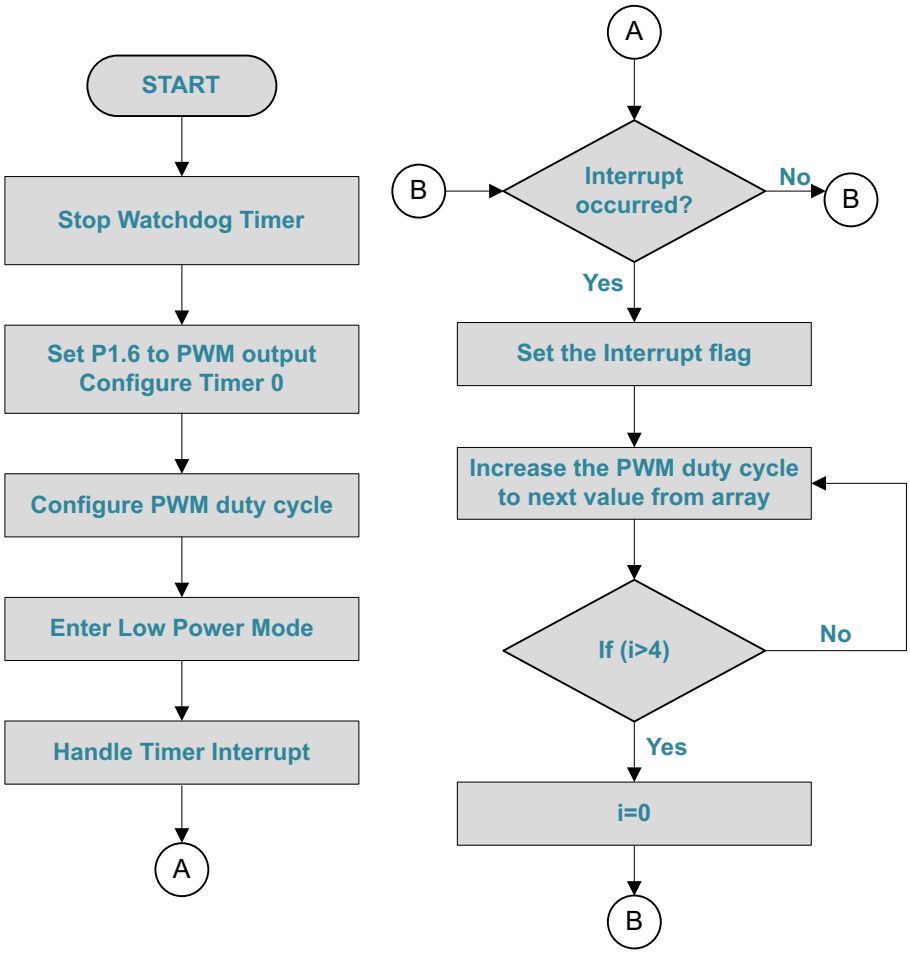


Figure 5-4 Flowchart for PWM Generation

5.4.2 C Program Code for PWM Generation

```
#include <msp430.h>
int a[5] = {0,32,64,128,255};
int i = 0;
void main(void){
    WDTCTL = WDTPW|WDTHOLD;         // Stop WDT
   //IE1 |= WDTIE;                   // enable Watchdog timer interrupts
    P1DIR |= BIT6;                  // Green LED for output
    P1SEL |= BIT6;                  // Green LED Pulse width modulation
    TA0CCR0 = 512;                  // PWM period
    TA0CCR1 = a[0];                 // PWM duty cycle
    TA0CCTL1 = OUTMOD_7;            // TA0CCR1 reset/set-high voltage
                                    // below count, low voltage when past
```

```
        TA0CTL = TASSEL_2 + MC_1 + TAIE +ID_3;
                                    // Timer A control set to SMCLK, 1MHz
                                    // and count up mode MC_1
        _bis_SR_register(LPM0_bits + GIE);     // Enter Low power mode 0
        while (1);
}
#pragma vector= TIMER0_A1_VECTOR    // Watchdog Timer ISR
  __interrupt void timer(void) {
        TA0CTL &= ~TAIFG;
        TA0CCR1 = a[++i];
        if (i>4)
             {
             i=0;
             }
}
```

5.4.3 Registers Used

The registers used are:

- **WDTCTL: 16-bit Watchdog Timer Control Register** - The Watchdog Timer (WDT) is typically used to trigger a system reset after a certain amount of time. The register is set to WDT_MDLY_32 for 32ms interval interrupt.
- **IE1: Interrupt Enable Register 1 -** This register enables the NMI interrupts and WDT interrupt. Here, the Watchdog timer interrupt is enabled.
- **P1DIR: 8-bit Direction Register** - Each bit in the direction register selects the direction of the corresponding pin as an input or an output. Here, bit 6 is set to 1 directing the Port Pin 1.6 as an output.
- **P1SEL: Function Select Register -** Each bit in the function select register is used to select the pin function - I/O port or peripheral module function. Here, the pin used as PWM output(P1.6) is specified.
- **TA0CTL: Timer A Control Register -** Each bit in this register configures Timer A operation. Here, SMCLK is used as clock source and the count up mode (MC_1) is selected for timer configuration.
- **TA0CCR0: Timer A Capture/Compare Register -** It holds the data for comparison to the timer value in compare mode. Here, the PWM period is set using this register.
- **TA0CCR1: Timer A Capture/Compare Register -** It holds the data for comparison to the timer value in compare mode. Here, this register set is used to the PWM duty cycle. This value will be equal to the reading taken from the potentiometer at any time instant and will alter the LED's brightness.
- **TA0CCTL1: Timer A Capture/Compare Control Register -** The Timer capture/compare register configures the capture/compare modes of the timer. Here, the PWM output mode is set to PWM Reset/Set mode (OUTMODE_7).

5.5 Procedure

1. Connect the MSP-EXP430G2 LaunchPad to the PC using the USB cable supplied.
2. Build, program and debug the code into the LaunchPad using CCS.

5.6 Observation

The green LED on our LaunchPad board starts off glowing very dimly and gradually gets brighter till it reaches a maximum. It then, goes back to its minimum brightness and the process repeats.

Figure 5-5 Variation in Brightness of Green LED due to PWM

We can suspend and resume the running code at different instances to view the value of our duty cycle for different brightness levels.

The register values for three different cases of the duty cycle are tabulated in **Table 5-2**. **Figure 5-6** shows the register values displayed in CCS watch window.

Table 5-2: Duty Cycle Calculation for Timer A Register Values

Sl.No	TA0CCR1	TA0CCR0	Duty Cycle = (TA0CCR1/TA0CCR0)* 100
1.	0x00C9	0x03E8	20%
2.	0x02E5	0x03E8	74%
3.	0x03C1	0x03E8	96%

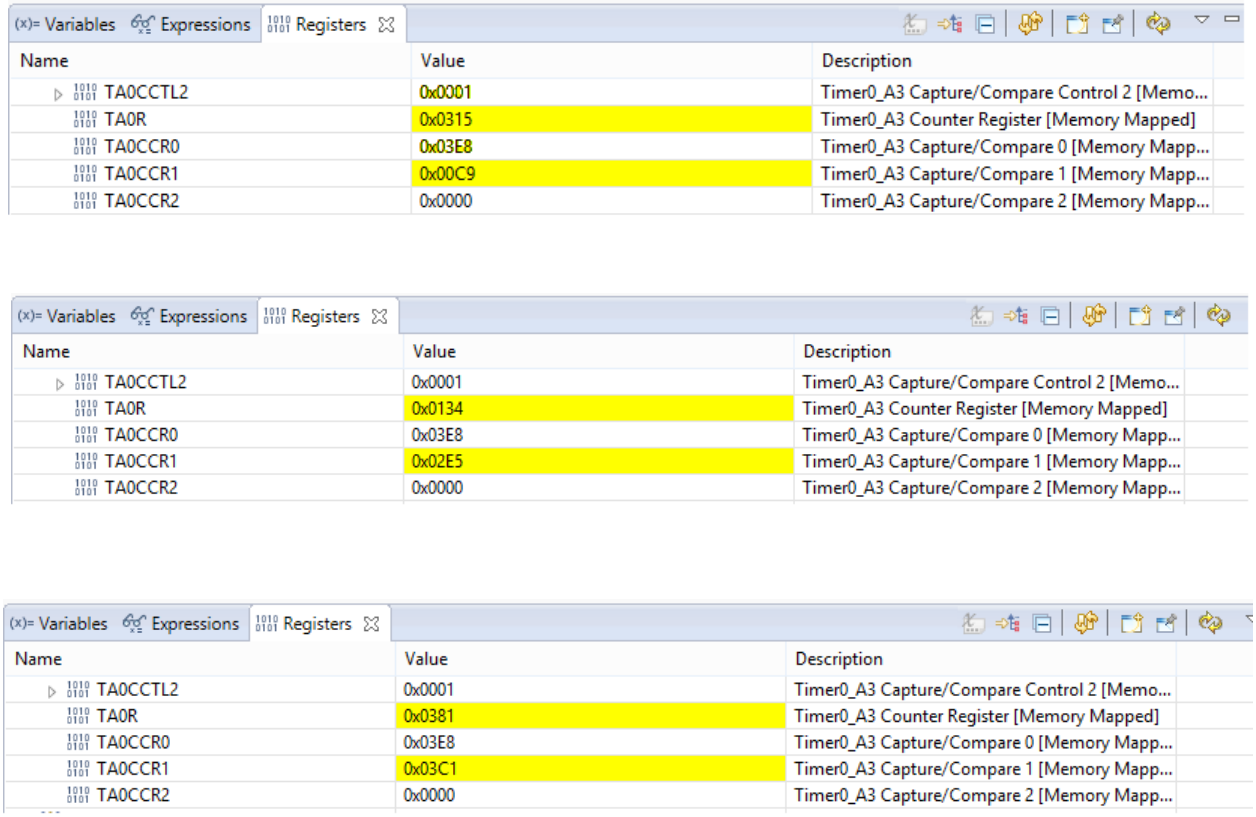


Figure 5-6 Register Values in CCS Watch Window for Duty Cycle Calculation

5.7 Summary

In this experiment we have learnt to configure and program the GPIO pins as PWM pins of MSP430G2553 and have successfully programmed the GPIO port P1.6 as PWM pin to blink the on-board green led of MSP430G2 Launchpad.

5.8 Exercise

1. Observe the PWM waveform on a particular pin using CRO.
2. What is the maximum resolution of PWM circuitry in MSP430G2 LaunchPad?
3. Change the above code to create a PWM signal of 75% duty cycle on particular PWM pin.

Experiment 6

Interfacing Potentiometer with MSP430

Topics **Page**

6.1 Objective

The main objective of this experiment is to control the on-board, red LED by the analog input from a potentiometer. This experiment will help you to learn and understand how to configure an ADC to interface with a potentiometer.

6.2 Introduction

The MSP430G2553 consists of ADC10, 10-bit analog-to-digital converter which converts the analog input into a maximum of 1024 discrete levels. The ADC10 module is capable of 8 external inputs and 4 internal analog sources. The ADC10 can be configured to convert a single channel or a block of contiguous channels and can optionally repeat its conversions automatically. Each conversion takes multiple cycles. The timing is controlled by an ADC clock. While a conversion is in progress, the busy flag set. After conversion, the result is stored in the ADC10MEM register. There is a possible interrupt signaled when a conversion is complete. There is also a **data transfer controller** that can steal cycles from the CPU and store a block of conversion results directly in memory.

The block diagram for the experiment is as shown in **Figure 6-1**. In this experiment, the analog input is provided by the potentiometer on analog channel A3 which is the same pin as P1.3. The resistance of the potentiometer is varied by turning its knob, thereby resulting in a varying analog voltage on A3.

This voltage is converted into 1024 discrete levels from 0V to 3V DC (since the V_{ref} is set to V_{cc}). The red LED turns on when the voltage crosses the threshold of $V_{cc}/2$ (or the digital level 512).

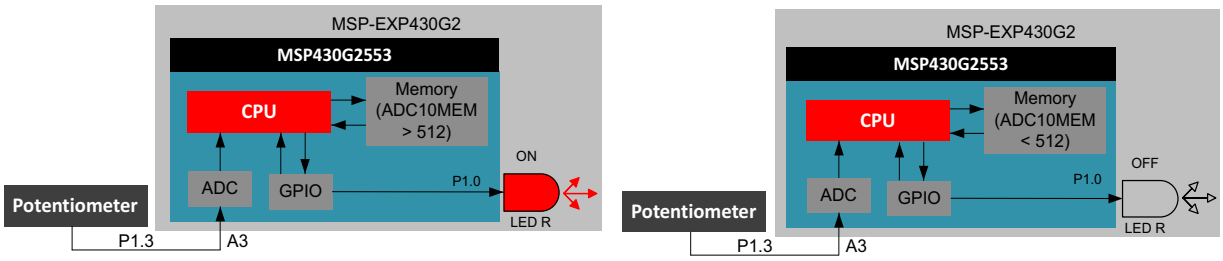


Figure 6-1 Functional Block Diagram

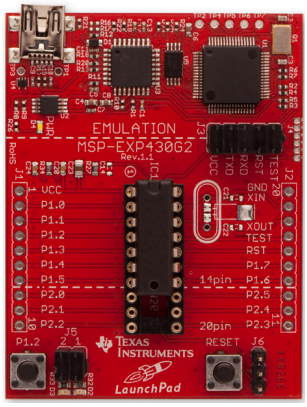
6.3 Component Requirements

6.3.1 Software Requirement

Code Composer Studio

6.3.2 Hardware Requirement

Table 6-1: Components Required for the Experiment

S.No	Components	Specifications	Image
1.	MSP430G2 LaunchPad	MSP-EXP430G2XL	
2.	USB cable		
3.	10,000 Ohm Potentiometer		

6.4 Software

The software for the experiment is written in C and developed using the CCS Integrated Development Environment (IDE). Refer to "**Project Creation and Build**" in the Getting Started section of this manual for project build and debug using CCS. The software is programmed into the target device MSP430G2553 on the MSP-EXP430G2 LaunchPad using the USB interface.

6.4.1 Flowchart

The flowchart for the code is as shown in **Figure 6-2**. The program code first disables the watchdog timer to prevent a restart on expiry of the WDT count. The GPIO port pin P1.0 connected to the red LED is configured as output. ADC10 is configured by two control registers, ADC10CTL0 and ADC10CTL1 with reference voltage as V_{CC} and input channel as A3 connected to pin P1.3. The ADC converter is enabled, to start sampling and conversion of analog input. If the converted value is greater than 512, the red LED is turned ON. Else, it is turned OFF.

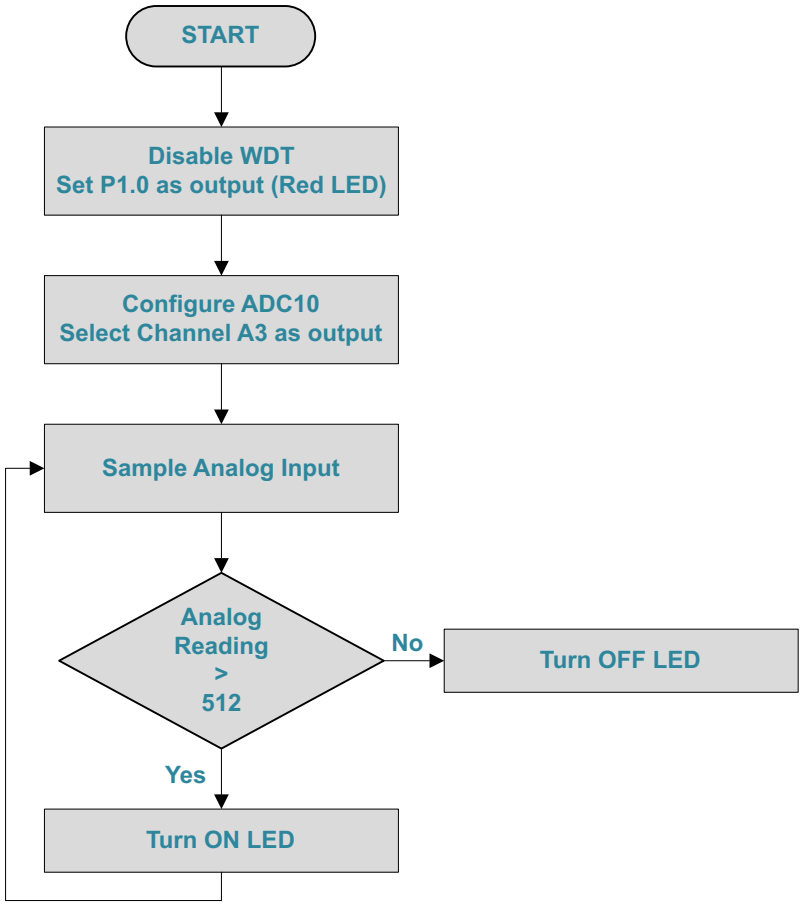


Figure 6-2 Flowchart for Interfacing the Potentiometer with MSP-EXP430G2XL

6.4.2 C Program Code for Interfacing Potentiometer

```
#include <msp430.h>
int main(void)
{
  WDTCTL = WDTPW + WDTHOLD;              // Stop WDT
  ADC10CTL0 = SREF_0 + ADC10SHT_2 + ADC10ON;
  ADC10CTL1 = INCH_3;                    // input A3
  ADC10AE0 |= 0x08;                      // PA.3 ADC option select
  P1DIR |= 0x01;                         // Set P1.0 to output direction

  while(1)
  {
    ADC10CTL0 |= ENC + ADC10SC;          // Sampling and conversion start
```

```
        if (ADC10MEM < 512)              // ADC10MEM = A3 > 512?
          P1OUT &= ~0x01;                // Clear P1.0 LED off
        else
          P1OUT |= 0x01;                 // Set P1.0 LED on
      }
    }
```

6.4.3 Registers Used

The registers used are:

- **WDTCTL: 16-bit Watchdog Timer Control Register** - The Watchdog Timer (WDT) is typically used to trigger a system reset after a certain amount of time. The register is set to WDT_MDLY_32 for 32ms interval interrupt.
- **ADC10CTL0:ADC10 Control Register 0 -** The ADC10 core is configured by the ADC10CTL0 and ADC10CTL1 registers.
- **ADC10CTL1:ADC10 Control Register 1 -** The ADC10 core is configured by the ADC10CTL0 and ADC10CTL1 registers.
- **ADC10AE: Analog (Input) Enable Control Register 0 -** The bits of this register enable the corresponding pin for analog input.
- **P1DIR: 8-bit Direction Register -** Each bit in the direction register selects the direction of the corresponding pin as an input or an output. Here, bit 0 is set to 1 directing the Port Pin 1.0 as an output.
- **ADC10MEM: Conversion Memory Register -** The 10-bit conversion results are stored in this register.

The various configuration parameters related to the various registers of the ADC10 used in the current program are tabulated in **Table 6-2**.

Table 6-2: Configuration Parameters for ADC10 Registers

Parameter	Denoted by	Location	Comments
Turn on ADC10	ADC10ON	ADC10CTL0, bit 4	Overall on/off for ADC
Enable Conversion	ENC	ADC10CTL0, bit 1	Enables conversion
Start Conversion	ADC10SC	ADC10CTL0, bit 0	Start conversion (writing a 1 starts sampling and conversion), returns to 0 automatically.
Analog Enable		ADC10AE0	Bit flags. A set bit enables the corresponding analog input.
Interrupt enable	ADC10IE	ADC10CTL0 bit 3	Used to service an interrupt when a value is received on the ADC. The corresponding fag that is set is ADC10IFG.
Input Channel Select	INCH	ADC10CTL1 bits 15-12	Specifies highest input channel

Table 6-2: Configuration Parameters for ADC10 Registers

Parameter	Denoted by	Location	Comments
Select Reference	SREF	ADC10CTL0 bits 15-13	Voltage references. Default is Vss and V_{CC}. Can be internally generated sources or external sources.
Reference Generator	REFON, REF2_5V	ADC10CTL0 bits 5,6	Enable optional internal reference voltage of either 1.5 or 2.5V
Sample and hold time	ADC10SHT	ADC10CTL0 bits 12-11	Time used for sample and hold. Depends on circuit time constants (see data sheet). Default is 4 ADC clock cycles

6.5 Procedure

1. Connect the MSP-EXP430G2 LaunchPad to the PC using the USB cable supplied.

2. Connect the positive lead of the potentiometer to the V_{cc} pin on the MSP-EXP430G2 LaunchPad and the negative lead of the potentiometer to the GND pin.

3. Connect the output lead of the potentiometer to pin P1.3 or Analog Channel A3.

4. Connect the jumpers on the MSP-EXP430G2 LaunchPad for RXD and TXD horizontally.

The setup appears as shown in **Figure 6-3**.

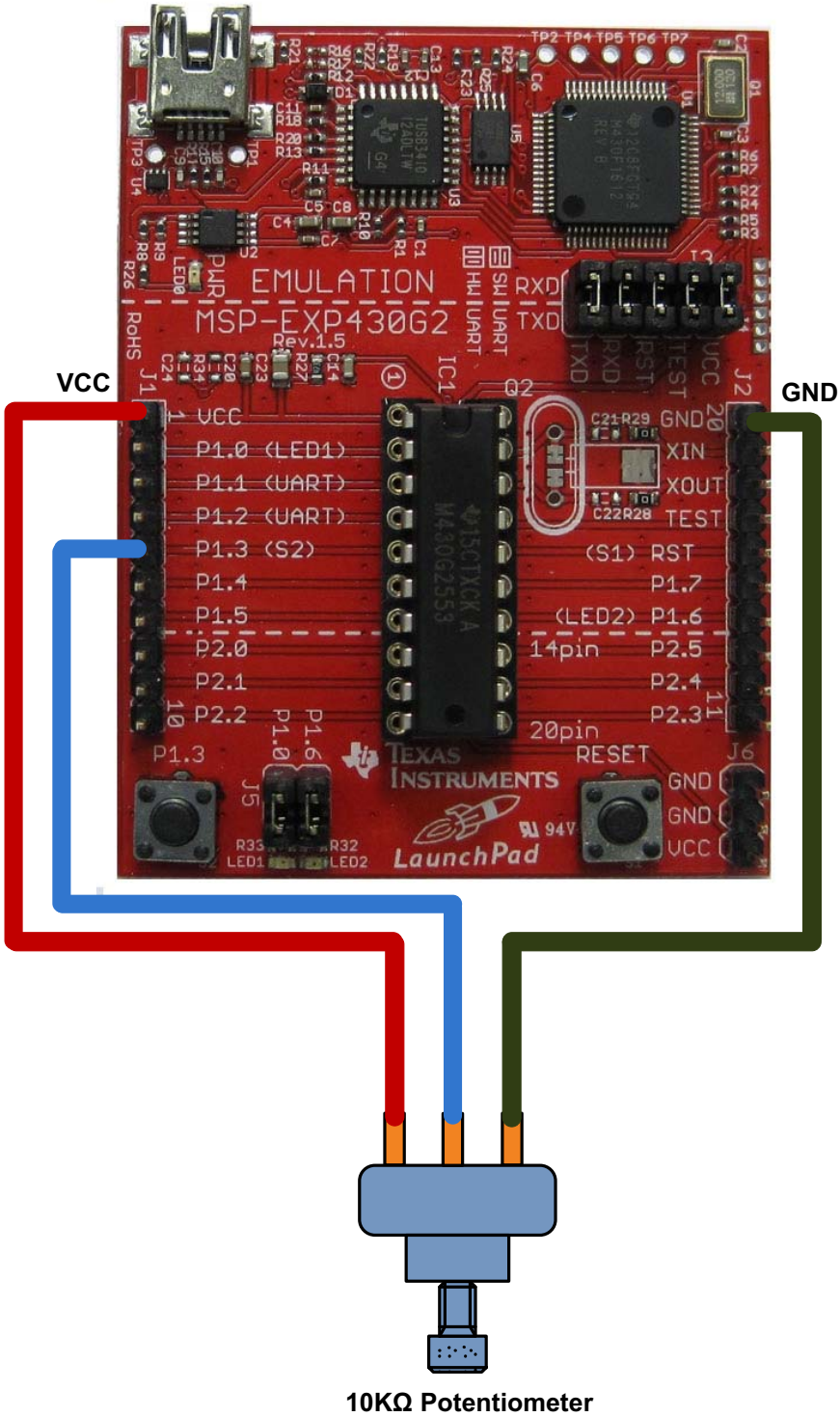


Figure 6-3 Hardware Setup

5. Build, program and debug the code into the LaunchPad using CCS.

6. Vary the potentiometer and observe the on board red LED.

6.6 Observation

The LED is initially OFF, when the resistance is at its maximum. As we turn the potentiometer, the LED turns ON after a certain point. This point is the ADC voltage level of 512.

The potentiometer reading can be viewed by watching the ADC10MEM register on CCS Debug Perspective as shown in **Figure 6-4.**

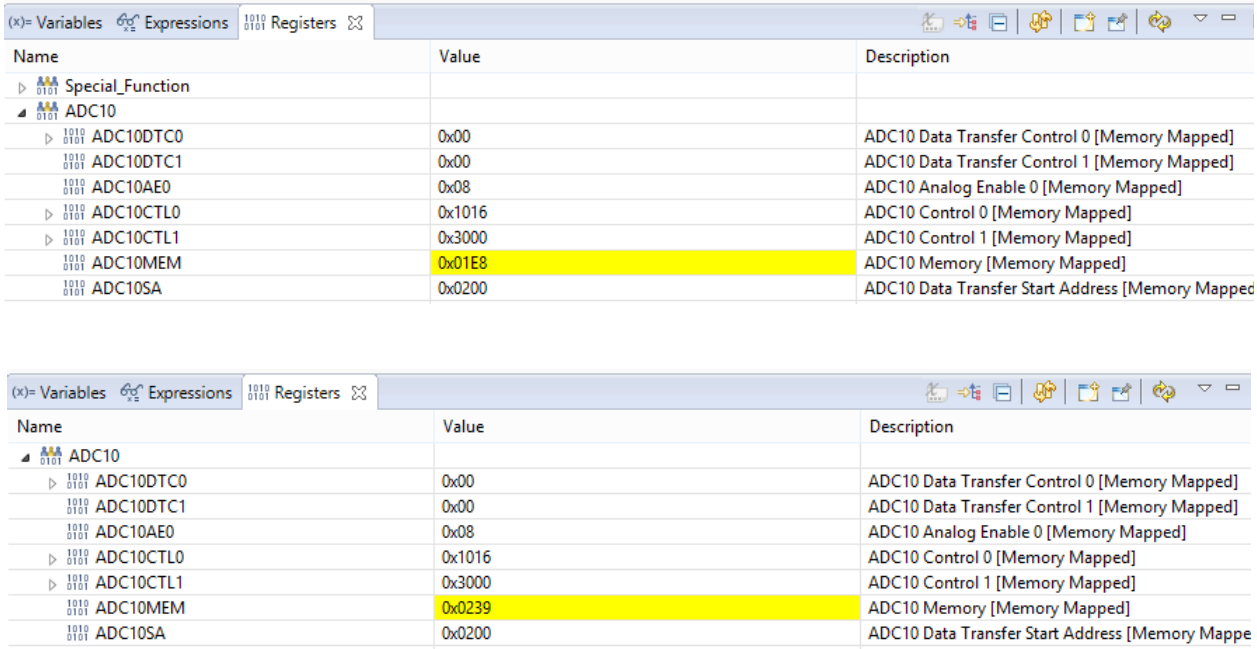


Figure 6-4 ADC Conversion Register Values in CCS Watch Window

6.7 Summary

In this experiment, we have learnt to configure and program the ADC to control the red LED based on the analog input from the potentiometer.

6.8 Exercise

1. Alter the threshold to 75% of V_{cc} for the LED to turn on.

2. Modify the code to change the Reference Voltage from V_{cc} to 2.5V.

Hint: Use Parameters SREF, REFON, REF2_5V.

Experiment 7

PWM Based Speed Control of Motor by Potentiometer

Topics **Page**

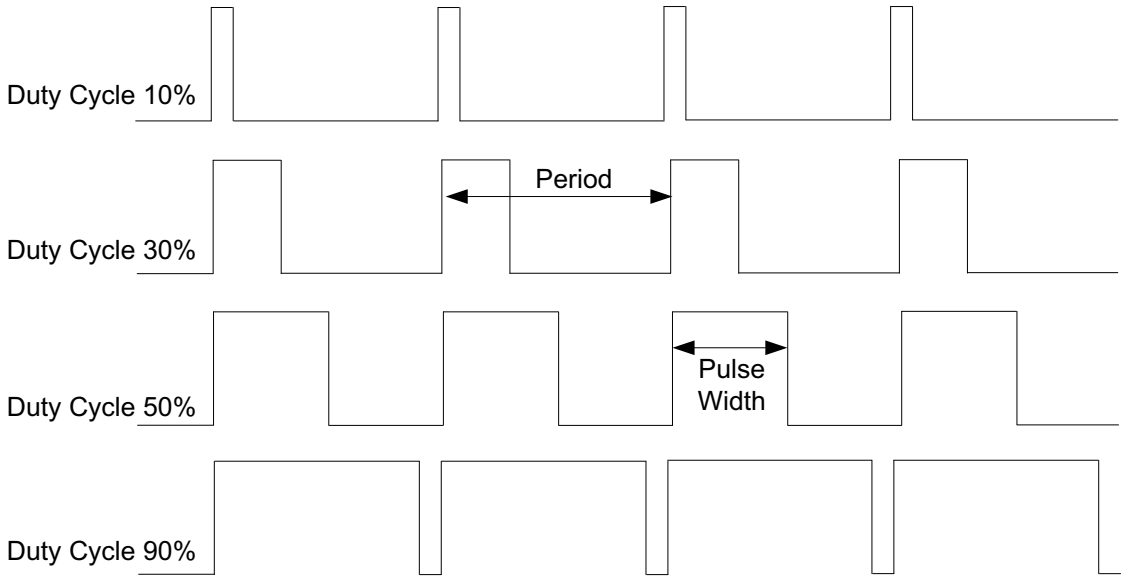
7.1 Objective

The main objective of this experiment is to control the speed of a DC Motor using the potentiometer. This experiment will help to learn and understand how to configure the PWM and ADC modules of the processor to control the DC motor using potentiometer input.

7.2 Introduction

7.2.1 Pulse Width Modulation

Pulse Width Modulation (PWM) is a method of digitally encoding analog signal levels. High-resolution digital counters are used to generate a square wave of a given frequency, and the duty cycle of the square wave is modulated to encode the analog signal. Duty cycle determines the time during which the signal pulse is HIGH. **Figure 7-1** shows the different waveforms for varying duty cycles of the signal.



Duty Cycle = Pulse Width x 100 / Period

Figure 7-1 PWM Signal for Varying Duty Cycle

A large analog voltage results in increased pulse width and vice versa. Hence, the duty cycle of the wave changes with the analog voltage that is encoded as shown in **Figure 7-2**.

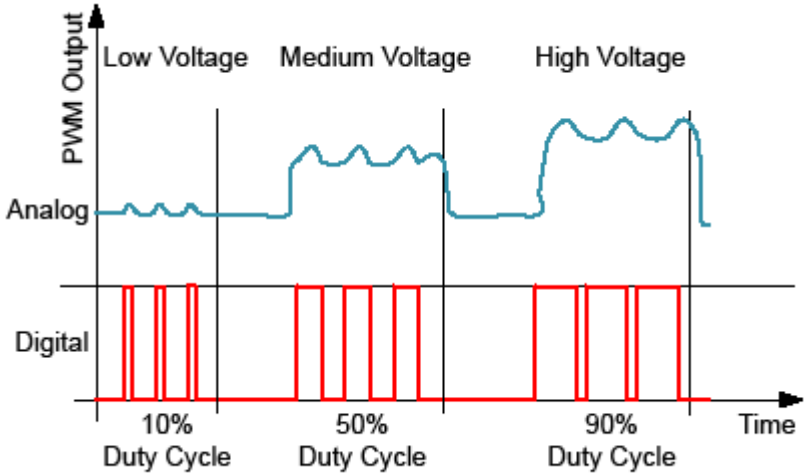


Figure 7-2 PWM Output for Analog Signal

The Timer A and Timer B of the MSP430G2553 are capable of PWM outputs in compare mode.

7.2.2 Analog-Digital Converter

The MSP430G2553 has ADC10, a 10-bit analog-to-digital converter which converts the analog input into a maximum of 1024 discrete levels. The ADC10 module is capable of 8 external inputs and 4 internal analog sources. The ADC10 can be configured to convert a single channel or a block of contiguous channels and can optionally repeat its conversions automatically. Each conversion takes multiple cycles. The timing is controlled by an ADC clock. While a conversion is in progress, the busy flag is set. After conversion, the result is stored in the ADC10MEM register. There is a possible interrupt signaled when a conversion is complete. There is also a 'data transfer controller' that can steal cycles from the CPU and store a block of conversion results directly in memory.

Low-power mode 0 (LPM0)

- CPU is disabled
- ACLK and SMCLK remain active, MCLK is disabled

The block diagram for this experiment is shown in **Figure 7-3**. In this experiment, the analog signal from potentiometer connected to P1.3 is read by the ADC10, and the converted digital value is used to vary the duty cycle of the PWM output. The PWM output in turn controls the speed of the DC motor which is connected to PWM pin P1.6. The PWM period is set to 1024 in the Timer A capture/compare register 0 (TA0CCR0). The digital value after conversion is stored in ADC10MEM which is passed to the TA0CCR1 register to set the PWM duty cycle. This value can vary from 0 to 1024 based on the position of the potentiometer. The timer interrupt is set to sample and convert the analog input to digital output. The device is kept in low power mode, until it is woken up by the interrupt.

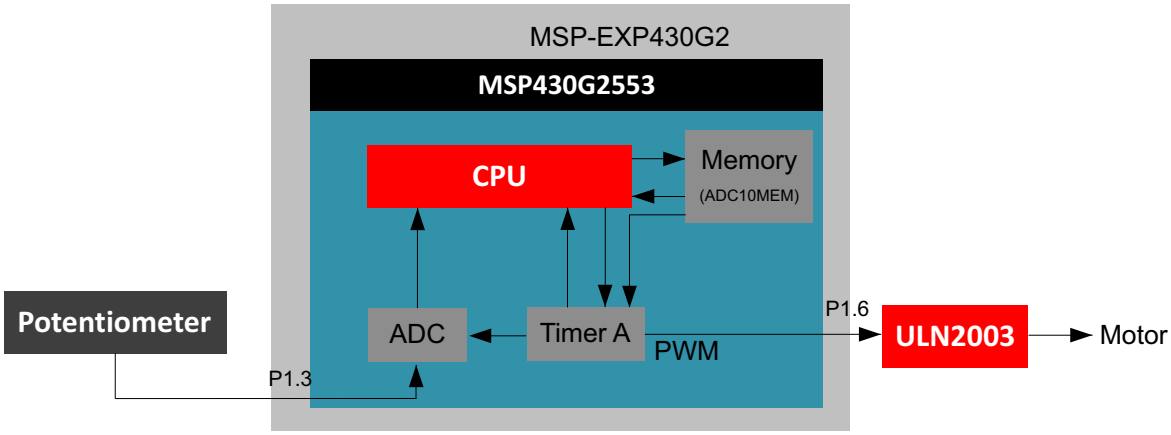


Figure 7-3 Functional Block Diagram

7.3 Component Requirements

7.3.1 Software Requirement

Code Composer Studio

7.3.2 Hardware Requirement

Table 7-1: Components Required for the Experiment

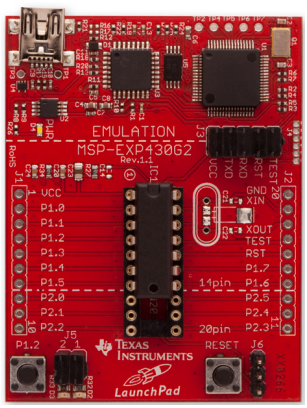
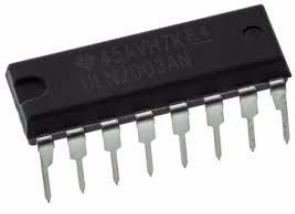
S.No	Components	Specifications	Image
1.	MSP430G2 LaunchPad	**MSP-EXP430G2XL**	
2.	USB cable		
3.	ULN2003AN	Motor Driver IC	
4.	DC Motor		

Table 7-1: Components Required for the Experiment

S.No	Components	Specifications	Image
5.	10,000 Ohm Potentiometer		

7.4 Software

The software for the experiment is written in C and developed using the CCS Integrated Development Environment (IDE). Refer to "**Project Creation and Build**" in the Getting Started section of this manual for project build and debug using CCS. The software is programmed into the target device MSP430G2553 on the MSP-EXP430G2 LaunchPad using the USB interface.

7.4.1 Flowchart

The flowchart for the code is shown in **Figure 7-4**. The program code first disables the watchdog timer to prevent a restart on expiry of the WDT count. The ADC10 core is configured by two control registers, ADC10CTL0 and ADC10CTL1 with reference voltage as V_{cc} and input channel as A3 connected to pin P1.3.

The port pin P1.6 connected to the DC motor is configured as PWM output of Timer A that is configured in Set/Reset output mode 7 with period 1024 (TA0CCR0) and required PWM duty cycle (TA0CCR1). Initially, the PWM duty cycle is set to 0.1%. Timer A control is set to use SMCLK as clock source for up counting. The global interrupt is enabled, and the processor is switched to low power mode 0.

On interrupt by the Timer, the ADC converter is enabled and sampling and conversion of analog input is started. The ADC converted value stored in ADC10MEM is copied to TA0CCR1 to set the PWM duty cycle for the output to the DC motor.

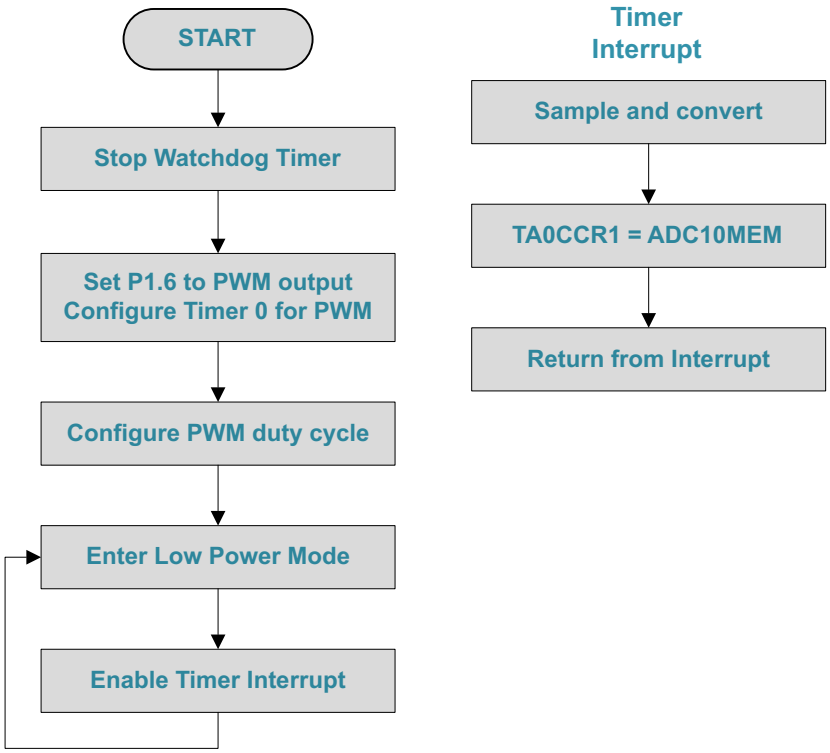


Figure 7-4 Flowchart for PWM Based Speed Control of Motor by Potentiometer

7.4.2 C Program Code for PWM Based Speed Control of Motor

```
#include <msp430.h>

int pwmDirection = 1;
void main(void){
    WDTCTL = WDTPW|WDTHOLD;          // Stop WDT
    ADC10CTL0 = SREF_0 + ADC10SHT_2 + ADC10ON;
    ADC10CTL1 = INCH_3;              // input A3
    ADC10AE0 |= 0x08;                // PA.3 ADC option select
    P1DIR |= BIT6;                   // Green LED for output
    P1SEL |= BIT6;                   // Green LED Pulse width modulation
    TA0CCR0 = 1024;                  // PWM period
    TA0CCR1 = 1;                    // PWM duty cycle,on 1/1000 initially
    TA0CCTL1 = OUTMOD_7;             // TA0CCR1 reset/set-high voltage
                                     // below count,low voltage when past
    TA0CTL = TASSEL_2 + MC_1 + TAIE +ID_3;
```

```
                                        // Timer A control set to SMCLK,1MHz
                                        // and count up mode MC_1
    _BIS_SR(LPM4_bits + GIE);
    while (1);                          // Enter Low power mode 0
}
#pragma vector= TIMER0_A1_VECTOR        // Watchdog Timer ISR
  __interrupt void timer(void) {

  ADC10CTL0 |= ENC + ADC10SC;
  TA0CCR1 = ADC10MEM;
}
```

7.4.3 Registers Used

The registers used are:

- **WDTCTL: 16-bit Watchdog Timer Control Register** - The Watchdog Timer (WDT) is typically used to trigger a system reset after a certain amount of time. The register is set to WDT_MDLY_32 for 32ms interval interrupt.
- **IE1: Interrupt Enable Register 1 -** This register enables the NMI interrupts and WDT interrupt. Here, the Watchdog timer interrupt is enabled.
- **P1DIR: 8-bit Direction Register** - Each bit in the direction register selects the direction of the corresponding pin as an input or an output. Here, bit 6 is set to 1 directing the Port Pin 1.6 as an output.
- **P1SEL: Function Select Register -** Each bit in the function select register is used to select the pin function - I/O port or peripheral module function. Here, the pin used as PWM output(P1.6) is specified.
- **TA0CTL: Timer A Control Register -** Each bit in this register configures Timer A operation. Here, SMCLK is used as clock source and the count up mode (MC_1) is selected for timer configuration.
- **TA0CCR0: Timer A Capture/Compare Register -** It holds the data for comparison to the timer value in compare mode. Here, the PWM period is set using this register.
- **TA0CCR1: Timer A Capture/Compare Register -** It holds the data for comparison to the timer value in compare mode. Here, this register set is used to the PWM duty cycle. This value will be varied to alter the LED's brightness.
- **TA0CCTL1: Timer A Capture/Compare Control Register -** The Timer capture/compare register configures the capture/compare modes of the timer. Here, the PWM output mode is set to PWM Reset/Set mode (OUTMODE_7).
- **ADC10AE: Analog (Input) Enable Control Register 0 -** The bits of this register enable the corresponding pin for analog input. Here, input channel A3 is enabled.
- **ADC10CTL0:ADC10 Control Register 0 -** The ADC10 core is configured by the ADC10CTL0 and ADC10CTL1 registers. The voltage reference, sample and hold times are configured in this register.
- **ADC10CTL1:ADC10 Control Register 1 -** The ADC10 core is configured by the ADC10CTL0 and ADC10CTL1 registers. Here, input channel A3 is enabled.

- **ADC10MEM: Conversion Memory Register -** The 10-bit conversion results are stored in this register. The value in this register is used to set the duty cycle value for PWM output of Timer A.

7.5 Procedure

7.5.1 Hardware Setup

Figure 7-5 shows the hardware connections for the potentiometer and the DC motor with the MSP-EXP430G2 LaunchPad.

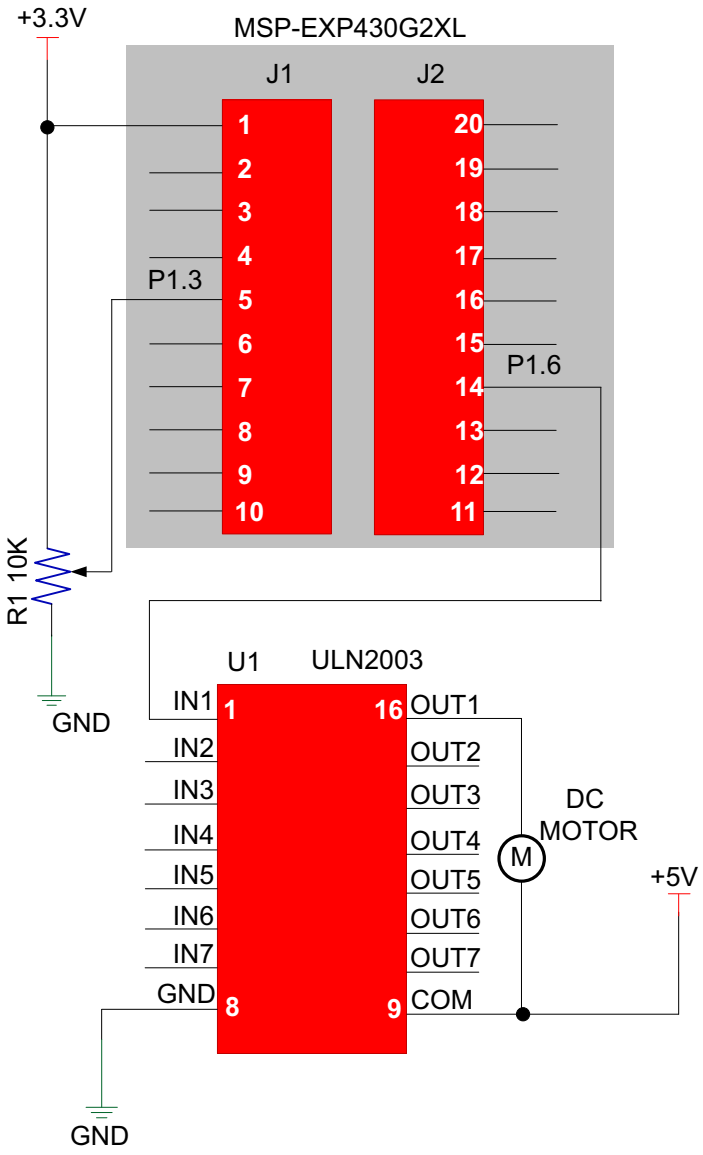


Figure 7-5 Connection Diagram for Potentiometer and DC Motor with MSP-EXP430G2XL

The procedure to be followed for hardware setup is as follows:

1. Position DIP IC ULN2003AN on a Breadboard.

2. Connect one terminal of the DC motor to the Common pin (IC Pin 9) of ULN2003.

3. Connect the junction of the above 2 terminals 5V Power from USB.

4. Connect the other terminal of the DC motor to the Drive Pin (IC Pin 16) of ULN 2003.

5. Connect the GND of the LaunchPad to the Ground Pin (IC Pin 8) of ULN2003.

6. Connect the PWM Output P1.6 to Input Signal (IC Pin 1) of ULN2003.

7. Connect one lead of the Potentiometer to VCC (J1 connector, Pin 1) on the LaunchPad.

8. Connect other lead of the Potentiometer to the GND Pin of ULN2003 (IC Pin 8).

9. Connect the center lead of the Potentiometer (variable analog output) to P1.3 which is the analog Input 0 of ADC10 module of MSP43G2553.

After connecting the hardware, the setup appears as shown in **Figure 7-6.**

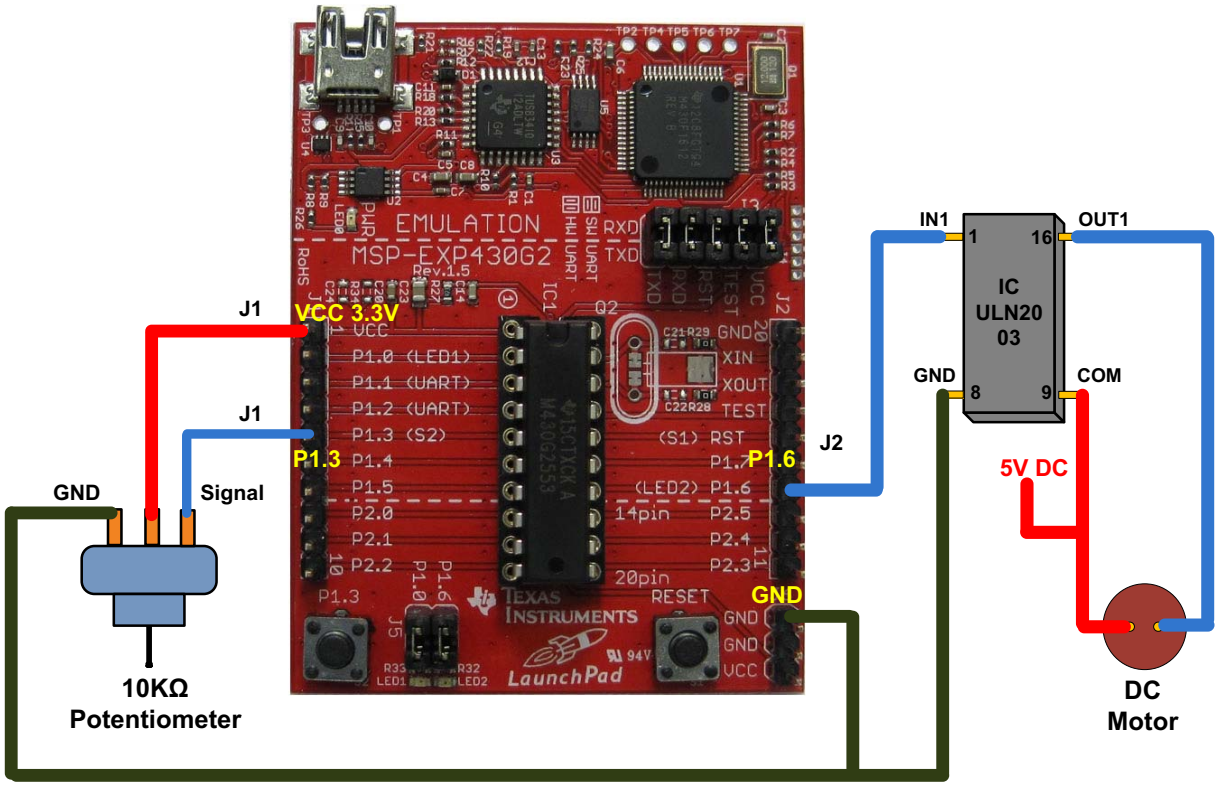


Figure 7-6 Hardware Setup

7.5.2 Software Execution

1. Build, program and debug the code into the LaunchPad using CCS.

2. Vary the potentiometer and observe the speed of the DC motor.

3. Also observe the corresponding digital output on the CCS window.

7.6 Observation

We see the speed of the DC motor varies as we change the position of the potentiometer knob.

If we suspend the program and simultaneously watch the registers ADC10MEM and TA0CCR1, we will find their values to be identical, as desired.

To watch specific registers, right-click on those registers in the Registers tab and select the **Watch** option. The watch window shows the values of the registers as shown in **Figure 7-6**.

(x)= Variables | Expressions | Registers

Name	Value	Description
ADC10		
ADC10DTC0	0x00	ADC10 Data Transfer Control 0 [Memory Mapped]
ADC10DTC1	0x00	ADC10 Data Transfer Control 1 [Memory Mapped]
ADC10AE0	0x08	ADC10 Analog Enable 0 [Memory Mapped]
ADC10CTL0	0x1016	ADC10 Control 0 [Memory Mapped]
ADC10CTL1	0x3000	ADC10 Control 1 [Memory Mapped]
ADC10MEM	0x03FE	ADC10 Memory [Memory Mapped]
ADC10SA	0x0200	ADC10 Data Transfer Start Address [Memory Mapped]

(x)= Variables | Expressions | Registers

Name	Value	Description
Timer0_A3		
TA0IV	0x0000	Timer0_A3 Interrupt Vector Word [Memory Mapped]
TA0CTL	0x0211	Timer0_A3 Control [Memory Mapped]
TA0CCTL0	0x0001	Timer0_A3 Capture/Compare Control 0 [Memory Map...
TA0CCTL1	0x00E1	Timer0_A3 Capture/Compare Control 1 [Memory Map...
TA0CCTL2	0x0001	Timer0_A3 Capture/Compare Control 2 [Memory Map...
TA0R	0x02EB	Timer0_A3 Counter Register [Memory Mapped]
TA0CCR0	0x0400	Timer0_A3 Capture/Compare 0 [Memory Mapped]
TA0CCR1	0x03FE	Timer0_A3 Capture/Compare 1 [Memory Mapped]
TA0CCR2	0x0000	Timer0_A3 Capture/Compare 2 [Memory Mapped]

Figure 7-7 ADC Conversion Register Values in CCS Watch Window

7.7 Summary

In this experiment, we have learnt to configure and program the ADC to control the speed of the DC motor based on the analog input from the potentiometer.

7.8 Exercise

1. Interface a Stepper motor with MSP-EXP430G2 LaunchPad to run it in a predetermined uniform speed.

2. Describe the applications of PWM in a digital power supply control.

3. Create Switch case code from the example code to run the DC Motor in 3 set of speeds.

Experiment 8

Using ULP Advisor on MSP430

Topics **Page**

8.1 Objective

The main objective of this experiment is to optimize the power efficiency of an application on MSP-EXP430G2 LaunchPad using ULP Advisor in CCS Studio. This experiment will help to learn and understand the ULP Advisor capabilities and usage of ULP Advisor to create optimized, power-efficient applications on the MSP-EXP430G2 LaunchPad.

8.2 Introduction

8.2.1 ULP Advisor

ULP (Ultra-Low Power) Advisor is a tool that provides advice on how to improve the energy efficiency of an application based on comparing the code with a list of ULP rules at compile time. The ULP Advisor helps the developer to fully utilize the ULP capabilities of MSP430 microcontrollers, This tool works at build time to identify possible areas where energy and power use is inefficient. A description of the ULP rule that is violated, 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 for increased power efficiency or continue to run the code as-is.

The ULP Advisor is built into Code Composer Studio™ IDE version 5.2 and newer. After compiling the code, a list of the ULP rule violations is provided in the Advice window. Unlike errors, these suggestions will not prevent the code from successfully compiling and downloading onto the target.

A list of some unique rule violations are shown in **Table 8-1**.

Table 8-1: ULP Rule Violations

Rule Violations	Detailed Description
(ULP $_{5.3}$)	Detected sprintf( ) operations
(ULP $_{5.2}$)	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

This experiment explains in detail how to use ULP Advisor on MSP430G2553. Three code files are used in this experiment and 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 back channel Universal Asynchronous Receiver/Transmitter (UART) at 9600bps.

The first code file - Inefficient.c, begins with an inefficient implementation of this application. The ULP Advisor is used to observe the extreme inefficiency of this version. Based on the feedback from the ULP Advisor, improvements are made for the somewhat efficient second version of the code - Efficient.c. The process is repeated, and ULP Advisor is used to compare this improved ver-

sion with the original. Finally, steps are taken to improve the code even further in the very efficient third version of the code - MostEfficient.c.

Table 8-2 briefs the descriptions of the experiment code.

Table 8-2: Experiment Code Descriptions

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

*The codes above are written for MSP430FR5969 LaunchPad. ULP Advisor tool helps us in evaluating the code for optimizing power and energy before debugging and is a feature of CCS.

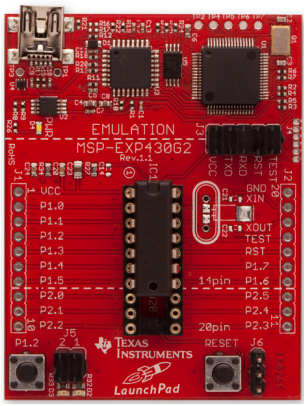
8.3 Component Requirements

8.3.1 Software Requirement

Code Composer Studio

8.3.2 Hardware Requirement

Table 8-3: Components Required for the Experiment

S.No	Components	Specifications	Image
1.	MSP430G2 LaunchPad	MSP-EXP430G2XL	
2.	USB cable		

8.4 Software

This experiment uses the **ULP_Case_Study** example in the CCS Integrated Development Environment (IDE). Refer to "**Project Creation and Build**" in the Getting Started section of this manual for project build and debug using CCS.

8.5 Procedure

1. Connect the MSP-EXP430G2 LaunchPad to the PC using the USB cable supplied.

2. Download the CCS example project ULP_Case_Study from **http://www.ti.com/lit/zip/slaa603** and extract using 7zip.

3. Import the project into your CCS workspace as shown in **Figure 8-1**.

a. Click **Project** → **Import Existing CCS Eclipse Project**.

b. Browse the directory for the downloaded file.

c. Check the box next to the **ULP_Case_Study** project in the **Discovered projects** window.

d. Uncheck the check box **Copy projects into workspace**.

e. Click **Finish**.

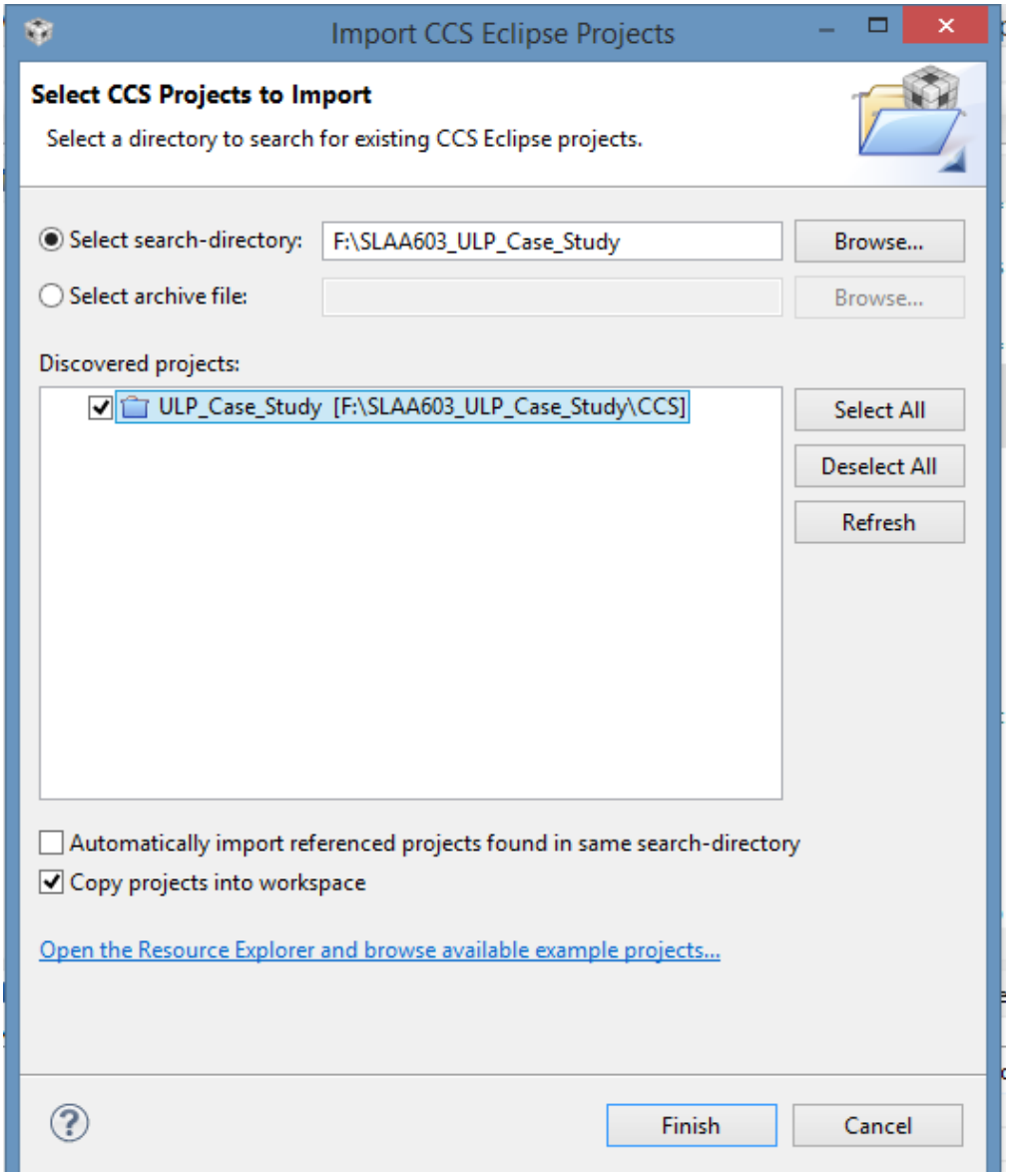


Figure 8-1 Importing the Project

4. To set as active project, click on the project name. The active build configuration should be **Inefficient**. If not, right click on the project name in the Project Explorer, then click **Build Configurations** ⟶ **Set Active** ⟶ **Inefficient** as shown in **Figure 8-2.**

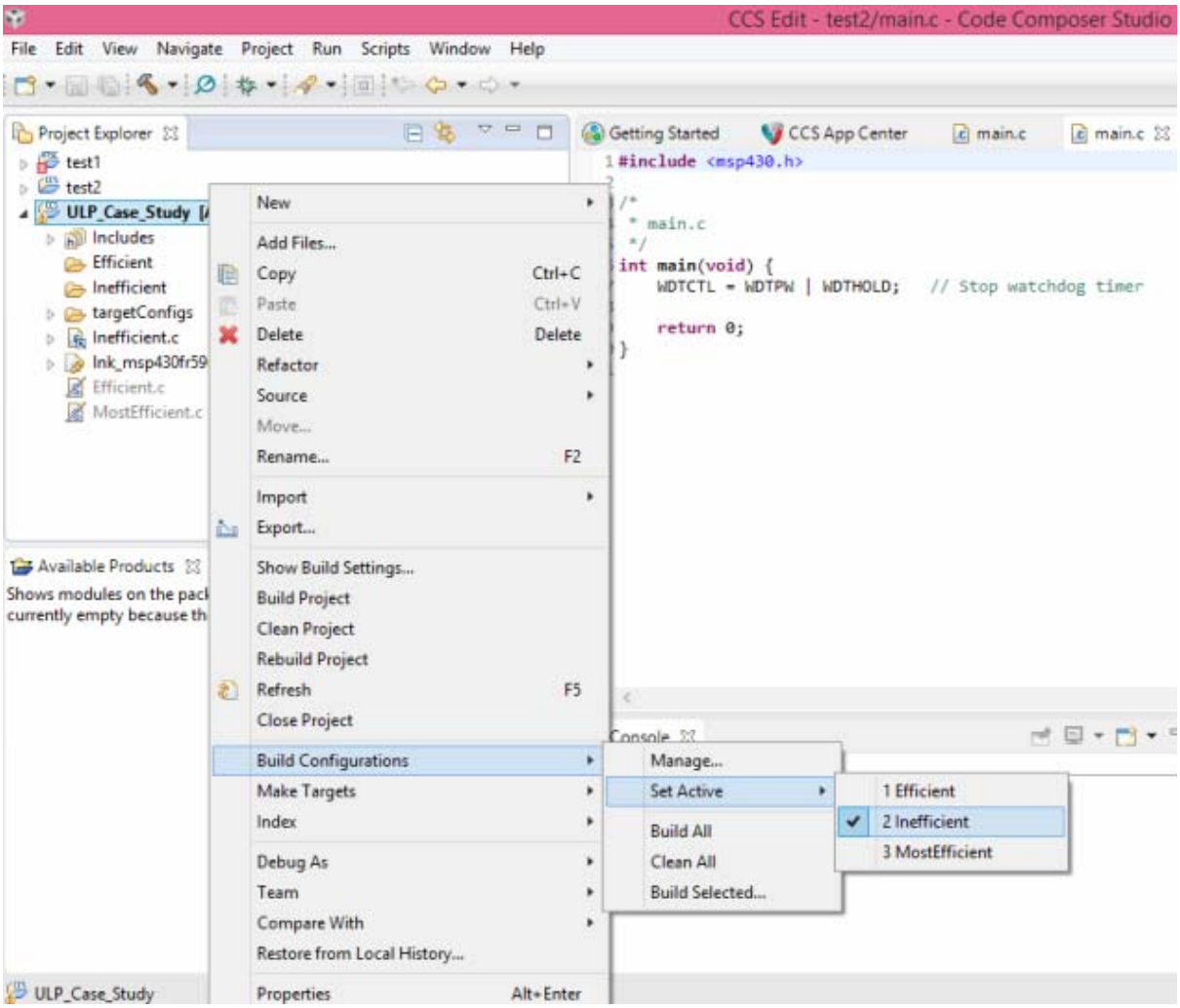


Figure 8-2 Selection of Build Configuration

5. Build the project.

6. The Advice window shows suggestions from the ULP Advisor under the Power (ULP) Advice heading. Click **View** ⟶ **Advice** as shown in **Figure 8-3**.

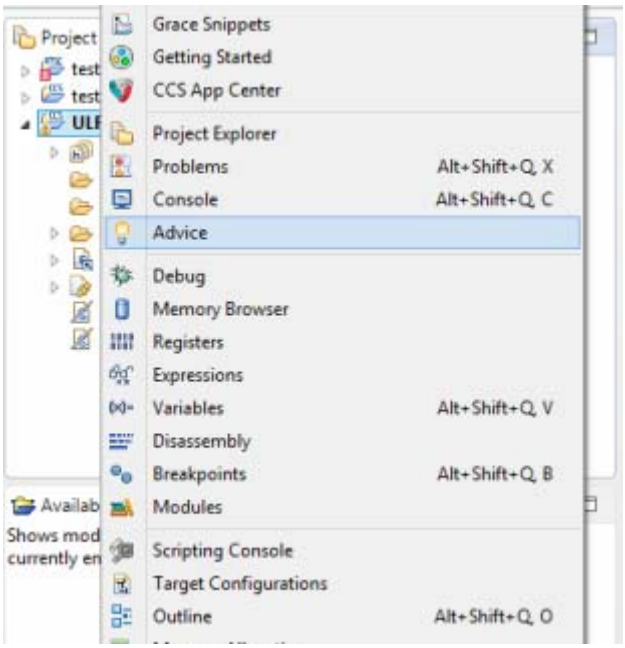


Figure 8-3 Opening the Advice Window

The Advice window appears as shown in **Figure 8-4**.

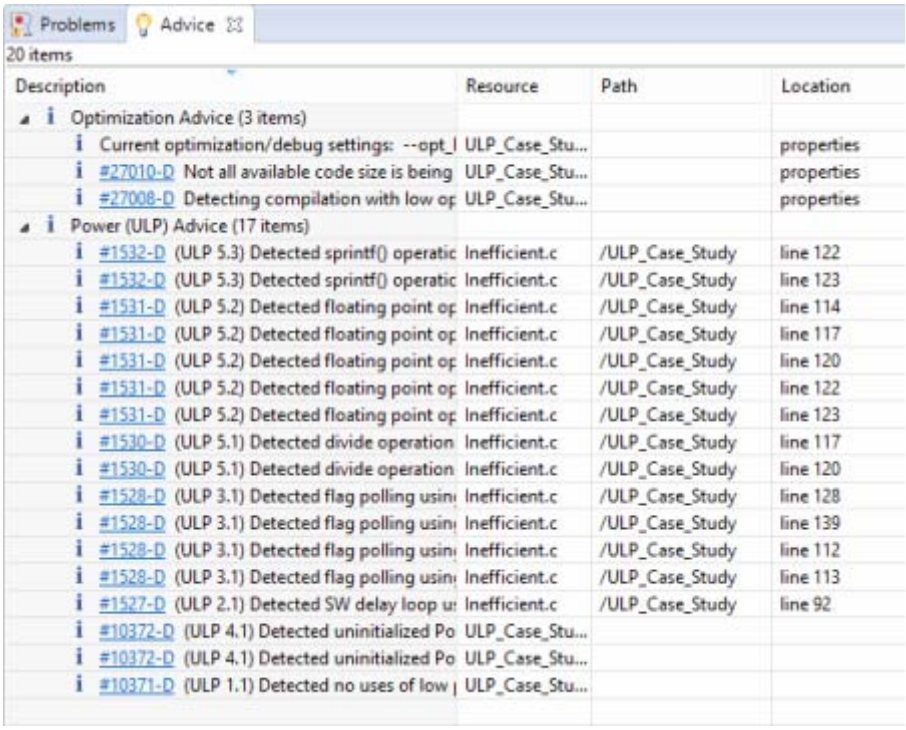


Figure 8-4 Advice Window

7. 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 Advice tab. All the information for this particular rule violation is provided as shown in **Figure 8-5**.

8. Scroll down to the **Remedy** section. Here, the first suggestion is to avoid using sprintf() altogether and work with the raw data.

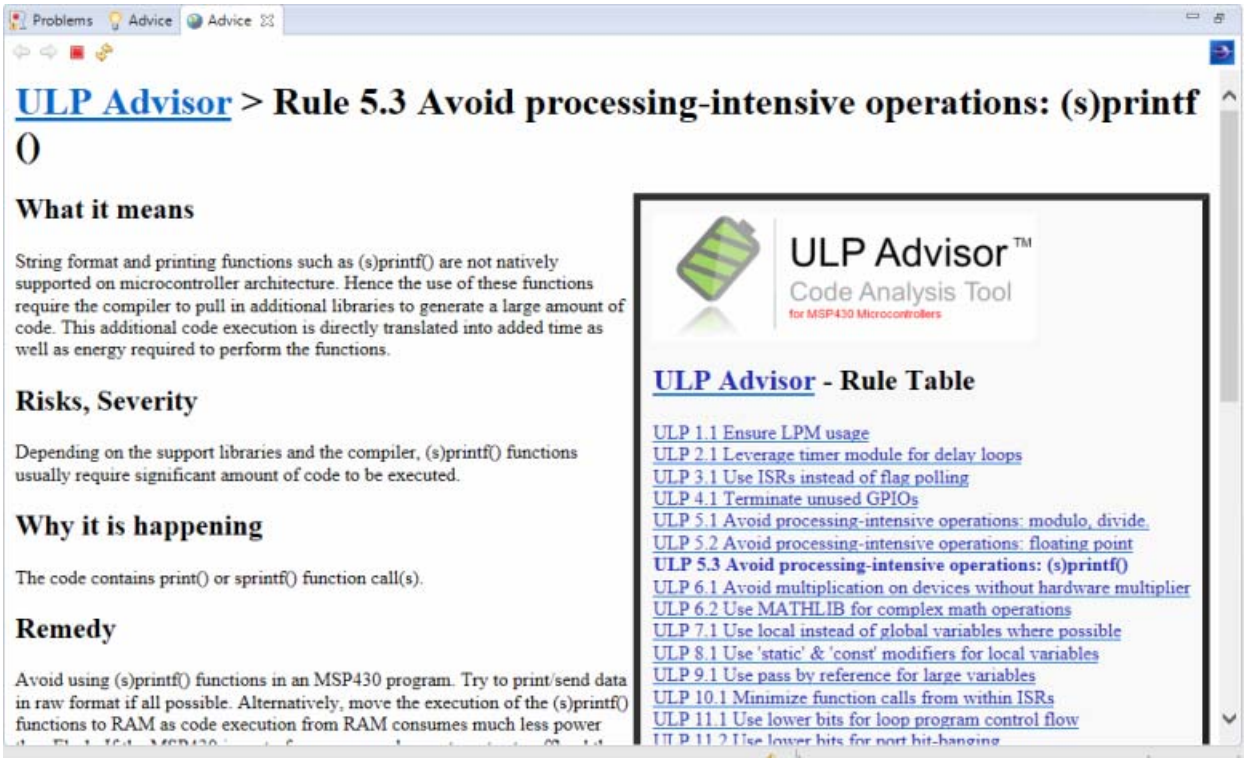


Figure 8-5 ULP Advisor Wiki Page

8.6 Observation

In the ULP Advisor of Inefficient.c, the first suggestion given is to avoid using sprintf(). The ULP Advisor implements the advice for the next version of code i,e., Efficient.c as shown in **Figure 8-6**.

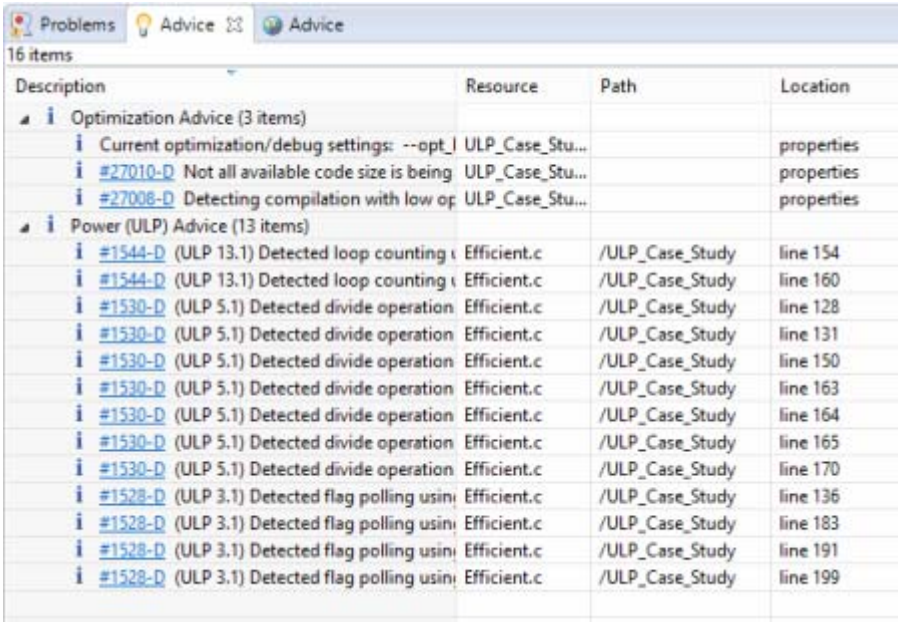


Figure 8-6 ULP Advice Window for Effiecient.c

There are no advices related to Power optimization in ULP Advisor of MostEfficient.c, as shown in **Figure 8-7** below.

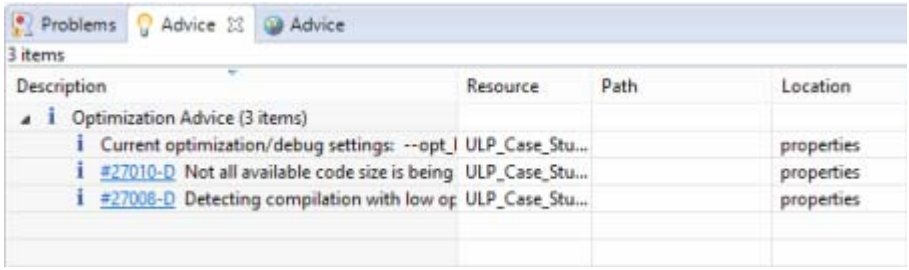


Figure 8-7 ULP Advice Window for MostEffiecient.c

8.7 Summary

In this experiment we have learnt to optimize the power efficiency of an application on MSP-EXP430G2 LaunchPad using ULP Advisor in CCS Studio.

8.8 Exercise

1. How does the ULP Advisor software help in designing power optimized code?
2. Which ULP rule violation helps us to detect a loop counting violation?

Experiment 9

Serial Communication

Topics **Page**

9.1 Objective

The main objective of this experiment is to use UART of the MSP430G2553 to communicate with the computer. This experiment will help to learn and understand the configuration of Universal Serial Communication Interface (USCI) module of MSP430G2553 for UART based serial communication.

9.2 Introduction

Serial communication can be divided into two categories:

- Synchronous serial communication
- Asynchronous serial communication

Synchronous serial communication uses a dedicated communication channel to send a clock signal which provides the timing between the transmitter and receiver. While in asynchronous communication, the timing is encoded within the signal. There are several versions of both synchronous and asynchronous serial communication. The MSP430G2553 supports 3 different types of communications using its USCI peripheral: Serial Peripheral Interface (SPI), Inter-Integrated Circuit Interface (I^2C) and UART.

The MSP430G2553 is an ultra-low power device and has one active mode and five software selectable low-power modes of operation. An interrupt event can wake up the device from any of the low-power modes, service the request, and restore back to the low-power mode on return from the interrupt program.

In UART mode, the USCI transmits and receives characters at a bit rate asynchronous to the paired device. Timing for the transmission of each character is based on the selected baud rate of the USCI. The transmit and receive functions use the same baud rate frequency.

The block diagram for the experiment is as shown in **Figure 9-1**. The UART baud rate is set to 9600. The UART specifies two pins, a TX pin (located at P1.2) and an RX pin (P1.1) for communication. The USCI-UART module is configured to transmit a preset message on an interrupt. An interrupt provided within the program code transmits the message to be displayed on the UART terminal of the computer.

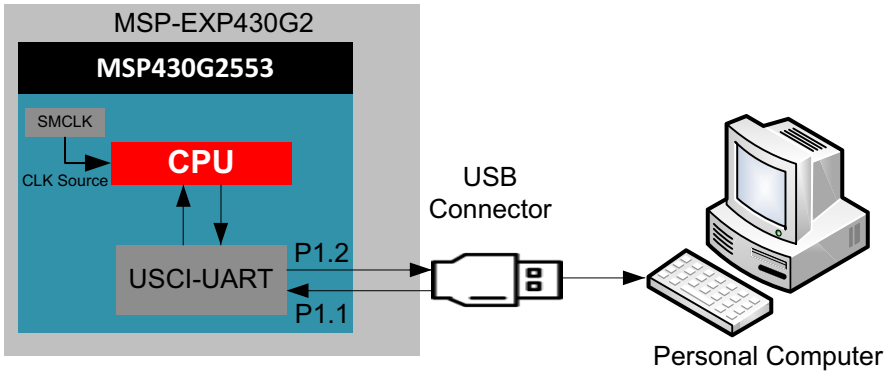


Figure 9-1 Functional Block Diagram

In our program, we configure our device to stay in Low power mode 1 (LPM1) for power optimization until it is woken up by the interrupt. The features of Low power mode 1 (LPM1):

- CPU is disabled
- ACLK and SMCLK remain active, MCLK is disabled
- DCO's DC generator is disabled if DCO not used in active mode

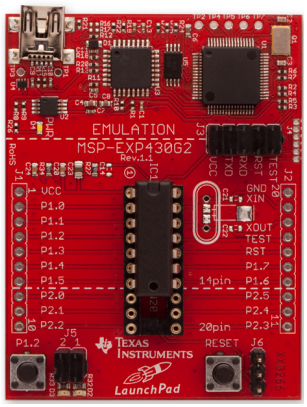
9.3 Component Requirements

9.3.1 Software Requirement

Code Composer Studio

9.3.2 Hardware Requirement

Table 9-1: Components Required for the Experiment

S.No	Components	Specifications	Image
1.	MSP430G2 LaunchPad	**MSP-EXP430G2XL**	
2.	USB cable		

9.4 Software

The software for the experiment is written in C and developed using the CCS Integrated Development Environment (IDE). Refer to "**Project Creation and Build**" in the Getting Started section of this manual for project build and debug using CCS. The software is programmed into the target device MSP430G2553 on the MSP-EXP430G2 LaunchPad using the USB interface.

9.4.1 UART Terminal Setup

1. In CCS window select **View→Other** and from the options given select **Terminal.**

2. When the terminal window opens, click on the **Settings** icon and set the values as shown in **Figure 9-2**.

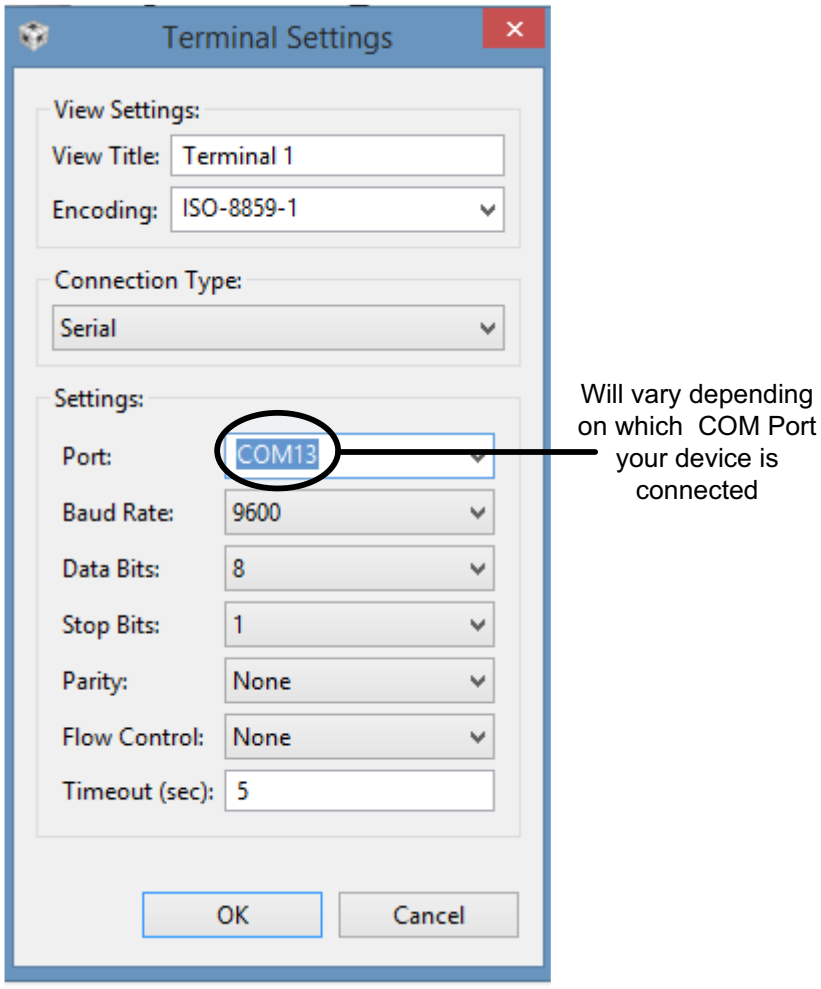


Figure 9-2 Terminal Window Settings

This Terminal will allow you to view the serial information received by the computer and will also allow you to type in the information that you wish to transmit to the MSP430G2553 via the UART.

9.4.2 Flowchart

The flowchart for the code is shown in **Figure 9-3**. The program code first disables the watch dog timer to prevent a restart on expiry of the WDT count. The port pins P1.1 and P1.2 are configured as Rx and Tx of UART respectively. The USCI SMCLK is selected as clock source, and the clock prescaler value in the baud rate control register is set to 104 to have 1MHz frequency and baud rate of 9600. The USCI-UART module is initialized, and the transmit interrupt is enabled. The global interrupt is enabled, and the processor is switched to low power mode 0.

The transmission of the received character is carried out using the USCI0RX interrupt. Interrupt vector addresses and further details about the various involved registers can be found in the Data-sheet.

The interrupt service routine is defined as follows:

```
#pragma vector=USCIAB0RX_VECTOR
__interrupt void USCI0RX_ISR(void)
```

The interrupt service routine for serial transmission transmits the received character from the keyboard which is stored in the receiver buffer UCA0RXBUF to the transmitter buffer UCA0TXBUF.

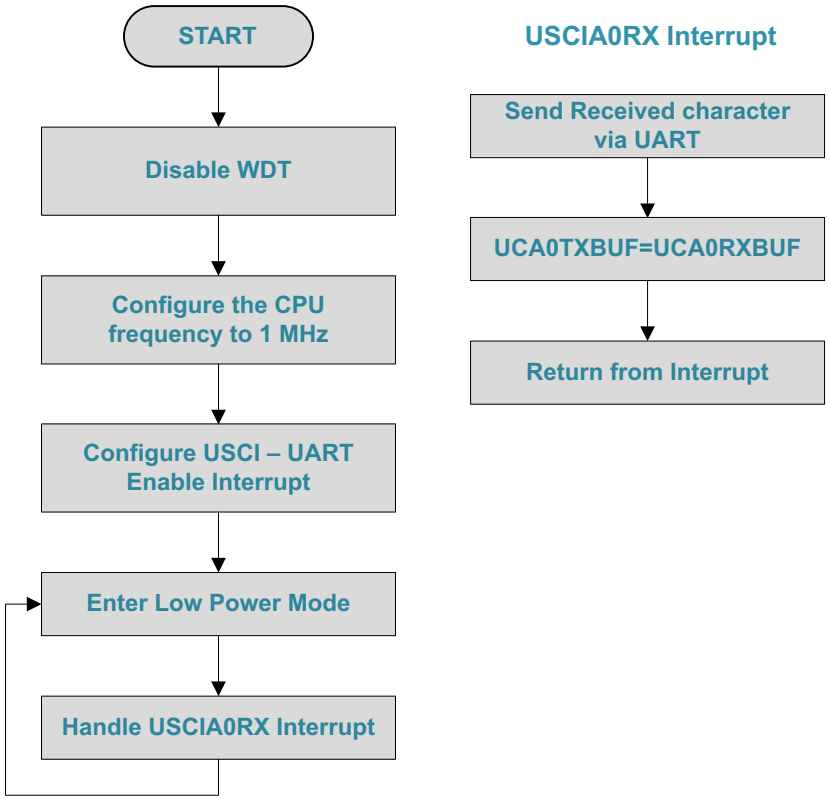


Figure 9-3 Flowchart for Serial Communication Using UART

9.4.3 C Program Code for Serial Communication Using UART

```
#include <msp430.h>
int main(void) {
     WDTCTL = WDTPW | WDTHOLD;     // Stop watchdog timer
     if (CALBC1_1MHZ==0xFF)        // If calibration constant erased
      {
       while(1);                   // do not load, trap CPU!!
      }
      DCOCTL = 0;                  //Select lowest DCOx and MODx settings
     BCSCTL1 = CALBC1_1MHZ;        // Set DCO
     DCOCTL = CALDCO_1MHZ;
     P1SEL = BIT1 + BIT2 ;         // P1.1 = RXD, P1.2=TXD
     P1SEL2 = BIT1 + BIT2 ;        // P1.1 = RXD, P1.2=TXD
     UCA0CTL1 |= UCSSEL_2;         // CLK = SMCLK
```

```
    UCA0BR0 = 104;                // 1MHz/9600 = 104.166
    UCA0BR1 = 0x00;
    UCA0MCTL = UCBRS0;              // Modulation UCBRx = 1
    UCA0CTL1 &= ~UCSWRST;           // **Initialize USCI state machine**
    IE2 |= UCA0RXIE;                // Enable USCI_A0 RX interrupt
  __bis_SR_register(LPM0_bits + GIE);    // Enter LPM0 w/ int
}

#pragma vector=USCIAB0RX_VECTOR
__interrupt void USCI0RX_ISR(void)
{
while (!(IFG2 & UCA0TXIFG));        // USCI_A0 TX buffer ready?
UCA0TXBUF = UCA0RXBUF;
}
```

9.4.4 Registers Used

The registers used are:

- **WDTCTL: 16-bit Watchdog Timer Control Register** - The Watchdog Timer (WDT) is typically used to trigger a system reset after a certain amount of time. The register is set to WDT_MDLY_32 for 32ms interval interrupt.
- **IE2: Interrupt Enable Register 2 -** This register enables the USCI transmit and receive interrupts.Here, the USCI interrupts are enabled.
- **P1SEL1 and P1SEL2: Function Select Register -** Each bit in the function select register is used to select the pin function - I/O port or peripheral module function. Here, the UART function on Pins P1.1 and P1.2 are enabled.
- **UCA0CTL0: USCI_A0 Control Register 0 -** This register configures the USCI module. Here the configuration is: No parity, LSB first, 8-bit data, 1 stop bit, UART, Asynchronous.
- **UCA0CTL1: USCI_A0 Control Register 1** - This register is used to configure clock source and enable software interrupts. Here, SMCLK is used as clock source.
- **UCA0BR0 & UCA0BR1: USCI_A0 Baud Rate Control Register 0 and 1** - These 16-bit registers are configured the clock prescaler setting of the baud rate generator. Here, the registers are configured to have 1MHz frequency and 9600 baud rate.
- **UCA0MCTL: USCI_A0 Modulation Control Register** - This register selects the modulation stages which determine the modulation pattern for BITCLK. Here, the 2nd Stage modulation = 1, and oversampling is turned off.
- **IFG2: Interrupt Flag Register 2** - This register consists of the transmit and receive interrupt flags. Here, the flags are cleared.

9.5 Procedure

1. Connect the MSP-EXP430G2 LaunchPad to the PC using the USB cable supplied.

2. Build, program and debug the code into the LaunchPad using CCS.

9.6 Observation

After compiling and running the program the Serial Terminal displays - **Echo Output:**. As the characters are typed in, they will be displayed on the terminal. The input entered from the keyboard is echoed back and shown on the terminal window. The keyboard entries in Tera Term are generally streamed only to the serial port and not to the display. Only the characters received via the serial port are displayed on this window. To confirm this, hold the RESET key after terminating debug session on the MSP430G2 Launchpad and try typing the input string. You will observe no display of the characters typed. We can thus infer that the program in the MSP-EXP430G2 LaunchPad is doing the echo operation.

On running the program you will be able to view a continuous loop of messages on the UART display as shown in **Figure 9-4**.

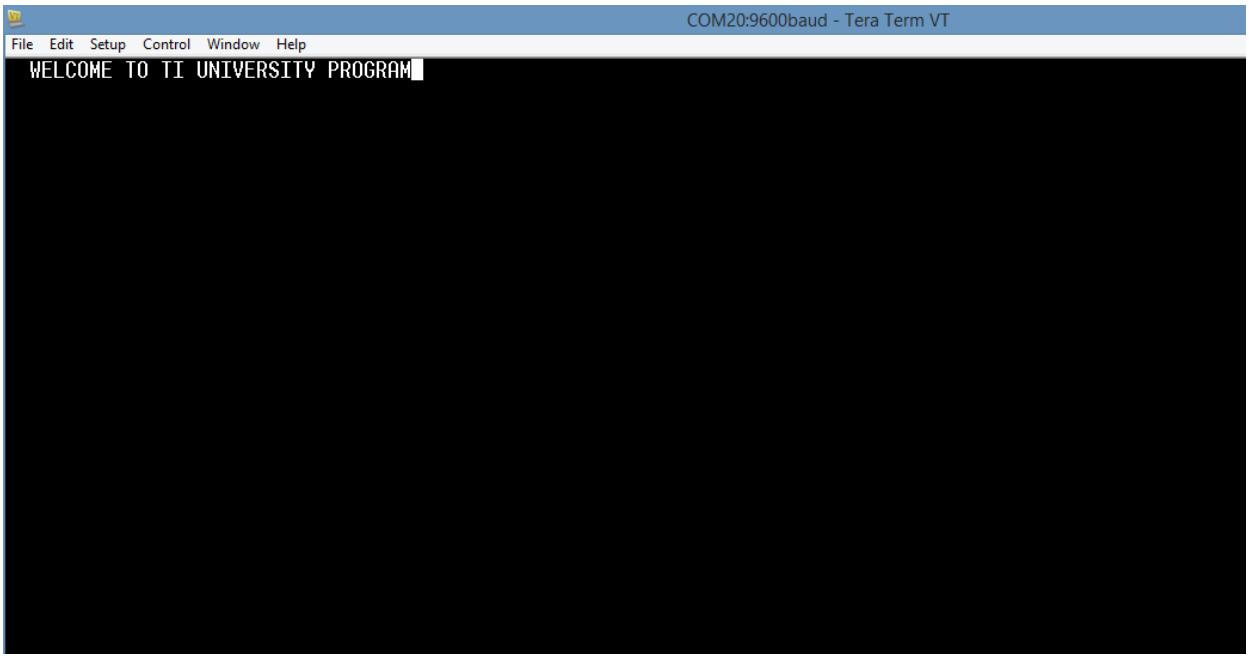


Figure 9-4 UART Display Terminal

Suspend the code and view the various USCI-UART registers.The watch window appears as shown in **Figure 9-5**.

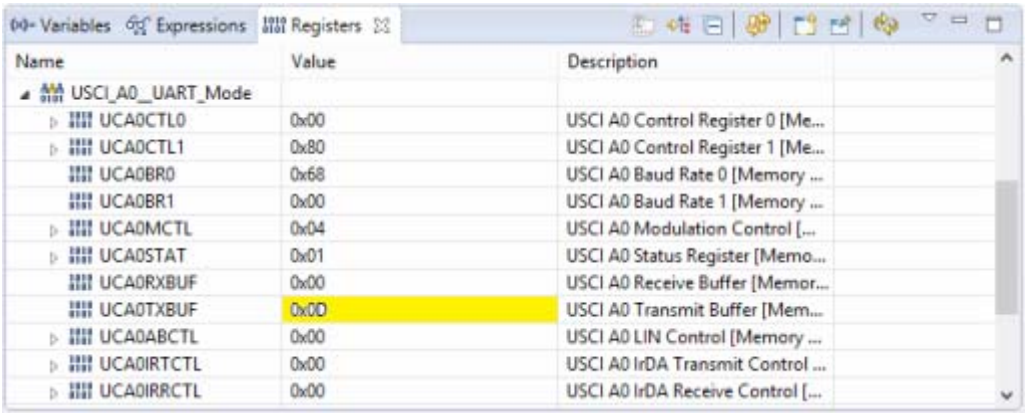


Figure 9-5 USCI UART Registers in CCS Watch Window

9.7 Summary

In this experiment, we have learnt to use UART to communicate with the computer.

9.8 Exercise

1. Modify the above code to transmit the set of strings to the serial terminal via UART as shown below:

char str1[]="MSP430G2 launchpad"

char str2[]= "Ultra low power mixed signal processing applications"

Topics **Page**

10.1 Objective

The main objective of this experiment is to establish the SPI master-slave communication using 3-wire mode in MSP430F5529 Launchpad. This experiment will help understand the configuration of USCI_A0 SPI 3-Wire Master Incremented Data in MSP430F5529.

10.2 Introduction

10.2.1 Universal Serial Communication Interface (USCI)

The MSP430F5529 consists of two Universal Serial Communication Interfaces (USCI) which support multiple serial communication modes as shown below.

The USCI_Ax modules support:

- UART mode
- Pulse shaping for IrDA communications
- Automatic baud rate detection for LIN communications
- Synchronous SPI mode

The USCI_Bx modules support:

- I^2C mode
- Synchronous SPI (Serial Peripheral Interface) mode

In synchronous SPI mode, the USCI connects the MSP430F5529 to an external system via three or four pins as shown in **Figure 10-1**.

- UCxSIMO (Slave In Master Out)
- UCxSOMI (Slave Out Master In)
- UCxCLK USCI SPI clock
- UCxSTE Slave Transmit Enable

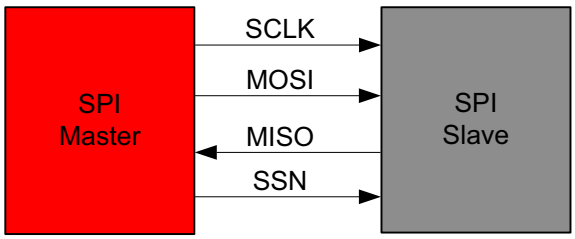


Figure 10-1 Synchronous SPI Mode of USCI

SPI mode is selected when the UCSYNC bit is set and SPI mode (3-pin or 4-pin) is selected with the UCMODEx bits.

The features of MSP430F5529 SPI are:

- 7 or 8-bit data length
- LSB-first or MSB-first data transmit and receive
- 3-pin and 4-pin SPI operation
- Master or slave modes

- Independent transmit and receive shift registers
- Separate transmit and receive buffer registers
- Continuous transmit and receive operation
- Selectable clock polarity and phase control
- Programmable clock frequency in master mode
- Independent interrupt capability for receive and transmit
- Slave operation in LPM4

The block diagram for the experiment is as shown in **Figure 10-2**. In this experiment the SPI master talks to SPI slave using 3-wire mode. The USCI_A0 module is configured as a SPI interface. Master sends the data starting at 0x01 to the slave mode. The slave will receive the data which is expected to be same as the previous data transmission.

In the master, USCI_RX ISR is used to handle communication with the CPU, normally in LPM0. In the slave, USCI_ RX ISR is used to handle communication with the CPU, normally in LPM4.

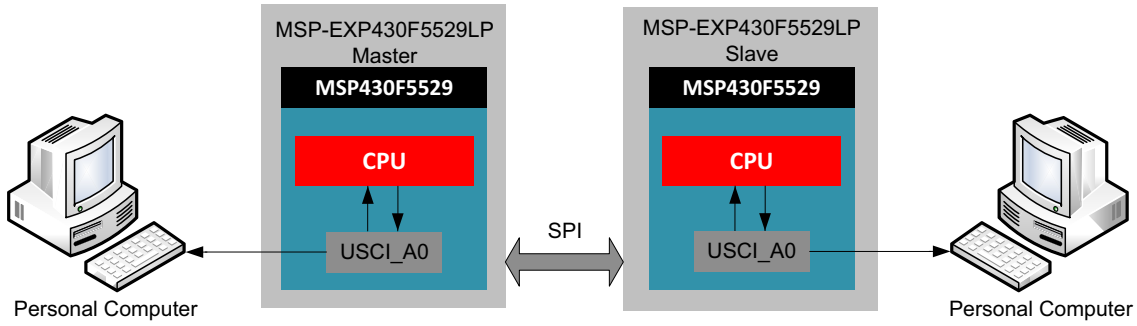


Figure 10-2 Functional Block Diagram

The master-slave connection for SPI communication is shown in **Figure 10-3**.

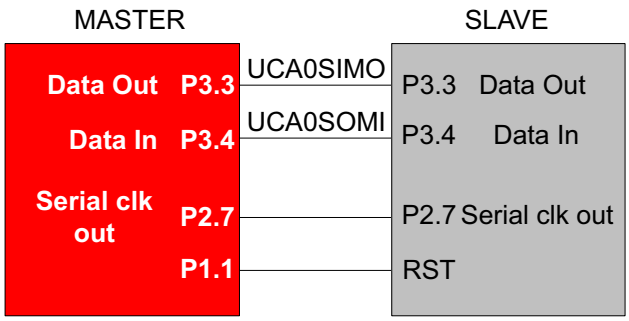


Figure 10-3 Connection Diagram for Master Slave Communication Using SPI Protocol

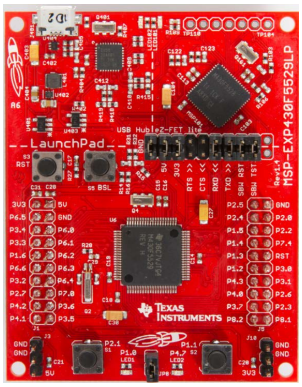
10.3 Component Requirements

10.3.1 Software Requirement

Code Composer Studio

10.3.2 Hardware Requirement

Table 10-1: Components Required for the Experiment

S.No	Components	Specifications	Image
1.	2 MSP430F5529 LaunchPads	MSP-EXP430F5529LP	
2.	USB cable		

10.4 Software

The software for the experiment is written in C and developed using the CCS Integrated Development Environment (IDE). Refer to "**Project Creation and Build**" in the Getting Started section of this manual for project build and debug using CCS. The software is programmed into the target device MSP430F5529 on the MSP-EXP430F5529 using the USB interface.

10.4.1 Flowchart

Master:

The flowchart for the code of the master is shown in **Figure 10-4**. The program code first disables the watch dog timer to prevent a processor restart on expiry of the WDT count. The port pin P1.1 is set as slave reset and port pin P1.0 is configured for the red LED. The port pins P3.3, P3.4 and P2.7 are configured for SPI communication. The USCI logic is held in reset state and the 3-pin, 8-bit synchronous SPI master is enabled. The SMCLK is selected as clock source and the USCI logic initiated. The USCI_RX interrupt is enabled in low power mode and the slave is reset via output pin P1.1 and allowed to initialize. The data values to be transmitted to the slave is also initialized and transmitted via the transmit buffer UCA0TXBUF. On receiving data from the slave, the USCI_RX interrupt occurs. If the data received is same as the data transmitted, the red LED is turned on and the data transmission is continued.

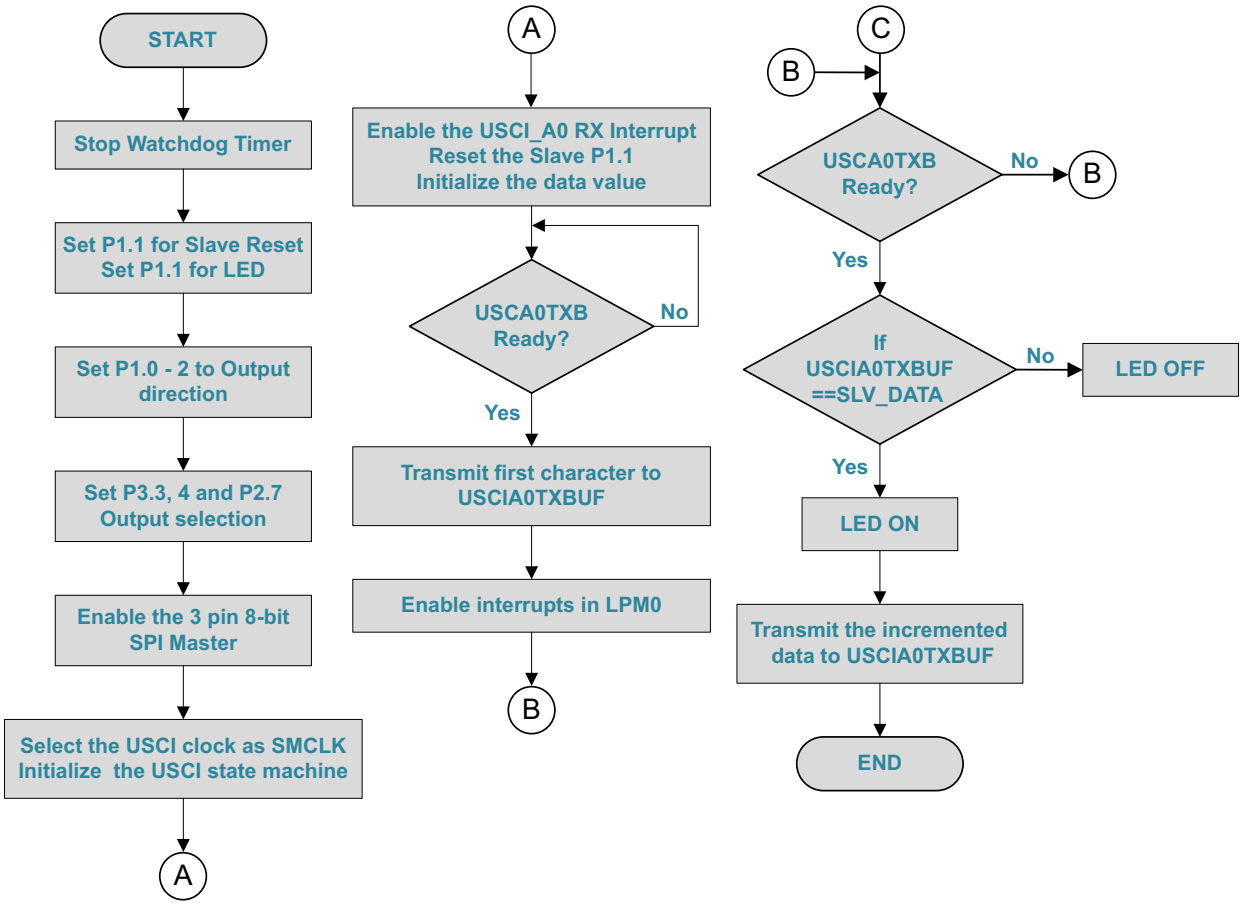


Figure 10-4 Flowchart for the Code of Master

Slave:

The flow chart for the code of the slave is shown in **Figure 10-5**. The program code first disables the watch dog timer to prevent a processor restart on expiry of the WDT count. Since, the connection is synchronous, the slave receives the clock source from the master on port pin P2.7. If a clock signal is detected from the master, the port pins P3.3, P3.4 and P2.7 are configured for SPI communication and the USCI logic is held in reset state. The 3-pin, 8-bit synchronous SPI master is then enabled. The USCI logic is initiated and the USCI Rx interrupt is enabled in low power mode 4. When the slave receives data from the master, the USCI Rx interrupt occurs. The received data is copied into the transmit buffer UCA0TXBUF for echo transmission to the master.

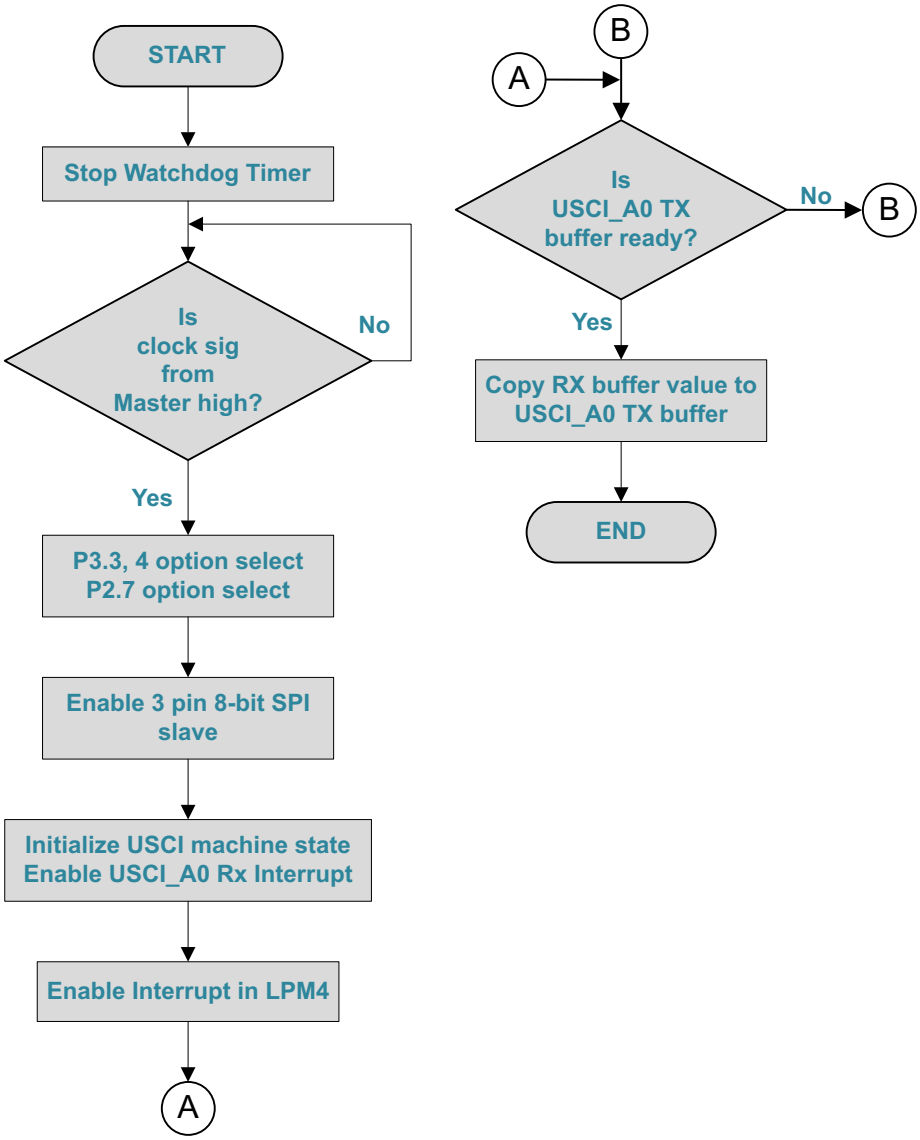


Figure 10-5 Flowchart for the Code of Slave

10.4.2 C Program Code for Master and Slave

Master:

```
#include <msp430.h>
unsigned char MST_Data,SLV_Data;
unsigned char temp;

int main(void)
{
  volatile unsigned int i;
```

```
  WDTCTL = WDTPW+WDTHOLD;          // Stop watchdog timer

  P1OUT |= 0x02;                   // Set P1.0 for LED
                                   // Set P1.1 for slavereset
  P1DIR |= 0x03;                   // Set P1.0-2 to output direction
 * P3SEL |= BIT3+BIT4;             // P3.3,4 option select
  P2SEL |= BIT7;                   // P2.7 option select

  UCA0CTL1 |= UCSWRST;             // **Put state machine in reset**
  UCA0CTL0 |= UCMST+UCSYNC+UCCKPL+UCMSB;
                                  // 3-pin, 8-bit SPI synchronous master
                                   // Clock polarity high, MSB
  UCA0CTL1 |= UCSSEL_2;            // SMCLK
  UCA0BR0 = 0x02;                  // /2
  UCA0BR1 = 0;
  UCA0MCTL = 0;                    // No modulation
  UCA0CTL1 &= ~UCSWRST;            // **Initialize USCI state machine**
  UCA0IE |= UCRXIE;                // Enable USCI_A0 RX interrupt
  P1OUT &= ~0x02;                  // Now with SPI signals initialized,
  P1OUT |= 0x02;                   // reset slave
  for(i=50;i>0;i--);               // Wait for slave to initialize
`  MST_Data = 0x01;                // Initialize data values
  SLV_Data = 0x00;

  while (!(UCA0IFG&UCTXIFG));      // USCI_A0 TX buffer ready?
  UCA0TXBUF = MST_Data;            // Transmit first character
  __bis_SR_register(LPM0_bits + GIE);       // CPU off, enable interrupts
}

#if defined(__TI_COMPILER_VERSION__) || defined(__IAR_SYSTEMS_ICC__)
#pragma vector=USCI_A0_VECTOR
__interrupt void USCI_A0_ISR(void)
#elif defined(__GNUC__)
void __attribute__ ((interrupt(USCI_A0_VECTOR))) USCI_A0_ISR (void)
```

```
#else
#error Compiler not supported!
#endif
{
  volatile unsigned int i;
  switch(__even_in_range(UCA0IV,4))
  {
    case 0: break;                  // Vector 0 - no interrupt
    case 2:                         // Vector 2 - RXIFG
      while (!(UCA0IFG&UCTXIFG));   // USCI_A0 TX buffer ready?

      if (UCA0RXBUF==SLV_Data)      // Test for correct character RX'd
        P1OUT |= 0x01;              // If correct, light LED
      else
        P1OUT &= ~0x01;             // If incorrect, clear LED

      MST_Data++;                   // Increment data
      SLV_Data++;
      UCA0TXBUF = MST_Data;         // Send next value

     for(i = 20; i>0; i--);         // Add time between transmissions to
                                    // make sure slave can process
                                    //information
      break;
    case 4: break;                  // Vector 4 - TXIFG
    default: break;
  }
}
```

Slave:

```
#include <msp430.h>

int main(void)
{
  WDTCTL = WDTPW+WDTHOLD;           // Stop watchdog timer

  while(!(P2IN&0x80));              // If clock sig from mstr stays low,
```

```
                                        // it is not yet in SPI mode
  P3SEL |= BIT3+BIT4;                   // P3.3,4 option select
  P2SEL |= BIT7;                        // P2.7 option select
  UCA0CTL1 |= UCSWRST;                  // **Put state machine in reset**
  UCA0CTL0 |= UCSYNC+UCCKPL+UCMSB;      // 3-pin, 8-bit SPI slave,
                                        // Clock polarity high, MSB
  UCA0CTL1 &= ~UCSWRST;                 // **Initialize USCI state machine**
  UCA0IE |= UCRXIE;                     // Enable USCI_A0 RX interrupt

  __bis_SR_register(LPM4_bits + GIE); // Enter LPM4, enable interrupts
}

// Echo character

#if defined(__TI_COMPILER_VERSION__) || defined(__IAR_SYSTEMS_ICC__)
#pragma vector=USCI_A0_VECTOR
__interrupt void USCI_A0_ISR(void)
#elif defined(__GNUC__)
void __attribute__ ((interrupt(USCI_A0_VECTOR))) USCI_A0_ISR (void)
#else
#error Compiler not supported!
#endif

{
  switch(__even_in_range(UCA0IV,4))
  {
    case 0:break;                       // Vector 0 - no interrupt
    case 2:                             // Vector 2 - RXIFG
      while (!(UCA0IFG&UCTXIFG));       // USCI_A0 TX buffer ready?
      UCA0TXBUF = UCA0RXBUF;
      break;
    case 4:break;                       // Vector 4 - TXIFG
    default: break;
  }
}
```

10.4.3 Registers Used

The Registers used in our program are:

- **WDTCTL: 16-bit Watchdog Timer Control Register** - The Watchdog Timer (WDT) is typically used to trigger a system reset after a certain amount of time. The register is set to WDT_MDLY_32 for 32ms interval interrupt.
- **P3SEL and P2SEL: Function Select Register -** Each bit in the function select register is used to select the pin function - I/O port or peripheral module function. Here, the pins P3.3, P3.4 and P2.7 are configured for SPI communication.
- **UCA0CTL0: USCI_A0 Control Register 0 -** This register configures the USCI module. Here the master and slave are configured for 3-bit synchronous SPI.
- **UCA0CTL1: USCI_A0 Control Register 1** - This register is used to configure clock source and enable software interrupts. The master uses SMCLK as clock source.
- **UCA0BR0 & UCA0BR1: USCI_A0 Baud Rate Control Register 0 and 1** - These 16-bit registers configure the clock prescaler setting of the bit rate. In the master, the registers are configured to have clock prescaler of 2.
- **UCA0IE**: **USCI_A0 Interrupt Enable Register -** This register enables the USCI transmit and receive interrupts.Here, the USC RxI interrupt is enabled.
- **UCA0TXBUF: USCI Transmit Buffer Register -** This register contains the data to be transmitted by the USCI module.
- **UCA0RXBUF: USCI Receive Buffer Register -** This register contains the data received by the USCI module.
- **UCA0IFG: USCI_A0 Interrupt Flag Register -** This flag is set when UCA0TXBUF is empty.

The various configuration parameters related to the various registers of the USCI_A0 used in the current program are summarized in **Table 10-2**.

Table 10-2: Configuration Parameters for Registers of USCI_A0

Parameter	Denoted by	Location	Comments
Software reset enable	UCSWRST	UCA0CTL1, bit 0	0 - Disabled. USCI reset released for operation. 1 - Enabled. USCI logic held in reset state.
Master mode select	UCMST	UCA0CTL0, bit 3	0 - Slave Mode 1 - Master Mode
Synchronous mode enable	UCSYNC	UCA0CTL0, bit 0	0 - Asynchronous mode 1 - Synchronous mode
Clock polarity select	UCCKPL	UCA0CTL0, bit 6	0 - The inactive state is low 1 - The inactive state is high
Controls the direction of the receive and transmit shift register	UCMSB	UCA0CTL0, bit 5	0 - LSB first 1 - MSB first

Table 10-2: Configuration Parameters for Registers of USCI_A0

Parameter	Denoted by	Location	Comments
USCI clock source select	UCSSEL_2	UCA0CTL1, bits 7-6	10 SMCLK is selected
Receive interrupt enable	UCRXIE	UCA01E, bit 0	0 - Interrupt disabled 1 - Interrupt enabled
Transmit interrupt flag	UCTXIFG	UCA01FG, bit 1	0 - No interrupt pending 1 - No interrupt pending

10.5 Procedure

1. Connect the MSP430 LaunchPads to the PC using the USB cable supplied.

2. Build, program and debug the code into the LaunchPad using CCS.

10.6 Observation

A successful SPI communication has been established between 2 MSP430 devices. This fact is verified by the blinking red LED in the master after receiving the exact echo data from the slave that the master had previously transmitted.

10.7 Summary

In this experiment, we have learnt to use the SPI communication to establish a connection between two MSP430 devices. We have also learnt to configure the USCI_A0 SPI 3-Wire Master Incremented Data in MSP430F5529.

10.8 Exercise

1. Which port pins of MSP430 can be configured for SPI communication?

2. What is the data transfer rate supported by MSP430 for SPI communication?

Experiment 11

Wi-Fi Email Application

Topics **Page**

11.1 Objective

The main objective of this experiment is to configure CC3100 Booster Pack as a Wireless Local Area Network (WLAN) Station to send Email over SMTP. This experiment will help you understand the WLAN concepts and CC3100 configuration to send Email over SMTP.

11.2 Introduction

A wireless local area network (WLAN) is a wireless computer network that connects two or more devices without wires within a confined area, for example: within a building. This stay connected without physical wiring constraints and also access Internet. WiFi is based on IEEE 802.11 standards including IEEE 802.11a and IEEE802.11b.

All nodes that connect over a wireless network are referred to as stations (STA). Wireless stations can be categorized into Wireless Access Points (AP) or clients. Access Points (AP) work as the base station for a wireless network. The Wireless clients could be any device such as computers, laptops, mobile devices, smart phones, etc.

The Simplelink WIFI CC3100 device is the industry's first Wi-Fi Certified chip used in the wireless networking solution that simplifies the implementation of Internet connectivity. This device supports WPA2 personal and enterprise security and WPS 2.0 and Embedded TCP/IP and TLS/SSL stacks, HTTP server, and multiple Internet protocols. The functional block diagram for the CC3100 is shown in **Figure 11-1**.

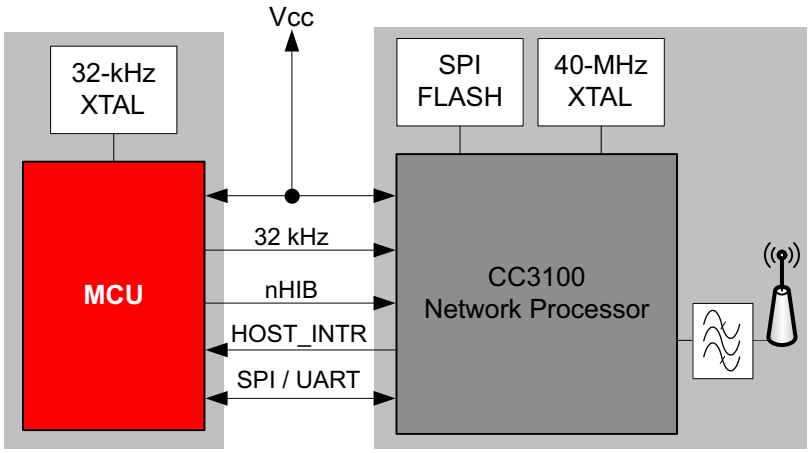


Figure 11-1 Functional block diagram of CC3100

The software overview of the Simplelink WiFi CC3100 is shown in **Figure 11-2**. The SimpleLink Host Driver includes a set of six simple Application Program Interface (API) modules:

- Device API – Manages hardware-related functionality such as start, stop, set, and get device configurations.
- WLAN API – Manages WLAN, 802.11 protocol-related functionality such as device mode (station, AP, or P2P), setting provisioning method, adding connection profiles, and setting connection policy.
- Socket API – The most common API set for user applications, and adheres to BSD socket APIs.
- NetApp API – Enables different networking services including the Hypertext Transfer Protocol (HTTP) server service, DHCP server service, and MDNS client\server service.

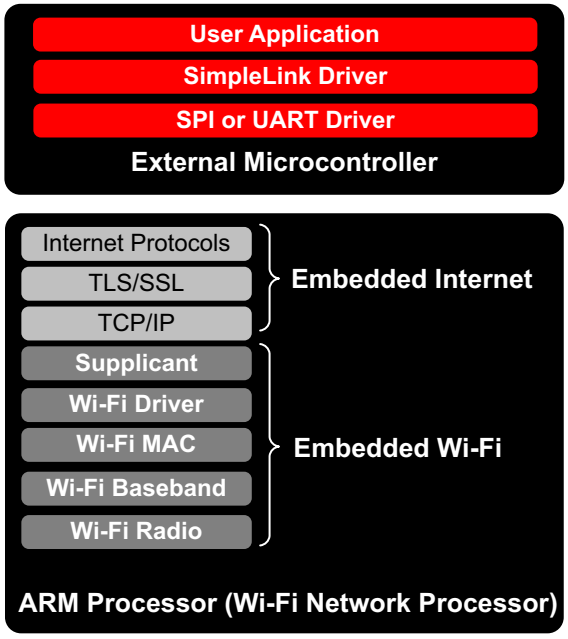


Figure 11-2 CC3100 Software Overview

- NetCfg API – Configures different networking parameters, such as setting the MAC address, acquiring the IP address by DHCP, and setting the static IP address.
- File System API – Provides access to the serial flash component for read and write operations of networking or user proprietary data.

The CC3100BOOST and the MSP430F5529 are connected via the SPI interface as shown in **Figure 11-3**.

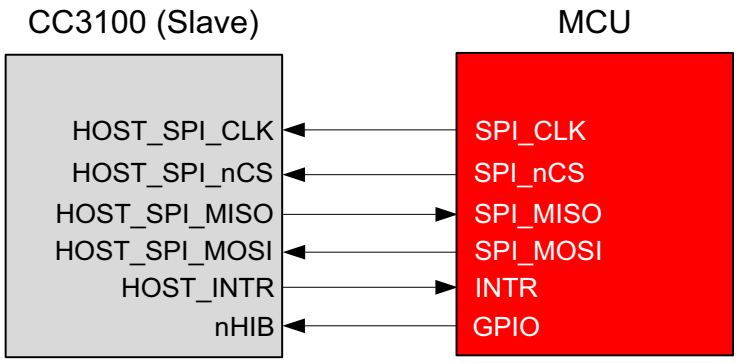


Figure 11-3 Connection Diagram for CC3100BOOST and MSP430F5529

The block diagram of Email application using CC3100 is as shown in **Figure 11-4**. This experiment demonstrates the configuration of CC3100 as AP to send an email over SMTP (Simple Mail Transfer Protocol). The application configures CC3100 to connect with an SMTP server and sends email to it. The SMTP server forwards it to the recipient's email-server and the recipient receives the email from his email-server using IMAP/POP3 and/or other proprietary protocol.

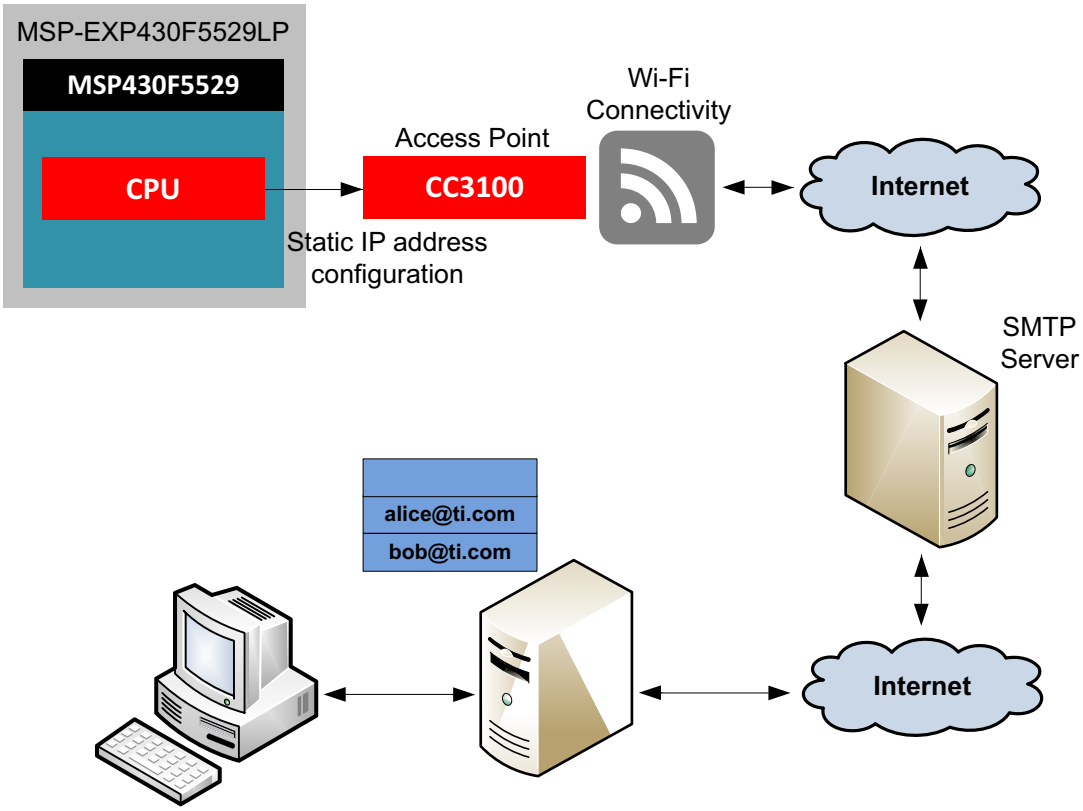


Figure 11-4 Functional Block Diagram

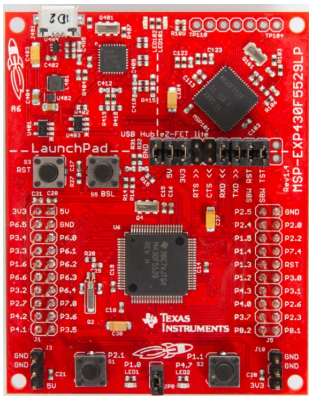
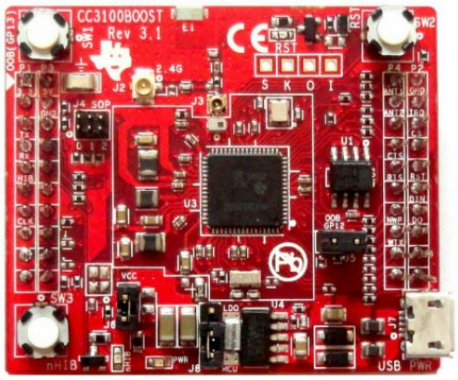
11.3 Component Requirements

11.3.1 Software Requirement

1. Code Composer Studio
2. CC3100 SDK
3. Tera Term Software (or any Serial Terminal Software)

11.3.2 Hardware Requirement

Table 11-1: Components Required for the Experiment

S.No	Components	Specifications	Image
1.	MSP430F5529 LaunchPad	**MSP-EXP430F5529LP**	
2.	USB cable		
3.	CC3100 Booster Pack	**CC3100BOOST**	
4.	Wireless Network connection with its name, security type and password with Internet connectivity		

11.4 Software

The software for the experiment is written in C and developed using the CCS Integrated Development Environment (IDE). Refer to "**Project Creation and Build**" in the Getting Started section of this manual for project build and debug using CCS. The software is programmed into the target device MSP430F5529 on the MSP-EXP430F5529LP using the USB interface.

11.4.1 Flowchart

The flowchart for the experiment is as shown in **Figure 11-5**. The CC3100 BoosterPack is powered up in the default state. The CC3100 is started as a station and a static IP address is configured to the device. On successful configuration of an IP address, the device is connected to the access point, and communication is established. Then, it checks for LAN and Internet connectivity. Once the connection is established, an email is sent through the SMTP.

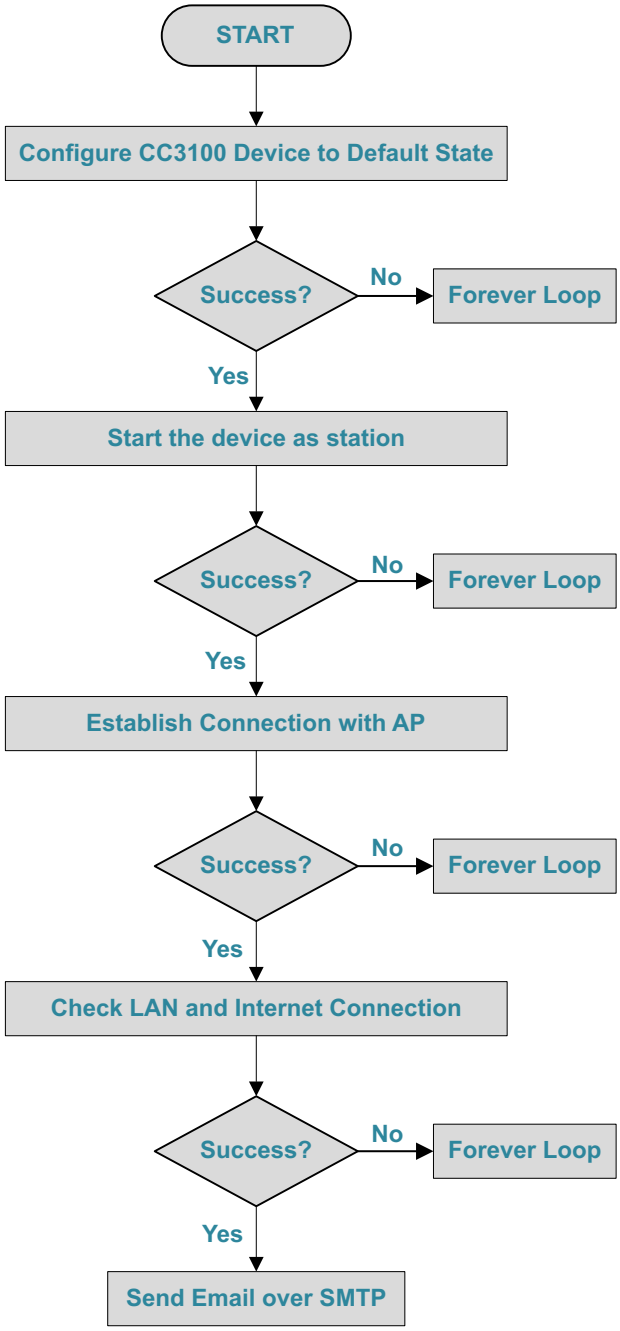


Figure 11-5 Flowchart to Configure CC3100 as LAN Station to Send Email over SMTP

11.4.2 Steps for Creating and Debugging the Project

1. Install CC3100 SDK.

2. Open CCS and create a new workspace.

3. Choose the Import Project link on the TI Resource Explorer page. Import "email_application" project from CC3100 SDK using the following steps:

a. Choose the **Import Project** link on the TI Resource Explorer page.

b. Select the **Browse** button in the Import CCS Eclipse Projects dialog.

c. Select the *email_application* within

C:\TI\CC3100SDK_1.1.0\cc3100-sdk\platform\msp430f5529lp\example_project_ccs\email_application

4. Open sl_common.h(line 50 of main.c) and change SSID_NAME, SEC_TYPE and PASSKEY as per the properties of your Access Point(AP). SimpleLink device will connect to this AP when the application is executed.

a. Expand the imported project

b. Click on includes.

c. Click on ***C:ti/CC3100SDK_1.1.0/cc3100sdk/examples/common***

d. Open sl_common.h

```
#include "simplelink.h"
#include "sl_common.h"                          << Line 50
```

5. Replace with SSID of your Access Point.

```
#define SSID_NAME  "XXXXXXXX"    /* Security type of the Access point */
```

6. Specify Security Type and Password as below if network is protected with security.

```
#define SEC_TYPE   SL_SEC_TYPE_WPA_WPA2 /*Security type of the Access point */
```

7. If network is protected with security, replace passkey with your network password.

```
#define PASSKEY       "XXXXXXXXXX"   /* Password in case of secure AP */
```

8. If the Security is open, replace the definitions as below:

```
#define SEC_TYPE   SL_SEC_TYPE_OPEN /*Security type of the Access point*/
```

9. If the Network has no password

```
#define PASSKEY       ""            /* No password for open type */
```

10. Open config.h and change values for USER and PASS for setting up the source email using the following steps:

a. Replace the username of your source email id

```
#define USER       "<username>"
```

Write the password of your email id

```
#define PASS       "<password>"
```

11. Edit the same config.h file and change the values of DESTINATION_EMAIL, EMAIL_SUBJECT and EMAIL_MESSAGE for setting up the email properties using the following steps:

a. Replace your destination email id

```
#define DESTINATION_EMAIL       "<destination_email>"
```

b. Write the subject of your mail

```
#define EMAIL_SUBJECT           "<email_subject>"
```

c. Write your email message

```
#define EMAIL_MESSAGE            "<email_body>"
```

12. Save, Debug and Run

11.5 Procedure

1. Connect MSP430F5529 Launchpad and CC3100 BoosterPack as shown below in **Figure 11-6**.

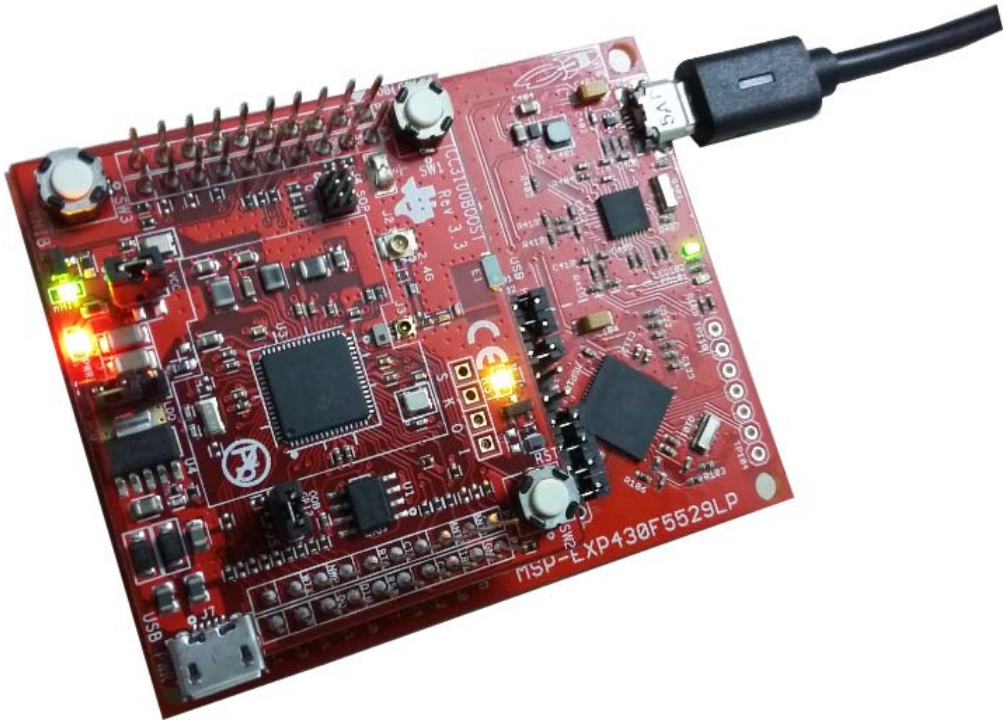


Figure 11-6 Hardware Setup

2. Build, program and debug the code into the LaunchPad using CCS.

11.6 Observation

Open Tera Term Terminal Software on the PC where the MSP-EXP430 LaunchPad is connected. The serial port parameters to be set are 9600 baud rate, 8 bits, No parity and 1 stop bit.The serial window outputs the debug messages received from the MSP-EXP430 Hardware as shown in **Figure 11-7**.

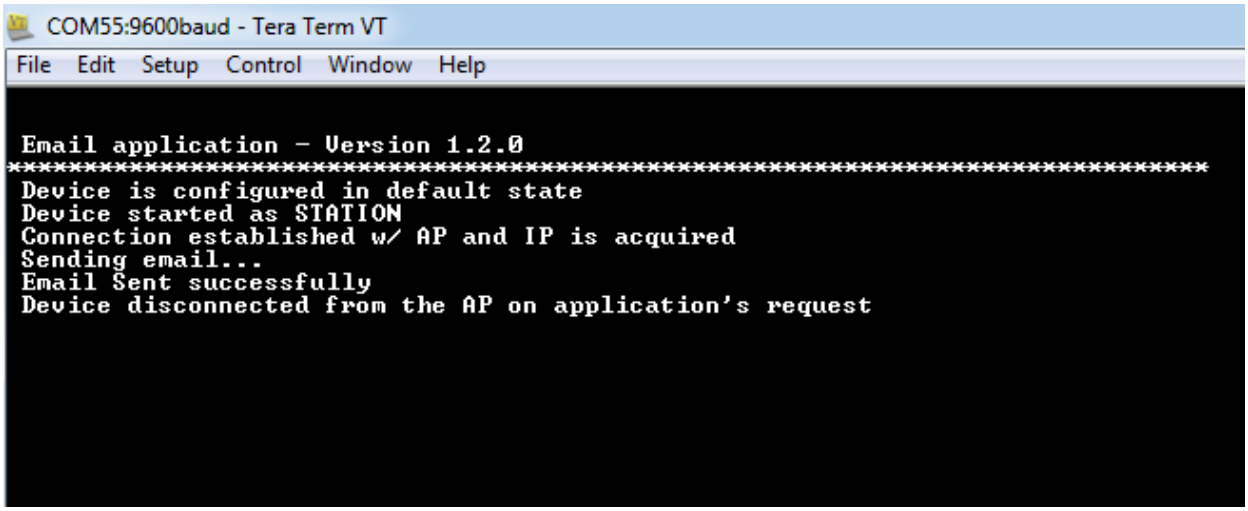


Figure 11-7 Debug Messages on Terminal Window for Sending Email

11.7 Summary

The CC3100 device is configured as a station and successfully connected to Internet to send an email over the SMTP server.

11.8 Exercise

1. In the terminal output window, we have received a debug message “Pinging…!”. Search in the code and change the message to “Pinging the website”. Repeat the experiment to observe this change in the Serial Window.

2. Design an experimental set up to communicate between two MSP430 based sensor nodes using Peer-to-peer communication.

Experiment 12

Energy Trace and Energy Trace++ Modes

Topics **Page**

12.1 Objective

The main objective of this experiment is to enable Energy Trace and Energy Trace++ modes in MSP-EXP430G2 LaunchPad by using MSP430FR5969. This experiment will help you learn how to analyze the Energy and Power graphs by enabling the Energy Trace Technology of MSP430 in CCS studio.

12.2 Introduction

12.2.1 Energy Trace

Energy Trace 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.

Integration of Energy Trace Technology in the development process:

- Real-time energy/power measurements
- Device-internal state analysis
- Requires specialized debugger circuitry
- Integrated visualization tool in IDE
- Performs at run time
- CCS v6.0 IAR 5.40.7 and newer

Energy Trace 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.

Energy Trace 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 Energy Trace is in use.

The two modes of Energy Capture are:

Energy Trace:

This mode allows the standalone use of the energy measurement feature with all MSP430 microcontrollers, even unsupported devices (meaning there is no built-in Energy Trace++ 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.

Energy Trace++:

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 as given in **Table 12-1**.

Table 12-1: Microcontratoller state for Various Energy Trace++ modes

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

This experiment explains in detail how to use Energy Trace technology on MSP-EXP430G2 LaunchPad. For demonstration purpose, the code from Experiment 7: PWM Based Speed Control of Motor by Potentiometer is used.

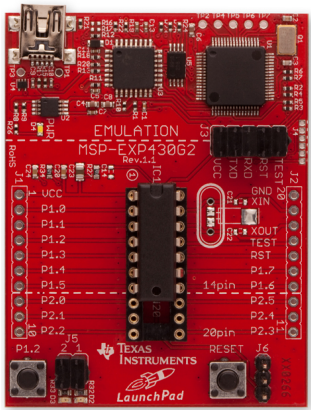
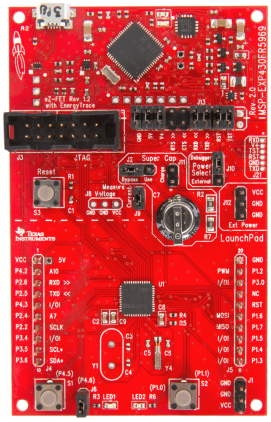
12.3 Component Requirements

12.3.1 Software Requirement

Code Composer Studio

12.3.2 Hardware Requirement

Table 12-2: Components Required for the Experiment

S.No	Components	Specifications	Image
1.	MSP430G2 LaunchPad	**MSP-EXP430G2XL**	
2.	MSP430FR5969 Launchpad	**MSP-EXP430FR5969**	
3.	USB cable	For MSP430FR5969LP	

12.4 Software

Any MSP430 evaluation module (EVM), experimenter's board (EXP), target socket module (TS), or application board can utilize the Energy Trace component despite the fact that the Energy Trace++ on-board circuitry is not present. To do this, the on-board emulation of a LaunchPad that contains the Energy Trace technology circuitry is used to program and debug the intended target. In this example, the MSPEXP430FR5969 which is having a Energy Trace technology on board circuitry is used as experimenter board and the MSP430G2553 is used as a target module to achieve Energy Trace Technology.

12.5 Procedure

12.5.1 Hardware Connection

1. Remove the RST, TST, V+, and GND jumpers from J13 on the MSP-EXP430FR5969 LaunchPad.

2. Remove the RST, TEST, and VCC jumpers from J3 on the MSP-EXP430G2 LaunchPad.

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 as in **Table 12-3**.

Table 12-3: Connection of Signals

MSP-EXP430FR5969 **Emulation**	**MSP-EXP430G2XL Device**
RST	RST
TST	TEST
V+	VCC
GND	GND

4. Target power must be supplied through the MSP-EXP430FR5969 LaunchPad that has the Energy Trace technology included in the on-board emulation. Plug-in the micro-USB to the MSP-EXP430FR5969.

5. Initialize the debug session for the MSP430G2553 to enable the Energy Trace mode.

12.5.2 Software Execution

1. Build the example code of **Experiment 7:PWM Based Speed Control of Motor by Potentiometer** that you want to analyze using Energy Trace technology.

2. Enable Energy Trace technology

a. Click Window → Preferences → Code Composer Studio → Advanced Tools → Energy Trace Technology

b. Make sure that the box next to **Enable** is checked

c. Select the **Energy Trace++** mode

d. Click **Apply** and **OK**.

3. Debug the example code, the Energy Trace window will open automatically.

If the window does not open automatically, click View → Others → MSP430 Energy Trace → Energy Trace Technology

4. Run the code. The Energy Trace Technology window will update the real time data.

12.6 Observation

Complete analysis of Energy Trace Technology for **Experiment 7:PWM Based Speed Control of Motor by Potentiometer** in terms of Energy, Power, Current, Voltage are given in **Table 12-4**.

Table 12-4: Analysis of Energy Trace Technology

Example Code	Energy in mJ	Mean Power in mW	Mean Current in mA	Mean Voltage in V
Experiment 8	17.57	19.55	5.46	3.58

The power and energy graph analyzed from Energy trace technology for **Experiment 7: PWM Based Speed Control of Motor by Potentiometer** are shown in **Figure 12-1**.

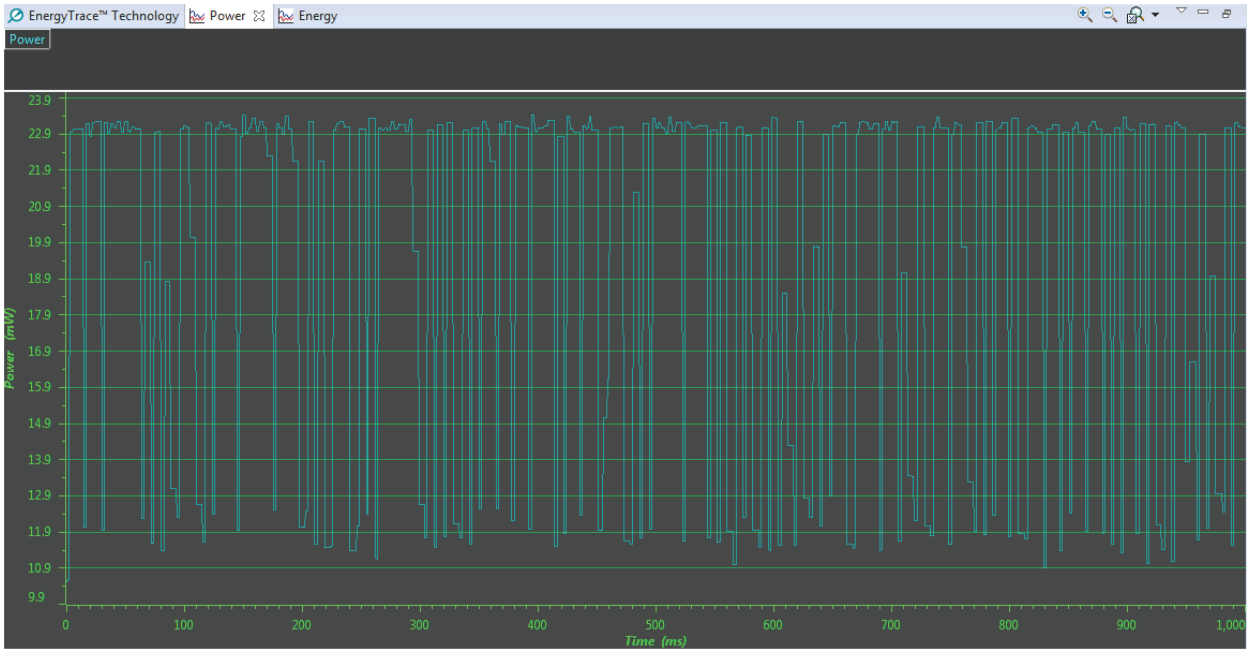


Figure 12-1 Experiment 7 Power Graph with 1sec Time Interval

The power graph shown in **Figure 12-1** highlights the time density of the DC-DC converter charge pulses which is equal to the energy consumption of the target microcontroller for a particular application. 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. The complete Energy profile for the application is shown in **Figure 12-2**.

.

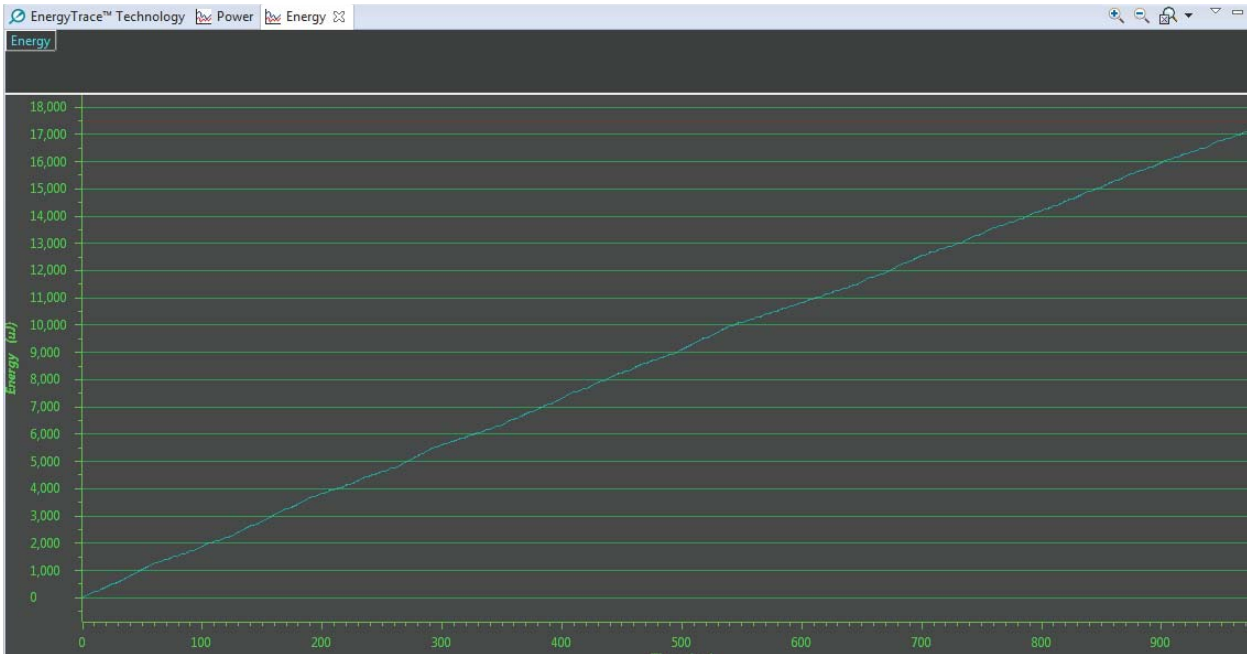


Figure 12-2 Experiment 7 Energy Graph with 1sec Time Interval

12.7 Summary

In this experiment, we have learnt to enable Energy Trace and Energy Trace++ modes in MSP-EXP430G2 LaunchPad by using MSP430FR5969.

12.8 Exercise

1. What is the difference between the Energy Trace and Energy Trace ++?
2. What hardware options available that supports Energy Trace++?

Experiment 13

Computation of Energy and Estimation of Battery Lifetime

Topics **Page**

13.1 Objective

The main objective of this experiment is to compute the total energy of MSP-EXP430G2 Launchpad running an application and to estimate the life time of a battery.

13.2 Introduction

To measure the total energy of the MSP-EXP430G2XL and estimated life time of a battery the Energy Trace Technology is must and should. Energy Trace 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. MSP430 devices with built-in Energy Trace+ [CPU State]+ [Peripheral States] (or in short, Energy Trace++™) technology allow real-time monitoring of many internal device states while user program code executes. Energy Trace++ technology is supported on selected MSP430 devices and debuggers.

The MSP-EXP430G2 LaunchPad does not support Energy Trace Technology. But it can utilize the Energy Trace component despite the fact that the Energy Trace++ on board circuitry is not present. To achieve this, the on-board emulation of a Launchpad that contains the Energy Trace technology circuitry is used to program and debug the intended target.

In this experiment, to program and debug the target device MSP-EXP430G2 LaunchPad, we use the MSP-EXP430FR5969 Launchpad which contains the Energy Trace Technology on-board circuitry. When the Energy Trace mode is enabled in the target device, the profile window shows some statistical data about the application that has been profiled.

The following are the parameters shown in the profile window:

- 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 the selected battery (in days) for the captured energy profile

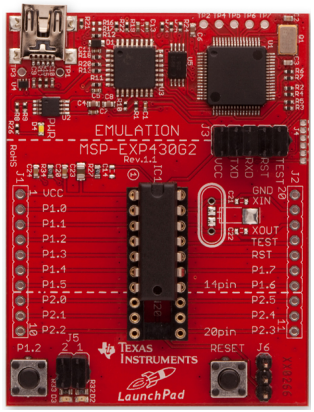
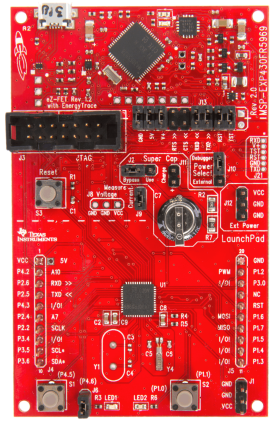
13.3 Component Requirements

13.3.1 Software Requirement

Code Composer Studio

13.3.2 Hardware Requirement

Table 13-1: Components Required for the Experiment

S.No	Components	Specifications	Image
1.	MSP430G2 LaunchPad	**MSP-EXP430G2XL**	
2.	USB cable		
3.	**MSP-EXP430FR5969**	MSP430FR5969 Launchpad plugged into your PC via the USB cable	
4.	Jumper Wires		

13.4 Software

In this experiment, we have used the example code of **Experiment 4: Interrupt Programming Through GPIO**.

13.5 Procedure

13.5.1 Hardware Connection

1. Remove the RST, TST, V+, and GND jumpers from J13 on the MSP-EXP430FR5969 LaunchPad.

2. Remove the RST, TEST, and VCC jumpers from J3 on the MSP-EXP430G2 LaunchPad.

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 as in **Table 13-2**.

Table 13-2: Connection of Signals

MSP-EXP430FR5969 Emulation	MSP-EXP430G2XL Device
RST	RST
TST	TEST
V+	VCC
GND	GND

4. Target power must be supplied through the MSP-EXP430FR5969 LaunchPad that has the Energy Trace technology included in the on-board emulation. Plug-in the micro-USB to the MSP-EXP430FR5969.

5. Initialize the debug session for the MSP430G2553 to enable the Energy Trace mode.

13.5.2 Software Execution

1. Build the example code of **Experiment 3: Low Power Modes and Current Measurement** for which you want to compute the total energy and estimate the life time of a battery using Energy Trace technology.

2. Enable Energy Trace technology

- Click Window → Preferences → Code Composer Studio → Advanced Tools → Energy Trace Technology
- Make sure that the box next to **Enable** is checked
- Select the **Energy Trace++** mode
- Click **Apply** and **OK**.

3. Debug the example code, the Energy Trace window will open automatically.

If the window does not open automatically, click View → Others → MSP430 Energy Trace → Energy Trace Technology

4. Run the code. The Energy Trace Technology window will update the real time data.

13.6 Observation

The Energy trace profile for active and stand by code is analyzed using Energy Trace technology. The Energy Trace profile window of the application in active mode provides us the estimated life-time of a battery of 46.6 days. If the same application run in standby mode, the estimated lifetime of a battery exceeds to 106.7 days.

The formula to calculate the battery lifetime assumes an ideal 3-V battery and does not account for temperature, aging, peak current and other factors that could negatively affect battery capacity.

Figure 13-1 and **Figure 13-2** show the Energy Trace Profile windows for the application in active and standby modes respectively.

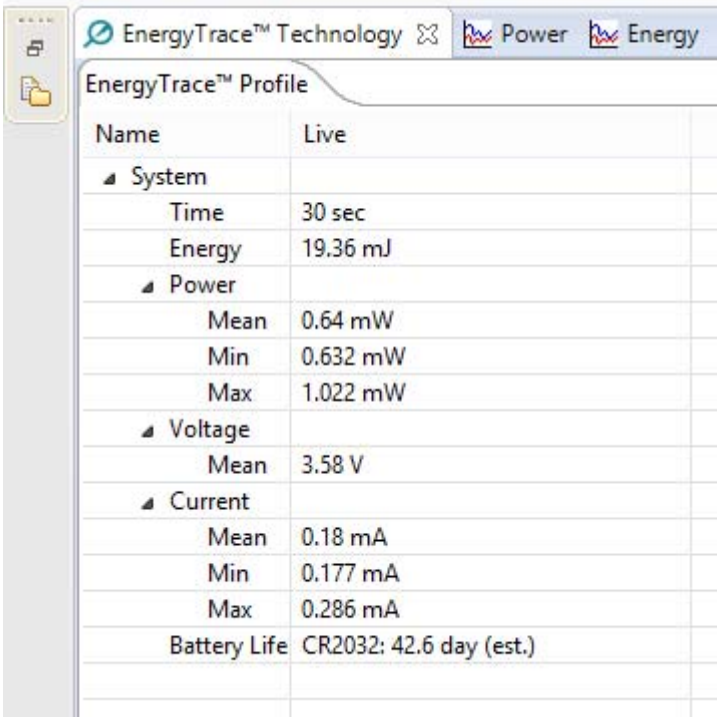


Figure 13-1 Energy Trace Profile Window in Active Mode

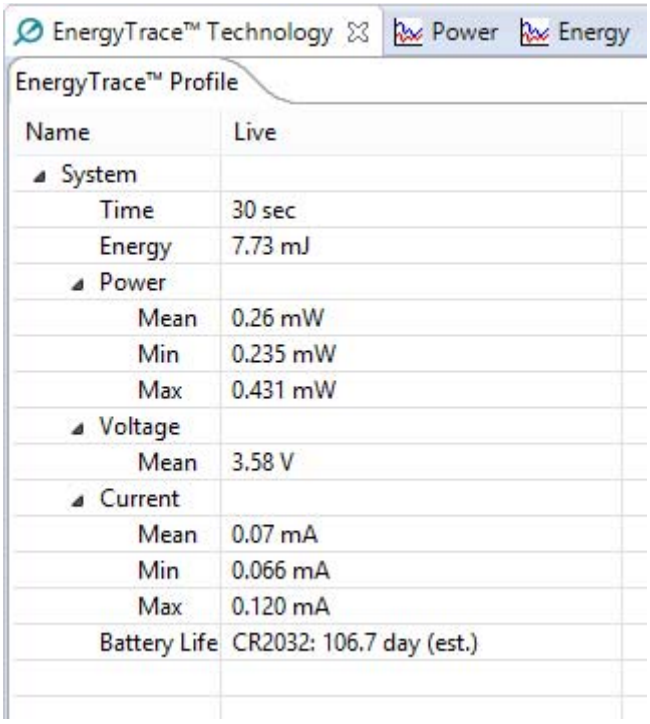


Figure 13-2 Energy Trace Profile Window in Standby Mode

The energy measurement and estimated lifetime of a battery for the application in both active and standby modes are tabulated in **Table 13-3**.

Table 13-3: Energy measurement and Estimated lifetime of a battery

Example Code	Energy in mJ for the 30sec time period	Estimated life time of a battery in days.
Active Mode	19.36	46.6
Standby Mode	7.73	106.7

13.7 Summary

In this experiment, we have learnt the computation of total energy and estimated life time of a battery in terms of days by Energy Trace technology of MSP-EXP430G2 LaunchPad with the help of MSP-EXP430FR5969.

13.8 Exercise

Compute the energy measurement and the estimated life time of a battery for Experiments 4 to 7.

Glossary

DCO	Digitally Controlled Oscillator
DHCP	Dynamic Hypertext Control Protocol
EVM	Evaluation Module
GIE	General Interrupt Enable
GPIO	General Purpose Input Output
HTTP	Hyper Text Transfer Protocol
I^2C	Inter-Integrated Circuit Interface
IDE	Integrated Development Environment
IE	Interrupt Enable Register
IFR	Interrupt Flag Register
ISR	Interrupt Service Routine
LPM	Low Power Mode
MAB	Memory Address Bus
MCU	Micro Controller Unit
MDB	Memory Data Bus
mDNS	Multicast Domain Name System
NMI	Non-maskable Interrupt
PWM	Pulse Width Modulation
SFR	Special Function Registers
SMTP	Simple Mail Transfer Protocol
SPI	Serial Peripheral Interface
SR	Status Register
TCP/IP	Transmission Control Protocol / Internet Protocol
TLS/SSL	Transport Layer Security / Secure Sockets Layer
UART	Universal Asynchronous Receiver / Transmitter
ULP Advisor	Ultra-Low-Power Advisor
USB	Universal Serial Bus
USCI	Universal Serial Communications Interface
WDT	Watchdog Timer
Wi-Fi	Wireless Fidelity
WLAN	Wireless Local Area Network
WPS	Wi-Fi Protected Setup