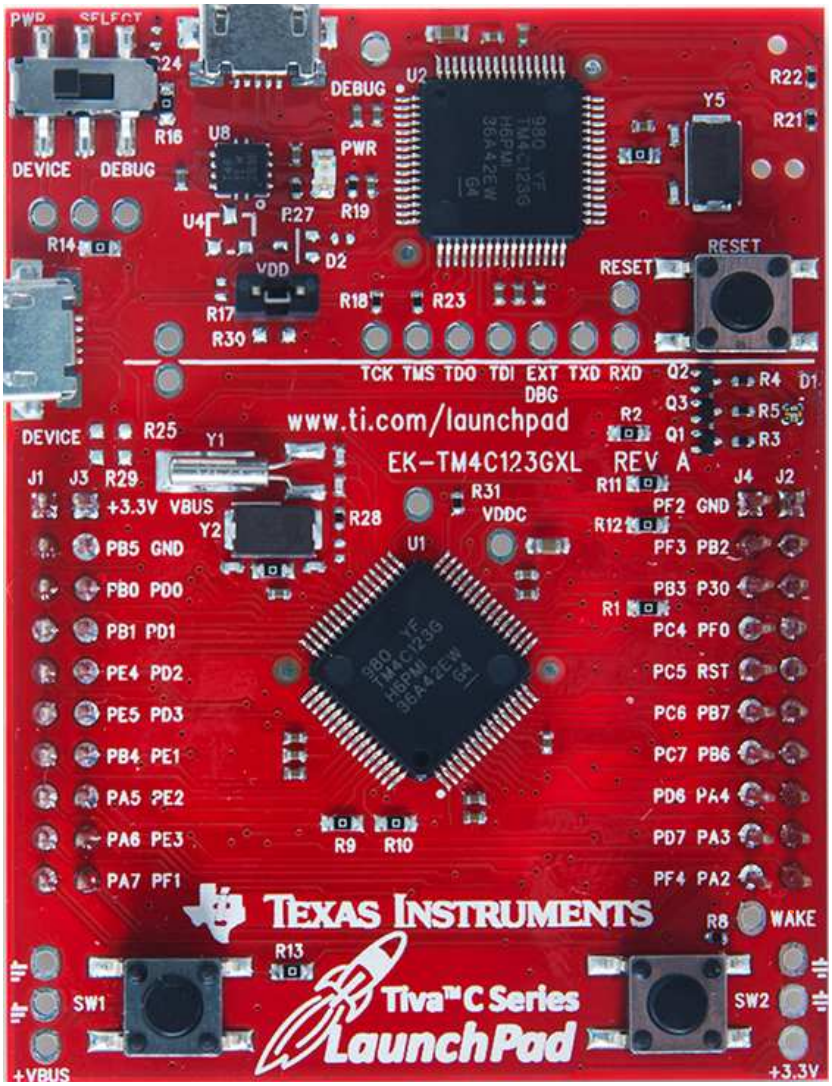
Embedded System Design using TM4C LaunchPad™ Development Kit

TEXAS INSTRUMENTS



TI LaunchPad™

SSQU015

Copyright

Texas Instruments Incorporated
108 Wild Basin, Suite 350
Austin, TX 78746
http://www.ti.com/tm4c
http://www-k.ext.ti.com/sc/technical-support/product-information-centers.htm

Contents

List of Figures

List of Tables

Preface

The Texas Instruments Tiva C Series microcontrollers (MCUs) are low-power, versatile and smart devices that feature different sets of peripherals and are used for various applications. The Tiva C series microcontrollers feature a powerful 32-bit ARM core, 80-MHz operation, and 100 DMIPS performance. The TM4C123x MCUs of the Tiva C Series provide a balance between the floating-point performance required to create highly responsive mixed-signal applications and the low-power architecture required to enable increasingly aggressive power budgets.

In this lab manual, all the experiments are performed using TM4C123GH6PM microcontrollers to realize widespread applications. The experiments take advantage of the wide range of parts and tools supported by Texas Instruments for design with the Tiva C Series microcontrollers and wireless communication products. The Tiva EK-TM4C123GXL LaunchPad kit is an easily available evaluation kit for users to get started with designing applications using Tiva C Series microcontroller. The TM4C123x MCUs are supported by TivaWare™ for C Series software that helps the user to start writing production-ready code easily and quickly and minimize the overall cost of software ownership.

Each experiment in this manual includes:

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

Experiments 1 to 7 use typical applications like GPIO, PWM, ADC, Interrupt, Low Power Modes, UART. Experiments 8 and 9 use different interfaces supported by the TM4C123GH6PM such as I2C and USB bulk transfer. Experiment 10 demonstrates the use of mathematical library for exhaustive computation. Experiment 11, 12 and 13 use applications for IoT (Internet of Things).

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

Getting Started

Topics **Page**

Overview

Introduction to Tiva C Series

The Texas Instruments Tiva C Series microcontrollers (MCUs) are low-power, versatile and smart devices. The Tiva C Series consists of several devices that feature different sets of peripherals and are targeted for various applications.

The Tiva C Series MCUs provide both high performance for mixed-signal applications and low power architecture for power critical applications. The Tiva C Series MCUs are supported by TivaWare™ for C Series for software development. The Tiva EK-TM4C123GXL Launchpad kit is an easily available evaluation kit for users to get started with designing applications using Tiva C Series microcontroller.

TM4C12x Microcontrollers

The Tiva C series microcontrollers feature a powerful 32-bit ARM core, 80-MHz operation, and 100 DMIPS[1] performance. The TM4C12x devices provide high performance and advanced integration. The TM4C123GH6PM microcontroller is used on the EK-TM4C123GXL.

The features of TM4C123GH6PM microcontroller are:

- It supports Thumb2 Technology, which means it supports 16/32-bit code. It consumes 26% less memory & is 25% faster when compared to pure 32-bit microcontrollers. This means, it provides the optimum mix of 16-bit and 32-bit microcontroller application requirements.
- It consists of a battery-backed hibernation module that provides logic to switch power off to the main processor and its peripherals while the processor is idle. The features of this module include low-battery detection, signaling and interrupt generation, with optional wake-up on low battery.
- The power consumption is as low as 370μA/MHz. It takes 500μs to wake up from low-power modes.
- The current consumption of Real Time Clock is 1.7μA and it also supports internal and external power control.
- It has six physical GPIO (General Purpose Input Output) blocks, six 16/32-bit and six 32/64-bit general purpose timers, 32 dedicated 32-bit single-precision registers, also addressable as 16 double-word registers.
- TM4C123GH6PM devices support flexible clocking system.
- It has an internal precision oscillator, external main oscillator with PLL(Phase Locked Loop) support, internal low frequency oscillator and real-time-clock through Hibernation module.
- It has saturated math for signal processing.
- It supports atomic bit manipulation and Read-Modify-Write using bit-banding.
- It has single cycle multiply and hardware divider module.
- The device has 256KB of Flash Memory, 32KB single-cycle SRAM and 2KB EEPROM. The memory is used efficiently because of unaligned data access.
- It has two motion control modules each with eight high-resolution PWM outputs.
- The device supports IEEE754 compliant single-precision floating-point unit.
- The debugger access that the device supports are JTAG and Serial Wire Debug - ETM (Embedded Trace Macrocell) available through Keil and IAR emulators.

1. DMIPS - Dhrystone Million Instructions per Second. It is a measure of processor performance.

- The serial connectivity includes peripherals such as USB 2.0 (OTG/Host/Device), 8 - UART (with IrDA, 9-bit and ISO7816 support), 6 - I^2C, 4 - SPI (Serial Port Interface), Microwire or TI synchronous serial interfaces and 2 - CAN.

The basic blocks in the Tiva C Series TM4C12x microcontrollers are shown in **Figure 1**.

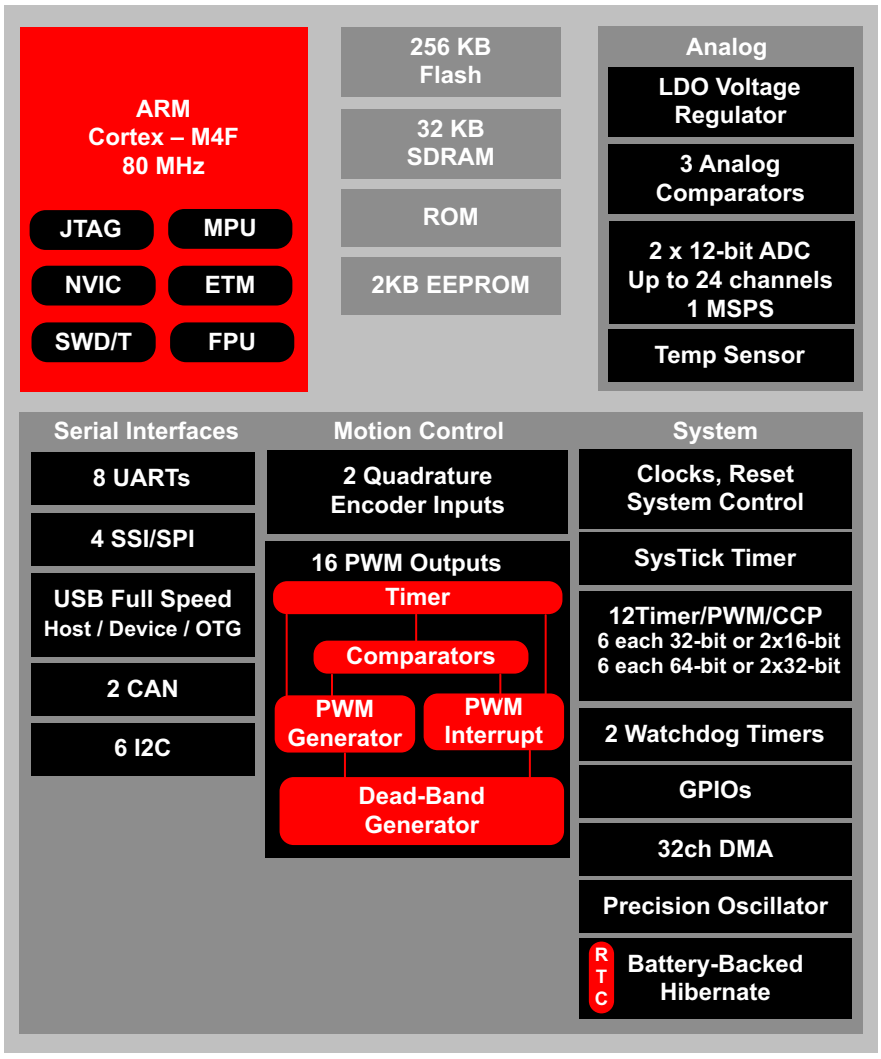


Figure 1 Tiva C Series TM4C12x Microcontroller Architecture

Figure 2 shows the pin diagram for TM4C123GH6PM microcontroller.

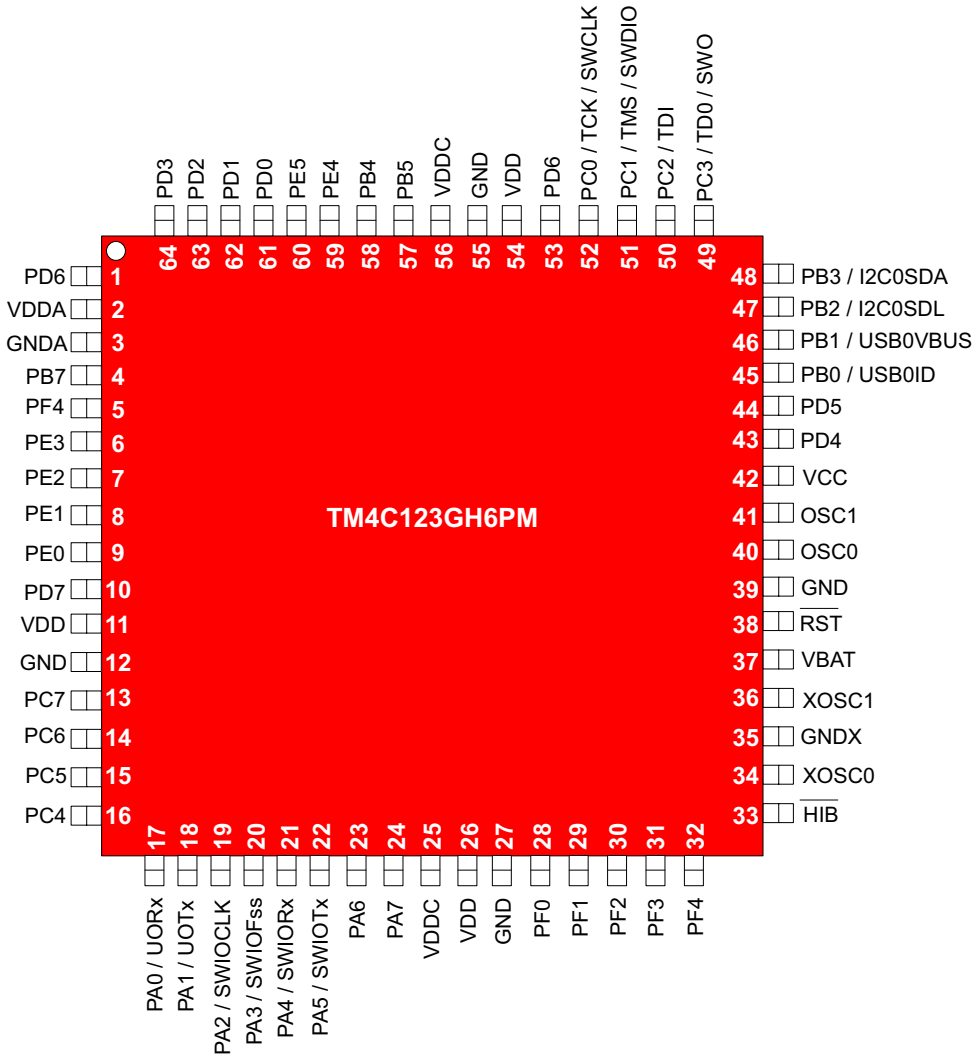


Figure 2 Pin Diagram of TM4C123GH6PM

ARM Cortex-M4F Processor

The ARM Cortex-M4F processor used in the Tiva C Series microcontrollers provides high performance while meeting low-power requirements. **Figure 3** shows the functional block diagram of the ARM Cortex-M4F processor.

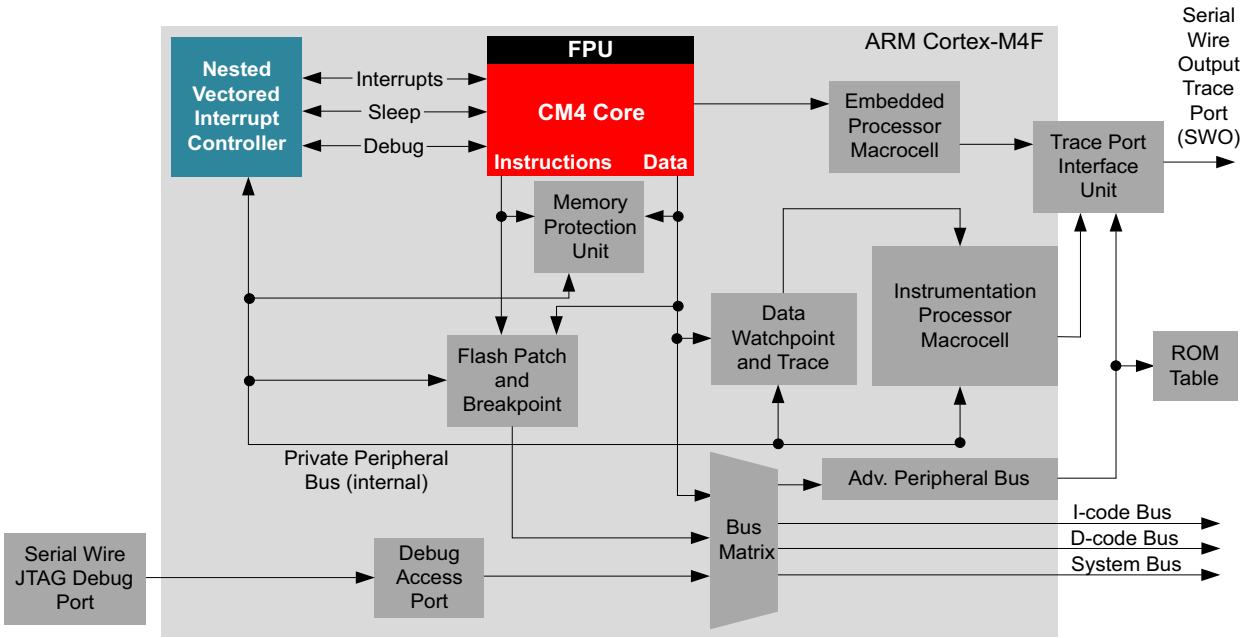


Figure 3 Functional Block Diagram of ARM Cortex-M4F Processor

The Cortex-M4F processor is a 3-stage pipeline Harvard architecture which is most suitable for embedded applications. The processor has a Memory Protection Unit (MPU) for memory control that can separate code, data and stack based on the security levels. The Cortex -M4F processor core interacts with the Nested Interrupt Controller (NVIC) that provides exceptional interrupt performance with eight interrupt priority levels and a Non-Maskable Interrupt (NMI). The interrupt latency is reduced significantly because of tight integration of the processor core with the NVIC.

Enhanced system debug is achieved in the Cortex -M4F processor with several supporting modules. The processor implements a JTAG (Joint Test Actions Group) or a 2-pin Serial Wire Debug (SWD) for hardware debug. For system trace, the processor incorporates an Instrumentation Trace Macrocell (ITM) with Data Watchpoint and Trace unit. The Embedded Trace Macrocell (ETM) enables full instruction trace.

The Flash Patch and Breakpoint Unit (FPB) module provides up to eight hardware breakpoint comparators that debuggers can use. The Trace Port Interface Unit (TPIU) connects the Cortex-M4F trace data to an off-chip Trace Port Analyzer.

Tiva C Series LaunchPad

Introduction

The Tiva C Series TM4C123GXL LaunchPad Evaluation Kit (EK-TM4C123GXL) is a low-cost evaluation platform for ARM Cortex-M4F-based microcontrollers from Texas Instruments. It is an easy to use startup kit that provides everything the user will need to evaluate the Tiva C Series microcontroller. The design of the EK-TM4C123GXL highlights the TM4C123GH6PM microcontroller with a USB 2.0 device interface and hibernation module. The EK-TM4C123GXL also features programmable user buttons and an RGB LED for custom applications. The EK-TM4C123GXL includes:

- A TM4C123G LaunchPad Evaluation board
- On-board In-Circuit Debug Interface (ICDI)
- USB Micro-B plug to USB-A plug cable
- ReadMe First quick-start guide

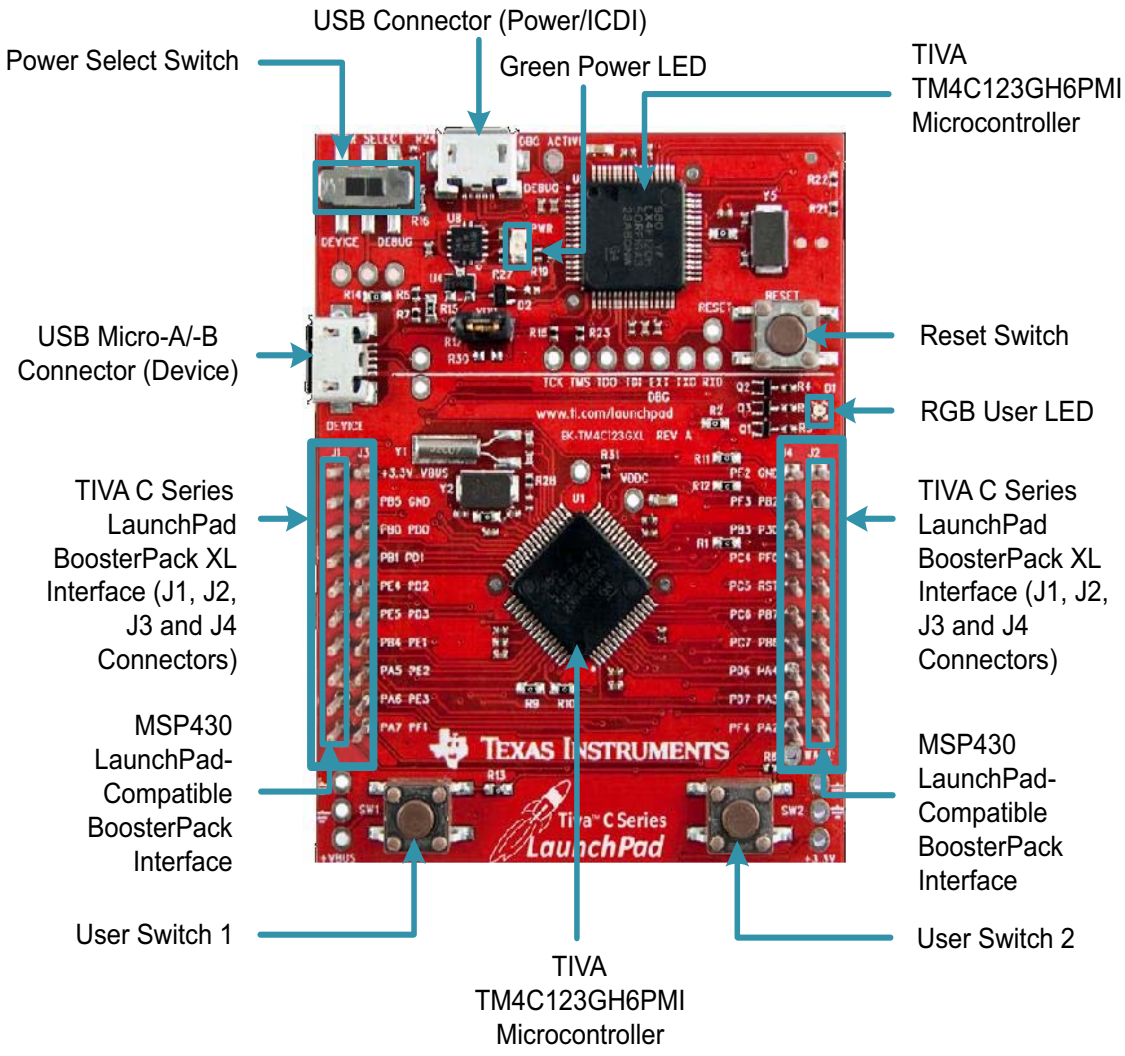


Figure 4 EK-TM4C123GXL

The features of the EK-TM4C123GXL is:

- ARM® Cortex™-M4F, 64-pin 80MHz TM4C123GH6PM processor
- On-board USB ICDI (In-Circuit Debug Interface)
- Micro AB USB port
- Device/ICDI power switch
- BoosterPack XL pinout also supports legacy BoosterPack pinout
- 2 user pushbuttons (SW2 is connected to the WAKE pin)
- Reset button
- 3 user LEDs (1 tri-color device)
- Current measurement test points
- 16MHz Main Oscillator crystal
- 32kHz Real Time Clock crystal
- 3.3V regulator
- Support for multiple IDEs

Hardware Setup

The LaunchPad experimenter board includes a pre-programmed TM4C123H6PM device which comes preinstalled in the target socket. Connect the LaunchPad Board ICDI USB connector (Refer **Figure 4**) to a free USB port on your PC using the USB cable.

When the LaunchPad is connected to your PC via USB, the driver installation starts automatically. If prompted for software, allow Windows to install the software automatically.

After driver installation, the LaunchPad board starts a demo application with an LED toggle sequence. The on-board emulator generates the supply voltage and all signals necessary to start the demo.

The Application Demo Program

The Tiva C Series LaunchPad comes pre-programmed with the RGB QuickStart application. This application demonstrates control of the on-board RGB LED, TM4C123GH6PM microcontroller's Hibernate functionality and serial communication with the Tiva C Series LaunchPad. Once the LaunchPad is connected to the PC via the ICDI USB port, the drivers are installed automatically and the demo application starts execution. The steps to be followed for the demo application are:

1. Check if the Power Select Switch in the upper left corner of the board is in the DEBUG position as shown in **Figure 5**.

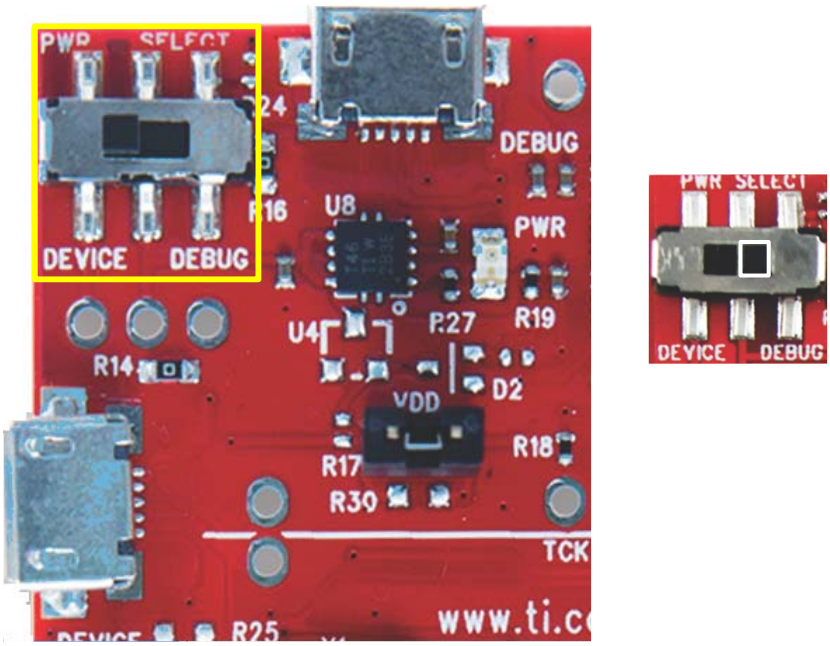


Figure 5 Power Select Switch Position

2. Press the bottom left button (SW1) or the bottom right button (SW2) to scan the ROYGBIV color spectrum of the RGB LED.

3. Leave the LaunchPad idle for five seconds. You will see a random color display of the RGB LED.

4. To enter Hibernation mode, press and hold SW1 and SW2 for three seconds.While in this mode, you can see the LED blink every three seconds.

5. To exit Hibernation mode, press SW2.

6. To control these functions serially using the UART, see the readme file in the qs_rgb project in the Tivaware™ examples for the EK-TM4C123GXL.

Documents for Reference

- LaunchPadUser Guide
- ROM User's Guide
- TM4C123GH6PM Data Sheet
- Peripheral Driver Library User Guide

Code Composer Studio

Overview

Code Composer Studio (CCS) is the integrated development environment for the whole span of TI Micro-controllers, 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.

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 6**.

- 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.

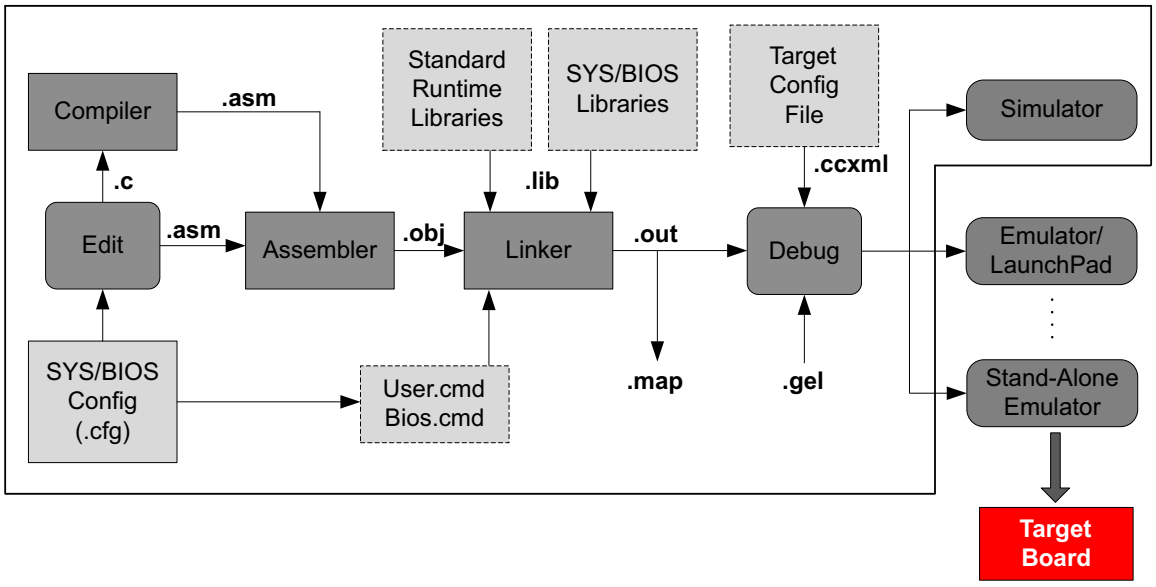


Figure 6 CCS Functional Overview

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 7**.

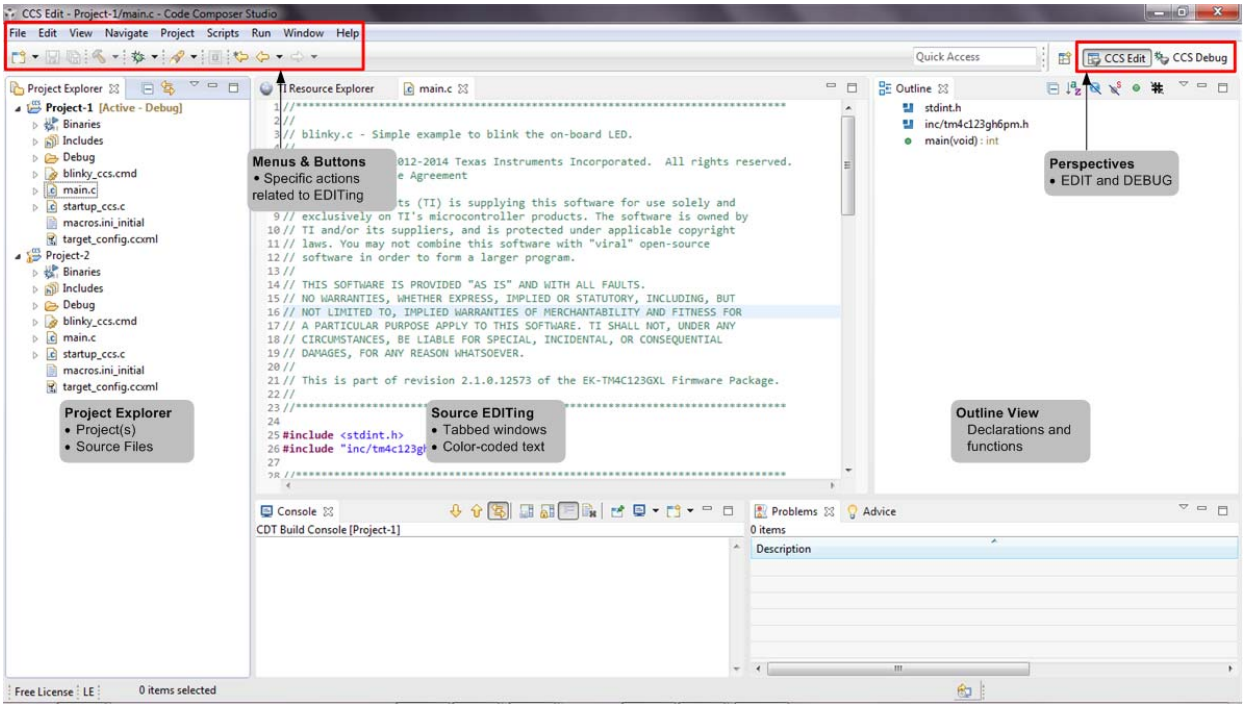


Figure 7 CCS Edit Perspective

The **Debug perspective** has menus, buttons and debug window related to debugging as shown in **Figure 8**.

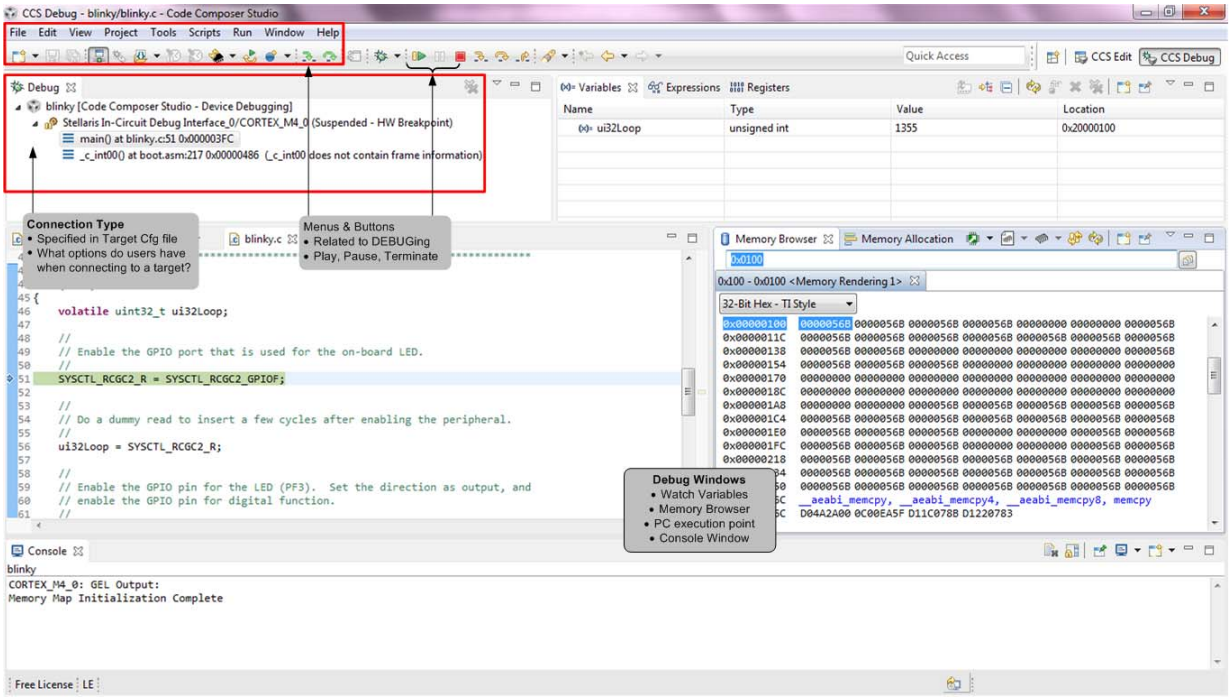


Figure 8 CCS Debug Perspective

Workspaces and Projects

A workspace contains settings and preferences, as well as links to the projects as shown in **Figure 9**.

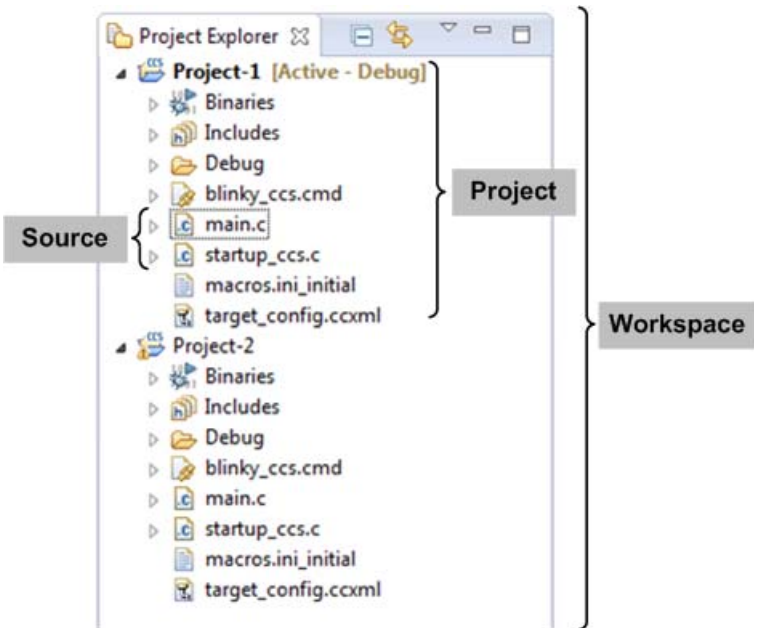


Figure 9 Workspace and Projects (as viewed in CCS)

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 10**.

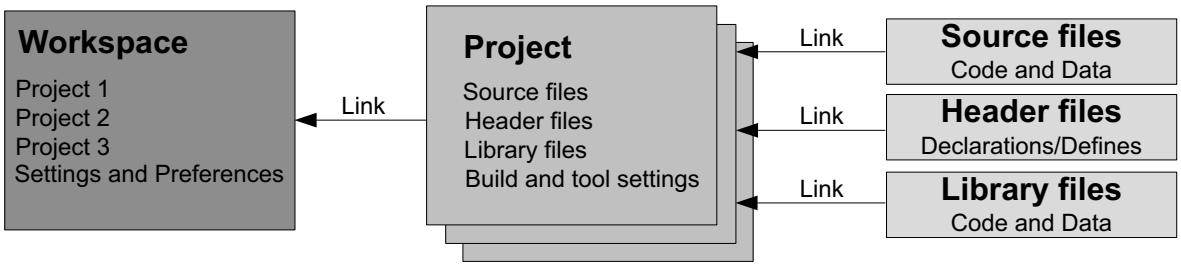


Figure 10 Workspace and Projects

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 TM4C123GH6PM hardware. The steps to be followed to create a project are:

1. Create a New Project from **Project** → **New Project**. Refer **Figure 11**.

a. Select the Target as Tiva TM4C123GH6PM from the drop down box.

b. Choose the Connection as Stellaris In-Circuit Debug Interface from the drop down box.

c. Enter a relevant Project name in the text box provided.

d. Click on Finish to create the new project.0.7

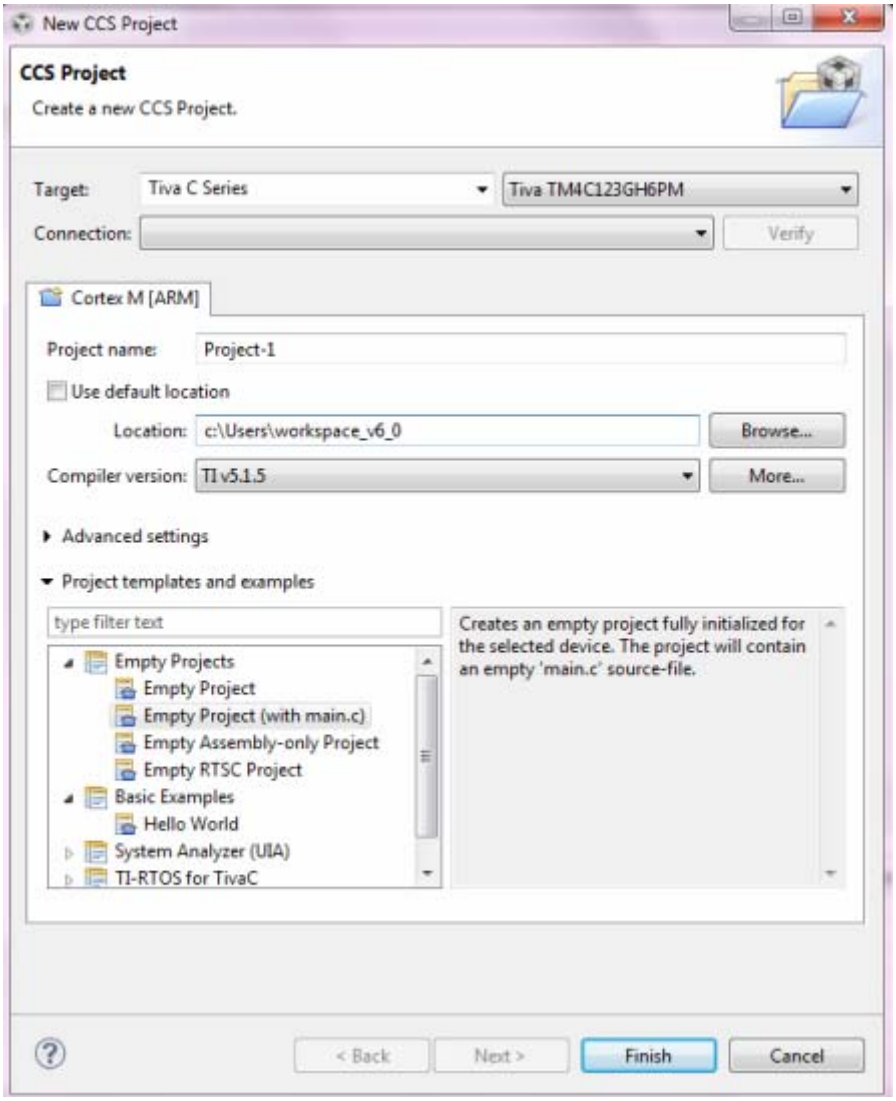


Figure 11 New CCS Project Creation

2. Go to Properties from **Project** → **Properties**.

a. Click on **Include Options** under **ARM Compiler** as shown in **Figure 12**.

b. Add the root of "TivaWare_C_Series-2.1.0.12573" to the #include search path.

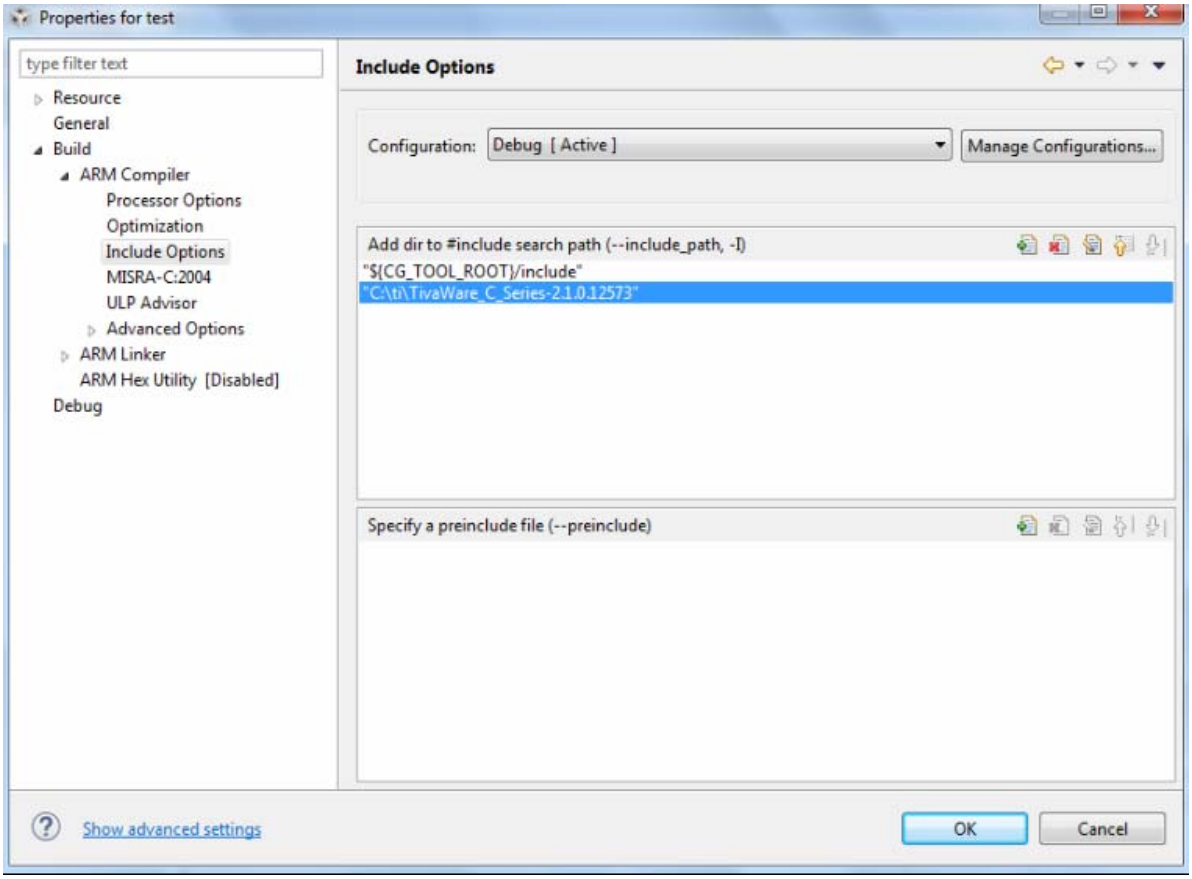


Figure 12 Project Properties Window

c. Click on **File Search Path** under **ARM Linker** as shown in **Figure 13**.

d. Add "**driverlib.lib**" from driverlib\ccs\Debug inside the TivaWare_C_Series installation directory to the "Include Library" section.

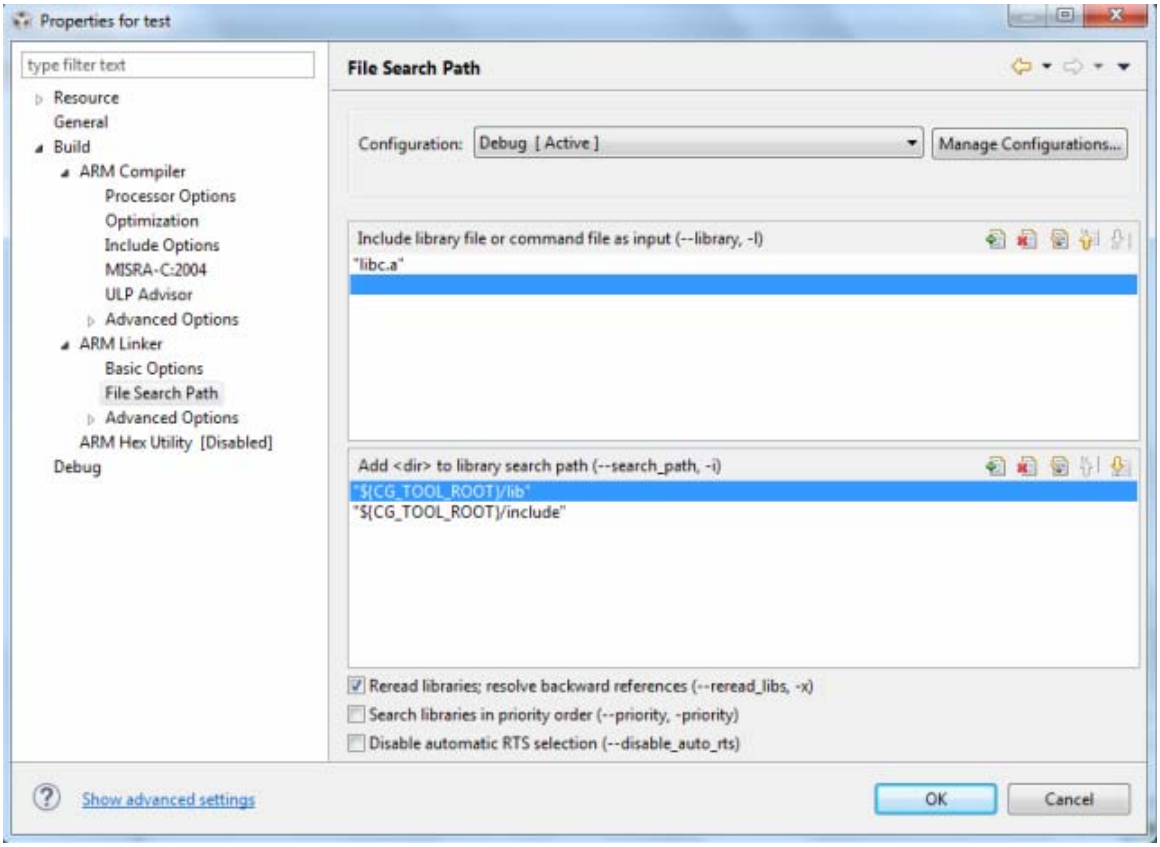


Figure 13 File Search Path

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

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 8**). Click and drag the perspective tabs to the left until you can completely see both the tabs.

After modification of source files, CCS can 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  . When the Ultra-Low-Power Advisor (ULP Advisor) appears, click the **Proceed** button. The program has completed execution through the C-environment initialization routine in the runtime support library and stopped at main() in main.c.

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 14 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 on the left of the opening brace of main() in the middle window as shown in **Figure 14**. The blue arrow indicates where the Program Counter (PC) is pointing to.

Click **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.

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

Important Notes for CCS

1. Make sure your Launch pad 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. For experiments which are taken from the library, any warning message indicating that the project was created with an earlier compiler version 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.

TivaWare Library for Tiva Platform

The Texas Instruments TivaWare peripheral driver library is a set of drivers for accessing the peripherals found on the Tiva family of ARM Cortex-M based microcontrollers. They are not drivers in the pure operating system sense, i.e., they do not have a common interface and do not connect into a global device driver infrastructure. TivaWare simplifies the use of the device's peripherals. The capabilities and organization of the drivers are governed by the following:

- Written entirely in C programming language except where absolutely not possible
- Allows the use of peripherals in its common mode of operation
- Efficient usage of memory and processor.
- Where possible, computations that can be performed at compile time are performed in the drivers instead of at run time. They can be built with more than one tool chain.

The peripheral driver library provides support for two programming models namely, the direct register access model and the software driver model. Each model can be used independently or together based on the needs of the application or the programming environment desired by the developer. Use of the direct register access model generally results in smaller and more efficient code than using the software driver model. However, the direct register access model requires detailed knowledge of the operation of each register and bit field, as well as their interactions and any sequencing required for proper operation of the peripheral, the developer is insulated from these details by the software driver model, generally requiring less time to develop applications. The direct register access model and software driver model can be used together in a single application, allowing the most appropriate model to be applied as needed to any particular situation within the application. The general API functions used in this lab manual are given in **Table 1**.

Table 1: API Functions Used in the Application Program

API Function	Parameters	Description
SysCtlClockSet(uint32_t ui32Config)	• **ui32Config** is the required configuration of the device clocking - logical OR of several different values: - System clock divider - Use of the PLL - External crystal frequency - Oscillator source	This function configures the clocking of the device. The input crystal frequency, oscillator to be used, use of the PLL, and the system clock divider are all configured with this function.
SysCtlPeripheralEnable(uint32_t ui32Peripheral)	• **ui32Peripheral** is the peripheral to enable.	This function enables a peripheral.
GPIOPinTypeGPIOOutput(uint32_t ui32Port, uint8_t ui8Pins)	• **ui32Port** is the base address of the GPIO port. • **ui8Pins** is the bit-packed representation of the pin(s).	Configures pin(s) for use as GPIO outputs.
GPIOPinWrite(uint32_t ui32Port, uint8_t ui8Pins, uint8_t ui8Val)	• **ui32Port** is the base address of the GPIO port. • **ui8Pins** is the bit-packed representation of the pin(s). • **ui8Val** is the value to write to the pin(s).	Writes a value to the specified pin(s).

Using Tera Term

Tera Term is open source freely downloadable terminal emulator software used for testing 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 the Tiva is connected in the Tera Term is displayed in the **New Connection** Window as shown in **Figure 15**. The serial port option is enabled only if the Tiva 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.

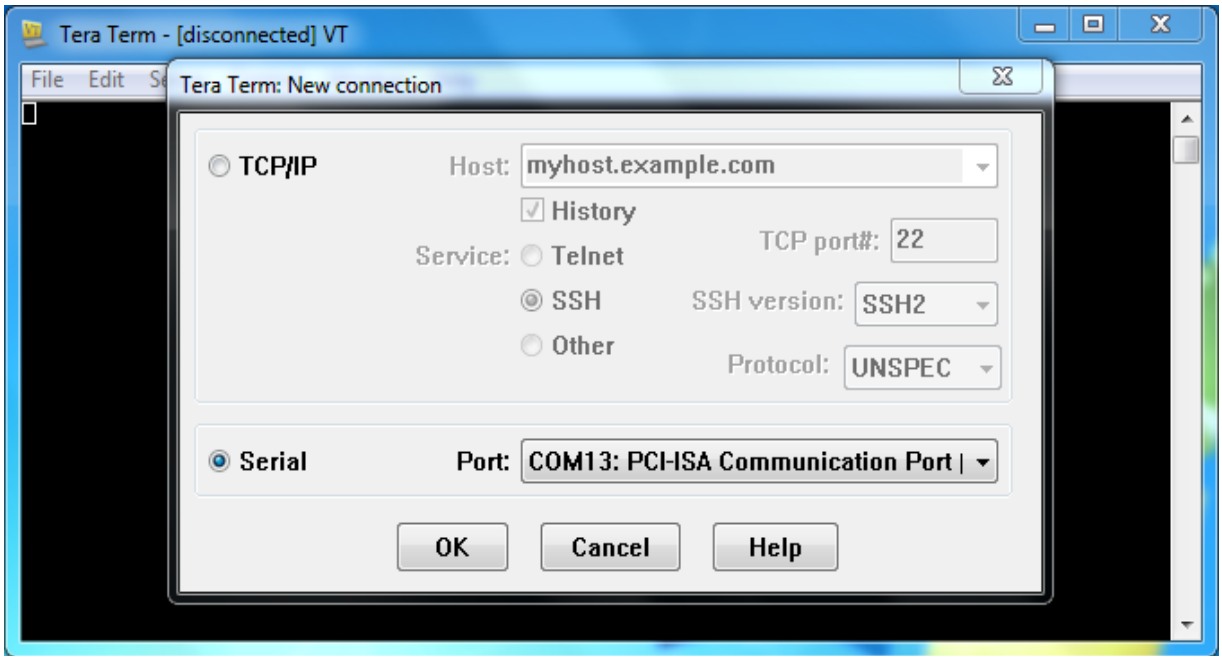


Figure 15 Tera Term New Connection Window

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 16**.

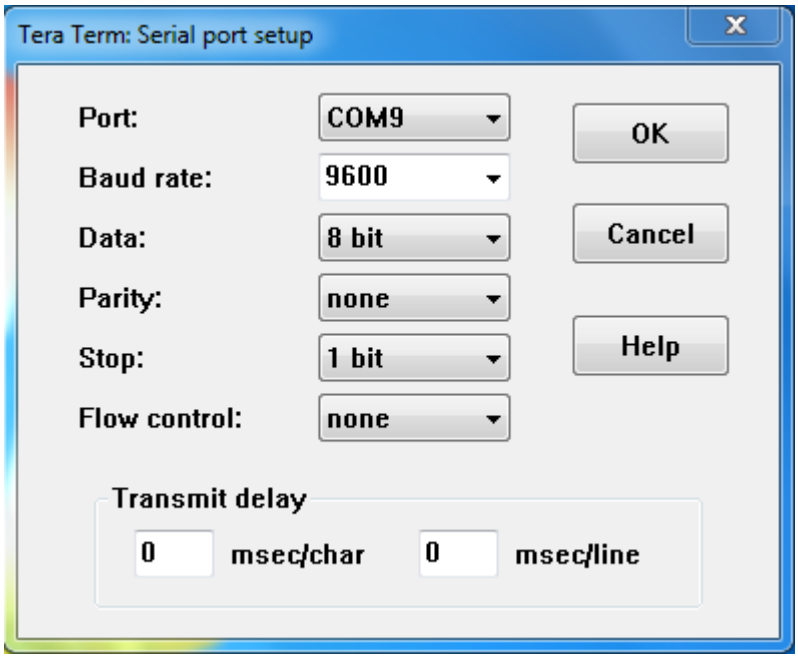


Figure 16 Tera Term Serial Port Setup Window

3. Select **Setup** → **Terminal** → **New Line** → **Receive** → **Auto** to configure the new line character as shown in **Figure 17**.

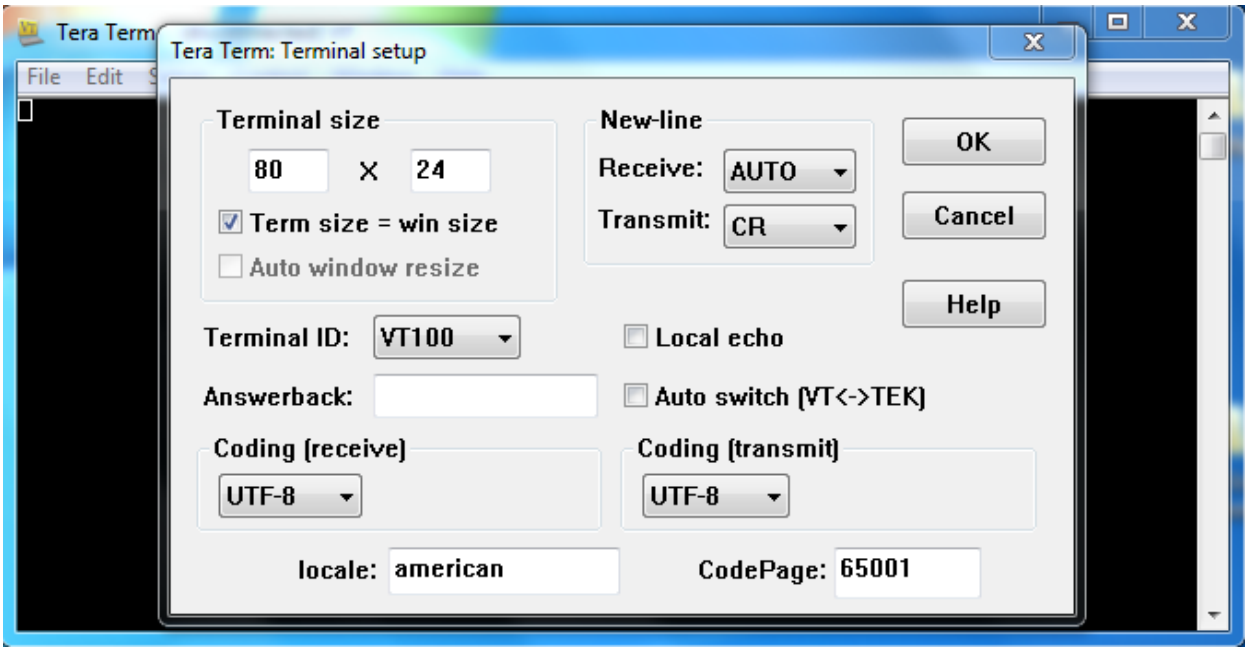


Figure 17 Tera Term Terminal Setup Window

Experiment 1

To Blink an Onboard LED

Topics **Page**

1.1 Objective

The main objective of this experiment is to configure the Tiva GPIO pins to blink the green on-board LED (connected to PF3) using C program.

1.2 Introduction

The EK-TM4C123GXL has three on-board LEDs which are connected to the GPIO pins PF1, PF2 and PF3 of the TM4C123GH6PM microcontroller. The software code in the TM4C123GH6PM toggles the PF3 output at fixed time intervals computed within the code. A HIGH on PF3 turns the LED On, while a LOW on PF3 turns the LED Off. Thus, a toggling output on the port pin PF3 blinks the LED connected to it. The functional block diagram as shown in **Figure 1-1** illustrates the working principle of the experiment.

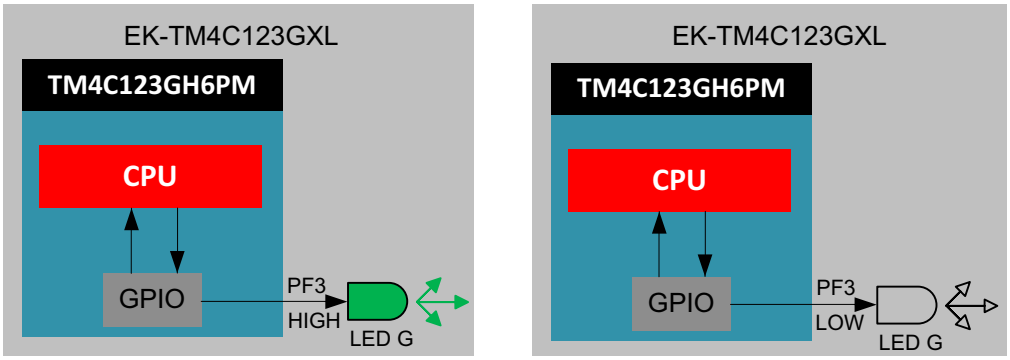


Figure 1-1 Functional Block Diagram

1.2.1 GPIO Module

The TM4C123GH6PM features a GPIO (General Purpose Input Output) module that is composed of six physical GPIO blocks. Each block corresponds to an individual GPIO port (Port A, Port B, Port C, Port D, Port E, Port F). The GPIO pins are connected to the on-board LEDs in the LaunchPad as shown in the EK-TM4C123GXL schematics in **Figure 1-2**.

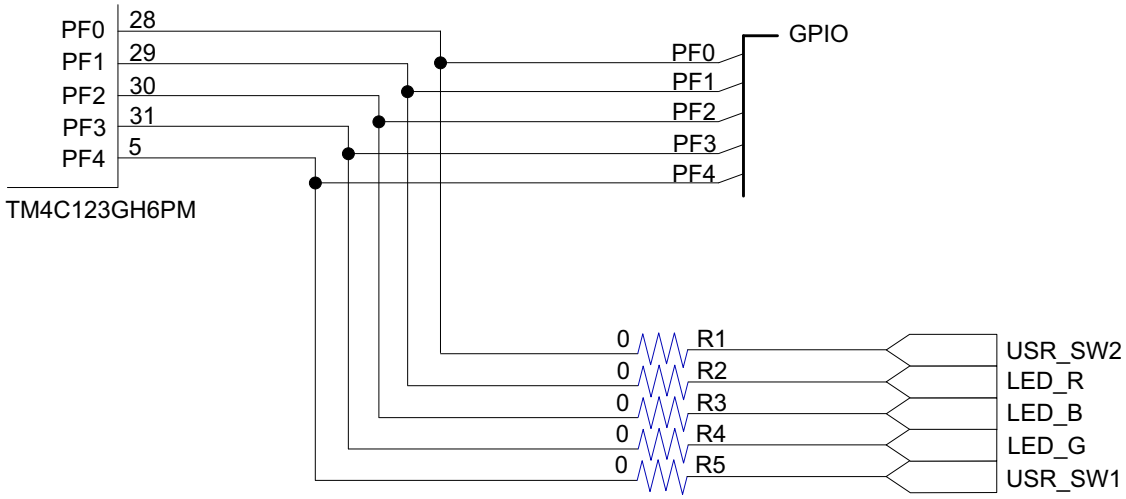


Figure 1-2 LaunchPad Schematics for GPIO Connected to LEDs

The features of GPIO are:

- Availability of up to 43 GPIOs, depending on configuration
- Highly flexible pin multiplexing allows usage of pins as GPIO or one of the several peripheral functions available
- Bit masking in both read and write operations through address lines
- Programmable control for GPIO interrupts:
  - Interrupt generation masking
  - Edge-triggered on rising, falling, or both
  - Level-sensitive on HIGH or LOW values
- Programmable weak pull-up, pull-down and open drain
- Programmable drive strength and slew rate control

Pin multiplexing of the GPIO signals allow alternate hardware functions of the pins. In the EK-TM4C123GXL, the GPIO pins are configured with pin functions as shown in **Table 1-1**.

Table 1-1: GPIO Pin Functions and USB Device Connected

GPIO Pin	Pin Function	USB Device
PF4	GPIO	SW1
PF0	GPIO	SW2
PF1	GPIO	RGB LED (Red)
PF2	GPIO	RGB LED (Blue)
PF3	GPIO	RGB LED (Green)

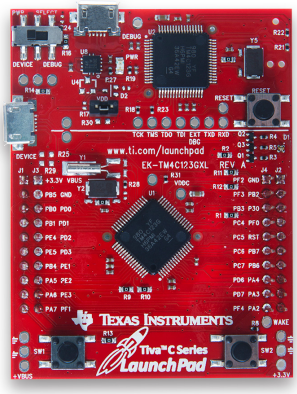
1.3 Component Requirements

1.3.1 Software Requirements

1. Code Composer Studio
2. TivaWare_C_Series

1.3.2 Hardware Requirements

Table 1-2: Components Required for the Experiment

S.No	Components	Specification	Image
1.	Tiva LaunchPad	EK-TM4C123GXL LaunchPad	
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 TM4C123GH6PM on the EK-TM4C123GXL using the USB interface.

1.4.1 Flowchart

The flowchart for the code is shown in **Figure 1-3**. The software written in C configures and enables the system clock to 40MHz. It then enables GPIO Port F of EK-TM4C123GXL and configures pin 3 (PF3) as output. A HIGH output at the GPIO PF3 turns the green LED ON connected to it and a LOW output turns the LED OFF. Hence, to blink the LED, the output is toggled at regular time interval using a while (1) loop. The time interval between the LED turning ON and OFF can be programmed by specifying the delay count value in the SysCtlDelay() function.

Calculation of Delay: The number of counts required to get a time delay is given by

Number of Counts = Time delay required * System Clock Frequency

In the program, for a time delay of 500ms

Number of counts = $500 * 10^{-3} * 40 * 10^{6} = 20 * 10^{6}$

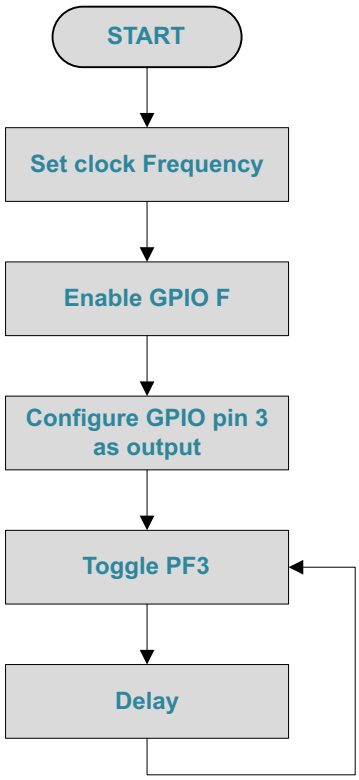


Figure 1-3 Flowchart for Blinking an Onboard LED

1.4.2 C Program Code to Blink an Onboard LED

```
#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_types.h"
#include "inc/hw_memmap.h"
#include "driverlib/sysctl.h"
#include "driverlib/pin_map.h"
#include "driverlib/debug.h"
#include "driverlib/gpio.h"

int main(void)
{
SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|
SYSCTL_OSC_MAIN);
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|
GPIO_PIN_3);
```

```
while(1){
GPIOPinWrite(GPIO_PORTF_BASE,GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 0x08);
SysCtlDelay(20000000);
GPIOPinWrite(GPIO_PORTF_BASE,GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 0x00);
SysCtlDelay(20000000);
}
}
```

Table 1-3: API Functions Used in the Application Program

API Function	Parameters	Description
SysCtlClock-Set(uint32_t ui32Con-fig)	• **ui32Config** is the required configuration of the device clocking - logical OR of sev-eral different values: - System clock divider - Use of the PLL - External crystal frequency - Oscillator source	This function config-ures the clocking of the device. The input crys-tal frequency, oscillator to be used, use of the PLL, and the system clock divider are all con-figured with this func-tion.
SysCtlPeripheralEn-able(uint32_t ui32Pe-ripheral)	• **ui32Peripheral** is the peripheral to enable.	This function enables a peripheral.
GPIOPinTypeGPIOOut-put(uint32_t ui32Port, uint8_t ui8Pins)	• **ui32Port** is the base address of the GPIO port. • **ui8Pins** is the bit-packed representation of the pin(s).	Configures pin(s) for use as GPIO outputs.
SysCtlDelay(uint32_t ui32Count)	• **ui32Count** is the number of delay loop iterations to per-form.	Provides a small delay.
GPIOPinWrite(uint32_t ui32Port, uint8_t ui8Pins, uint8_t ui8Val)	• **ui32Port** is the base address of the GPIO port. • **ui8Pins** is the bit-packed representation of the pin(s). • **ui8Val** is the value to write to the pin(s).	Writes a value to the specified pin(s).

1.5 Procedure

1. Connect the EK-TM4C123GXL 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 green LED.

1.6 Observation

The green LED blinks on successful execution of the program. The rate of blinking can be altered by changing the delay count value in the program.

1.7 Summary

In this experiment, we have learnt to configure and program the GPIO pins of Tiva and have successfully programmed the GPIO pin PF3 to blink the on-board green LED.

1.8 Exercise

1. Program the TM4C123GH6PM to blink any other two on-board LED's of EK-TM4C123GXL.
2. Program the TM4C123GH6PM to blink all on-board LED's of EK-TM4C123GXL at a time.

Experiment 2

Interrupt Programming with GPIO

Topics **Page**

2.1 Objective

The main objective of this experiment is to toggle an LED by configuring the timer interrupt of the TM4C123GH6PM microcontroller. This experiment will help to understand the Timer Interrupt and its configuration.

2.2 Introduction

In this experiment, the timer peripheral of the TM4C123GH6PM processor is configured in periodic mode. In periodic mode, the timer load register is loaded with a preset value and the timer count is decremented for each clock cycle.When the timer count reaches zero, an interrupt is generated. On each interrupt, the processor reads the current status of the LED connected to a GPIO port and toggles it. Consequently, the GPIO output is programmed by the timer interrupt. The functional block diagram as shown in **Figure 2-1** illustrates the working principle of the experiment.

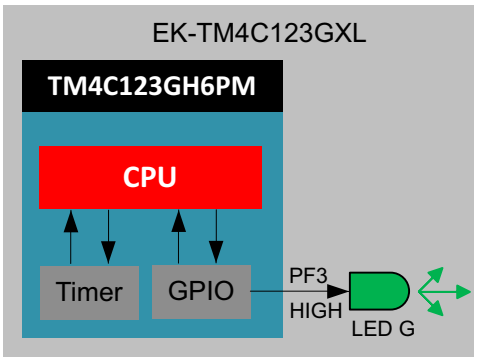


Figure 2-1 Functional Block Diagram

2.2.1 Timer

The programmable timer peripheral is used to count or time external events that drive the timer input pins. The GPT (General Purpose Timer) Module is one of the timing resource available on the Tiva™ C Series microcontrollers. Other timer resources include the System Timer and the PWM timer in the PWM modules. The features of the general purpose timer peripheral of the TM4C123GH6PM MCU are:

- Six 16/32-bit general purpose timers and six 32/64-bit wide general purpose timers
- Twelve 16/32-bit and twelve 32/64-bit capture/compare/PWM pins
- Timer modes supported:
  - One-shot
  - Periodic
  - Input edge count or time capture with 16-bit prescaler
  - PWM generation (separated only)
  - Real-Time Clock (concatenated only)
- Count up or down
- Simple PWM Support for timer synchronization, daisy-chains, and stalling during debug
- Can trigger ADC samples or DMA transfers

In this experiment, the timer is used in periodic mode. The capabilities of the general purpose timer in periodic mode are given in the **Table 2-1**.

Table 2-1: Capabilities of General Purpose Timer in Periodic Mode

Timer Use	Count Direction	Counter Size		Prescaler Size		Prescaler Behaviour (Count Diection)
		16/32-bit GPTM	32/64-bit Wide GPTM	16/32-bit GPTM	32/64-bit Wide GPTM	
Individual Up or Down 16	bit 32	bit 8	bit 16	bit	Individual Up or Down 16	Timer Extension (Up), Prescaler (Down
Concate-nated	Up or Down	32-bit	64-bit	-	-	N/A

2.2.2 GPIO Module

Tiva TM4C123GH6PM MCUs feature a GPIO (General Purpose Input Output) module that is composed of six physical GPIO blocks. Each block corresponds to an individual GPIO port (Port A, Port B, Port C, Port D, Port E, Port F). The GPIO pins are connected to the on-board LEDs in the LaunchPad as shown in the LaunchPad schematics in **Figure 2-2**.

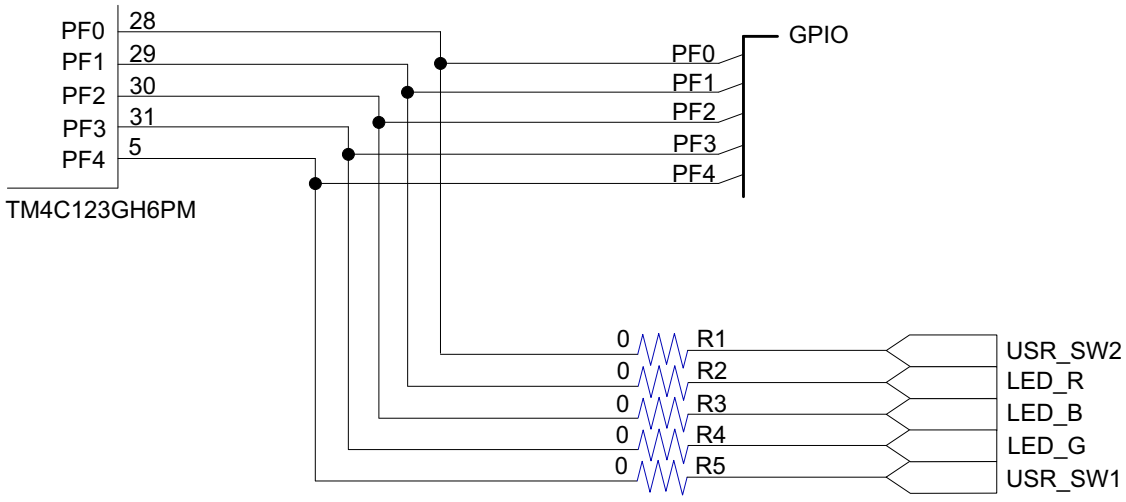


Figure 2-2 LaunchPad Schematics for GPIO Connection with LEDs

The features of GPIO are:

- Availability of up to 43 GPIOs, depending on configuration
- Highly flexible pin multiplexing allows usage of pins as GPIO or one of the several peripheral functions available
- Bit masking in both read and write operations through address lines
- Programmable control for GPIO interrupts:
  - Interrupt generation masking

- Edge-triggered on rising, falling, or both
- Level-sensitive on High or Low values

- Programmable weak pull-up, pull-down and open drain
- Programmable drive strength and slew rate control

The GPIO pins are configured with pin functions as shown in **Table 2-2** in the EK-TM4C123GXL.

Table 2-2: GPIO Pin Functions and Connected USB Devices

GPIO Pin	Pin Function	USB Device
PF4	GPIO	SW1
PF0	GPIO	SW2
PF1	GPIO	RGB LED (Red)
PF2	GPIO	RGB LED (Blue)
PF3	GPIO	RGB LED (Green)

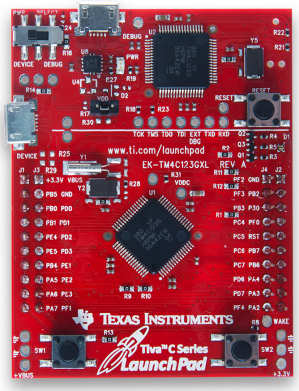
2.3 Component Requirements

2.3.1 Software Requirement

1. Code Composer Studio
2. TivaWare_C_Series

2.3.2 Hardware Requirement

Table 2-3: Components Required for the Experiment

S.No	Components	Specification	Image
1.	Tiva LaunchPad	EK-TM4C123GXL LaunchPad	
2.	USB Cable		

2.4 Software

The software for the experiment is written in C language 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 TM4C123GH6PM on the EK-TM4C123GXL using the USB interface.

2.4.1 Flowchart

The flowchart for the code is shown in **Figure 2-3**. The software written in C configures and enables the system clock to 40MHz. It then enables GPIO Port F of EK-TM4C123GXL and configures pin 3 (PF3) connected to the green LED as output. The timer is configured in periodic mode and the timer interrupt is enabled. On interrupt in periodic mode, the Timer Interrupt service Routine (ISR) reads the GPIO pin connected to the LED. If the current status of the LED is HIGH, then the processor sends a LOW to the LED and vice-versa. Thus, the green LED is toggled for each timer interrupt whose frequency is based on the timer load register value variable ui32Period in the code.

Calculation of timer period ui32Period

The timer counts for every cycle of the system clock frequency. The number of timer counts required to obtain a given frequency is calculated by

Number of clock cycles = System Clock Frequency / Desired Frequency

In the program, to toggle the GPIO at 10Hz and 50% duty cycle,

ui32Period = Number of clock cycles * Duty cycle = (40 MHz /10 Hz) * (50/100) =2 * 10^6 counts

---

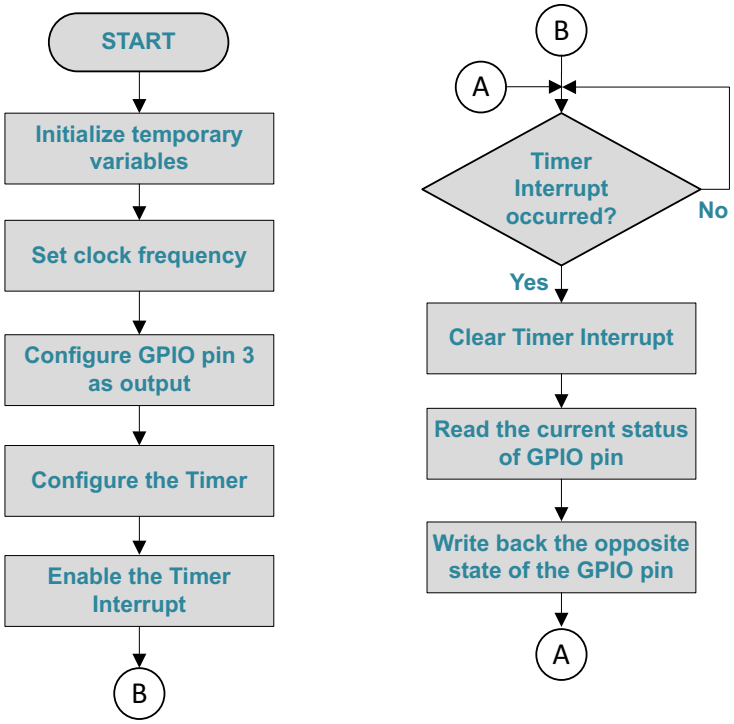


Figure 2-3 Flowchart for Interrupt Programming with GPIO

2.4.2 C Program Code for Interrupt Programming with GPIO

```
#include <stdint.h>
#include <stdbool.h>
#include "inc/tm4c123gh6pm.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/sysctl.h"
#include "driverlib/interrupt.h"
#include "driverlib/gpio.h"
#include "driverlib/timer.h"


int main(void)
{
uint32_t ui32Period;
SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|
SYSCTL_OSC_MAIN);
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|
GPIO_PIN_3);
SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);
TimerConfigure(TIMER0_BASE, TIMER_CFG_PERIODIC);
ui32Period = (SysCtlClockGet() / 10) / 2;
TimerLoadSet(TIMER0_BASE, TIMER_A, ui32Period -1);
IntEnable(INT_TIMER0A);
TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
IntMasterEnable();
TimerEnable(TIMER0_BASE, TIMER_A);
while(1)
{
}
}
void Timer0IntHandler(void)
{
// Clear the timer interrupt
TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
// Read the current state of the GPIO pin and+
```

```
// write back the opposite state
if(GPIOPinRead(GPIO_PORTF_BASE, GPIO_PIN_2))
{
GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 0);
}
else
{
GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 4);
}
```

Table 2-4: API Functions used in the Application Program

API Function	Parameters	Description
TimerConfig-ure(uint32_t ui32Base, uint32_t ui32Config)	• **ui32Base** is the base address of the timer module • **ui32Config** is the configu-ration for the timer	This function config-ures the operating mode of the timer(s).
SysCtlClockGet(void)		This function deter-mines the clock rate of the processor clock
TimerLoadSet(uint32_t ui32Base, uint32_t ui32Timer, uint32_t ui32Value)	• **ui32Base** is the base address of the timer module • **ui32Timer** specifies the timer(s) to adjust; must be one of **T**IMER_A, TIMER_B, or TIMER_BOTH. Only TIM-ER_A should be used when the timer is configured for full-width operation • **ui32Value** is the load value	This function config-ures the timer load value
TimerIntEnable (uint32_t ui32Base, uint32_t ui32IntFlags)	• **ui32Base** is the base address of the timer module • **ui32IntFlags** is the bit mask of the interrupt sources to be enabled	This function enables the indicated timer inter-rupt sources
TimerEnable(uint32_t ui32Base, uint32_t ui32Timer)	• **ui32Base** is the base address of the timer module • **ui32Timer** specifies the timer(s) to enable; must be one of TIMER_A, TIMER_B, or TIMER_BOTH	This function enables operation of the timer module

2.5 Procedure

1. Connect the EK-TM4C123GXL to the PC using the USB cable supplied.

2. Build, program and debug the code into the EK-TM4C123GXL using CCS to view the status of the green LED.

2.6 Observation

The green LED blinks on successful execution of the program. The rate of blinking can be altered by changing the ui32Period variable in the program.

2.7 Summary

In this experiment, we have successfully configured the Timer interrupt and written an ISR for the general purpose timer to change the GPIO pin status.

2.8 Exercise

Modify the program for a different timer toggling frequency by loading a new value of ui32Period in the third argument of TimerLoadSet(TIMER0_BASE, TIMER_A, ui32Period -1) function.

Experiment 3

Hibernation and Wakeup on RTC Interrupt

Topics **Page**

3.1 Objective

The main objective of this experiment is to understand and configure the Hibernation module of the TM4C123GH6PM device to place the device in a low power state and then to wake up the device on RTC (Real-Time Clock) interrupt.

3.2 Introduction

The TM4C123GH6PM consists of a battery-backed hibernation module that provides logic to switch power off to the main processor and its peripherals while the processor is idle. In this experiment, the hibernation module of the TM4C123GH6PM processor is turned on by the software code. This is indicated by the green LED connected to a GPIO port (PF3). When the LED is ON, the device is in wake up mode, and when the LED is OFF the device is in hibernate mode and the Hibernation module is enabled. The device in hibernation can be woken up by two ways:

- An external signal, SW2 input to the GPIO wake up pin
- RTC wake up
- In this experiment, the RTC is used to wake up the processor after 5 seconds. The RTC is used in the RTC Match-Seconds mode. (Refer to TM4C123GH6PM Data Sheet for more details).

The functional block diagram as shown in **Figure 3-1** illustrates the working principle of the experiment.

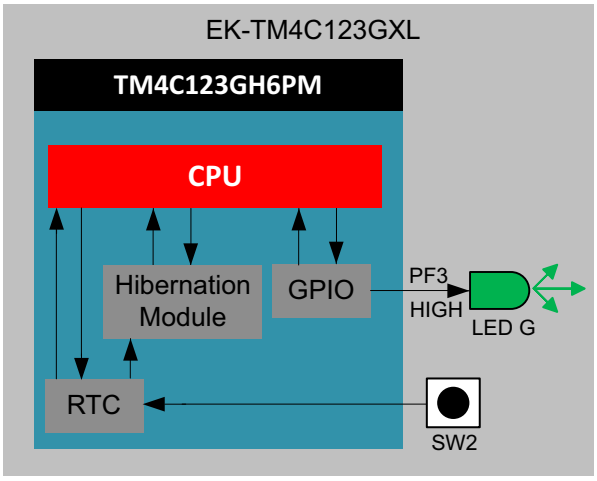


Figure 3-1 Functional Block Diagram

3.2.1 Hibernation Module

The Hibernation Module manages removal and restoration of power to the device and hence helps reduce system power consumption. When the processor and peripherals are idle, power can be completely cut off with only the Hibernation module powered up. Power can be restored based on an external signal or after a certain time using the built-in Real-Time Clock (RTC). Power for the Hibernation module can be independently supplied from an external battery or an auxiliary power supply.

Once in hibernation, the module signals an external voltage regulator to turn the power back on when an external pin (WAKE) is asserted or when the internal RTC reaches a certain value. The

Hibernation module can also detect when the battery voltage is low and optionally prevent hibernation or wake from hibernation when the battery voltage falls below a certain threshold.

The Hibernation module of the Tiva TM4C123GH6PM MCU has the following features:

- 32-bit real-time seconds counter (RTC) with 1/32,768 second resolution and a 15-bit sub-seconds counter
  - 32-bit RTC seconds match register and a 15-bit sub seconds match for timed wake-up and interrupt generation with 1/32,768 second resolution
  - RTC predivider trim to make fine adjustments to the clock rate
- Two mechanisms for power control
  - System power control using discrete external regulator
  - On-chip power control using internal switches under register control
- Dedicated pin to wake up the processor using an external signal
- RTC operational and hibernation memory valid as long as V_{DD} or V_{BAT} is valid
- Low-battery detection, signaling, and interrupt generation, with optional wake on low battery
- GPIO pin state can be retained during hibernation
- Clock source from a 32.768-kHz external crystal or oscillator
- Sixteen 32-bit words of battery-backed memory to save state during hibernation
- Programmable interrupts for:
  - RTC match
  - External wake
  - Low battery

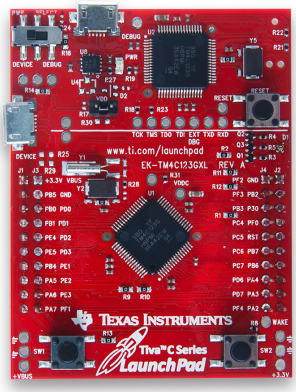
3.3 Component Requirements

3.3.1 Software Requirement

1. Code Composer Studio
2. TivaWare_C_Series

3.3.2 Hardware Requirement

Table 3-1: Components Required for the Experiment

S.No	Components	Specification	Image
1.	Tiva LaunchPad	EK-TM4C123GXL LaunchPad	
2.	USB Cable		

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 TM4C123GH6PM on the EK-TM4C123GXL using the USB interface.

3.4.1 Flowchart

The flowchart for the experiment is shown in **Figure 3-2**. The C program code enables and configures the system clock to 40MHz. The green LED on the EK-TM4C123GXL (connected to GPIO PF3) indicates if the device is in hibernate mode or wake up mode. For this purpose, the GPIO Port F of TM4C123GH6PM processor is enabled and pin 3 (PF3) configured as output. A HIGH is written on the GPIO pin to turn the green LED on.

The C code then enables the Hibernation module and configures the clock that feeds the module. It enables the GPIO pin state to be retained during hibernation and provides a 4-second delay for the user to observe the LED which is ON. The time interval for which the LED is ON can be programmed by specifying a the delay count value in the SysCtlDelay() function.

Calculation of delay

The number of counts required to get a time delay is given by

Number of Counts = Time delay required * System Clock Frequency

In hibernation mode, the processor uses the clock from the external crystal or oscillator. Hence in the program, the clock source used in hibernation mode is 16MHz. To get a required time delay of 4 second

Number of counts = $4 * 16 * 10^6 = 64 * 10^6$

The program configures the device to be woken up either by an input on the wake-up pin or by the RTC. It sets the RTC wake up parameters and turns the RTC on. The wake-up time is set to 5 seconds. The wake-up pin on the EK-TM4C123GXL is connected to SW2 on the board. When the switch SW2 is pressed and held, the device is woken up from the hibernation mode. Hence, the device can be woken up by either the switch SW2 or the RTC.

After setting the wake-up pin parameters, the device enters the hibernation mode and the green LED turns off. The Hibernation module is powered by a battery or an auxiliary power supply. If the supply to the Hibernation module is low or not present, the device may not enter the hibernate mode. Once in hibernate mode, the device can be woken up by pressing and holding the switch SW2 or by the RTC which is set for 5 seconds, whichever occurs earlier.

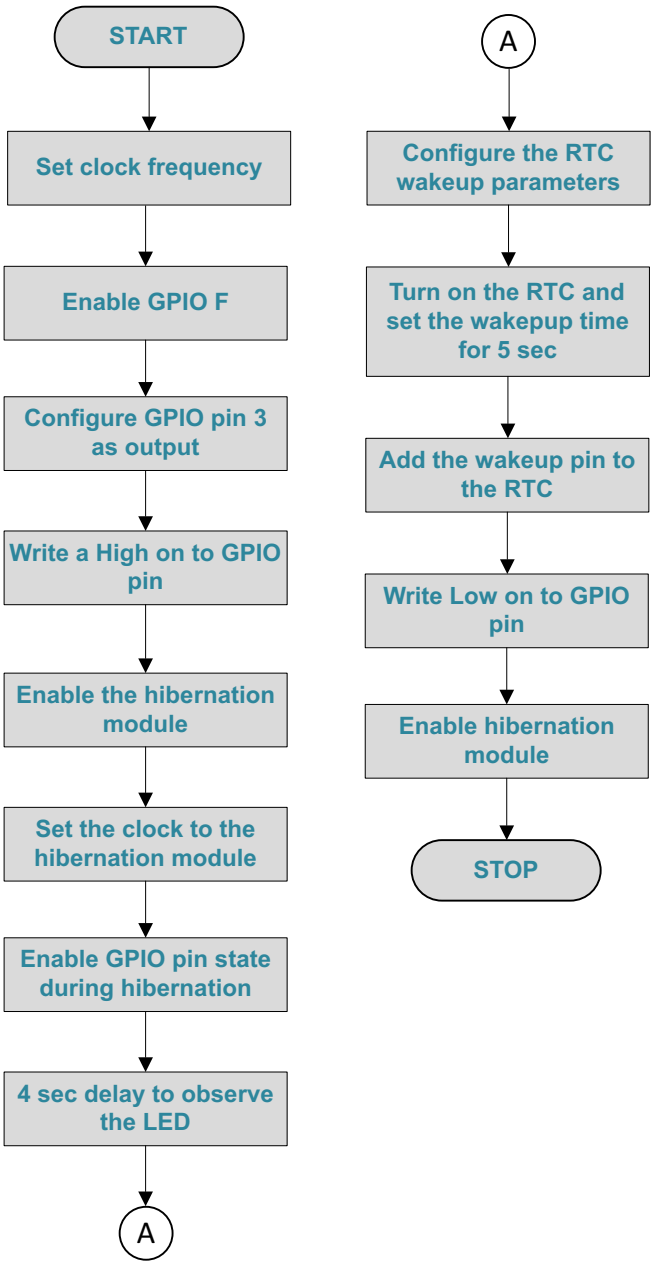


Figure 3-2 Flowchart for Hibernate Mode and Wake up using RTC

3.4.2 C Program Code for Hibernate Mode and Wake up using RTC

```
#include <stdint.h>
#include <stdbool.h>
#include "utils/ustdlib.h"
#include "inc/hw_types.h"
#include "inc/hw_memmap.h"
#include "driverlib/sysctl.h"
#include "driverlib/pin_map.h"
#include "driverlib/debug.h"
#include "driverlib/hibernate.h"
#include "driverlib/gpio.h"
int main(void)
{
SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|
SYSCTL_OSC_MAIN);
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|
GPIO_PIN_3);
GPIOPinWrite(GPIO_PORTF_BASE,GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 0x08);
SysCtlPeripheralEnable(SYSCTL_PERIPH_HIBERNATE);
HibernateEnableExpClk(SysCtlClockGet());
HibernateGPIORetentionEnable();
SysCtlDelay(64000000);
HibernateRTCSet(0);
HibernateRTCEnable();
HibernateRTCMatchSet(0,5);
HibernateWakeSet(HIBERNATE_WAKE_PIN | HIBERNATE_WAKE_RTC);
GPIOPinWrite(GPIO_PORTF_BASE,GPIO_PIN_3, 0x00);
HibernateRequest();
while(1)
{
}
}
```

Table 3-2: API Functions Used in the Application Program

API Function	Parameters	Description
HibernateEnableExpClk(uint32_t ui32HibClk)	• **lui32HibClk** is the rate of the clock supplied to the Hibernation module.	This function enables the Hibernation module for operation.
HibernateGPIORetentionDisable(void)		This function disables the retention of the GPIO pin state during hibernation and allows the GPIO pins to be controlled by the system.
HibernateRTCSet (uint32_t ui32RTCValue)	• **ui32RTCValue i**s the new value for the RTC.	This function sets the value of the RTC
HibernateRTCEnable (void)		This function enables the RTC in the Hibernation module.
HibernateRTCMatchSet (uint32_t ui32Match, uint32_t ui32Value)	• **ui32Match** is the index of the match register • **ui32Value** is the value for the match register	This function sets a match register for the RTC.
HibernateWakeSet (uint32_t ui32WakeFlags)	• **ui32WakeFlags** specifies which conditions should be used for waking.	Configures the wake conditions for the Hibernation module.
HibernateRequest(void)		This function requests the Hibernation module to disable the external regulator, thus removing power from the processor and all peripherals.

3.5 Procedure

1. Connect the EK-TM4C123GXL to the PC using the USB cable supplied.

2. Build, program and debug the code to view the status of the green LED.

3. After 4 seconds, the green LED will switch off, indicating that the TM4C123GH6PM device has gone into hibernation.

4. Observe the status of the LED. After 5 seconds (RTC wake up time set in the code), the LED turns ON, indicating the RTC has woken the processor.

5. Also you can press and hold the SW2 button located at the lower right corner of the EK-TM4C123GXL to wake up the processor at any time.

6. On wake up the green LED will turn ON again.

3.6 Observation

The green LED turns ON for 4 seconds, then OFF for about 5 seconds. This process repeats. The real-time-clock (RTC) wakes the processor from hibernate mode after 5 seconds. Also, note the processor can be woken up with SW2 at any time. The green LED being ON indicates that the processor is in wake-up condition and green LED being OFF indicates the processor is in Hibernate mode.

3.7 Summary

In this experiment, we have understood and configured the hibernation module of TM4C123GH6PM. Also, we have learnt to implement the RTC interrupt to wake up the processor from hibernation.

3.8 Exercise

Write a program to configure hibernation mode and wake up the EK-TM4C123GXL when SW2 of EK-TM4C123GXL is pressed.

Experiment 4

Interfacing Potentiometer with TIVA GPIO

Topics **Page**

4.1 Objective

The main objective of this experiment is to interface a potentiometer with EK-TM4C123GXL GPIO (PE3) by configuring it as an Analog Input (AN0) and to observe its corresponding 12-bit digital value. In this experiment, we will also understand the Analog-to-Digital Conversion (ADC) and processing of the analog signals in a digital environment.

4.2 Introduction

In this experiment, the potentiometer is interfaced to the TM4C123GH6PM processor via a GPIO (PE3) configured as analog input. The potentiometer input is read by the 12-bit ADC peripheral of the processor and the analog input voltage is converted to corresponding digital values. These 12-bit digital values are stored in a temporary register.The functional block diagram as shown in **Figure 4-1** illustrates the working principle of the experiment.

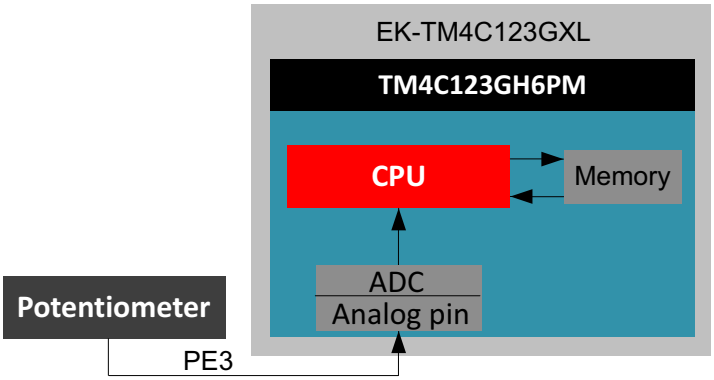


Figure 4-1 Functional Block Diagram

4.2.1 Analog to Digital Conversion Module

TM4C123GH6PM MCUs consists of two ADC modules (ADC0 and ADC1) that can be used to convert continuous analog voltages to discrete digital values. Each ADC module has a 12-bit resolution, operates independently, can execute different sample sequences, can sample any of the shared analog input channels and generate interrupts & triggers based on the conversion process.

The features of the ADC module are:

- Two 12-bit 1MSPS ADCs giving a digital range of 0 to 4095
- 12 shared analog input channels
- Single ended & differential input configurations
- On-chip temperature sensor
- Maximum sample rate of one million samples/second (1MSPS)
- Fixed references (VDDA/GNDA) due to pin-count limitations
- 4 programmable sample conversion sequencers per ADC
- Separate analog power & ground pins
- Flexible trigger control
- 2x to 64x hardware averaging
- 8 Digital comparators per ADC

- 2 Analog comparators
- Optional phase shift in sample time between ADC modules is programmable from 22.5° to 337.5°

The TM4C123GH6PM ADC collects sample data by using a programmable sequence-based approach. Each sample sequence is a fully programmed series of consecutive (back-to-back) samples, allowing the ADC to collect data from multiple input sources without having to be re-configured or serviced by the processor. The programming of each sample in the sample sequence includes parameters such as the input source and mode (differential versus single-ended input), interrupt generation on sample completion, and the indicator for the last sample in the sequence. Each ADC module of the TM4C123GH6PM has four sample sequencers that handle the sampling control and data capture. All of the sequencers are identical in implementation except for the number of samples that can be captured and the depth of the FIFO (First In First Out).

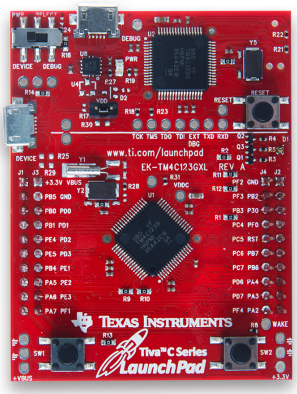
4.3 Component Requirements

4.3.1 Software Requirement

1. Code Composer Studio
2. TivaWare_C_Series

4.3.2 Hardware Requirement

Table 4-1: Components Required for the Experiment

S.No	Components	Specifications	Images
1.	Tiva LaunchPad	EK-TM4C123GXL LaunchPad	
2.	USB cable		
3.	Potentiometer	10kΩ, 1/2W	
4.	Multimeter		

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 TM4C123GH6PM on the EK-TM4C123GXL using the USB interface.

4.4.1 Flowchart

The flowchart for the program is shown in **Figure 4-2**. The software controls the GPIO and ADC peripherals of the processor to convert the potentiometer input into digital values and store the converted 12-bit digital values in an array variable. A temporary array variable 'ui32ADC0Value' is initialized in the program for storing the 12-bit digital output of the ADC. The system clock is set to 40MHz and the GPIO port E (PE3) configured and enabled as analog input to the ADC0 module which is used for A/D conversion. The ADC0 module is configured to interrupt on conversion and then enabled. The program then waits for an interrupt from the ADC0. The ADC0 module reads the analog input from the potentiometer and converts to digital output and interrupts the processor. On interrupt from the ADC0, the program enables the processor trigger event and stores the converted digital value in a temporary variable. The process continues for the next analog input from the potentiometer.

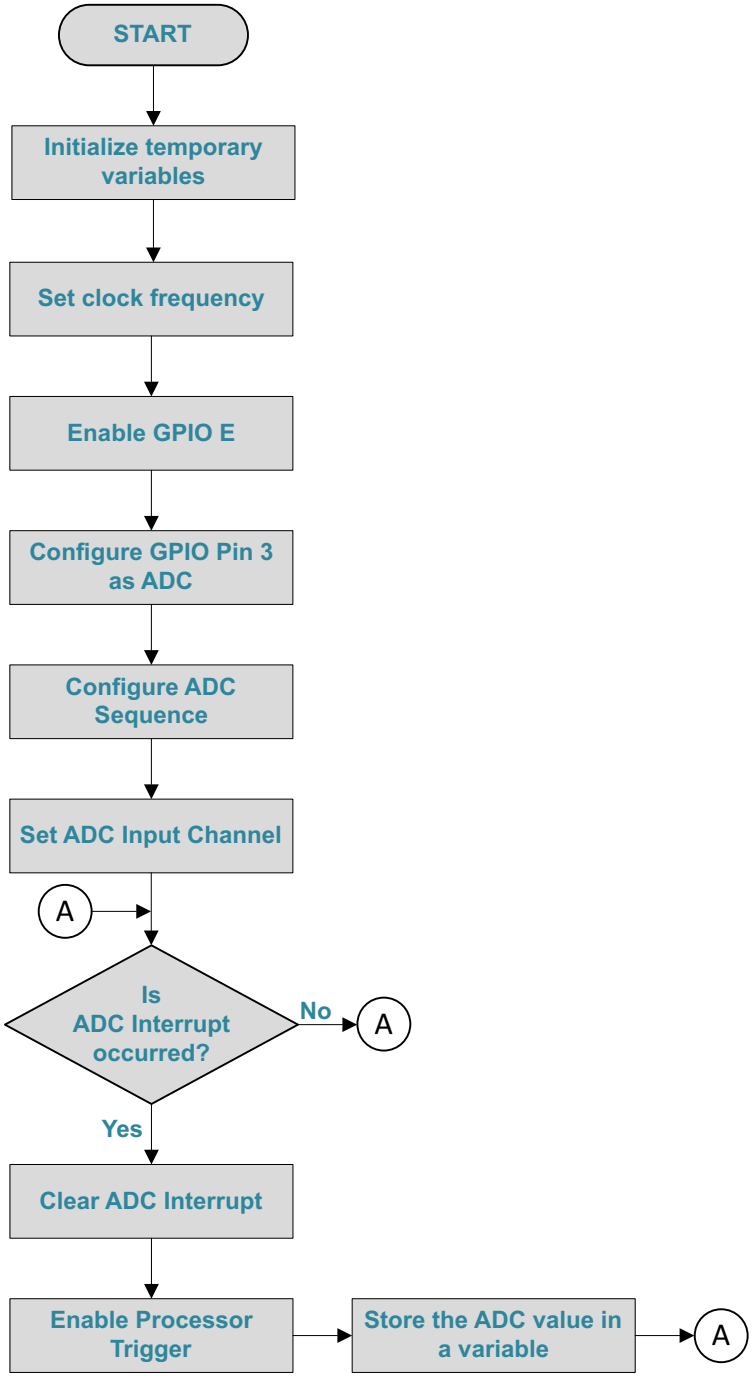


Figure 4-2 Flowchart for ADC Conversion of Potentiometer Input via GPIO

4.4.2 C Program for ADC Conversion of Potentiometer Input via GPIO

```
// HEADER FILES
#include<stdint.h>
#include<stdbool.h>
#include"inc/hw_memmap.h"
#include"driverlib/gpio.h"
#include"inc/hw_types.h"
#include"driverlib/debug.h"
#include"driverlib/sysctl.h"
#include"driverlib/adc.h"
// TO STORE THE VALUE IN VARIABLE ui32ADC0Value FOR EVERY SAMPLING
uint32_tui32ADC0Value[1];
intmain(void)
{
SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_OSC_MAIN|
SYSCTL_XTAL_16MHZ); // SYSTEM CLOCK AT 40MHZ
SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);    // ENABLE ADC0 MODULE
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);   // ENABLE GPIO for ADC0
MODULE
GPIOPinTypeADC(GPIO_PORTE_BASE,GPIO_PIN_3);// ENABLE AN0 OF ADC0 MODULE
// ADC0 MODULE, TRIGGER IS PROCESSOR EVENT, SEQUENCER 0 IS CONFIGURED
ADCSequenceConfigure(ADC0_BASE, 1, ADC_TRIGGER_PROCESSOR, 0);
// ADC0 MODULE, SEQUENCER 0 , FOR 1 SAMPLING, INPUT IS FROM CHANNEL 0 PE3
ADCSequenceStepConfigure(ADC0_BASE, 1, 0, ADC_CTL_CH0);
// ENABLE THE SEQUENCE 1 FOR ADC0
ADCSequenceEnable(ADC0_BASE, 1);
    while(1)
    {
    // CLEAR INTERRUPT FLAG FOR ADC0, SEQUENCER 1
    ADCIntClear(ADC0_BASE, 1);
    // TRIGGER IS GIVEN FOR ADC 0 MODULE, SEQUENCER 1
    ADCProcessorTrigger(ADC0_BASE, 1);
// STORE THE CONVERTED VALUE FOR ALL DIFFERENT SAMPLING IN ARRAY
//ui32ADC0Value
ADCSequenceDataGet(ADC0_BASE, 1, ui32ADC0Value);
  }
}
```

Table 4-2: API Functions Used in the Application Program

API Function	Parameters	Description
GPIOPin-TypeADC(uint32_t ui32Port, uint8_t ui8Pins)	• **ui32Port** is the base address of the GPIO port • **lui8Pins** is the bit-packed representation of the pin(s)	Configures pin(s) for use as analog-to-digital converter inputs.
ADCSequenceConfig-ure(uint32_t ui32Base, uint32_t ui32Sequence-Num, uint32_t ui32Trigger, uint32_t ui32Priority)	• **ui32Base** is the base address of the ADC module • **ui32SequenceNum** is the sample sequence number • **ui32Trigger** is the trigger source that initiates the sample sequence; must be one of the **ADC_TRIGGER**_ values **ui32Priority** is the relative priority of the sample sequence with respect to the other sample sequences	Configures the trigger source and priority of a sample sequence.
ADCSequenceStep-Configure(uint32_t ui32Base, uint32_t ui32Sequence-Num, uint32_t ui32Step, uint32_t ui32Config)	• **ui32Base** is the base address of the ADC module • **ui32SequenceNum** is the sample sequence number • **ui32Step** is the step to be configured • **ui32Config** is the configuration of this step; must be a logical OR of **ADC_CTL_TS, ADC_CTL_IE, ADC_CT-L_END, ADC_CTL_D**, one of the input channel selects (**ADC_CTL_CH0** through **ADC_CTL_CH23**), and one of the digital comparator selects **(ADC_CTL_CMP0** through **ADC_CTL_CMP7**)	Configure a step of the sample sequencer.
ADCSequenceEn-able(uint32_t ui32Base, uint32_t ui32Sequence-Num)	• **ui32Base** is the base address of the ADC module • **ui32SequenceNum** is the sample sequence number	Enables a sample sequence.
ADCIntClear(uint32_t ui32Base, uint32_t ui32Sequence-Num)	• **ui32Base** is the base address of the ADC module • **ui32SequenceNum** is the sample sequence number	Clears sample sequence interrupt source.

Table 4-2: API Functions Used in the Application Program

API Function	Parameters	Description
ADCProcessorTrigger(uint32_t ui32Base, uint32_t ui32SequenceNum)	• **ui32Base** is the base address of the ADC module • **ui32SequenceNum** is the sample sequence number, with **ADC_TRIGGER_WAIT** or **ADC_TRIGGER_SIGNAL** optionally ORed into it	Causes a processor trigger for a sample sequence.
ADCSequenceDataGet(uint32_t ui32Base, uint32_t ui32SequenceNum, uint32_t *pui32Buffer)	• **ui32Base** is the base address of the ADC module • **ui32SequenceNum** is the sample sequence number • **pui32Buffer** is the address where the data is stored	Gets the captured data for a sample sequence.

4.5 Procedure

4.5.1 Hardware Setup

Figure 4-3 shows the hardware connections for the potentiometer with the EK-TM4C123GXL.

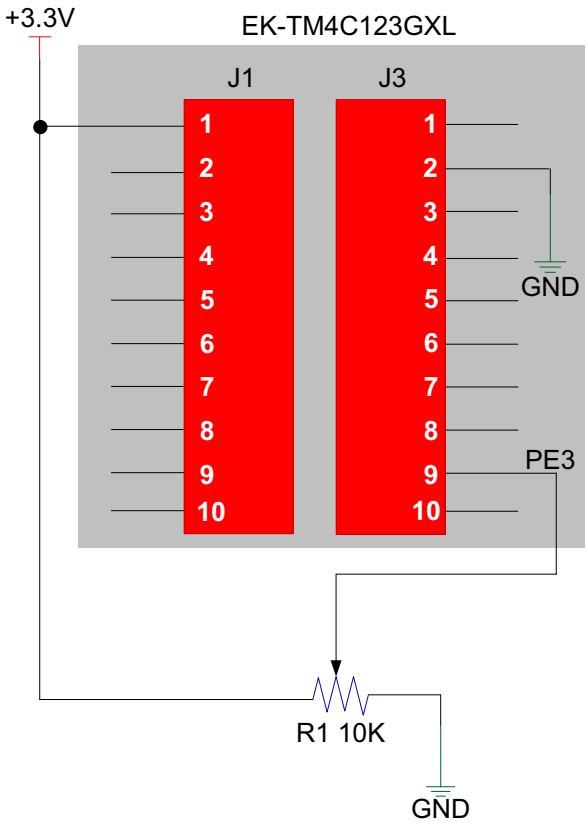


Figure 4-3 Connection Diagram for Potentiometer with EK-TM4C123GXL

The procedure to be followed for the hardware connections is:

1. Connect one lead of the potentiometer (Vcc) to +3.3V DC Supply Voltage (J1 connector Pin 1).

2. Connect the other lead of the potentiometer to the GND pin (J3 connector, Pin 2).

3. Connect the center lead of the potentiometer to pin PE3 which is the Analog Channel AN0 (J3 connector, Pin 9).

4. The Multimeter can be probed at the center lead of the potentiometer to observe the analog voltage input.

5. This configuration varies the Analog Voltage from 0V to 3.3V depending on the wiper position on PE3 which is the AN0 or Analog Input 0 of the TM4C123GH6PM.

6. The current configuration connects the VREFP of ADC to VDDA which is 3.3V and VREFP to GND which is 0V.

Design Calculations

For a 12-bit ADC, the range of the conversion values is from 0 to 4095 (0 to 0xfff in Binary).

With the Conversion voltage span and the bit range we can calculate the following

Resolution = 3.3V/ 4096 = 3300mV / 4096 = 0.8mV

The resolution is the change in voltage per unit change in ADC code

For example, if the digital code is equivalent to 1000 in decimal,

The equivalent analog voltage for 1000 = Resolution * decimal equivalent of the digital code = 0.8mV *1000 = 800 mV = 0.8V

After connecting the hardware, the setup appears as shown in **Figure 4-4**.

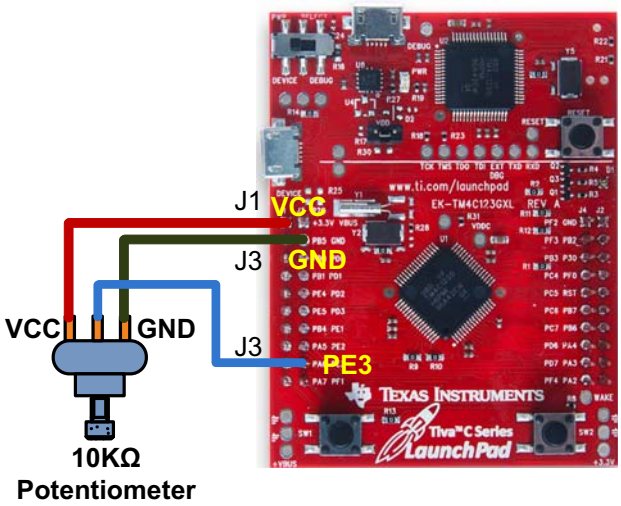


Figure 4-4 Hardware Setup

4.5.2 Implementing the Software

1. Build, program and debug the code.

2. Vary the position of the potentiometer wiper and observe the corresponding digital output stored in the register ui32ADC0Value on the CCS watch window.

4.6 Observation

Based on the wiper position of the potentiometer, the analog input varies. The corresponding digital output is stored in the variable ui32ADC0Value and is observed on the CCS watch window.

Figure 4-5 shows the snapshots of digital output for three different positions of the potentiometer wiper:

- Minimum position
- Center position and
- Maximum position

Digital output for other intermediate positions of the potentiometer can be observed and tabulated to understand the Analog-to-Digital Conversion function and its corresponding values.

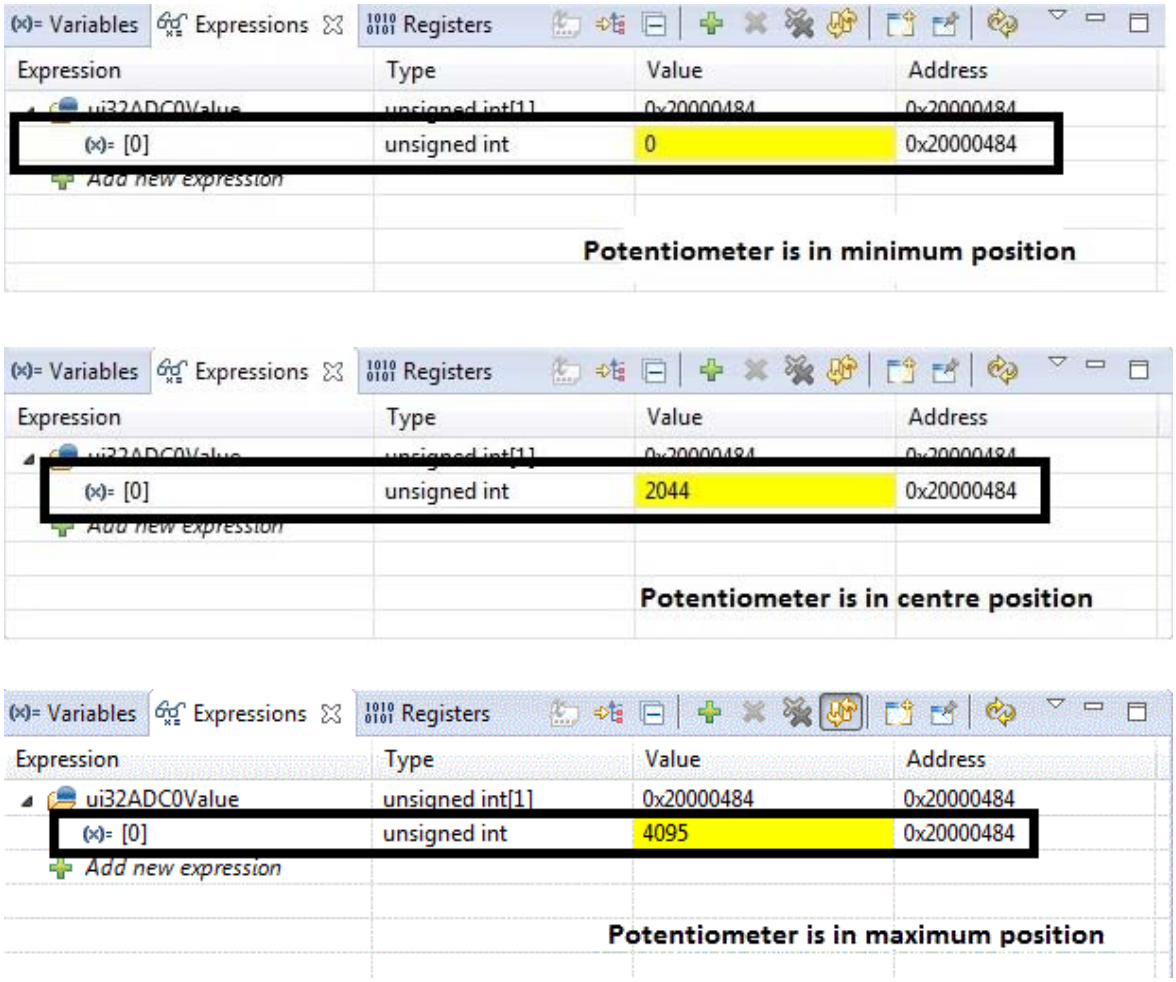


Figure 4-5 Digital Output for Three Different Positions of the Potentiometer

4.7 Summary

In this experiment, we have successfully read the potentiometer input via GPIO and used the on chip ADC to convert the analog voltage to a digital value. We have also learnt to configure the ADC for analog to digital conversion.

4.8 Exercise

1. Tabulate ten different positions of the potentiometer and note down the digital value. Also, calculate the equivalent analog value.

2. Use the ADC module to obtain the analog value from the internal temperature sensor.

3. Configure Dual ADC modules to read from 2 analog inputs (could be from 2 potentiometers).

4. What are the trigger control mechanisms for this ADC?

5. What does the resolution refer on ADC Specification?

6. The current sampling method is single ended sampling. This ADC could also be configured to do differential sampling. What is the difference between the two methods of sampling?

Texas Instruments

Experiment 5
PWM Generation

Topics **Page**

5.1 Objective

The main objective of this experiment is to generate a Pulse Width Modulation (PWM) signal using the PWM Module in the TM4C123GH6PM processor.

5.2 Introduction

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. Typical applications for PWM are switching power supplies, motor control, servo positioning and lighting control.

Figure 5-1 shows a PWM signal for a simple analog signal. A large analog voltage increases the pulse width of the digital signal and a small analog voltage decreases the pulse width.. Hence, the analog voltage is encoded in the duty cycle of the digital signal.

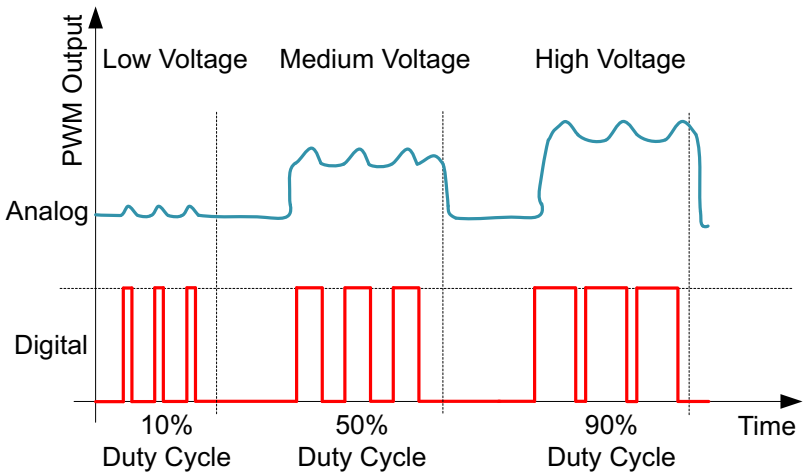


Figure 5-1 PWM Output for Analog Signal

The TM4C123GH6PM has two PWM modules. Each PWM module consists of:

- Four PWM generator blocks
- A control block which determines the polarity of the signals and which signals are passed to the pins

Each PWM generator block produces:

- Two independent output signals of the same frequency
- A pair of complementary signals with dead-band generation (for H-bridge circuit protection)
- Eight outputs total

The Port D pin 0 (PD0) of the TM4C123GH6PM processor has multiple functions. It can be used as one of the following:

- GPIO
- ADC input
- I^2C module 3 clock
- Motion Control Module 0 PWM

- Motion Control Module 1 PWM
- SSI Module Clock
- Timer 2 Capture/Compare

In this experiment, the GPIO pin PD0 is configured as M1PWM0 (Motion Control Module 1 PWM0). The output signal on this pin is controlled by the Module 1 PWM Generator 0. Different pulse widths (duty cycle) for the PWM signal is set in the software program and resultant PWM signal is observed using the oscilloscope. The functional block diagram as shown in **Figure 5-2** illustrates the working principle of the experiment.

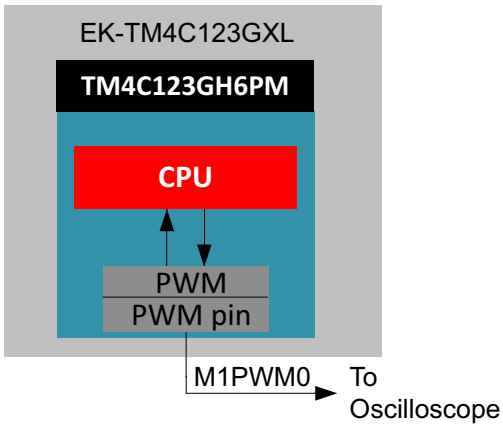


Figure 5-2 Functional Block Diagram

5.3 Component Requirements

5.3.1 Software Requirement

1. Code Composer Studio
2. TivaWare_C_Series

5.3.2 Hardware Requirement

Table 5-1: Components Required for the Experiment

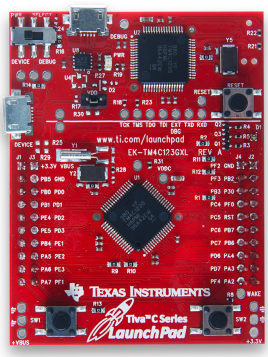
S.No	Components	Specifications	Images
1.	Tiva LaunchPad	EK-TM4C123GXL LaunchPad	

Table 5-1: Components Required for the Experiment

S.No	Components	Specifications	Images
2.	USB cable		
3.	Oscilloscope		

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 TM4C123GH6PM on the EK-TM4C123GXL using the USB interface.

5.4.1 Flowchart

The flowchart for the experiment is shown in **Figure 5-3**. The C program code enables and configures the system clock to 40MHz. It sets the PWM frequency based on **Equation 5.2**.

Design calculations

The PWM clock is the system clock divided by a factor, and the factor is configured to 64 in this program. To correlate with variation as a percentage change, the PWM period is set at 100. This allows the PWM period to change from 1 to 100.

System Clock = 40MHz

PWM Clock = System Clock /Dividing factor = 40MHz /64 = 625kHz (5.1)

The dividing factor can range from 2 to 64 for the PWM generator.

PWM Period = 100

PWM frequency = PWM Clock / PWM Period (5.2)

PWM frequency = 625 kHz / 100 = 6.25kHz

It initializes temporary variables for PWM pulse width adjustment. The GPIO port D pin PD0 is configured and enabled as PWM output. The code then configures and enables the PWM generator with counting mode of operation, period to 100 and enables bit 0 as PWM output (the output state of the PWM module). A PWM signal can be observed at output pin PD0 of the processor. Further, the code produces PWM outputs with varying pulse width in steps of 10% of duty cycle of the PWM generator. This is implemented in a while loop with each PWM signal output for 300 ms.

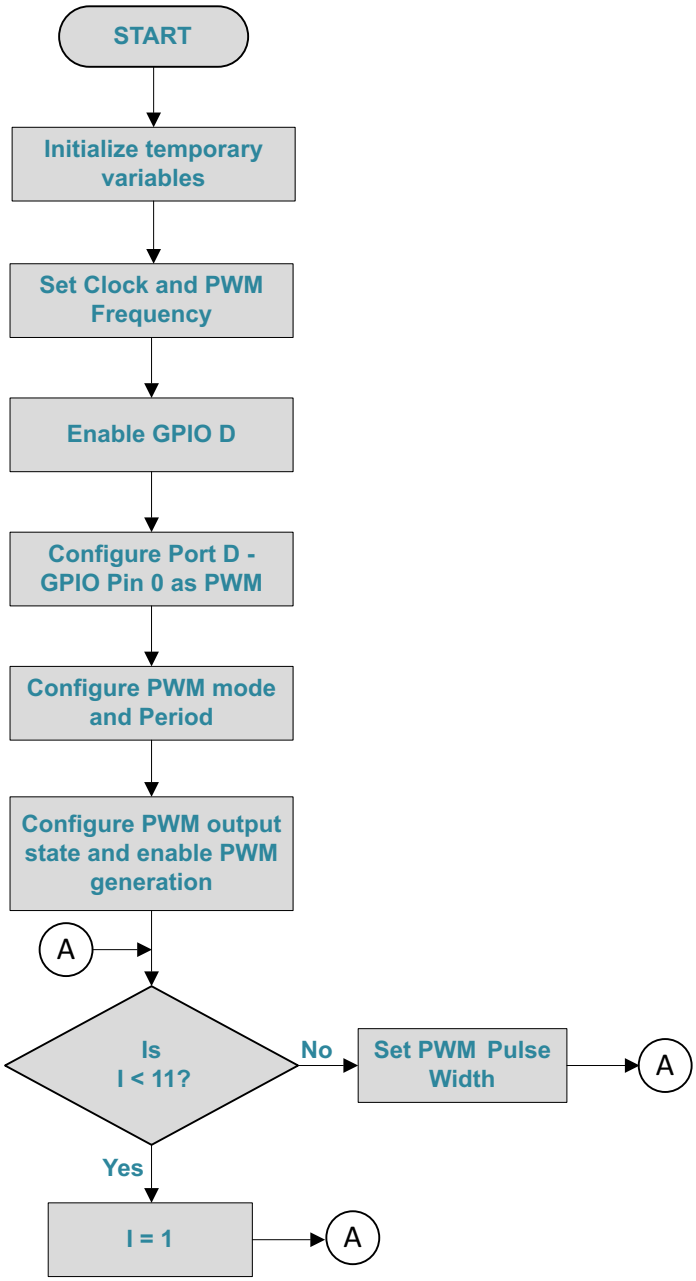


Figure 5-3 Flowchart for PWM Generation

5.4.2 C Program Code for PWM Generation

```
// HEADER FILES
#include<stdint.h>
#include<stdbool.h>
#include"inc/hw_memmap.h"
#include"inc/hw_types.h"
```

```
#include"driverlib/sysctl.h"
#include"driverlib/gpio.h"
#include"driverlib/debug.h"
#include"driverlib/pwm.h"
#include"driverlib/pin_map.h"
#include"inc/hw_gpio.h"
#include"driverlib/rom.h"

//MAIN FUNCTION
intmain(void)
{
inti=0;
volatileuint8_t ui8Adjust;//VARIABLE FOR PWM_WIDTH SET AS % FROM 0 TO 100
//CLOCK SETTINGS - SET SYSTEM CLOCK TO 40MHz
SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_OSC_MAIN|
SYSCTL_XTAL_16MHZ);
//SET PWM CLOCK AS SYSTEM CLOCK DIVIDED BY 64
SysCtlPWMClockSet(SYSCTL_PWMDIV_64);
//PERIPHERAL CONFIGURATION
SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM1);  //PWM PERIPHERAL ENABLE
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD); //GPIO FOR PWM1
GPIOPinTypePWM(GPIO_PORTD_BASE, GPIO_PIN_0); //GPIO PD0 FOR PWM1
GPIOPinConfigure(GPIO_PD0_M1PWM0);           //PD0 AS M1PWM0
PWMGenConfigure(PWM1_BASE, PWM_GEN_0, PWM_GEN_MODE_DOWN);
//SET PWM GENERATOR WITH MODEOF OPERATION AS COUNTING
PWMGenPeriodSet(PWM1_BASE, PWM_GEN_0,100); //SET PERIOD OF PWM GENERATOR
PWMOutputState(PWM1_BASE, PWM_OUT_0_BIT, true);     //ENABLE BIT0 OUTPUT
PWMGenEnable(PWM1_BASE, PWM_GEN_0);         //ENABLE PWM_GEN_0 GENERATOR
while(1)
{
// PWM_WIDTH CHANGE FROM 0% - 100% IN STEPS OF 10%
for(i=1;i<11;i++)
{
ui8Adjust = i*10;   //Step size of 10%
                    //GPIO FOR PWM1
```

```
PWMPulseWidthSet(PWM1_BASE, PWM_OUT_0, ui8Adjust); //SET THE PULSE WIDTH
SysCtlDelay(4000000);                              //DELAY OF 300MILLISECONDS
}
}
}
```

Table 5-2: API Functions Used in the Application Program

API Function	Parameters	Description
SysCtlPWMClock-Set(uint32_t ui32Con-fig)	• **ui32Config** is the configu-ration for the PWM clock; it must be one of **SYSCTL_PW-MDIV_1, SYSCTL_PWM-DIV_2, SYSCTL_PWMDIV_4, SYSCTL_PWMDIV_8, lSY-SCTL_PWMDIV_16, SYSCT-L_PWMDIV_32**, or **SYSCTL_PWMDIV_64**	Sets the PWM clock configuration.
GPIOPinTy-pePWM(uint32_t ui32Port, uint8_t ui8Pins)	• **ui32Port** is the base address of the GPIO port • **ui8Pins** is the bit-packed representation of the pin(s)	Configures pin(s) for use by the PWM periph-eral.
PWMGenConfig-ure(uint32_t ui32Base, uint32_t ui32Gen, uint32_t ui32Config)	• **ui32Base** is the base address of the PWM module • **ui32Gen** is the PWM gen-erator to configure. This parameter must be one of **PWM_GEN_0, PWM_-GEN_1, PWM_GEN_2**, or **PWM_GEN_3** • **ui32Config** is the configu-ration for the PWM generator	Configures a PWM gen-erator.
PWMGenPeriod-Set(uint32_t ui32Base, uint32_t ui32Gen, uint32_t ui32Period)	• **ui32Base** is the base address of the PWM module • **ui32Gen** is the PWM gen-erator to be modified. This parameter must be one of **PWM_GEN_0, PWM_-GEN_1, PWM_GEN_2**, or **PWM_GEN_3** • **ui32Period** specifies the period of PWM generator out-put, measured in clock ticks	Sets the period of a PWM generator.
GPIOPin-TypeADC(uint32_t ui32Port, uint8_t ui8Pins)	• **ui32Port** is the base address of the GPIO port • **lui8Pins** is the bit-packed representation of the pin(s)	Configures pin(s) for use as analog-to-digital converter inputs.

Table 5-2: API Functions Used in the Application Program

API Function	Parameters	Description
PWMGenEnable(uint32_t ui32Base, uint32_t ui32Gen)	• **ui32Base** is the base address of the PWM module • **ui32Gen** is the PWM generator to be enabled. This parameter must be one of **PWM_GEN_0, PWM_GEN_1, PWM_GEN_2**, or **PWM_GEN_3**	Enables the timer/counter for a PWM generator block.
PWMPulseWidthSet(uint32_t ui32Base, uint32_t ui32PWMOut, uint32_t ui32Width)	• **ui32Base** is the base address of the PWM module • **ui32PWMOut** is the PWM output to modify. This parameter must be one of **PWM_OUT_0, PWM_OUT_1, PWM_OUT_2, PWM_OUT_3, PWM_OUT_4, PWM_OUT_5, PWM_OUT_6,** or **PWM_OUT_7** • **ui32Width** specifies the width of the positive portion of the pulse	Sets the pulse width for the specified PWM output.

5.5 Procedure

5.5.1 Hardware Setup

Figure 5-4 shows the hardware connections for the EK-TM4C123GXL.

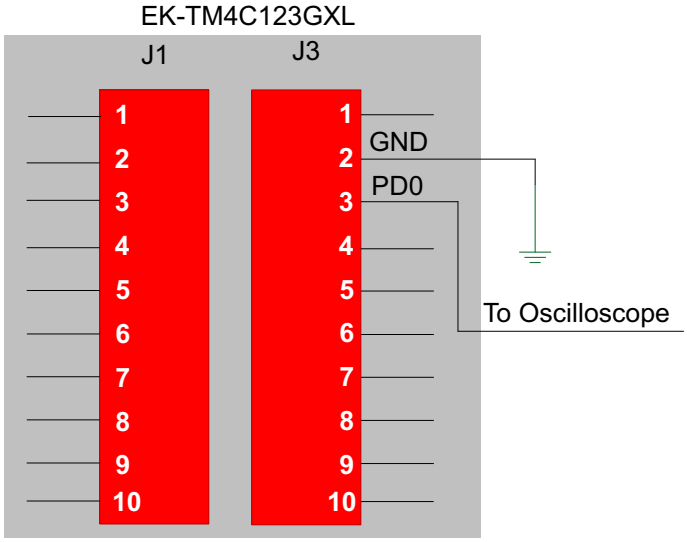


Figure 5-4 Connection Diagram for Oscilloscope with EK-TM4C123GXL

The procedure to be followed for the connections is:

1. The PWM output M1PWM0 is available on PD0 (J3 connector Pin3).

2. Probe pin 3 of J3 connector with an Oscilloscope.

3. Connect the ground at J3 connector Pin 2.

After connecting the hardware, the setup appears as shown in **Figure 5-5**.

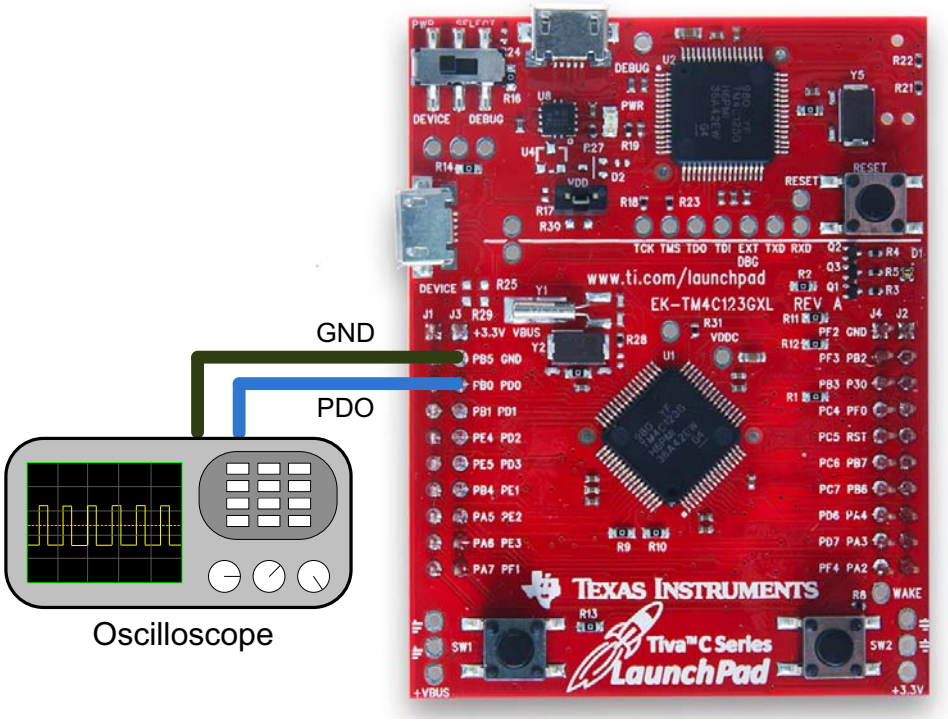


Figure 5-5 Hardware Setup

5.5.2 Implementing the Software

1. Build, program and debug the code.

2. Observe the output using an oscilloscope at J3 connector pin 3.

5.6 Observation

The oscilloscope captures for PWM generated output at 10%, 20%, 50% and 60% duty cycle are as shown in **Figure 5-6**, **Figure 5-7**, **Figure 5-8** and **Figure 5-9** respectively.

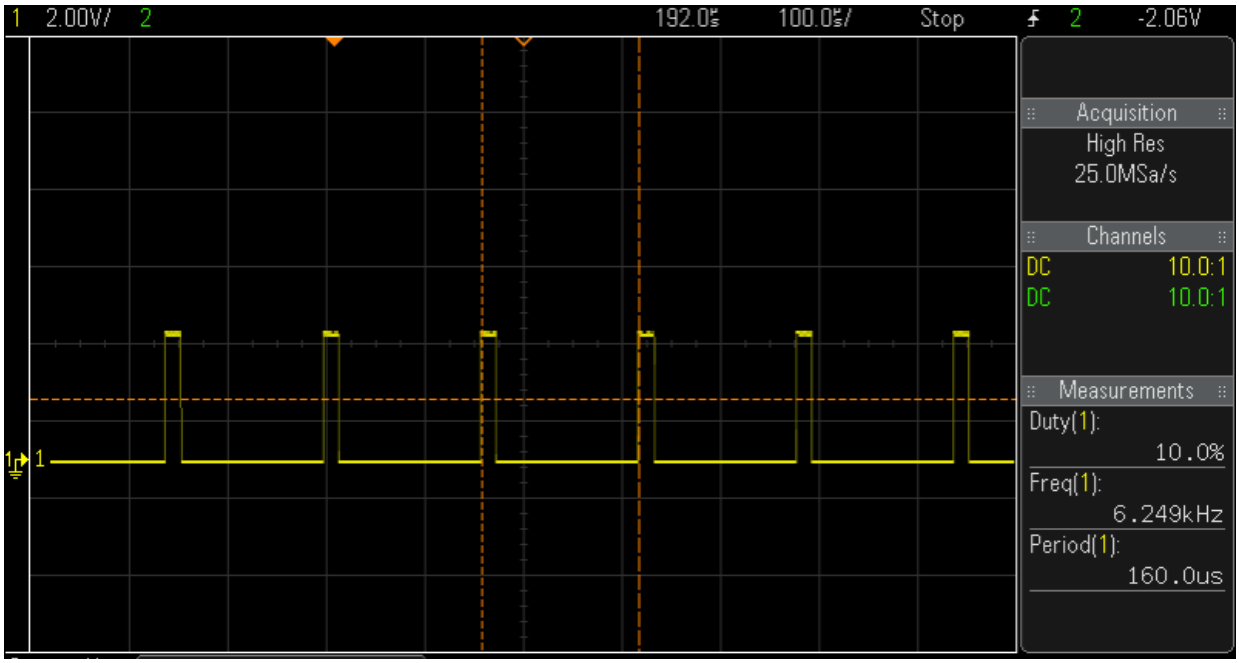


Figure 5-6 PWM Output for 10% Duty Cycle

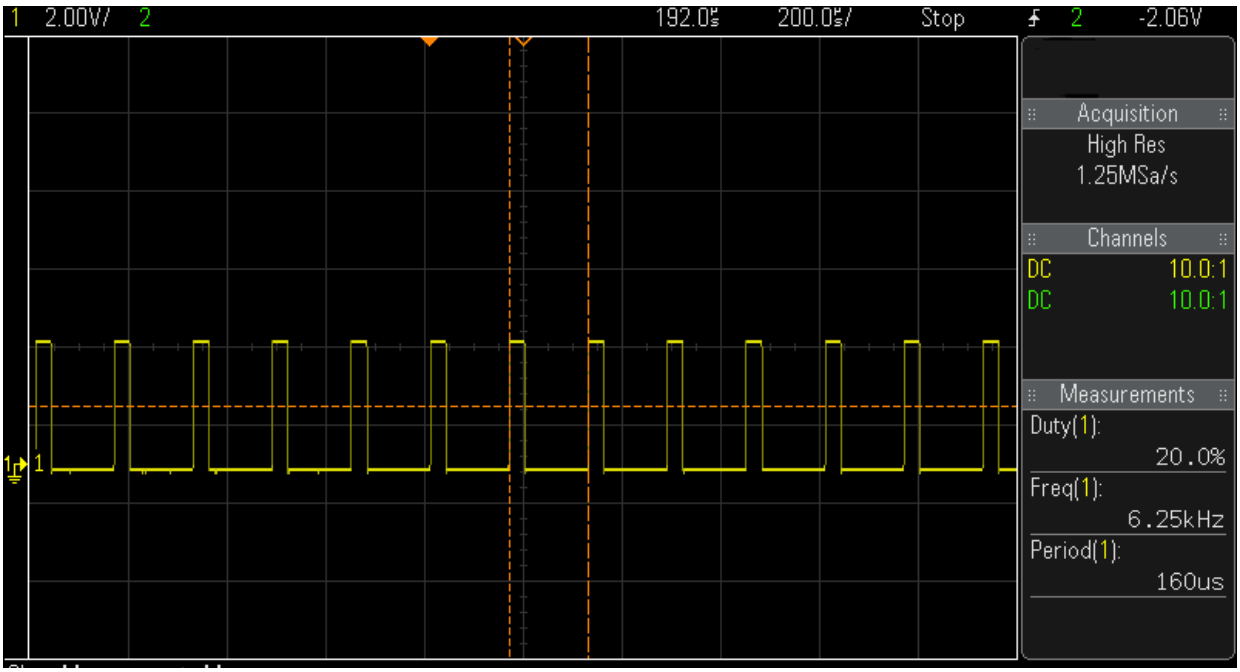


Figure 5-7 PWM Output for 20% Duty Cycle

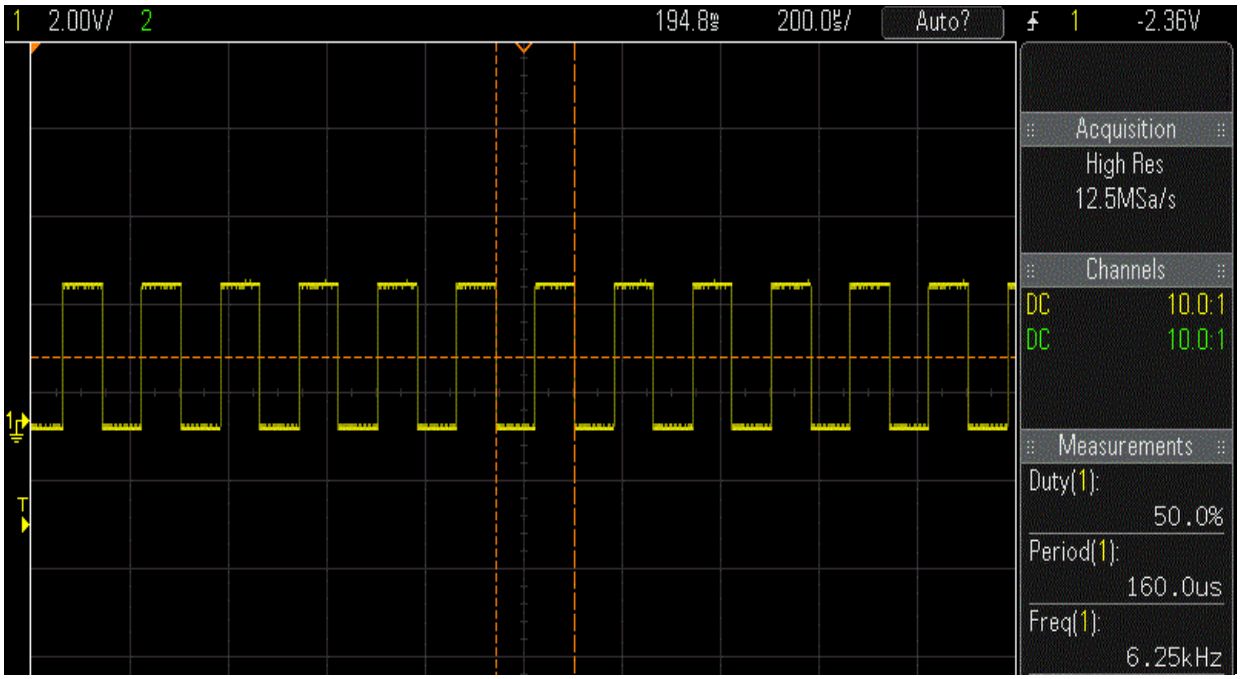


Figure 5-8 PWM Output for 50% Duty Cycle

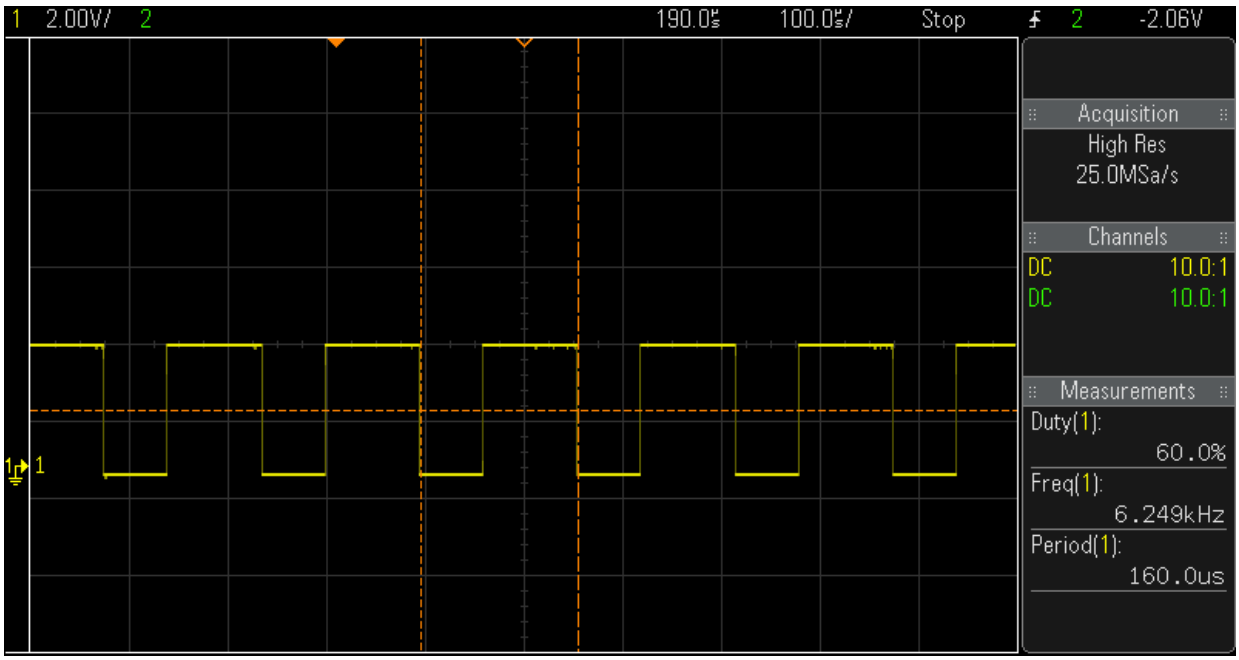


Figure 5-9 PWM Output for 60% Duty Cycle

5.7 Summary

We have successfully generated Pulse Width Modulated output by configuring the internal PWM of the TM4C123GH6PM processor. Also, we have generated PWM outputs for varying duty cycles.

5.8 Exercise

1. Change the software to output a set Duty Cycle, which can be user programmed.

Hint: Comment the ***for*** loop and ***Delay function*** and set ui8Adjust to Duty Cycle%.

Tabulate the Duty Cycle from 10% to 90% in steps of 10% and note down the ON duration and OFF Duration for each of them in Oscilloscope.

2. Change the frequency of the PWM Output from 6.25KHz to 10KHz and perform the tabulation again.

3. Generate Complementary Signals and route it to two pins and observe the waveforms.

4. What is dead band generation mean and where is it applied?

5. Is it possible to construct a DAC from a PWM? Identify the additional components and connection diagram for the same.

6. Sketch the gate control sequence of 3-phase Inverter Bridge and how many PWM generator blocks are required? Can we generate this from EK-TM4C123GXL?

Experiment 6

PWM based Speed Control of DC Motor using Potentiometer

Topics **Page**

6.1 Objective

The main objective of this experiment is to control the speed of a DC Motor with a PWM signal based on the potentiometer output. In this experiment we will also understand working with both the ADC and PWM peripherals to control a DC Motor with PWM.

6.2 Introduction

A DC motor is driven by a PWM signal that controls the speed of the motor. The TM4C123GH6PM generates the PWM signal for DC motor control. The user can control the speed of the DC motor via a potentiometer interfaced to the GPIO of the processor.The TM4C123GH6PM generates the PWM signal corresponding to the potentiometer output.In this experiment, the analog output of the potentiometer is connected to a GPIO port configured as input. This signal is input to the on chip ADC peripheral that converts analog input to 12-bit digital value. The ADC value is read and directly passed on to the duty cycle configuration of the PWM register. The period of the PWM output is set to 4096 (0xFFF) to match the resolution of the ADC and the duty cycle of the PWM output corresponds to the output from the analog potentiometer. The PWM signal controls the DC motor connected through the ULN2003, High Voltage, High Current Darlington Transistor Array, motor driver. Thus, the PWM signal that corresponds to the output of analog potentiometer is used for speed control of the DC motor. The functional block diagram as shown in **Figure 6-1** illustrates the working principle of the experiment.

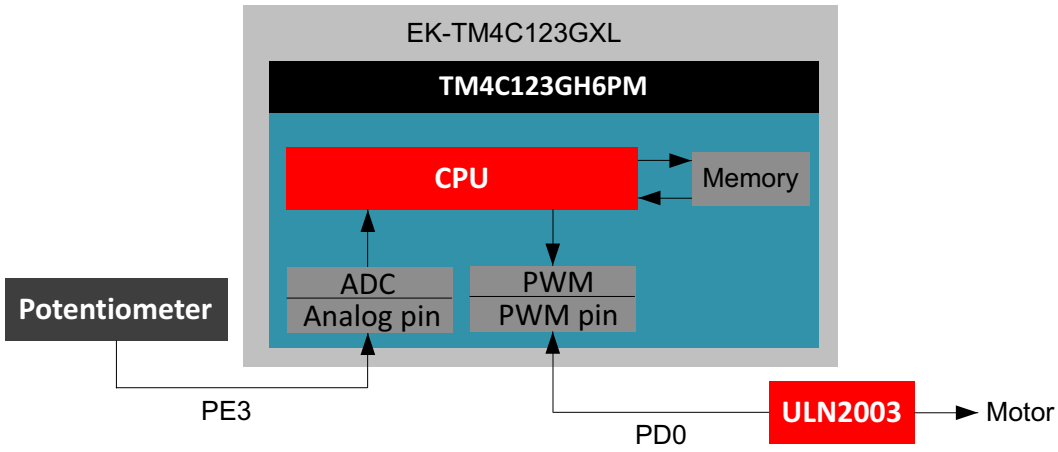


Figure 6-1 Functional Block Diagram

6.2.1 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 that square wave is modulated to encode the analog signal. Typical applications for PWM are switching power supplies, motor control, servo positioning and lighting control.

Using a PWM signal is one of the methods to control the speed of a DC motor. The speed of the motor depends on the speed in which the motor is switched ON and OFF. For example, a motor is continuously switched ON for a 100% duty cycle control signal and therefore runs in full speed. For 50% duty cycle, the motor is ON for half the time period and OFF for the remaining half period. Thus the effective voltage applied to the motor will be 50% and the speed is also reduced to half. Varying the duty cycle will vary the average voltage and hence speed of the DC motor as shown in **Figure 6-2.**

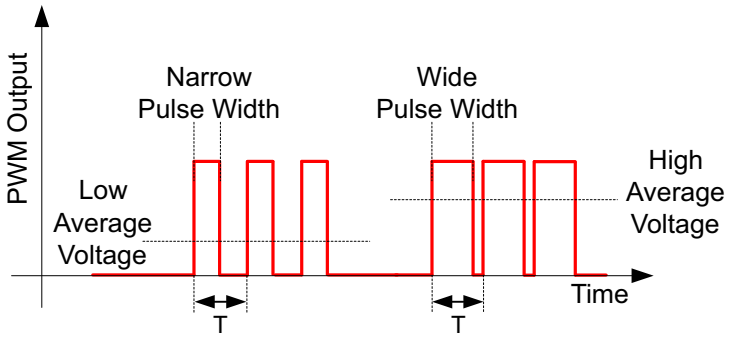


Figure 6-2 Analog Output for PWM Signal

The TM4C123GH6PM has two PWM modules. Each PWM module consists of:

- Four PWM generator blocks
- A control block which determines the polarity of the signals and which signals are passed to the pins

Each of the PWM generator block produces:

- Two independent output signals of the same frequency
- A pair of complementary signals with dead-band generation (for H-bridge circuit protection)
- Eight outputs total

6.2.2 ADC Module

TM4C123GH6PM MCUs feature two ADC modules (ADC0 and ADC1) that can be used to convert continuous analog voltages to discrete digital values. Each ADC module has a 12-bit resolution, operates independently, can execute different sample sequences, sample any of the shared analog input channels and generate interrupts & triggers based on the conversion process.

The features of the ADC module are:

- Two 12-bit 1MSPS ADCs giving a digital range of 0 to 4095
- 12 shared analog input channels
- Single ended & differential input configurations
- On-chip temperature sensor
- Maximum sample rate of one million samples/second (1MSPS)
- Fixed references (VDDA/GNDA) due to pin-count limitations
- 4 programmable sample conversion sequencers per ADC
- Separate analog power and ground pins
- Flexible trigger control
- 2x to 64x hardware averaging
- 8 Digital comparators per ADC
- 2 Analog comparators
- Optional phase shift in sample time between ADC modules is programmable from 22.5° to 337.5°

The TM4C123GH6PM ADC collects sample data by using a programmable sequence-based approach. Each sample sequence is a fully programmed series of consecutive (back-to-back) samples, allowing the ADC to collect data from multiple input sources without having to be re-configured or serviced by the processor. The programming of each sample in the sample sequence includes parameters such as the input source and mode (differential versus single-ended input), interrupt generation on sample completion, and the indicator for the last sample in the sequence. Each ADC module of the TM4C123GH6PM has four sample sequencers that handle the sampling control and data capture. All of the sequencers are identical in implementation except for the number of samples that can be captured and the depth of the FIFO.

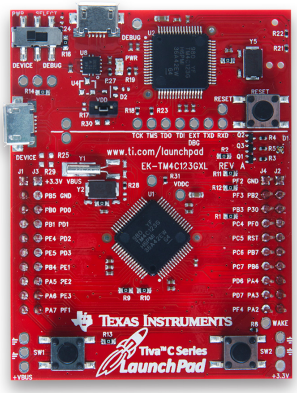
6.3 Component Requirements

6.3.1 Software Requirement

1. Code Composer Studio
2. TivaWare_C_Series

6.3.2 Hardware Requirement

Table 6-1: Components Required for the Experiment

S.No	Components	Specifications	Images
1.	Tiva LaunchPad	EK-TM4C123GXL LaunchPad	
2.	USB cable		
3.	10KΩ Potentiometer,	½ W	
4.	IC ULN2003 (DIP package)		
5.	Bread Board, Connecting wires		
6.	Multimeter		
7.	Oscilloscope		
8.	DC Motor	5V DC,100mA	

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 TM4C123GH6PM on the EK-TM4C123GXL using the USB interface.

6.4.1 Flowchart

The software programmed into the processor controls the GPIO, ADC and PWM peripherals of the processor to run the DC motor based on the input from a potentiometer. The flowchart for the program code is shown in **Figure 6-3**. A temporary array variable is initialized in the program to store the digital values. The system clock is set to 40MHz and PWM frequency set as per **Equation 6.2**. The GPIO ports D and E are enabled and configured. GPIO Port D pin 0 (PD0) is configured as PWM output and GPIO port E pin3 (PE3) is configured as analog input to the ADC module0.

The code then configures and enables the PWM generator with counting mode of operation, period to 4096 and enables bit 0 as PWM output (the output state of the PWM module). The ADC module0 is configured to use sample sequencer 1 and to interrupt on conversion. The program enables the sequencer and waits for an interrupt from the ADC0.

The ADC peripheral reads the analog input from the potentiometer and gives the converted digital output with an interrupt to the processor. On interrupt from the ADC0, the program triggers the ADC0 module sequencer 1 and stores the digital value in a temporary variable. The digital value is passed to the PWM generator as pulse width to generate a PWM signal wave. The code clears the interrupt flag and waits for the next interrupt from the ADC (next input from the potentiometer). The PWM signal corresponding to the potentiometer input is fed to the DC motor for speed control.

Design Calculations

- The calculations for the configuration of ADC and PWM are given below.

- This configuration varies the Analog Voltage from 0V to 3.3V depending on the wiper position on PE3 which is the AN0 or Analog Input 0 of the TM4C123GH6PM.

- The current configuration connects the VREFP of ADC to VDDA which is 3.3V and VREFP to GND which is 0V.

- The range of the conversion values is from 0 to 4095 which is 0 to 0xfff in Binary since it is a 12-bit ADC.

- With the Conversion voltage span and the bit range we can calculate the following:

- Resolution = 3.3V/ 4096 = 3300mV / 4096 = 0.8mV
- Voltage per ADC code = 3.3V / 4096
  - If Digital Code = 1000 in decimal, multiply this value by resolution
  - Equivalent Voltage for 1000 = 0.8mV *1000 = 800 mV = 0.8V

- Varying the Potentiometer changes the Digital Value between 0 and 4095.

- The PWM period is set to 4095 to match the resolution of the 12-bit ADC.

- The ADC value is read and directly passed onto the duty cycle configuration of the PWM register. It is also possible to have other values but necessary scaling has to be done.

Calculation for PWM

- System Clock = 40MHz

- PWM Clock = System Clock /64 = 625kHz (6.1)
- PWM Period = 4095
- PWM frequency = PWM Clock / PWM Period = 152.63Hz (6.2)

- The PWM can be varied as the ADC varies with a 12-bit resolution, and the PWM frequency is 152.63Hz.

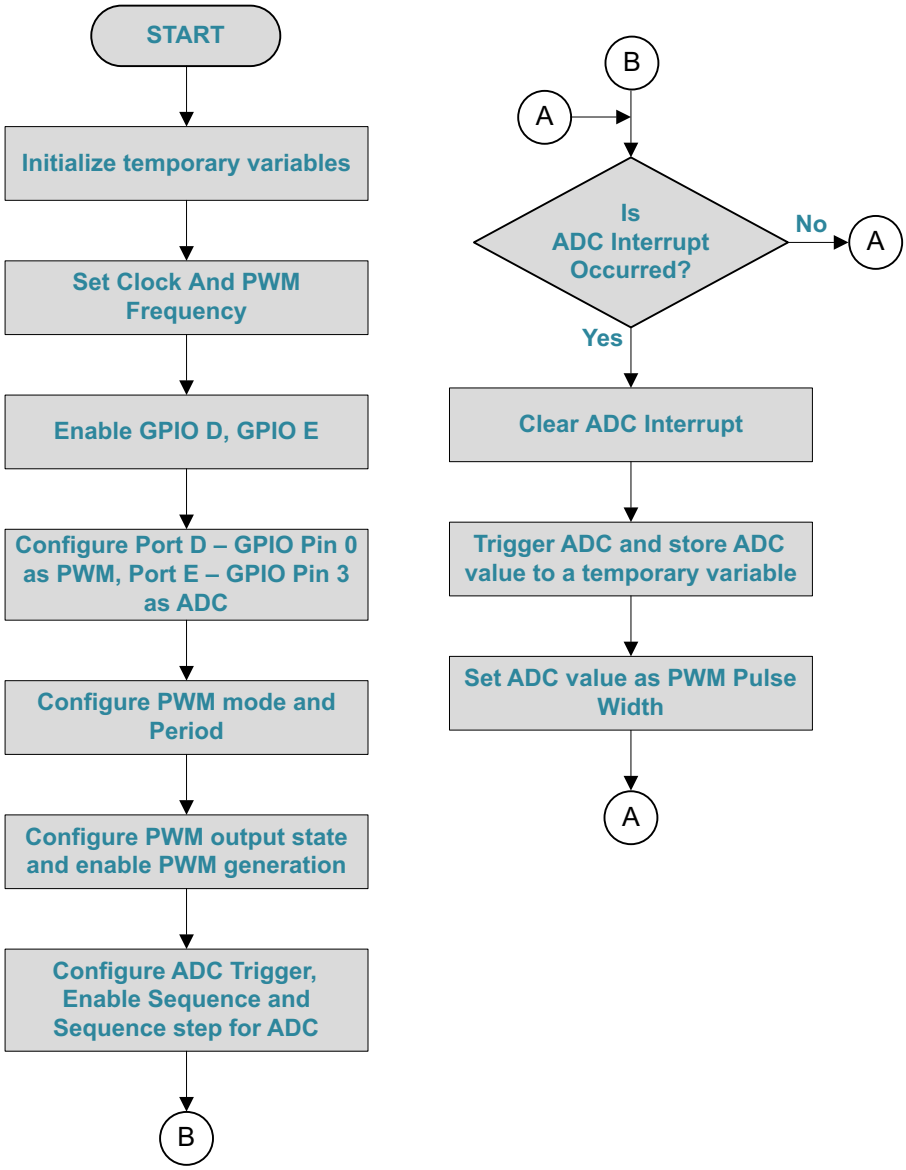


Figure 6-3 Flowchart for PWM based Speed Control of DC Motor using Potentiometer

6.4.2 C Program Code for PWM based Speed Control of DC Motor

```
// HEADER FILES //
#include<stdint.h>
#include<stdbool.h>
#include"inc/hw_memmap.h"
#include"inc/hw_types.h"
#include"driverlib/debug.h"
#include"driverlib/sysctl.h"
#include"driverlib/adc.h"
#include"inc/hw_types.h"
#include"driverlib/gpio.h"
#include"driverlib/pwm.h"
#include"driverlib/pin_map.h"
#include"inc/hw_gpio.h"
#include"driverlib/rom.h"


uint32_tui32ADC0Value[1];    // TO STORE THE VALUE IN VARIABLE
ui32ADC0Value FOR EVERY SAMPLING


intmain(void)
{
SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_OSC_MAIN|
SYSCTL_XTAL_16MHZ);  // SET SYSTEM CLOCK AT 40MHZ
SysCtlPWMClockSet(SYSCTL_PWMDIV_64); //SET PWM CLOCK AT SYSTEM CLOCK
DIVIDED BY 64
SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM1);       //ENABLE PWM1 MODULE
SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);       //ENABLE ADC0 MODULE
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);      //ENABLE GPIO FOR ADC0
MODULE
GPIOPinTypeADC(GPIO_PORTE_BASE,GPIO_PIN_3);      //CONFIGURE PE3 AS AN0
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);      //ENABLE GPIO FOR PWM1
MODULE
GPIOPinTypePWM(GPIO_PORTD_BASE, GPIO_PIN_0);      //CONFIGURE PD0 AS PWM
OUTPUT
GPIOPinConfigure(GPIO_PD0_M1PWM0);//SET PD0 AS M1PWM0
PWMGenConfigure(PWM1_BASE, PWM_GEN_0, PWM_GEN_MODE_DOWN);
//SET PWM GENERATOR WITH MODEOF OPERATION AS COUNTING
PWMGenPeriodSet(PWM1_BASE, PWM_GEN_0,4095);
```

```
   //SET THE PERIOD OF PWM GENERATOR
PWMOutputState(PWM1_BASE, PWM_OUT_0_BIT, true); //ENABLE BIT0 OUTPUT
PWMGenEnable(PWM1_BASE, PWM_GEN_0);              //ENABLE PWM GENERATOR
ADCSequenceConfigure(ADC0_BASE, 1, ADC_TRIGGER_PROCESSOR, 0);
// ADC0 MODULE, TRIGGER IS PROCESSOR EVENT, SEQUENCER 0 IS CONFIGURED
ADCSequenceStepConfigure(ADC0_BASE, 1, 0, ADC_CTL_CH0);
// ADC0 MODULE, SEQUENCER 0 , FOR 1 SAMPLING, INPUT IS FROM CHANNEL 0 PE3
ADCSequenceEnable(ADC0_BASE, 1);      // ENABLE THE SEQUENCE 1 FOR ADC0
while(1)
{
ADCIntClear(ADC0_BASE, 1);  // CLEAR INTERRUPT FLAG FOR ADC0, SEQUENCER 1
ADCProcessorTrigger(ADC0_BASE, 1);  // TRIGGER IS GIVEN FOR ADC0 MODULE,
SEQUENCER1
// STORE THE CONVERTED VALUE FOR ALL DIFFERENT SAMPLING IN ARRAY
ui32ADC0Value
ADCSequenceDataGet(ADC0_BASE, 1, ui32ADC0Value);
PWMPulseWidthSet(PWM1_BASE, PWM_OUT_0, ui32ADC0Value[0]);
 // SET THE PULSE WIDTH
}
}
```

Table 6-2: API Functions Used in the Application Program

API Function	Parameters	Description
GPIOPinTypeADC (uint32_t ui32Port, uint8_t ui8Pins)	• **ui32Port** is the base address of the GPIO port • **lui8Pins** is the bit-packed representation of the pin(s)	Configures pin(s) for use as analog-to-digital converter inputs.
ADCSequenceConfig-ure (uint32_t ui32Base, uint32_t ui32Sequence-Num, uint32_t ui32Trigger, uint32_t ui32Priority)	• **ui32Base** is the base address of the ADC module • **ui32SequenceNum** is the sample sequence number • **ui32Trigger** is the trigger source that initiates the sam-ple sequence; must be one of the **ADC_TRIGGER_** values • **ui32Priority** is the relative priority of the sample sequence with respect to the other sample sequences	Configures the trigger source and priority of a sample sequence.

Table 6-2: API Functions Used in the Application Program

API Function	Parameters	Description
ADCSequenceStep-Configure(uint32_t ui32Base, uint32_t ui32Sequence-Num, uint32_t ui32Step, uint32_t ui32Config)	• **ui32Base** is the base address of the ADC module • **ui32SequenceNum** is the sample sequence number • **ui32Step** is the step to be configured • **ui32Config** is the configuration of this step; must be a logical OR of **ADC_CTL_TS, ADC_CTL_IE, ADC_CTL_END, ADC_CTL_D,** one of the input channel selects **(ADC_CTL_CH0** through **ADC_CTL_CH23),** and one of the digital comparator selects **(ADC_CTL_CMP0** through **ADC_CTL_CMP7)**	Configure a step of the sample sequencer.
ADCSequenceEnable (uint32_t ui32Base, uint32_t ui32Sequence-Num)	• **ui32Base** is the base address of the ADC module • **ui32SequenceNum** is the sample sequence number	Enables a sample sequence.
ADCIntClear(uint32_t ui32Base, uint32_t ui32Sequence-Num)	• **ui32Base** is the base address of the ADC module • **ui32SequenceNum** is the sample sequence number	Clears sample sequence interrupt source.
ADCProcessorTrigger (uint32_t ui32Base, uint32_t ui32Sequence-Num)	• **ui32Base** is the base address of the ADC module • **ui32SequenceNum** is the sample sequence number, with **ADC_TRIGGER_WAIT** or **ADC_TRIGGER_SIGNAL** optionally ORed into it	Causes a processor trigger for a sample sequence.
ADCSequenceDataGet (uint32_t ui32Base, uint32_t ui32Sequence-Num, uint32_t *pui32Buffer)	• **ui32Base** is the base address of the ADC module • **ui32SequenceNum** is the sample sequence number • **pui32Buffer** is the address where the data is stored	Gets the captured data for a sample sequence.

Table 6-2: API Functions Used in the Application Program

API Function	Parameters	Description
SysCtlPWMClockSet (uint32_t ui32Config)	**ui32Config** is the configuration for the PWM clock; it must be one of **SYSCTL_PWMDIV_1, SYSCTL_PWMDIV_2, SYSCTL_PWMDIV_4, SYSCTL_PWMDIV_8, ISYSCTL_PWMDIV_16, SYSCTL_PWMDIV_32,** or **SYSCTL_PWMDIV_64**	Sets the PWM clock configuration.
GPIOPinTypePWM (uint32_t ui32Port, uint8_t ui8Pins)	• **ui32Port** is the base address of the GPIO port • **ui8Pins** is the bit-packed representation of the pin(s)	Configures pin(s) for use by the PWM peripheral.
PWMGenConfigure (uint32_t ui32Base, uint32_t ui32Gen, uint32_t ui32Config)	• **ui32Base** is the base address of the PWM module • **ui32Gen** is the PWM generator to configure. This parameter must be one of **PWM_GEN_0, PWM_GEN_1, PWM_GEN_2,** or **PWM_GEN_3** • **ui32Config** is the configuration for the PWM generator	Configures a PWM generator.
PWMGenPeriodSet (uint32_t ui32Base, uint32_t ui32Gen, uint32_t ui32Period)	• **ui32Base** is the base address of the PWM module • **ui32Gen** is the PWM generator to be modified. This parameter must be one of **PWM_GEN_0, PWM_GEN_1, PWM_GEN_2,** or **PWM_GEN_3** • **ui32Period** specifies the period of PWM generator output, measured in clock ticks	Sets the period of a PWM generator.

6.5 Procedure

6.5.1 Hardware Setup

Figure 6-4 shows the hardware connections for the potentiometer and the DC motor with the EK-TM4C123GXL.

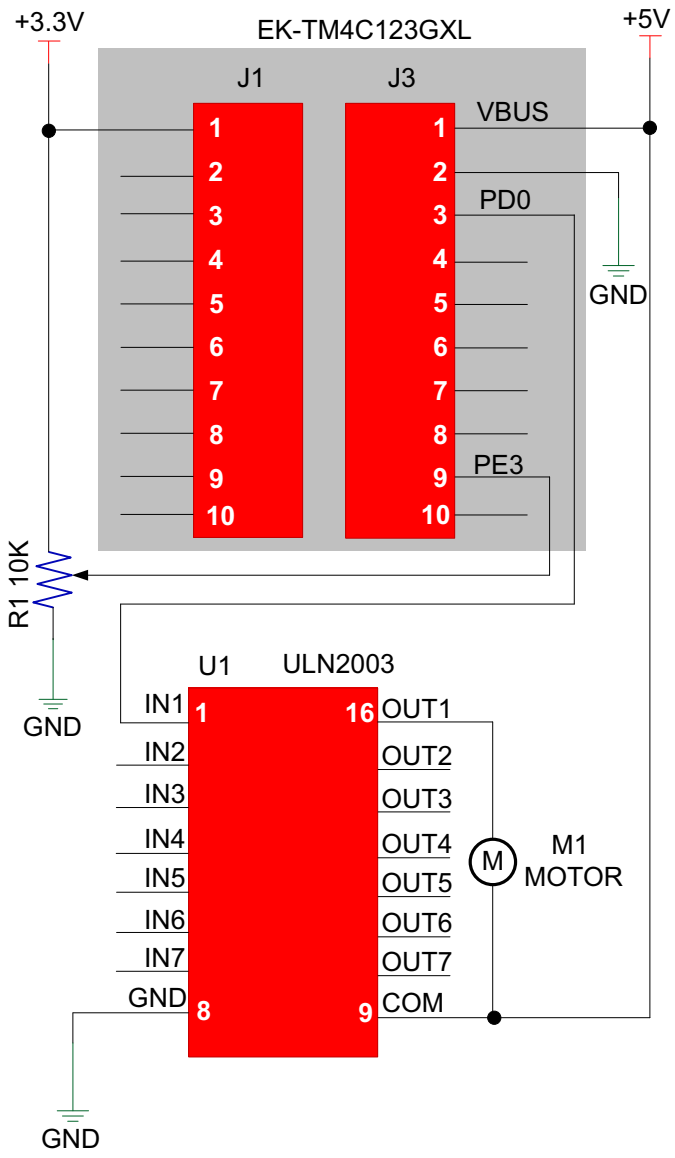


Figure 6-4 Connection Diagram for Potentiometer and DC Motor with EK-TM4C123GXL

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

1. Position DIP IC ULN2003A 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 to the VBUS (5V Power from USB) (available at J3 Connector Pin 1).

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

5. Connect J3 Connector Pin 2 (GND) to the Ground Pin (IC Pin 8) of ULN2003.

6. Connect the PWM Output PD0 (J3 connector Pin 3) to Input Signal (IC Pin 1) of ULN2003.

7. Connect one lead of the Potentiometer to the +3.3V Supply Voltage (J1 connector, Pin 1).

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 PE3 (J3 Connector Pin 9) which is the AN0 or Analog Input 0 of the Tiva TM4C123GH6PM.

After connecting the hardware, the setup appears as shown in **Figure 6-5**.

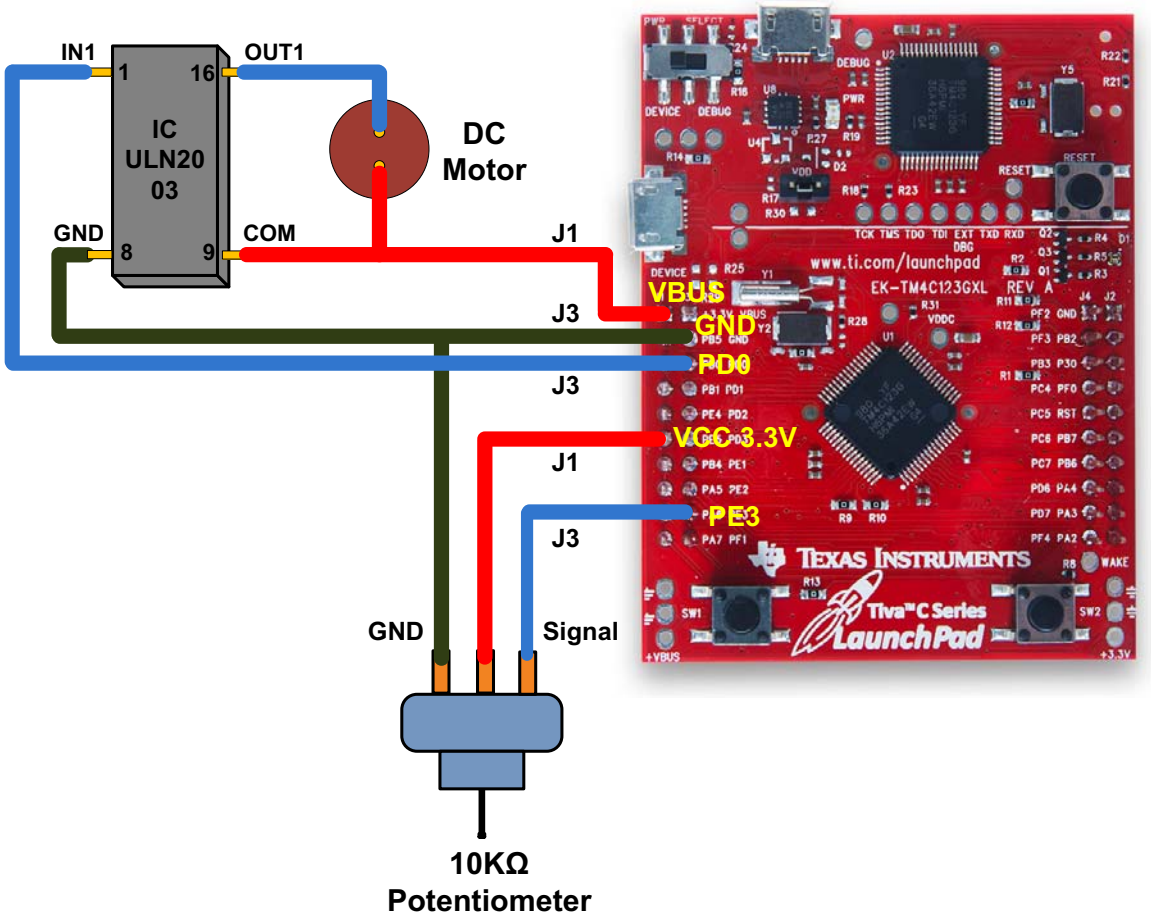


Figure 6-5 Hardware Setup

6.5.2 Software Execution

1. Build, program and debug the code.

2. Vary the position of the potentiometer wiper and observe the corresponding digital output on the CCS window.

3. Also, observe the PWM waveform at J3 connector pin 3 using an oscilloscope.

4. Observe the motion of the DC motor connected to the PWM output.

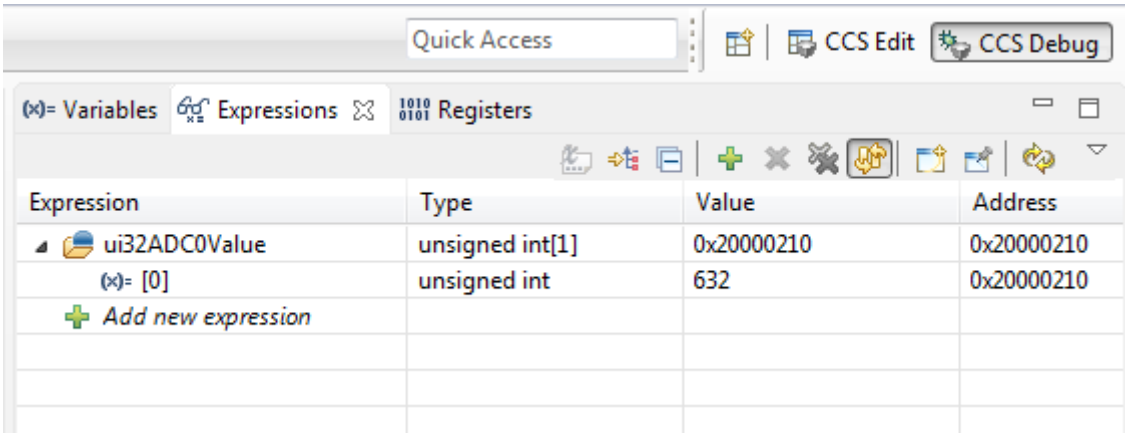
6.6 Observation

The digital value for two arbitrary positions of the potentiometer wiper and the corresponding PWM outputs are as shown in **Figure 6-6** and **Figure 6-7**.

1. The digital value of the analog signal is 632.

The corresponding duty cycle% is:

(632 / 4096) * 100 =15.4%



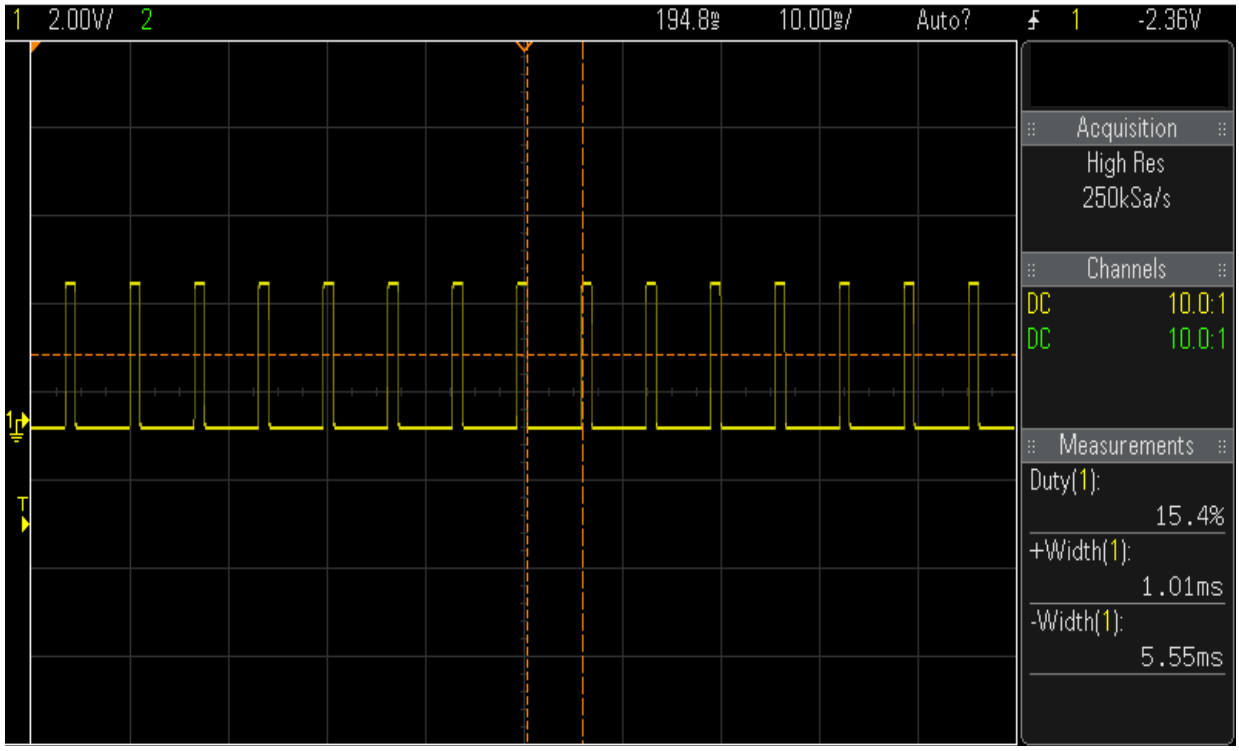
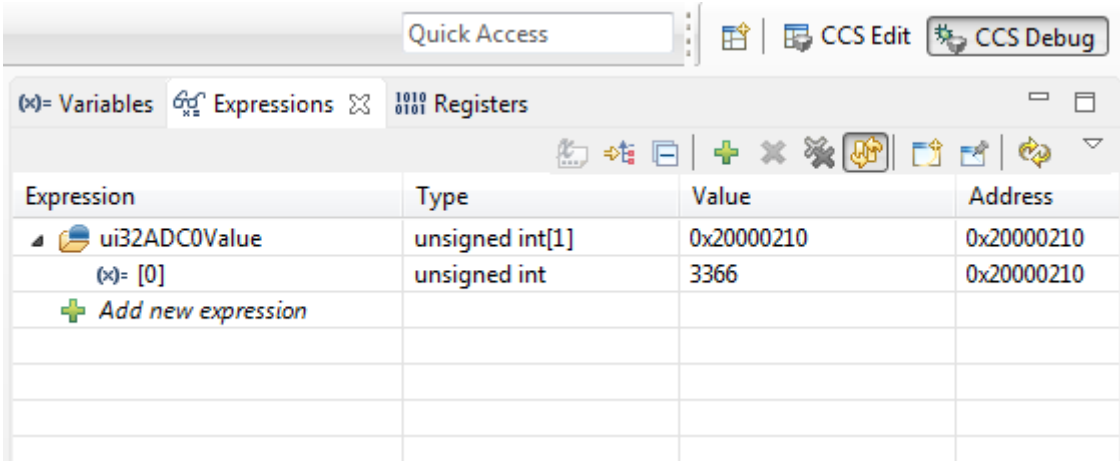


Figure 6-6 Digital Output and PWM Signal for Position 1

2. The digital value of the analog signal is 3366 and the corresponding duty cycle% is:

(3366 / 4096) * 100 = 82%



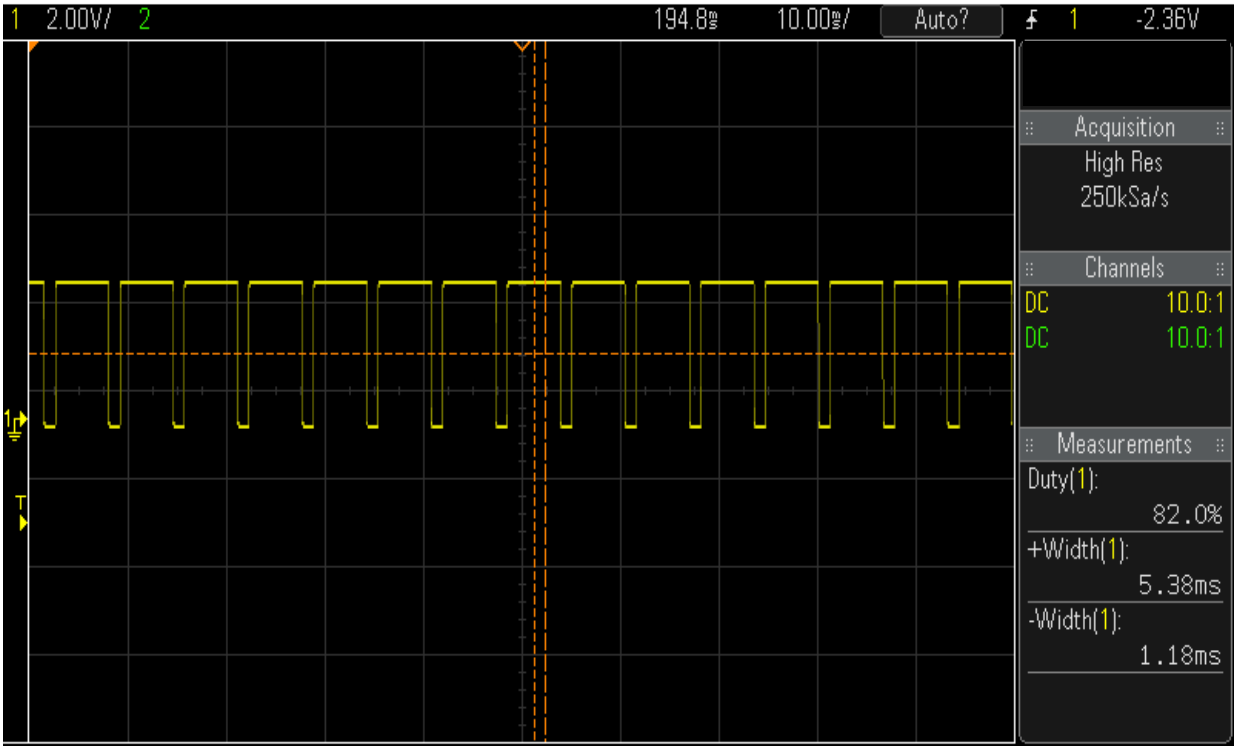


Figure 6-7 Digital Output and PWM Signal for Position 2

Also the DC motor speed is observed to be different for the two outputs. The DC motor speed is greater in the second case as compared to the first case.

6.7 Summary

We have successfully implemented a DC motor control by generation of Pulse Width Modulation signal based on a potentiometer analog input. We have also learnt how to use the ADC and PWM peripherals together for the required application.

6.8 Exercise

1. With the same ADC Input configure 2 PWM generator modules with 2 different frequencies.

2. Read the Internal Temperature Sensor and control a DC Motor that can be deployed to control a fan based on the ambient temperature.

3. What is the resolution of the PWM in this experiment?

4. What would be the maximum frequency that can be generated from the PWM generator?

5. Briefly explain an application in 5 lines which you think will deploy an ADC and a PWM.

Texas Instruments

Experiment 7
UART - ECHO!

Topics **Page**

7.1 Objective

The main objective this experiment is to connect the EK-TM4C123GXL to a PC terminal and send an echo of the data input back to the PC using UART. In this experiment, we will also understand the basics of serial communication to send and receive data.

7.2 Introduction

The SCI (Serial Communication Interface) modules support digital communications between the CPU and other asynchronous peripherals that use the standard non-return-to-zero (NRZ) format. The SCI receiver and transmitter each have a 4-level deep FIFO for reducing servicing overhead. Each FIFO has its own separate enable and interrupt bits. Both can be operated independently for half-duplex communication, or simultaneously for full-duplex communication. The bit rate is programmable to different speeds through a 16-bit baud-select register.

In this experiment, the SCI module is configured to transmit the received data from an external device along with some message to indicate the reception. The functional block diagram as shown in **Figure 7-1** illustrates the working principle of the experiment. The UART of the EK-TM4C123GXL is connected to the serial port of the PC that runs over the debug USB cable. The serial port can be monitored by the PC with the help of Tera Term or any other serial interface software. In this experiment, the UART receives serial data via the USB cable and displays the data received on the serial monitor (Tera Term) through serial communication.

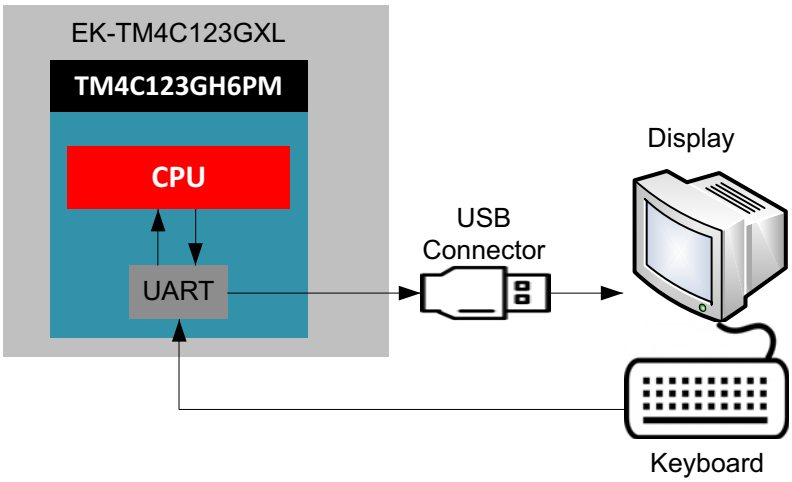


Figure 7-1 Functional Block Diagram

7.2.1 UART Module

The TM4C123GH6PM controller includes eight Universal Asynchronous Receiver/Transmitter (UART) with programmable baud-rate generator, allowing speeds of up to 5 Mbps for regular speed (divide by 16) and 10 Mbps for high speed (divide by 8). It has separate 16x8 transmit (TX) and receive (RX) FIFOs (First In First Out) to reduce CPU interrupt service loading. It has a programmable FIFO length, including 1-byte deep operation providing conventional double-buffered interface. The FIFO trigger levels available are of 1/8, 1/4, 1/2, 3/4, and 7/8. It also has standard asynchronous communication bits for start, stop, and parity and detects and generates line-break.

The characteristics of fully programmable serial interface are:

- 5, 6, 7, or 8 data bits

- Even, odd, stick, or no-parity bit generation/detection
- 1 or 2 stop bit generation

It supports Standard FIFO-level and End-of-Transmission interrupts and efficient transfers using Micro Direct Memory Access Controller (μDMA)[1] which has separate channels for transmit and receive. The EK-TM4C123GXL has an internal USB to UART converter to connect to the UART of the processor.

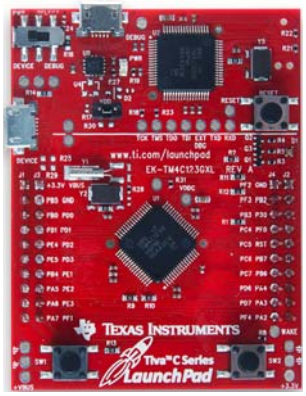
7.3 Component Requirements

7.3.1 Software Requirement

1. Code Composer Studio
2. Serial Terminal Software (eg: Tera Term)
3. TivaWare_C_Series

7.3.2 Hardware Requirement

Table 7-1: Components Required for the Experiment

S.No	Components	Specifications	Images
1.	Tiva LaunchPad	EK-TM4C123GXL LaunchPad	
2.	USB cable		

1. The TM4C123GH6PM microcontroller includes a Direct Memory Access (DMA) controller, known as micro-DMA (μDMA). The μDMA controller provides a way to offload data transfer tasks from the Cortex™-M4F processor, allowing for more efficient use of the processor and the available bus bandwidth.

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 TM4C123GH6PM on the EK-TM4C123GXL using the USB interface.

7.4.1 TeraTerm Setup

The serial port configuration for the Tera Term is:

Baud rate is 115200, Parity is none, 8 Data Bits and Stop Bit is 1.

Select the configuration and Press **OK** as shown in **Figure 7-2**.

Note: COM Port varies in different PCs based on the hardware

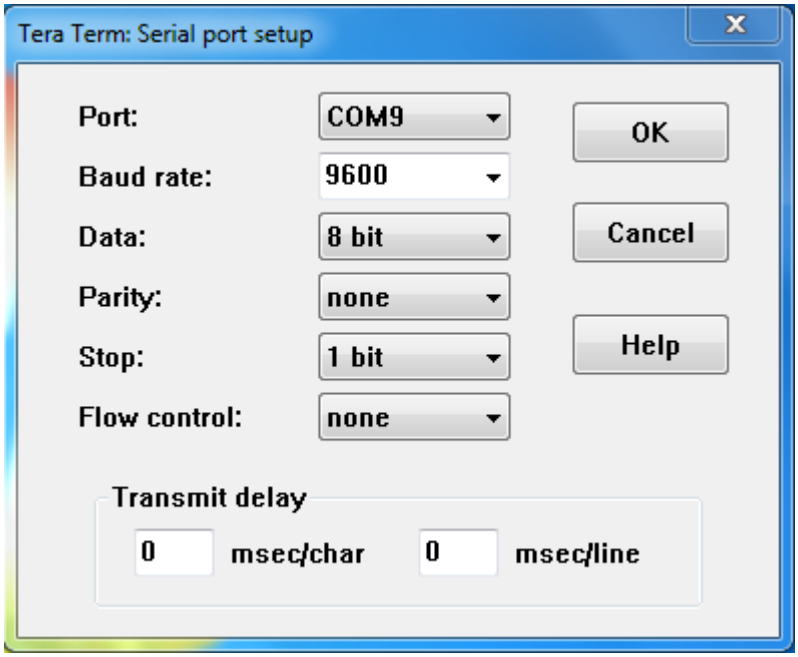


Figure 7-2 Tera Term Serial Port Setup Window

7.4.2 Flowchart

The flowchart for the program is shown in **Figure 7-3**. The system clock is configured and enabled to 40MHz. The GPIO Port A pins 0 and 1 are enabled and configured as UART receiver and transmitter respectively. The UART peripheral is configured with Baud rate 115200, Parity none, 8 Data Bits and Stop Bit 1. The default message "Echo output" is transmitted by the UART to the serial terminal in the PC. The program then waits for serial input from the PC using a while loop. When the UART receives an input, it sends the same data to PC terminal as output. Thus, the input is echoed on the Terminal window.

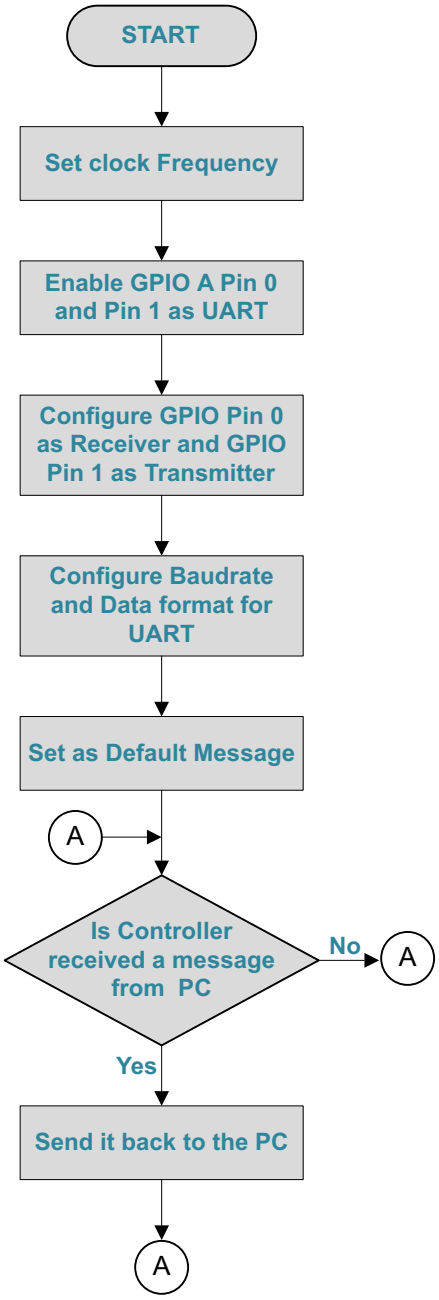


Figure 7-3 Flowchart for UART - Echo

7.4.3 C Program Code for UART - Echo

```
// HEADER FILES
#include<stdint.h>
#include<stdbool.h>
#include"inc/hw_memmap.h"
#include"inc/hw_types.h"
```

```
#include"driverlib/gpio.h"
#include"driverlib/pin_map.h"
#include"driverlib/sysctl.h"
#include"driverlib/uart.h"
#define GPIO_PA0_U0RX  0x00000001       // UART PIN ADDRESS FOR UART RX
#define GPIO_PA1_U0TX  0x00000401       // UART PIN ADDRESS FOR UART TX
intmain(void)
{
// SYSTEM CLOCK AT 40 MHZ
SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_OSC_MAIN|
SYSCTL_XTAL_16MHZ);
// ENABLE PERIPHERAL UART 0
SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
 // ENABLE GPIO PORT A,FOR UART
GPIOPinConfigure(GPIO_PA0_U0RX);        // PA0 IS CONFIGURED TO UART RX
GPIOPinConfigure(GPIO_PA1_U0TX);        // PA1 IS CONFIGURED TO UART TX
GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);
// CONFIGURE UART, BAUD RATE 115200, DATA BITS 8, STOP BIT 1, PARITY NONE
UARTConfigSetExpClk(UART0_BASE, SysCtlClockGet(), 115200,
(UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE | UART_CONFIG_PAR_NONE));

UARTCharPut(UART0_BASE, 'E');
UARTCharPut(UART0_BASE, 'c');
UARTCharPut(UART0_BASE, 'h');
UARTCharPut(UART0_BASE, 'o');           // SEND "Echo Output: " IN UART
UARTCharPut(UART0_BASE, ' ');
UARTCharPut(UART0_BASE, 'O');
UARTCharPut(UART0_BASE, 'u');
UARTCharPut(UART0_BASE, 't');
UARTCharPut(UART0_BASE, 'p');
UARTCharPut(UART0_BASE, 'u');
UARTCharPut(UART0_BASE, 't');
UARTCharPut(UART0_BASE, ': ');
UARTCharPut(UART0_BASE, ' ');
UARTCharPut(UART0_BASE, '\n');
```

```
while (1)
    {
//UART ECHO - what is received is transmitted back //
if (UARTCharsAvail(UART0_BASE)) UARTCharPut(UART0_BASE,
UARTCharGet(UART0_BASE));
 }
}
```

Table 7-2: API Functions Used in the Application Program

API Function	Parameters	Description
GPIOPinTypeU-ART(uint32_t ui32Port, uint8_t ui8Pins)	• **ui32Port** is the base address of the GPIO port • **ui8Pins** is the bit-packed representation of the pin(s)	Configures pin(s) for use by the UART peripheral
UARTConfigSetExp-Clk(uint32_t ui32Base, uint32_t ui32UARTClk, uint32_t ui32Baud, uint32_t ui32Config)	• **ui32Base** is the base address of the UART port • **ui32UARTClk** is the rate of the clock supplied to the UART module • **ui32Baud** is the desired baud rate • **ui32Config** is the data for-mat for the port (number of data bits, number of stop bits, and parity)	Sets the configuration of a UART.
UART-CharsAvail(uint32_t ui32Base)	• **ui32Base** is the base address of the UART port • **ucData** is the character to be transmitted	Waits to send a charac-ter from the specified port.
UART-CharsAvail(uint32_t ui32Base)	• **ui32Base** is the base address of the UART port	Determines if there are any characters in the receive FIFO.
UARTCharGetNon-Blocking(uint32_t ui32Base)	• **ui32Base** is the base address of the UART port	Receives a character from the specified port.

7.5 Procedure

1. Connect the EK-TM4C123GXL to the PC using the USB cable supplied.

2. Build, program and debug the code.

3. Open Tera Term UART terminal window and configure which is explained in **Section 7.4.1**.

4. Type characters on the keyboard and observe the terminal window.

7.6 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. Any alphanumeric string could be sent for testing.

To confirm this , hold the RESET key on the EK-TM4C123GXL and try typing the input string. You will observe no display of the characters typed. We can thus infer that the program in the EK-TM4C123GXL is doing the echo operation.

Figure 7-4 shows display of typed data (UART input) **TEXAS INSTRUMENTS INC.** being echoed back on the serial terminal window.

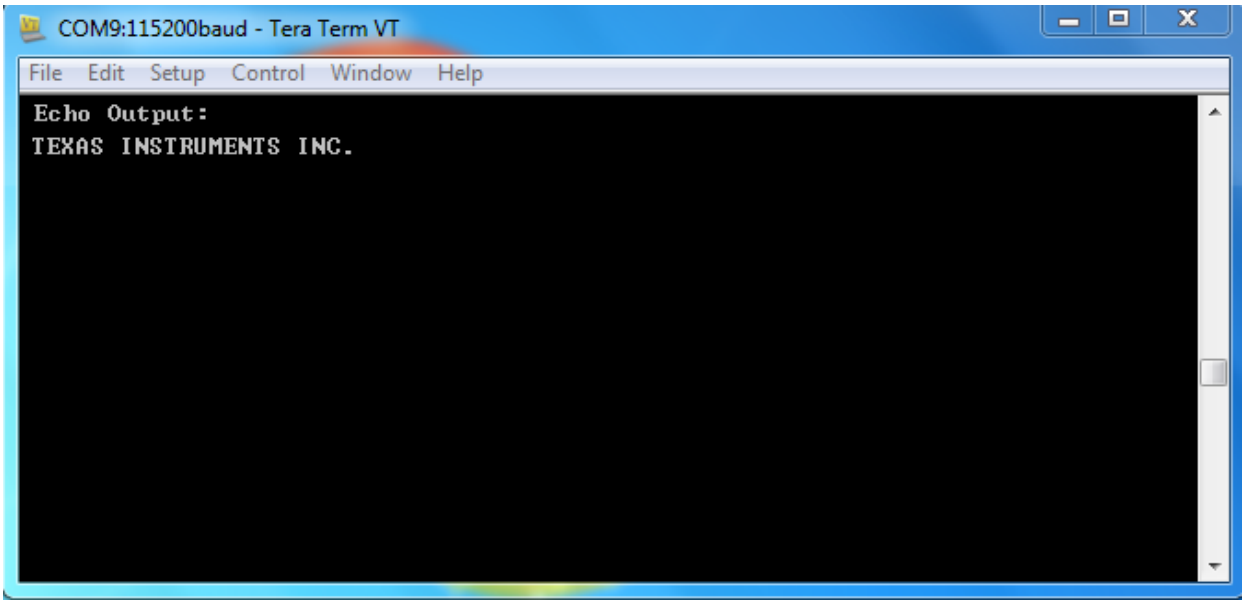


Figure 7-4 Output on Serial Terminal Window

7.7 Summary

We have successfully configured the serial port to send and receive bytes and monitor the data on a serial terminal. We have also learnt how the UART peripheral can be used for serial interface.

7.8 Exercise

1. Change the baud rate to 19200 and repeat the experiment.

Hint: If the serial terminal baud rate has not changed, junk characters will be received. So, it is important that two serially communicating partners to have the same configuration.

---

2. What is the maximum baud rate that can be set in the UART peripheral of Tiva?

3. Modify the software to display "Switch pressed" by pressing a user input switch on the LaunchPad.

Experiment 8

Interfacing an Accelerometer with TIVA using I^2C

Topics **Page**

8.1 Objective

The main objective of this experiment is to interface an accelerometer in BOOSTXL-SENSHUB Booster Pack with TM4C123GH6PM using I^2C interface. In this experiment, we will also learn how to obtain the data from the accelerometer and display on a serial terminal.

8.2 Introduction

The Texas Instruments Sensor Hub BoosterPack (BOOSTXL-SENSHUB) is an add-on board designed to plug into Tiva™ C Series EK-TM4C123GXL along with all of TI's Launch Pads. It allows developers to create products with up to nine axes of motion tracking and multiple environmental sensing capabilities.

The Sensor Hub BoosterPack comprises of the following sensors:

- **TMP006 Infrared Temperature Sensor** - Performs non-contact temperature measurement
- **BMP180 Digital Pressure Sensor** - Obtains air pressure and temperature measurements
- **MPU-9150 9-axis Motion Sensor** - Consists of 3 axis Accelerometer, 3 axis gyro meter and 3 axis magnetometer for motion sensing
- **ISL29023 Ambient & Infrared Light Sensor** - Obtains ambient and infrared light measurements
- **SHT21 humidity and ambient temperature sensor**- Obtains temperature and relative humidity of the environment

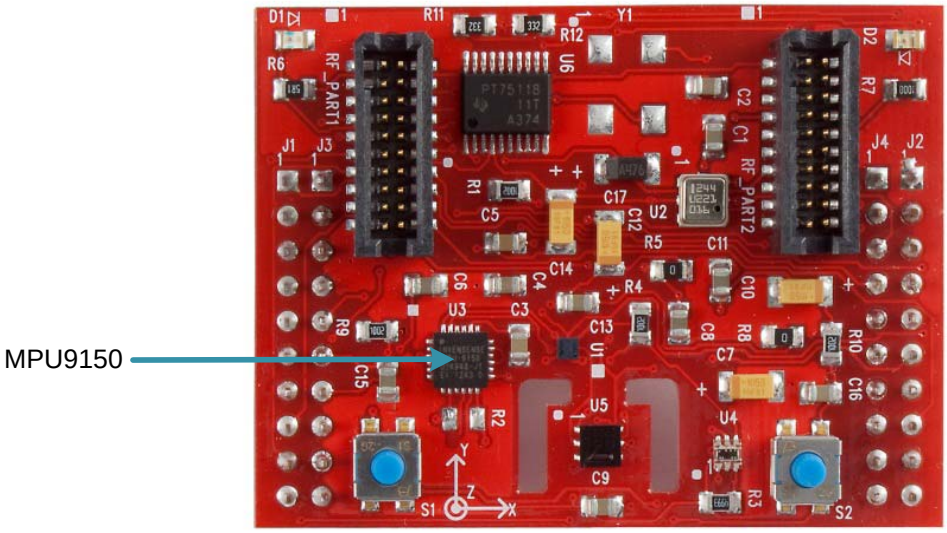


Figure 8-1 MPU9150 Sensor on the Sensor Hub BoosterPack

As shown in **Figure 8-1**, the IC MPU9150 present inside the Sensor Hub BoosterPack is the 9-axis motion sensor that performs motion sensing. The 3-axis MEMS (Micromechanical System) accelerometer measures the earth gravity minus the acceleration. There are a host of applications for the accelerometer sensor such as image stabilization in vision applications, orientation sensing as done in smart phones, target monitoring, vibration alarm etc.

Note: The most common acceleration, and one that we are constantly exposed to, is the acceleration that is due to gravity. This is a common reference value from which all other accelerations are measured (known as g, which is ~9.8m/s^2).

The MPU9150 (in the Sensor Hub BoosterPack) communicates with the microcontroller TM4C123GH6PM (of the Tiva C Series EK-TM4C123GXL) using I^{2}C interface. The Inter-Integrated Circuit (I^{2}C) bus of the TM4C123GH6PM provides bi-directional data transfer through a two-wire design (a Serial Data Line SDA and a Serial Clock Line SCL), and interfaces to external I^{2}C devices. The advantages of I^{2}C interface over other interfaces are:

- It requires only two signal lines
- It has a wide range of transmission speeds
- Each I^{2}C device is addressable independently
- The devices have a master/slave relationship with capability of having multiple masters and/or multiple slaves

This experiment demonstrates the basic use of the TivaWare sensor library, EK-TM4C123GXL and Sensor Hub Booster Pack. In this experiment, MPU-9150 sensor fuses the nine axis measurements into a set of Euler angles: roll, pitch and yaw (See **Figure 8-2**).

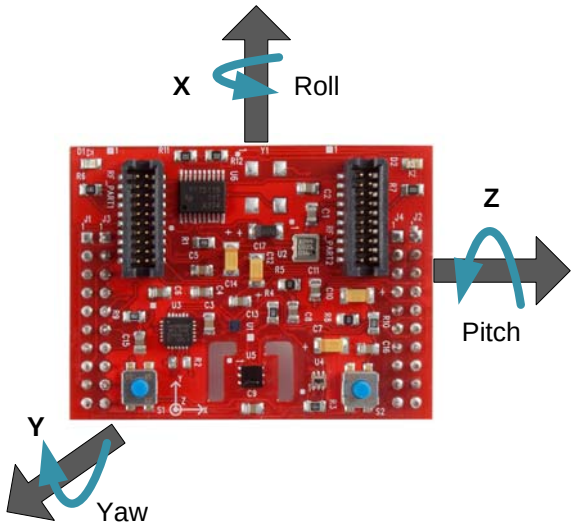


Figure 8-2 Set of Euler Angles

It also produces the rotation quaternion[1]. The Direct Cosine Matrix (DCM) algorithm (complimentary filtered DCM) in the library file compdcm_mpu9150 provided as part of the TivaWare Sensor Library, fuses the MPU-9150 sensor measurements into a single set of Euler angles[2].

The scope of the experiment is to monitor only the accelerometer values. An accelerometer sensor output value is a scalar corresponding to the magnitude of the acceleration vector. The raw sensor measurements after computation in the Sensor Library will be in float type and is converted

1. Quaternions are mathematical notations for representing orientations and rotations of objects in three dimensions.
2. A set of three angles that describe the orientation of an object in 3- Dimensional space.

to real before printing on the serial terminal. The raw accelerometer (in g) reading will be loaded in the variables pfAccel, pfAccel+1, pfAccel+2(three axis, x, y, z value).

The functional block diagram for interfacing the accelerometer with the EK-TM4C123GXL is shown in **Figure 8-3**.

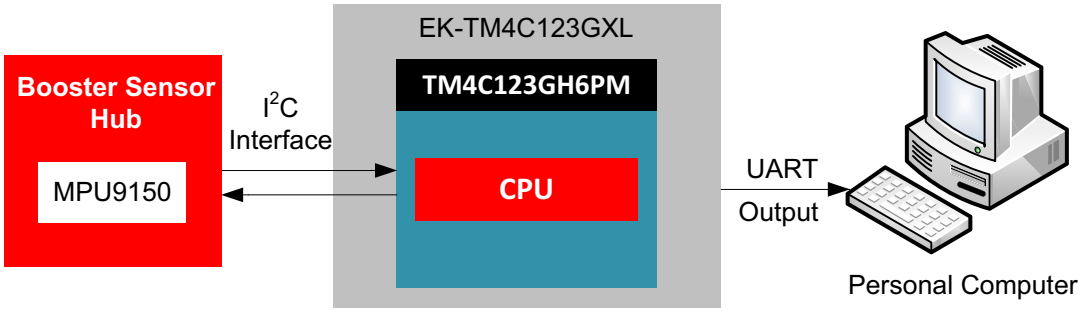


Figure 8-3 Functional Block Diagram

8.3 Component Requirements

8.3.1 Software Requirement

1. Code Composer Studio
2. TivaWare_C_Series-2.1.0.12573 version or above
3. Serial Terminal Software (ex. Tera Term)

8.3.2 Hardware Requirement

Table 8-1: Components Required for the Experiment

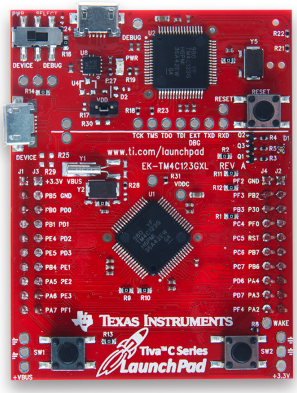
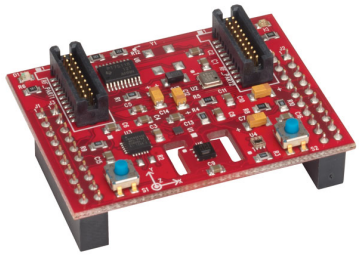
S.No	Components	Specifications	Images
1.	Tiva LaunchPad	EK-TM4C123GXL LaunchPad	
2.	USB cable		

Table 8-1: Components Required for the Experiment

S.No	Components	Specifications	Images
3.	BOOSTXL-SENSHUB BoosterPack		

8.4 Software

The software used in this experiment is available as a part of the TivaWare Sensor Library. The fusion mechanism demonstrated in the program is complimentary-filtered Direct Cosine Matrix (DCM) algorithm which is provided as part of the Sensor Library. It fuses the nine axis measurements into a set of Euler angles: roll, pitch and yaw. It also produces the rotation quaternion. The software is programmed into the target device TM4C123GH6PM on the EK-TM4C123GXL using the USB interface. The TM4C123GH6PM controls the BOOSTXL-SENSHUB for motion sensing.

8.4.1 Flowchart

The flowchart for the program is shown in **Figure 8-4**. The main program first initializes the GPIO, UART and I2C peripherals of the EK-TM4C123GXL. Then, it initializes the driver for the MPU9150 motion sensor available in TivaWare. The motion sensor is configured with application specific settings and enabled.

When the MPU9150 senses the acceleration, it sends a floating point acceleration value and sets a data ready flag. For a period interval, the IC MPU9150 sets the data ready flag in the controller through the MOTION INT pin. The acceleration values are read by the processor and converted to integer. To prevent the overflow of the UART buffer, a skip counter is used so that the UART will print the data only once for the 10 readings available. These integer values are sent to the serial terminal via UART for display on the terminal window.

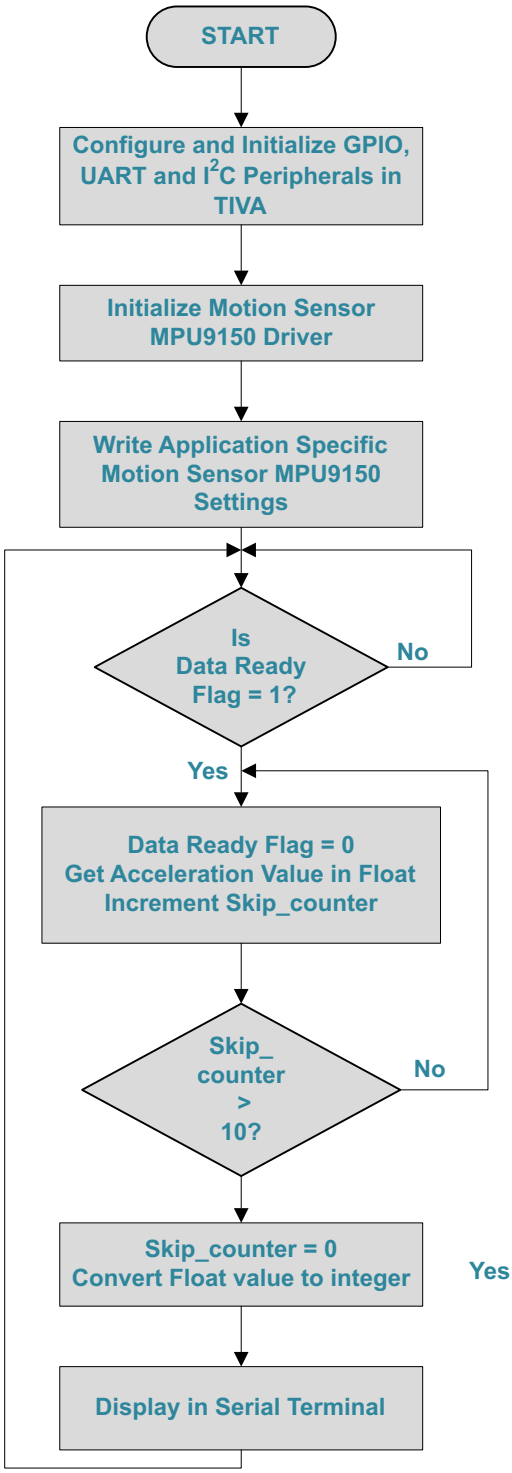


Figure 8-4 Flowchart for Interfacing an Accelerometer with EK-TM4C123GXL

8.4.2 Steps for Creating, Building and Debugging Project:

1. Install TivaWare.

2. Open CCS and create a new workspace.

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

4. Import *compdcm_mpu9150* project from TivaWare using the following steps:

- Choose the **Import Project** link on the TI Resource Explorer page
- Select the **Browse** button in the Import CCS Eclipse Projects dialog box
- Select the *compdcm_mpu9150* directory within

C:\ti\TivaWare_C_Series-2.1.0.12573\examples\boards\ek-tm4c123gxl-boostxl-SENSHUB\compdcm_mpu9150

5. Open compdcm_mpu9150.c and comment the portion from the Line No. 659 to 690.

The printing of all these values are commented and hence prevented to be sent to the serial terminal.

```
/*  // Print the acceleration numbers in the table
UARTprintf("\033[5;17H%3d.%03d", i32IPart[0], i32FPart[0]);
UARTprintf("\033[5;40H%3d.%03d", i32IPart[1], i32FPart[1]);
UARTprintf("\033[5;63H%3d.%03d", i32IPart[2], i32FPart[2]);

    // Print the angular velocities in the table.
UARTprintf("\033[7;17H%3d.%03d", i32IPart[3], i32FPart[3]);
UARTprintf("\033[7;40H%3d.%03d", i32IPart[4], i32FPart[4]);
UARTprintf("\033[7;63H%3d.%03d", i32IPart[5], i32FPart[5]);

    // Print the magnetic data in the table.
UARTprintf("\033[9;17H%3d.%03d", i32IPart[6], i32FPart[6]);
UARTprintf("\033[9;40H%3d.%03d", i32IPart[7], i32FPart[7]);
UARTprintf("\033[9;63H%3d.%03d", i32IPart[8], i32FPart[8]);

    // Print the Eulers in a table.
UARTprintf("\033[14;17H%3d.%03d", i32IPart[9], i32FPart[9]);
UARTprintf("\033[14;40H%3d.%03d", i32IPart[10], i32FPart[10]);
UARTprintf("\033[14;63H%3d.%03d", i32IPart[11], i32FPart[11]);

    // Print the quaternions in a table format.
UARTprintf("\033[19;14H%3d.%03d", i32IPart[12], i32FPart[12]);
UARTprintf("\033[19;32H%3d.%03d", i32IPart[13], i32FPart[13]);
```

```
UARTprintf("\033[19;50H%3d.%03d", i32IPart[14], i32FPart[14]);
UARTprintf("\033[19;68H%3d.%03d", i32IPart[15], i32FPart[15]);
*/
```

6. Add the 3 lines of code given below after the commented portion. This is the code to print the x, y and z values in the serial terminal. The code is now modified to print only the 3 axis values as x, y and z.

```
    // Print the x,y,z measured in a table format.
UARTprintf("x=%3d.%03d\n", i32IPart[0], i32FPart[0]);
UARTprintf("y=%3d.%03d\n", i32IPart[1], i32FPart[1]);
UARTprintf("z=%3d.%03d\n", i32IPart[2], i32FPart[2]);
```

7. Save, Build, Debug the Project and Run.

Note: This modification prints only the x, y and z axes which can be read on the serial terminal. If the modification is not done, then all the additional parameters will be displayed.

8.5 Procedure

8.5.1 Hardware setup

Connect TM4C123GXL and BOOSTXL-SENSHUB Booster Pack as shown in the **Figure 8-5**.

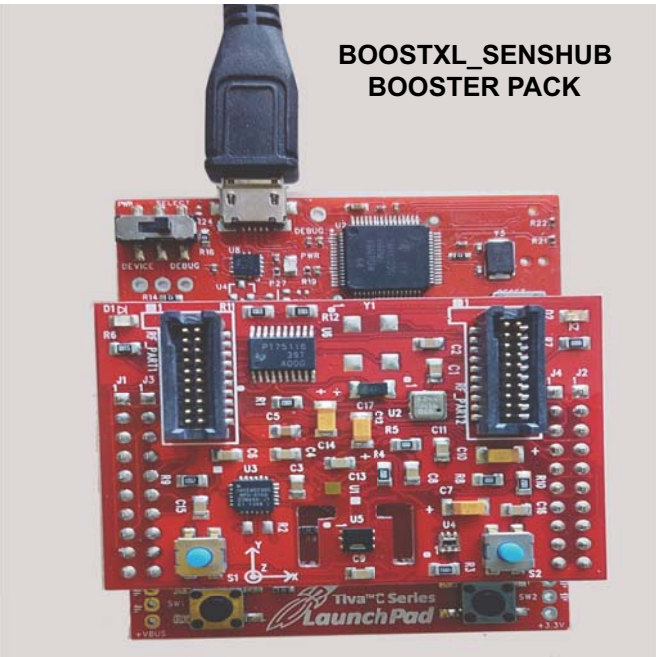


Figure 8-5 Hardware Setup

8.5.2 Software Execution

1. Configure Tera Term serial terminal for Baud rate 115200, Data bits 8, Stop Bit 1 and No Parity.

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

8.6 Observation

The RGB LED on the LaunchPad begins to blink at 1Hz after initialization is completed. The accelerometer sensor measurements are displayed on the terminal as shown in **Figure 8-6**.

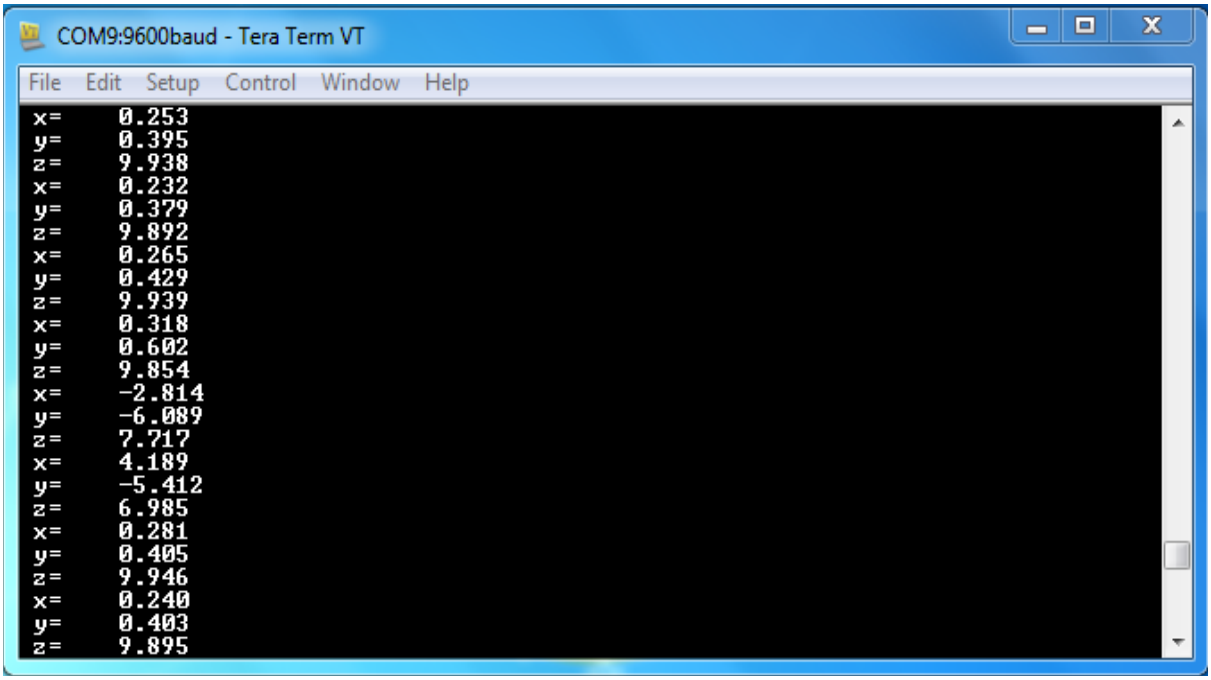


Figure 8-6 Accelerometer Sensor Output on Terminal Window

Tilt the LaunchPad with BoosterPack in different axes and observe the change in the x, y and z in the terminal software.

8.7 Summary

In this experiment, we have interfaced to the accelerometer sensor and read the values with change in position change on the serial terminal. This could be used as positional information in certain applications. We have also learnt to use a sensor library available in TivaWare for the required measurements.

8.8 Exercise

1. Change MPU9150_ACCEL_CONFIG_AFS_SEL_2G to MPU9150_ACCEL_CONFIG_AFS_SEL_4G **on line 461 of the same source file**.

```
g_sMPU9150Inst.pui8Data[0] = MPU9150_CONFIG_DLPF_CFG_94_98;
 g_sMPU9150Inst.pui8Data[1] = MPU9150_GYRO_CONFIG_FS_SEL_250;
g_sMPU9150Inst.pui8Data[2] = (MPU9150_ACCEL_CONFIG_ACCEL_HPF_5HZ |
        MPU9150_ACCEL_CONFIG_AFS_SEL_2G);<<Line 46 to be changed and observed
```

```
MPU9150Write(&g_sMPU9150Inst, MPU9150_O_CONFIG, g_sMPU9150Inst.pui8Data, 3,
             MPU9150AppCallback, &g_sMPU9150Inst);
```

Observe the difference and document it.

2. Change the value of PRINT_SKIP_COUNT to 100 and see the difference in the output.

```
// Global counter to control and slow down the rate of data to the terminal.
//
//*****************************************************************************
#define PRINT_SKIP_COUNT        10<< to be changed to 100 for observation and restored.

uint32_t g_ui32PrintSkipCounter;
//*****************************************************************************
```

3. Turn an LED ON when the acceleration value in the x axis crosses a certain limit, say +5.

4. What is the precaution taken in this experiment in order to avoid the overflow of UART buffer?

Experiment 9

USB Bulk Transfer Mode

Topics **Page**

9.1 Objective

The goal of this experiment is to transfer data using bulk transfer-mode with the USB 2.0 peripheral of the TM4C123GH6PM processor. In this experiment, we will understand and work with Bulk transfer mode of Universal Serial Bus (USB).

9.2 Introduction

This experiment demonstrates the bulk transfer of data via USB using the TivaWare USB Bulk example application (usb_dev_bulk). The application initially finds the TM4C123GH6PM device on the USB bus. If the TM4C123GH6PM is found, it prompts the user to enter strings on the serial terminal. The strings are then sent to the EK-TM4C123GXL board via USB which again returns the data to the USB host for display on the serial terminal. The functional block diagram as shown in **Figure 9-1**illustrates the working principle of the experiment.

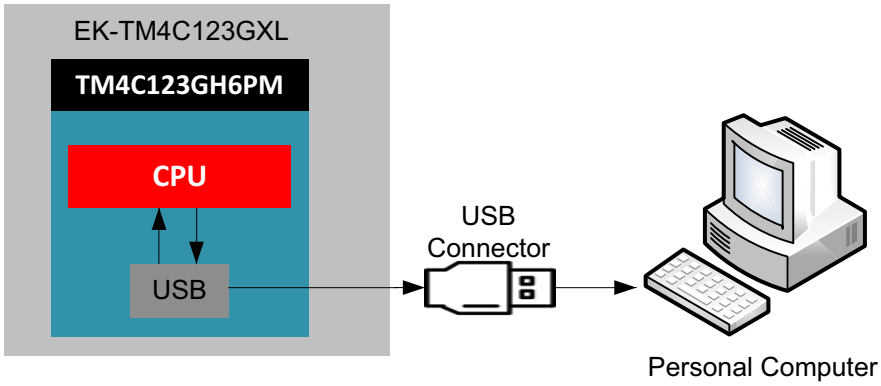


Figure 9-1 Functional Block Diagram

9.2.1 *Universal Serial Bus Controller*

The TM4C123GH6PM USB controller operates as a full-speed or low-speed function controller during point-to-point communications with USB Host, Device, or OTG (On the Go) functions. The controller complies with the USB 2.0 standard, which includes SUSPEND and RESUME signaling. 16 endpoints including two hard-wired for control transfers (one endpoint for IN and one endpoint for OUT) plus 14 endpoints defined by firmware along with a dynamic sizable FIFO that support multiple packet queuing. The presence of µDMA access to the FIFO allows minimal interference from system software. Software-controlled connect and disconnect allows flexibility during USB device start-up. The controller complies with OTG Standard's Session Request Protocol (SRP) and Host Negotiation Protocol (HNP). The USB has four different transfer modes for different applications:

- Control transfer - For command and status operations
- Interrupt transfer - For transfer of small data
- Bulk transfer - To transfer large amount of data in bursts
- Isochronous - To transfer data continuously and periodically

The TM4C123GH6PM USB module has the following features:

- Complies with USB-IF (Implementer's Forum) certification standards
- USB 2.0 full-speed (12 Mbps) and low-speed (1.5 Mbps) operation with integrated PHY

- 4 transfer types: Control, Interrupt, Bulk, and Isochronous
- 16 endpoints
  - 1 dedicated control IN endpoint and 1 dedicated control OUT endpoint
  - 7 configurable IN endpoints and 7 configurable OUT endpoints
- 4 KB dedicated endpoint memory: one endpoint may be defined for double-buffered 1023-byte isochronous packet size
- VBUS droop and valid ID detection and interrupt
- Efficient transfers using Micro Direct Memory Access Controller (µDMA)
  - Separate channels for transmit and receive for up to three IN endpoints and three OUT endpoints
  - Channel requests asserted when FIFO contains required amount of data

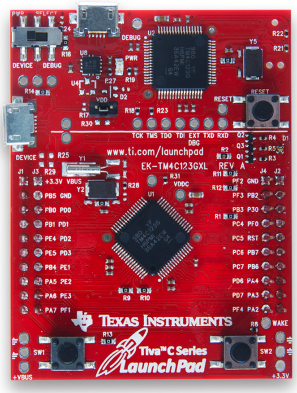
9.3 Component Requirements

9.3.1 *Software Requirement*

1. Code Composer Studio
2. USB Bulk Example Application from TI website
3. TivaWare_C_Series

9.3.2 *Hardware Requirement*

Table 9-1: Components Required for the Experiment

S.No	Components	Specification	Image
1.	Tiva LaunchPad	EK-TM4C123GXL LaunchPad	
2.	USB Cable		

9.4 Software

The software used in this experiment is available as a part of the TivaWare Sensor Library. The target device used is the TM4C123GH6PM on the EK-TM4C123GXL. The software is programmed into the EK-TM4C123GXL using the USB interface.

9.4.1 Flowchart

The flowchart for the application is shown in Figure 9-2. The application initializes the GPIO to USB. It initializes the USB transmit and receives buffers and sets the USB stack mode to device mode. The application checks for USB connection with the PC. If connected, it waits for user input. On receiving data from the PC, the case of alphabetical characters is inverted and data is transferred to the transmit buffer to resend to the PC terminal for display.

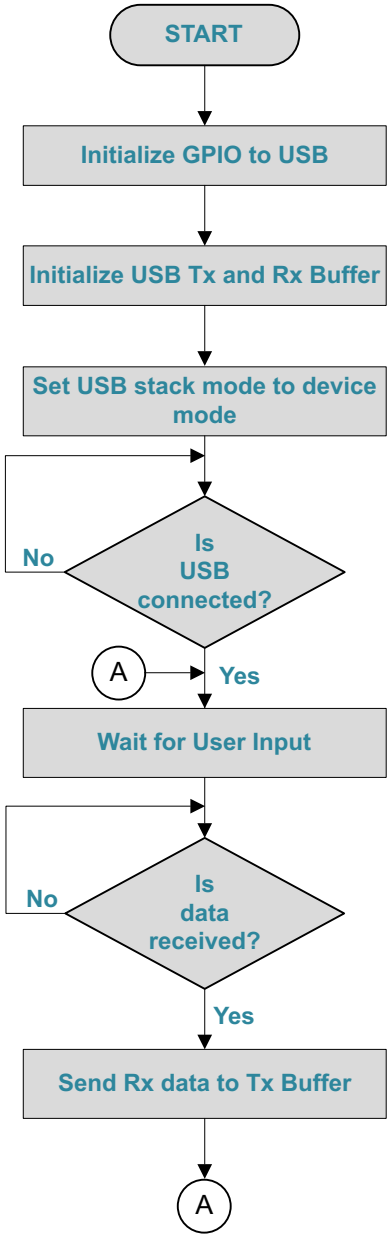


Figure 9-2 Flowchart for USB Bulk Transfer Mode of the TM4C123GH6PM Processor

9.5 Procedure

1. Install TivaWare.

2. Open CCS and create a new workspace.

3. Choose the **Import Project** link on the TI Resource Explorer page. Import usb_dev_bulk project from TivaWare using the following steps:

- Choose the **Import Project** link on the TI Resource Explorer page.
- Select the **Browse** button in the Import CCS Eclipse Projects dialog
- Select the below link for importing the USB bulk example as shown in **Figure 9-3.**

C:\ti\TivaWare_C_Series-2.1.0.12573\examples\boards\ek-tm4c123gxl\usb_dev_bulk

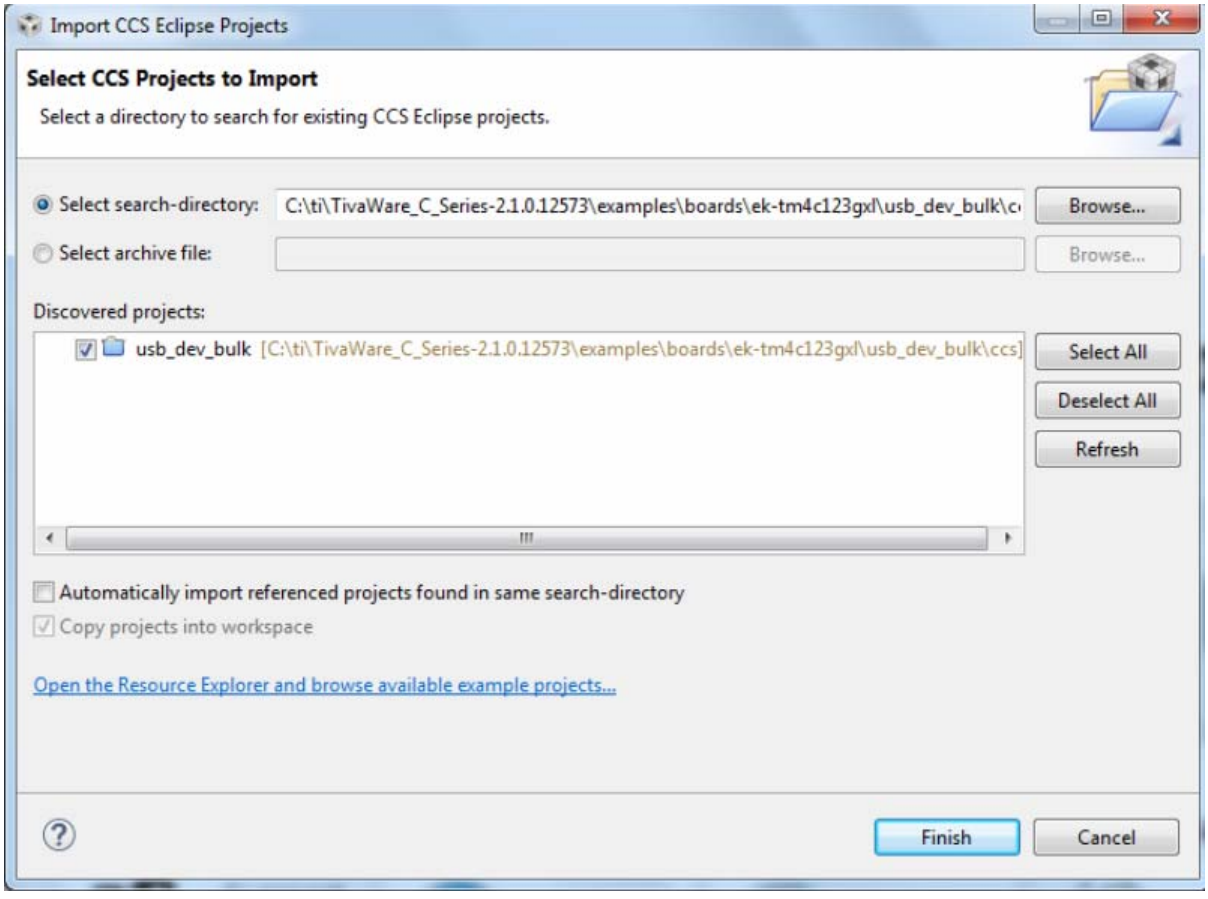


Figure 9-3 Importing USB Bulk Example

4. Build, Debug the Project and Run.

5. Click the Terminate button, when CCS returns to the CCS Edit perspective.

6. Unplug the USB cable from the DEBUG port of the LaunchPad.

7. Move the PWR SELECT switch on the board to the DEVICE position (nearest to the outside of the board) as shown in **Figure 9-4**.

8. Plug your USB cable into the USB DEVICE connector on the side of the Launch Pad board as shown in **Figure 9-4**.

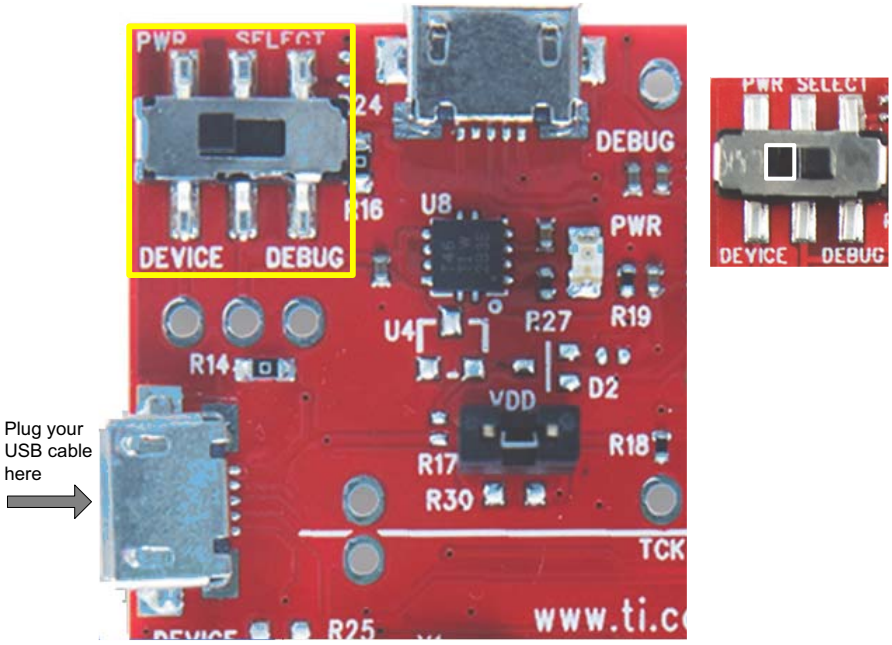


Figure 9-4 Power Switch Position and USB Device Connector on LaunchPad Board

9. The green LED in the emulator section of the Launch Pad should be lit, indicating that the power is applied to the board.

10. Within a short duration, the computer will detect that a **generic bulk device** has been plugged into the USB port.

11. Install the driver for this device from C:\TI\Tivaware_C_Series-1.1\windows_drivers. Alter the path if the CCS and TivaWare are installed in other drivers or a different path.

12. The Windows side USB examples have to be installed from www.ti.com/sw-usb-win.

13. Click **Start** → **All Programs** → **Texas Instruments** → **Stellaris** → **USB Examples** → **USB Bulk example**.

14. The terminal window appears as shown in **Figure 9-5.**

15. Type the text string that you want to transfer in Bulk Mode via USB to the LaunchPad board. Typing EXIT exits from this application.

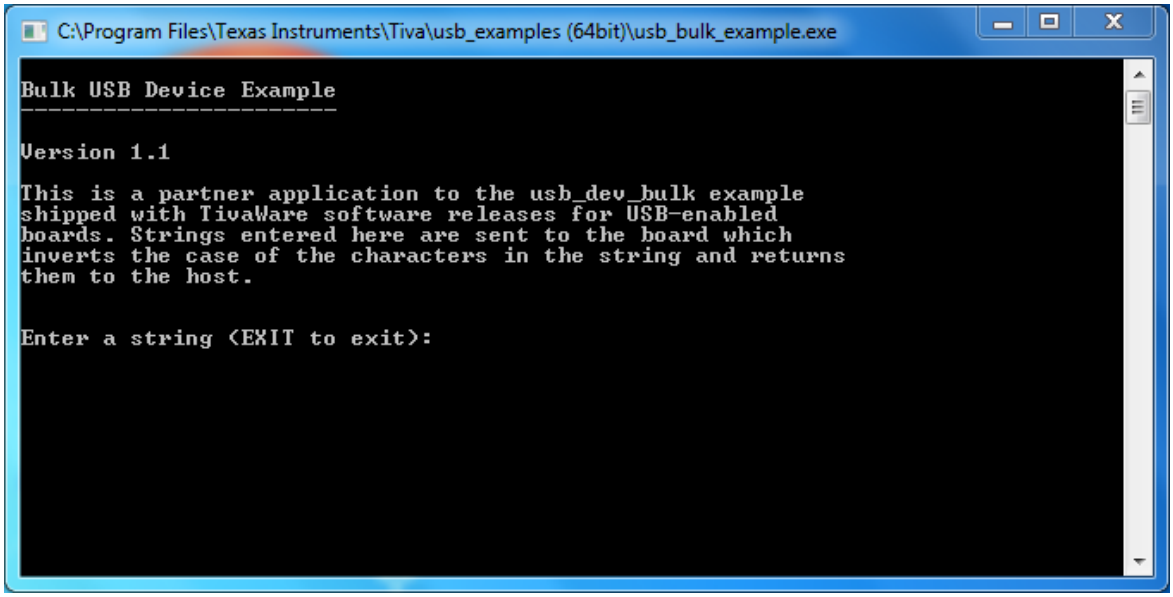


Figure 9-5 Terminal Window for USB Bulk Transfer

9.6 Observation

The data is transferred in Bulk Mode, and the count of the number of bytes transferred is displayed in the terminal window as shown in **Figure 9-6**. In device mode we can only send maximum of 256 byes for each session. It can also be observed that case conversion happens on the typed text which is also indicated on the terminal window. Typing EXIT exits from the application.

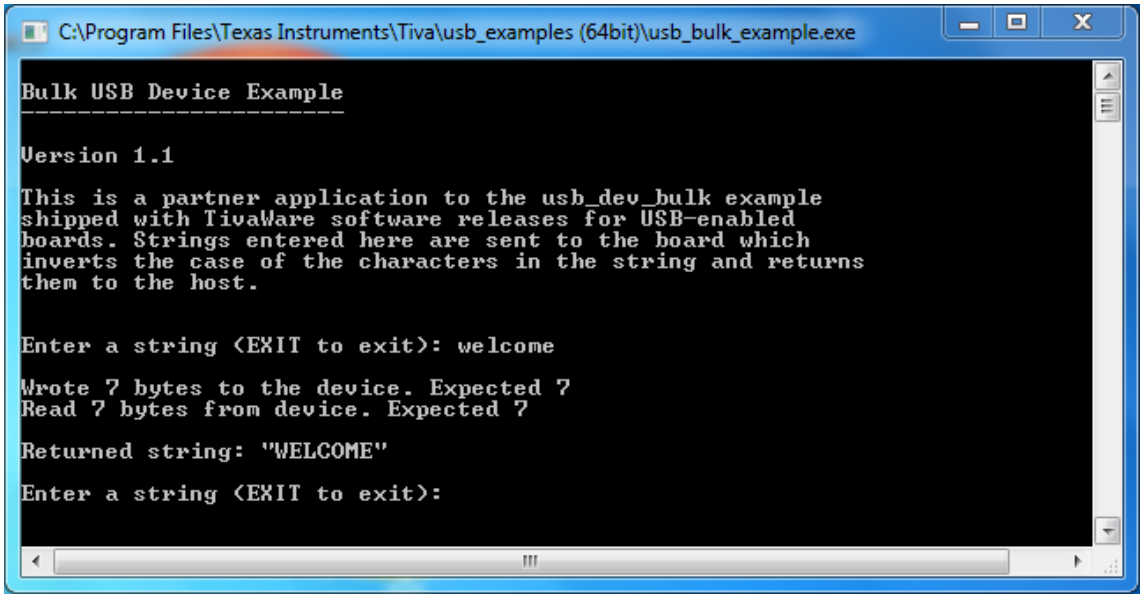


Figure 9-6 Output Terminal Window for USB Bulk Transfer

9.7 Summary

In this experiment, we have successfully implemented the transfer of data in Bulk Transfer mode of USB 2.0 with EK-TM4C123GXL.

9.8 Exercise

1. What are the different modes offered by USB 2.0?
2. What are the typical devices that use Bulk transfer mode?

Experiment 10

Using IQmath Library

Topics **Page**

10.1 Objective

The goal of this experiment is to find the angle and hypotenuse of a right angle triangle using IQmath library of TivaWare. In this experiment, we will learn computing precision math with float and double data types and work with IQmath Library.

10.2 Introduction

10.2.1 IQmath Library

IQmath (High Accuracy Mathematical Functions) Library is a collection of highly optimized and high precision mathematical function library for C/C++ programmers to seamlessly port the floating-point algorithm into fixed point code. These routines are typically used in computationally intensive real-time applications where optimal execution speed and high accuracy is critical. By using these routines, execution speeds that are considerably faster than equivalent code written in standard ANSI C language can be achieved. In addition, by providing ready-to-use high precision functions, IQmath library can significantly shorten the application development time.

10.2.2 Trigonometry

A Right-Angled Triangle is a triangle in which one angle is at a right angle (90-degree). The relation between the sides and angles of a right triangle is the basis for trigonometry. The side opposite the right angle is called the Hypotenuse as shown in **Figure 10-1**.

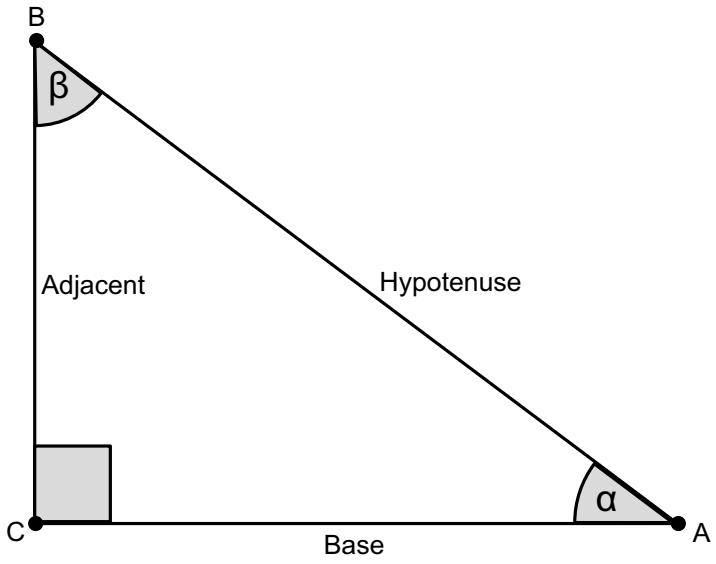


Figure 10-1 A Right Angled Triangle

1. To find Hypotenuse:

According to Pythagoras theorem,

$base^2 + adjacent^2 = hypotenuse^2$ (10.1)

We can use the above equation to find hypotenuse of the right-angled triangle. To find out the square root of a number using IQ numbers, IQmath library provides a function *_IQmag*. It is only needed to pass the base and adjacent variables to the *_IQmag* function.

2. To find Alpha (α):

$\sin(\alpha)$ = adjacent / hypotenuse

$\alpha = \sin^{-1}$(adjacent / hypotenuse) (10.2)

To find the *sin-1(x)* IQmath library provides a function*_IQasin*.

3. To find Beta(β):

$\cos(\beta)$ = adjacent / hypotenuse

$\beta = \cos^{-1}$(adjacent / hypotenuse) (10.3)

To find the $cos^{-1}(x)$ IQmath library provides a function *_IQacos*.

This experiment uses the IQmath library of Texas Instruments to perform mathematical computations to find the angle and adjacent of a right angled triangle. The IQmath functions can be used in the C program by including the IQmathLib.h header file and linking the code with the IQmath.lib object code library.

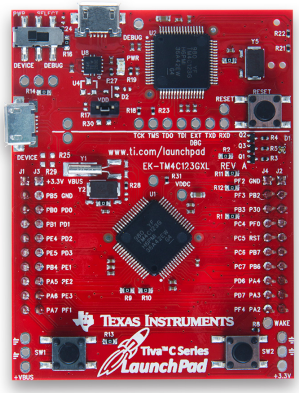
10.3 Component Requirements

10.3.1 Software Requirement

1. Code Composer Studio
2. TivaWare_C_Series-2.1.0.12573 or above version

10.3.2 Hardware Requirement

Table 10-1: Components Required for the Experiment

S.No	Components	Specifications	Images
1.	Tiva LaunchPad	EK-TM4C123GXL LaunchPad	
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 TM4C123GH6PM on the EK-TM4C123GXL using the USB interface.

10.4.1 Flowchart

The flowchart for the program is shown in **Figure 10-2**. The IQ variables required for calculation are defined. Floating point variables are defined to store and view the result. The system clock is configured to 40MHz. In this program, the base of the triangle is set to 4 and adjacent is 3. The hypotenuse and the angles α and β are calculated using IQmath library functions. The result obtained is in radians. The angles in radians are converted to degrees and the results in IQ format are converted to double by multiplying with 2^{-16}. The result is stored in a variable which can be viewed in the watch window.

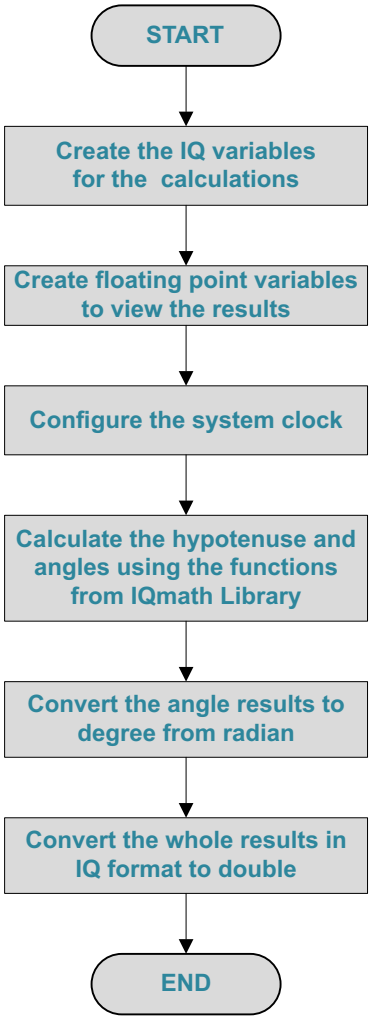


Figure 10-2 Flowchart for Computing Angle and Hypotenuse of A Right Angled Triangle

10.4.2 Addition of IQmath Library

The steps to be followed to add the IQmath Library into the project are:

1. Click on **File Search Path** under **ARM Linker**.

2. Add "**IQmathLib.lib**" from IQmath\ccs\IQmathLib\Debug inside the TivaWare_C_Series installation directory to the "Include Library" section as shown in **Figure 10-3**.

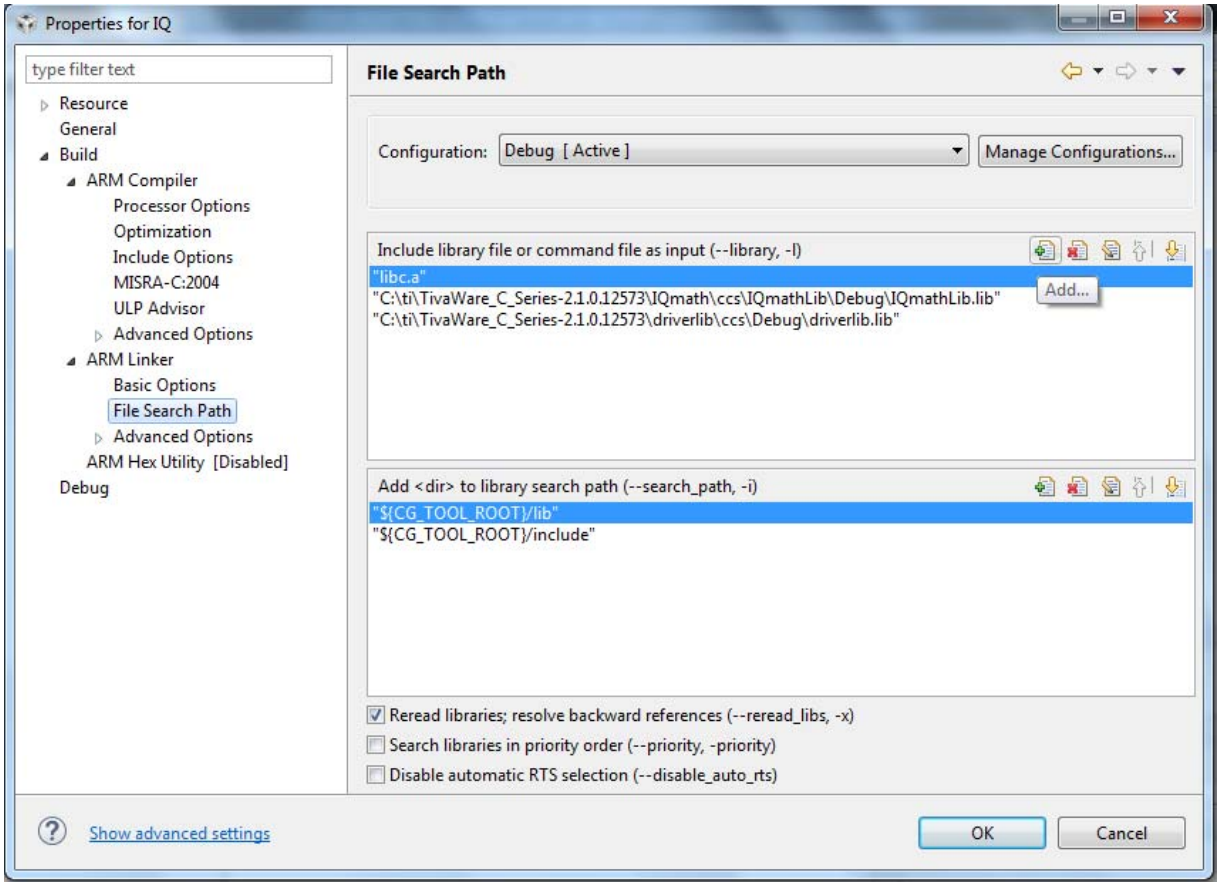


Figure 10-3 Including IQmath Library in Properties Window

10.4.3 C Program Code for Calculation using IQmath Library

```
#include<stdint.h>
#include<stdbool.h>
#include"inc/hw_memmap.h"
#include"inc/hw_types.h"
#include"driverlib/sysctl.h"
#include"math.h"
#include"IQmath/IQmathLib.h"
//one radian in degree
#define DEGREE 57.2957795
//Declared as global variables to access in watch windowon ccs
doubleg_result_hypo;
doubleg_result_angle_alpha_R, g_result_angle_beta_R;
doubleg_result_angle_alpha_D, g_result_angle_beta_D;
```

```
intmain(void)
{
//Local Variables
double IQ_Num_2_Float_Mul_Const;
_iq16 base, adj, hypo, angle_alpha_R, angle_beta_R, angle_alpha_D,
angle_beta_D;
//Configure system clock to 80MHz
SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ| SYSCT-
L_OSC_MAIN);
//Set base of the right triangle to 3cm
base=  _IQ16(3);
//Set adjacent of the right triangle to 4cm
adj=  _IQ16(4);
//Computes the square root of A^2 + B^2 using IQ numbers
hypo =  _IQ16mag(base, adj);
//Results in radian, which is in IQ number format
angle_alpha_R   = _IQ16asin(_IQ16div(adj, hypo));
angle_beta_R     = _IQ16acos(_IQ16div(adj, hypo));
//Results in degree, which is in IQ number format
angle_alpha_D   = _IQ16mpy(angle_alpha_R, _IQ16(DEGREE));
angle_beta_D     = _IQ16mpy(angle_beta_R,  _IQ16(DEGREE));
//Convert IQ number to double: multiply IQ number by 2^-n
IQ_Num_2_Float_Mul_Const = pow(2, -16);
g_result_hypo=  hypo * IQ_Num_2_Float_Mul_Const;
g_result_angle_alpha_D=  angle_alpha_D * IQ_Num_2_Float_Mul_Const;
g_result_angle_alpha_R=  angle_alpha_R * IQ_Num_2_Float_Mul_Const;
g_result_angle_beta_D=  angle_beta_D   * IQ_Num_2_Float_Mul_Const;
g_result_angle_beta_R=  angle_beta_R   * IQ_Num_2_Float_Mul_Const;
while(1);  //program halted here
return 0;
}
```

Table 10-2: IQmath Functions Used in the Application Program

IQmath Function	Description
_IQN	_iqN is the C-code data type definition equating a long, a 32-bit value representing a IQN number, where N=1:30
_iqN _IQNmag(_iqN A, _iqN B)	Magnitude Square: sqrt(A^2 + B^2)
_iqN _IQNasin(_iqN A)	High precision ASIN (output in radians)
_iqN _IQNacos(_iqN A)	High precision ACOS (output in radians)
_iqN _IQNmpy(_iqN A, _iqN B)	IQ Multiplication

10.5 Procedure

Build, program and debug the code.

10.6 Observation

Figure 10-4 shows the result for a right angled triangle of base 3 cm and adjacent 4 cm.

Angle alpha and beta are provided both in radian (_R) and degree (_D). All these variables have to be added to the watch window to observe the results before running the program.

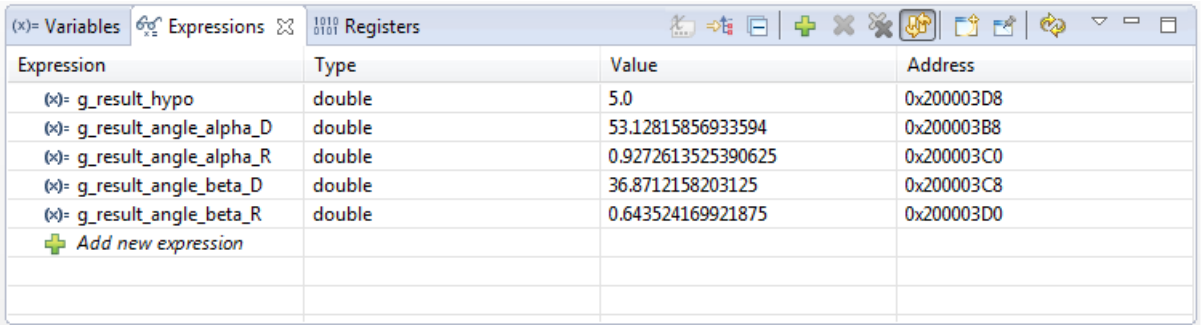


Figure 10-4 Output Result for Right Angled Triangle of Base 3cm and Adjacent 4cm

10.7 Summary

In this experiment, we have successfully used the IQmath library to find the angle and hypotenuse of a right-angled triangle. We have also learnt how to add IQmath library to our project to perform high accuracy mathematical calculations.

10.8 Exercise

1. Change the base and adjacent values in the program to other values, build the program and observe the values in the watch window.

2. For the same values used above, change double declaration to float and change the functions to compute in float. Tabulate the results and compare with double precision. The watch window

results as shown in **Figure 10-5** are for the values base 3cm and adjacent 4cm. Record your observation for different values. Observe change in precision between float and double.

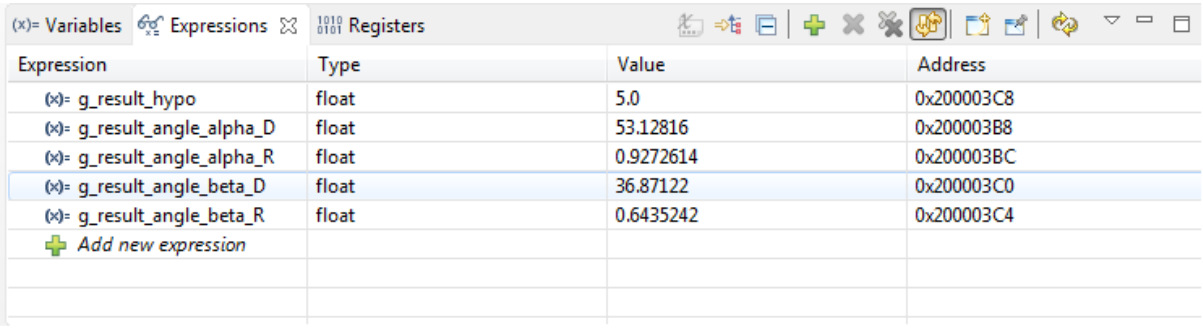


Figure 10-5 Output Result for Computation in Float

3. Open IQmathLib.h and browse through the available functions. What function is to be used if the IQ number used in the program is to be converted to a string?

Experiment 11

Setting of Static IP Address

Topics **Page**

11.1 Objective

The goal of this experiment is to configure a static IP address for CC3100 Booster Pack plugged into the EK-TM4C123GXL.

11.2 Introduction

An Internet Protocol address (IP address) is a numerical label assigned to each device (e.g., PC, Printer) connected to a computer network which uses the Internet Protocol for communication. IP addresses are required by any network adapter on any computer that needs to connect to the Internet or any another computing device in the network. Addresses are given out to network computers in either dynamic or static.

Addresses which are provided dynamically are done from a server on the network called a "DHCP" (Dynamic Host Configuration Protocol) Server. The server accesses a database that contains all of the IP addresses available for use on the network. The DHCP server ensures that the same IP address is not given to two different computers. By default, all computer operating systems use DHCP to configure their IP settings. Any of the computing devices that are plugged into the network will try to obtain an IP address from the DHCP.

The dynamic address keeps changing for every session. If the requirement is to have a fixed address, then a static IP address is to be sought which comes with a cost. More details on the advantages and disadvantages of Dynamic and Static IP can be referred from the Internet.

This experiment demonstrates the configuration of a static IP address for CC3100. The device acts as a station and connects to the Access Point (AP) with the configured static IP. The static IP address is stored inside the non-volatile memory of the CC3100. The functional block diagram as shown in **Figure 11-1** illustrates the working principle of the experiment.

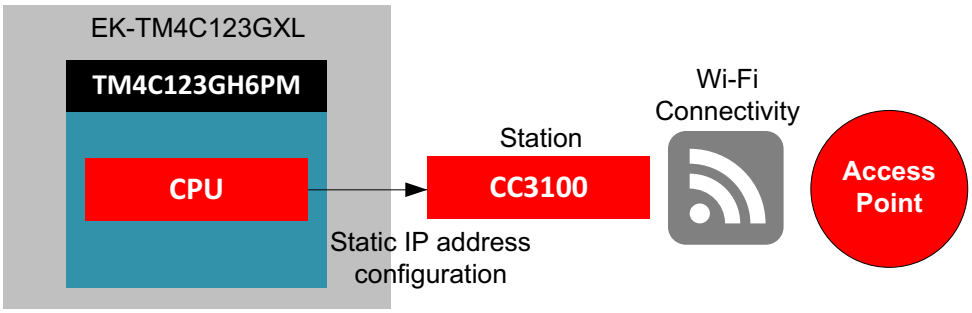


Figure 11-1 Functional Block Diagram

11.2.1 CC3100 Booster Pack

The SimpleLink Wi-Fi CC3100 solution provides the flexibility to add Wi-Fi to any microcontroller (MCU). This Internet-on-a-chip solution contains all that is needed to easily create IoT[1] solutions - security, quick connection, cloud support, publicly available documentation, E2E support forums and more. The CC3100 BoosterPack can be directly connected to a compatible LaunchPad using the standard 2x20 pin connectors.

1. The Internet of Things (IoT) is a network of objects embedded with electronics, software, sensors, and network connectivity, for exchange of data.

The features of CC3100 are:

- CC3100 Wi-Fi Network Processor in QFN package
- 2 20-pin stackable connectors (BoosterPack headers) to connect to TI LaunchPads and other BoosterPacks
- On-board chip antenna with option for U.FL-based testing
- Power from on-board LDO using USB OR 3.3V from MCU LaunchPad
- 0.8 megabit serial flash
- 40 MHz crystal, 32 KHz crystal and oscillator
- U.FL and chip antenna
- USB

The Wi-Fi subsystem has several configurable parameters that control its behavior. The host driver uses different APIs to configure these parameters. If not configured, by default the CC3100 device will operate as a station. The IP address, which is a network parameter, is configured to the Wi-Fi subsystem by the host (here, TM4C123GH6PM MCU of the EK-TM4C123GXL). In case of static IP address configuration, the user can set the IP address.

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

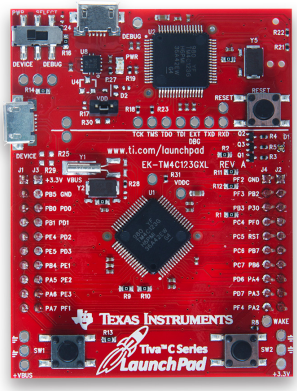
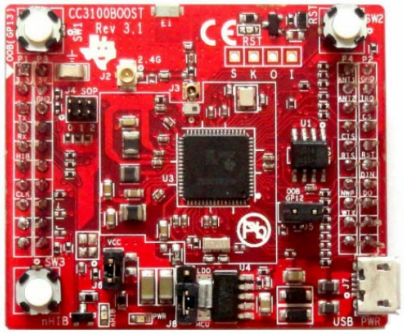
S.No	Components	Specifications	Images
1.	Tiva LaunchPad	EK-TM4C123GXL LaunchPad	
2.	USB cable		

Table 11-1: Components Required for the Experiment

S.No	Components	Specifications	Images
3.	CC3100 Booster pack		
4.	Wireless Network connection with its name, security type and password	To act as Access Point	

11.4 Software

The software is programmed into the target device TM4C123GH6PM on the EK-TM4C123GXL using the USB interface. The TM4C123GH6PM controls the CC3100BOOST for Wi-Fi transmission.

11.4.1 Flowchart

The flowchart for the experiment is shown in **Figure 11-2**. The CC3100 Booster pack 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 established.

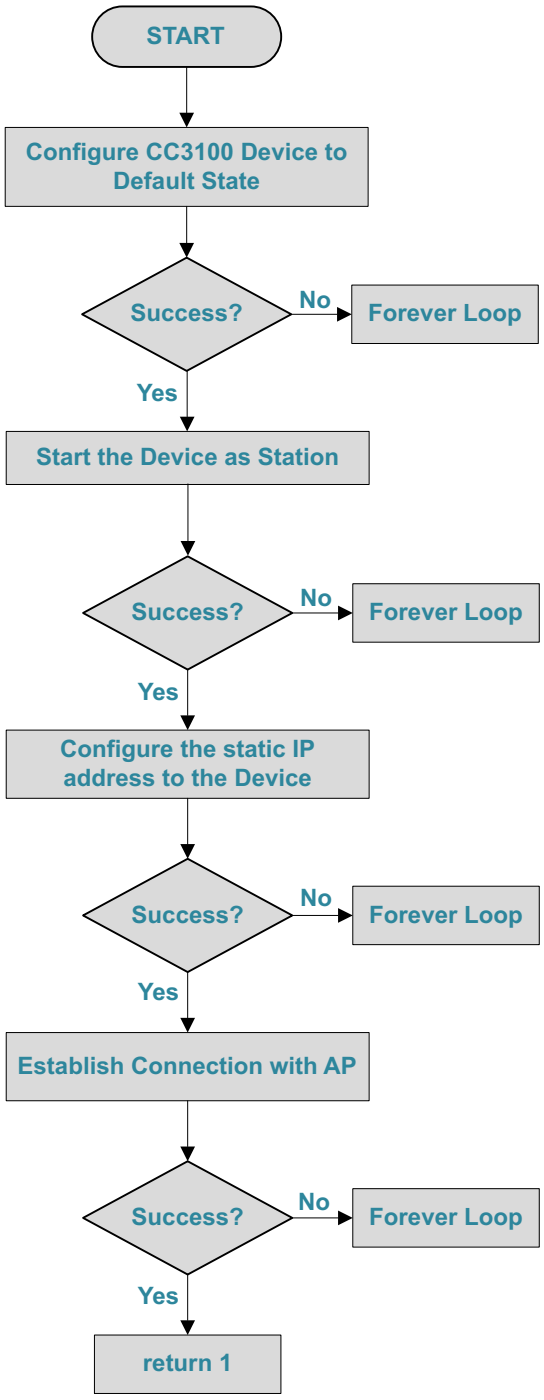


Figure 11-2 Flowchart for Configuration of Static IP address to the Device

11.5 Procedure

11.5.1 Hardware Setup

Connect EK-TM4C123GXL and CC3100 Booster pack as shown in **Figure 11-3**. The Booster pack directly plugs into the LaunchPad. Note that the USB cable is directly connected to the Booster-Pack to power it only. For debugging, the USB cable on the EK-TM4C123GXL is also required.

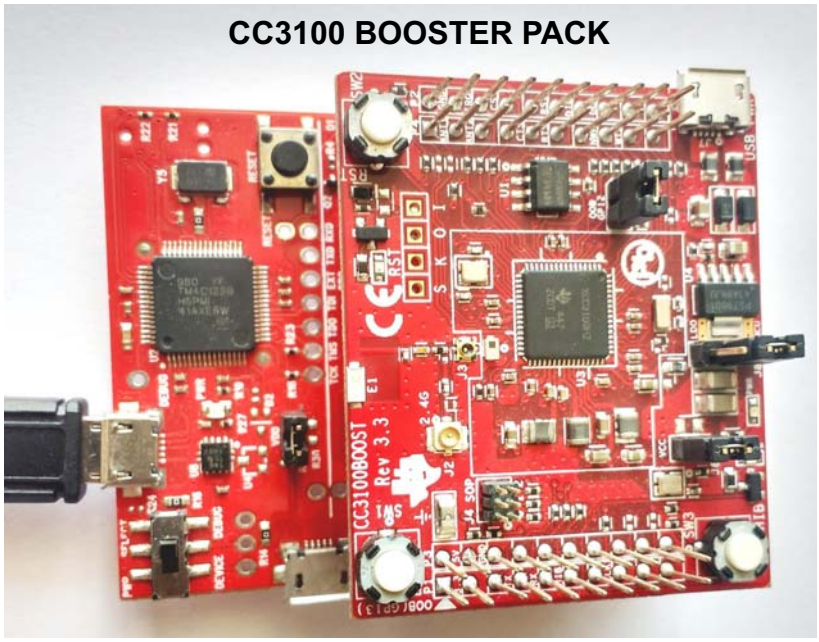


Figure 11-3 Hardware Setup

11.5.2 Steps for Creating, Building and Debugging 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 "getting started with wlanap" 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 **getting_started_wih_wlan_ ap**directory within

C:\TI\CC3100SDK_1.1.0\cc3100-sdk\platform\tiva-c-launchpad\example_project_ccs\getting_started_wih_wlan_ap

4. Create a copy of the project in the same workspace and rename the copy as **ip_configuration**. To copy the project just right click on the project name, click copy, then right click on the project explorer window and paste it.

5. Go to **C:\TI\CC3100SDK_1.1.0\cc3100-sdk\examples\ip_configuration**. Open the main.c, copy the code and paste it in the main.c of **ip_configuration** just created.

6. Edit following parameters in sl_common.h(line 49 of main.c) - The device connects to this Access Point (AP) on getting the corresponding command from application. Open sl_common.h

by clicking on sl_common.h while pressing ctrl key. The sl_common.h can be also opened by right clicking on sl_common.h and select "open declaration".

```
#include "simplelink.h"

#include "sl_common.h"

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

<< Line 49 Replace STEPS with your Access Point Name.

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 */
```

If the Security is open type with no security replace the definitions as below

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

Replace 123456 with your network password, if network is protected with security

```
#define PASSKEY       "123456" /* Password in case of secure AP */
```

If the Network has no password

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

7. Connect the Access Point on your PC with the following steps.

   a. Goto Control Panel → Network and Internet → Network and Sharing Center

   b. Click on Wireless Network Connection as shown in **Figure 11-4**.

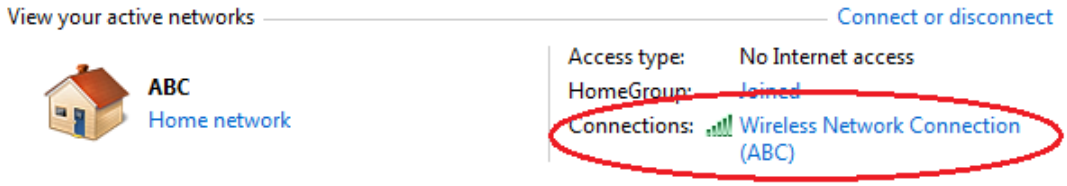


Figure 11-4 Connection to Access Point

   c. Click on Details as shown in **Figure 11-5**.

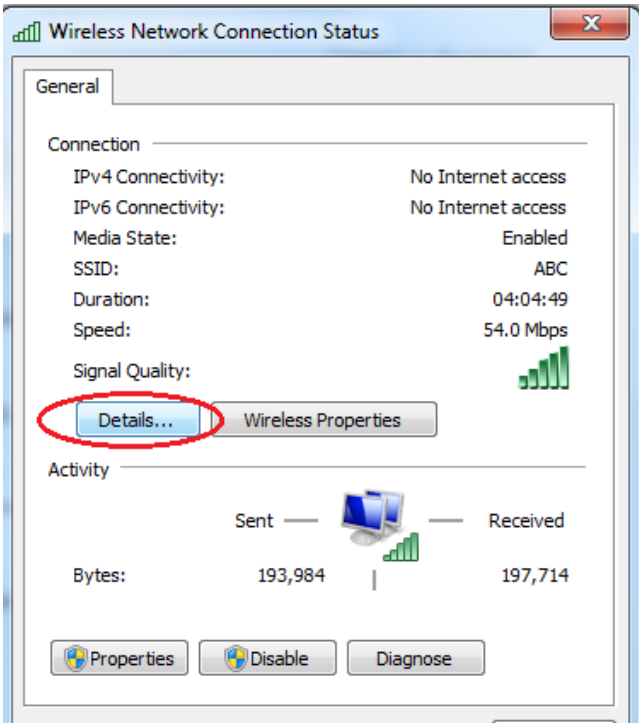


Figure 11-5 Wireless Network Connection Status

d. From the Network Connection Details window, note the IPv4 Address format, MASK, DNS and GATEWAY address of your Access Point.

8. The static IP address is defined in the main.c, line 59. This will be the IP address with which the device will be connected to the Access Point(AP).

```
/* Static IP to be configured, last value can be changed to your own
value */
#define CONFIG_IP       SL_IPV4_VAL(xxx,xxx,xxx,45)
/* Subnet Mask for the station */
#define AP_MASK         SL_IPV4_VAL(xxx,xxx,xxx,xxx)
/* Default Gateway address */
#define AP_GATEWAY      SL_IPV4_VAL(xxx,xxx,xxx,xxx)
/* DNS Server Address */
#define AP_DNS          SL_IPV4_VAL(xxx,xxx,xxx,xxx)
```

Note: xxx denotes the values noted down in the previous step.

9. Double click on the project name go to spi and open spi.c.

- Comment the line 76
  `ROM_GPIOPinWrite(GPIO_PORTE_BASE,GPIO_PIN_0, PIN_HIGH);`
  `/* Enable pull up on PB1, CC3100 UART RX */`

```
//GPIOPadConfigSet(GPIO_PORTB_BASE, GPIO_PIN_1, GPIO_STRENGTH_4MA, //
GPIO_PIN_TYPE_STD_WPU);
```

<< Line 76 to be commented

```
SysCtlPeripheralEnable(SYSCTL_PERIPH_SSI2);
```

10. Open the project property by right clicking on project name and selecting properties.

Update TIVAWARE_ROOT variable available under Resource → Linked Resources with TivaWare root directory as shown in **Figure 11-6**.

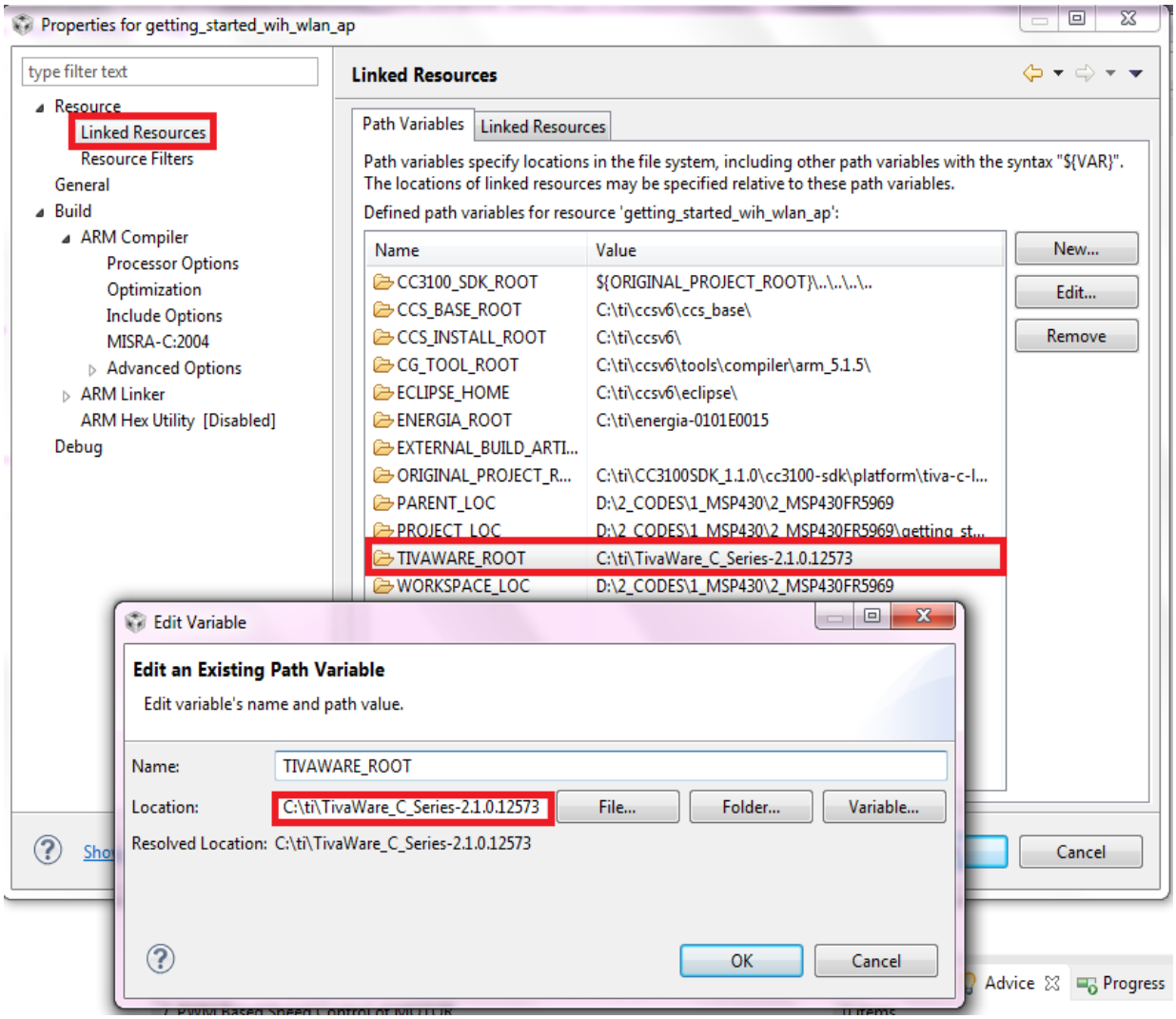


Figure 11-6 Updating TIVAWARE_ROOT Variable in Project Properties

11. Save, Debug and Run.

11.6 Observation

1. Open Tera Term Terminal Software on the PC where the EK-TM4C123GXL is connected. The serial window outputs the debug messages received from the Tiva Hardware. The serial port parameters to be set are 115200 baud rate, data bits 8, No parity and 1 stop bit.

2. The device connects to the 'STEPS' access-point with the Static IP defined in 'CONFIG_IP' as shown in **Figure 11-7**.

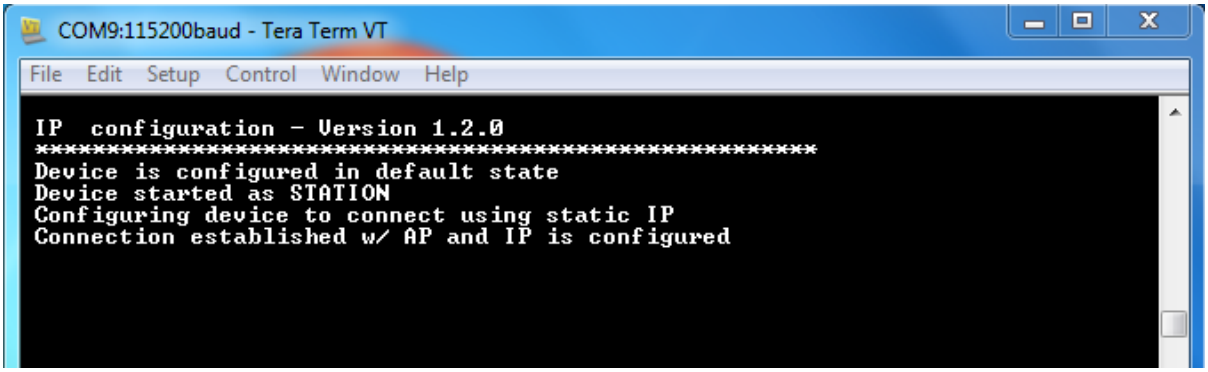


Figure 11-7 Debug Messages on Terminal Window for Connection to the STEPS Access Point

3. The connection could be checked by pinging CC3100 on the static IP address as shown in **Figure 11-8**.

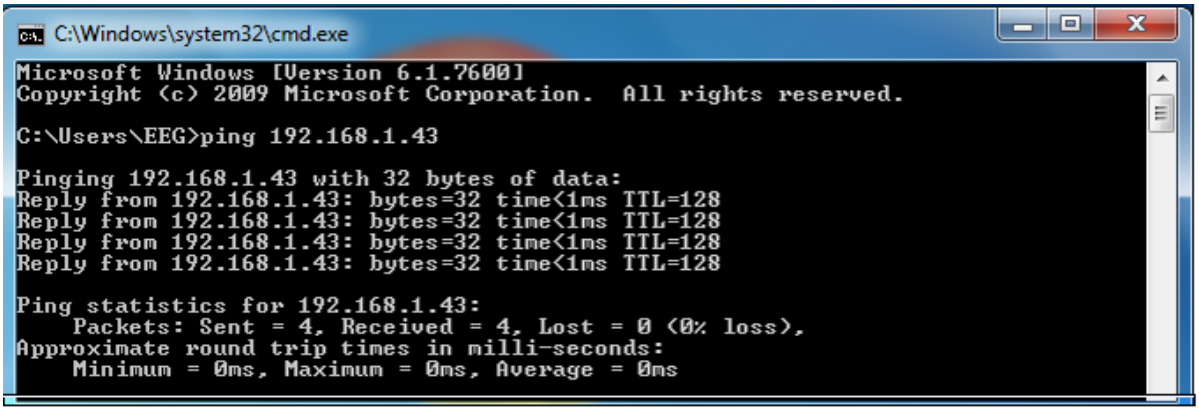


Figure 11-8 Pinging to CC3100 on the Static IP Address by the Tera Term

11.7 Summary

In this experiment we have successfully configured a static IP address to the CC3100 device and connected to an Access Point (AP).

11.8 Exercise

1. Try to ping the same IP address before connecting to the Access Point (AP) and note down the observation.

2. What is the difference between static IP address and dynamic IP address?

Experiment 12

CC3100 as WLAN Station

Topics **Page**

12.1 Objective

The goal of this experiment is to configure the CC3100 Booster Pack as a Wireless Local Area Network (WLAN) Station. In this experiment, we will also understand the WLAN concepts and communication between Station and Access Point.

12.2 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 facilitates the users to stay connected without physical wiring constraints and also access the Internet. Wi-Fi (Wireless Fidelity) 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. A simple WLAN setup consists of the access point and stations as shown in **Figure 12-1**. The wireless clients could be any device such as computers, laptops, mobile devices, smart phones, etc.

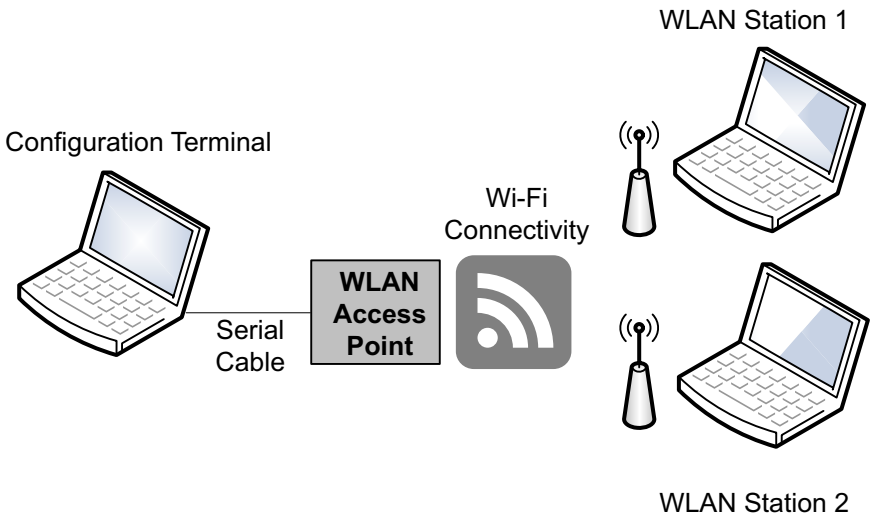


Figure 12-1 A WLAN Setup

This experiment demonstrates to configure CC3100 in WLAN-Station mode and connect to a Wi-Fi access-point. The application connects to an access-point and pings the gateway. It also checks for internet connectivity by pinging the website www.ti.com. The functional block diagram as shown in **Figure 12-2** illustrates the working principle of the experiment.

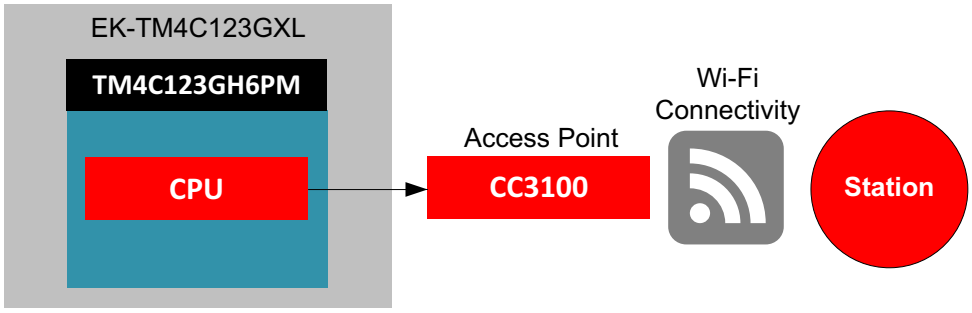


Figure 12-2 Functional Block Diagram

12.3 Component Requirements

12.3.1 Software Requirement

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

12.3.2 Hardware Requirement

Table 12-1: Components Required for the Experiment

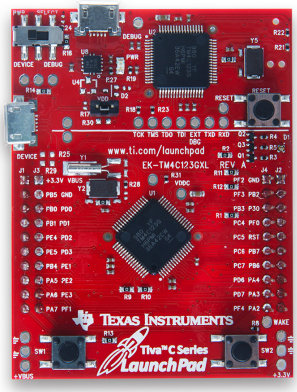
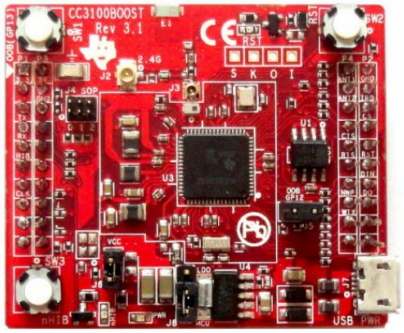
S.No	Components	Specification	Image
1.	Tiva LaunchPad	EK-TM4C123GXL LaunchPad	
2.	USB Cable		

Table 12-1: Components Required for the Experiment

S.No	Components	Specification	Image
3.	CC3100 Booster pack	CC3100BOOST	
4.	Wireless Nework connection with its name, security type and password	With Internet connectivity	

12.4 Software

The software is programmed into the target device TM4C123GH6PM on the EK-TM4C123GXL using the USB interface. The TM4C123GH6PM controls the operation of CC3100BOOST as WLAN station.

12.4.1 Flowchart

The flowchart for the experiment is shown in **Figure 12-3**. The CC3100 Booster pack is powered up in the default state. The CC3100BOOST 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 and tries to connect to the **www.ti.com** website.

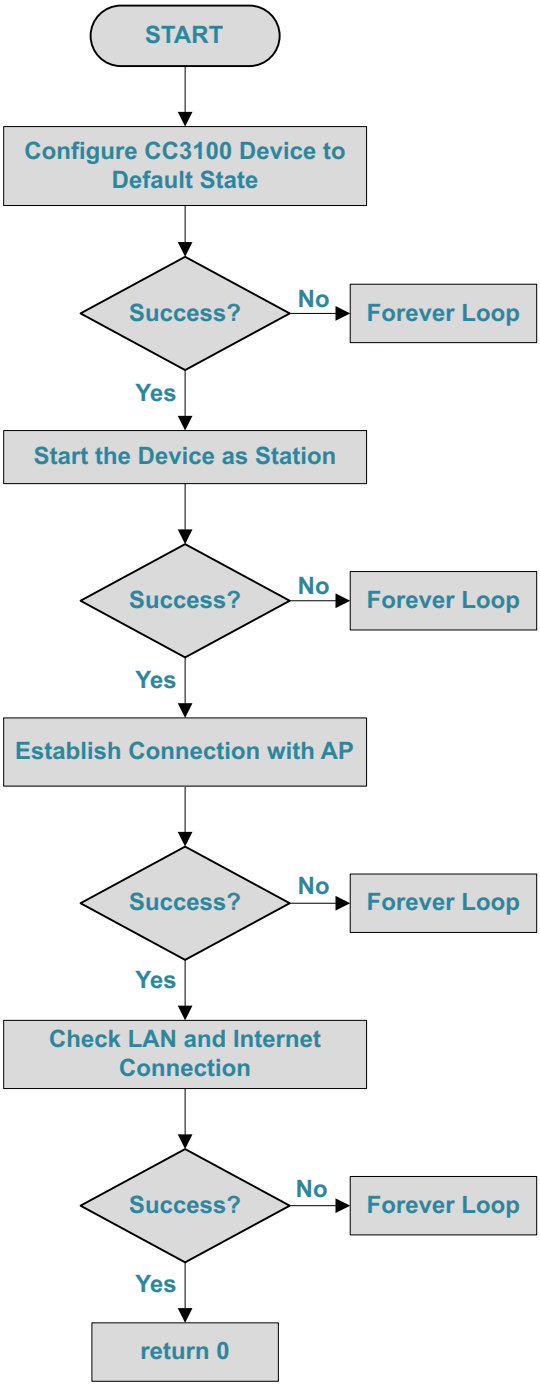


Figure 12-3 Flowchart for Configuring CC3100 as a WLAN Station to Connect to the Internet

12.5 Procedure

12.5.1 Hardware Setup

Connect EK-TM4C123GXL and CC3100BOOST as shown in **Figure 12-4**.

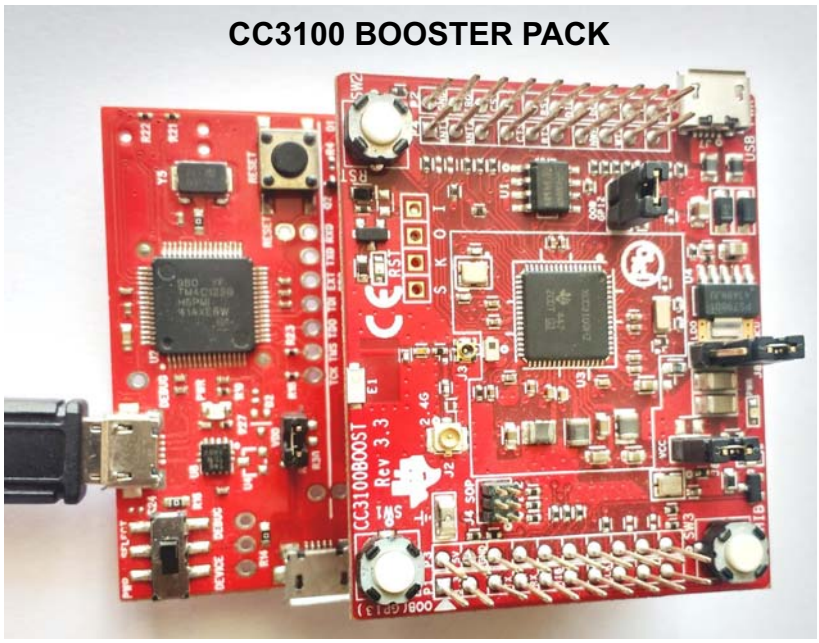


Figure 12-4 Hardware Setup

12.5.2 Steps for Creating, Building and Debugging 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 "getting started with WLAN station" project from CC3100 SDK using the following steps:

- Choose the Import Project link on the TI Resource Explorer page
- Select the Browse button in the Import CCS Eclipse Projects dialog
- Select the *getting_started_wih_wlan_station*directory within **C:\TI\CC3100SDK_1.1.0\cc3100-sdk\platform\tiva-c-launchpad\example_project_ccs\getting_started_wih_wlan_station**

4. Double click on the project name, go to spi and open spi.c.

- comment the line 76

```
ROM_GPIOPinWrite(GPIO_PORTE_BASE,GPIO_PIN_0, PIN_HIGH);
/* Enable pull up on PB1, CC3100 UART RX */
//GPIOPadConfigSet(GPIO_PORTB_BASE, GPIO_PIN_1, GPIO_STRENGTH_4MA, GPIO_PIN_TYPE_STD_WPU);
SysCtlPeripheralEnable(SYSCTL_PERIPH_SSI2);
```

<< Line 76 to be commented

5. 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. Open sl_common.h by clicking on sl_common.h while pressing ctrl key. sl_common.h can also be opened by right clicking on sl_common.h and select "open declaration".

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

Replace STEPS with your Access Point Name.

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

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 */
```

If the Security is open type with no security replace the definitions as below.

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

Replace 123456 with your network password, if network is protected with security.

```
#define PASSKEY      "123456"/* Password in case of secure AP */
```

If the Network has no password.

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

6. Open the project property by right clicking on project name and selecting properties.

Update TIVAWARE_ROOT variable available under Resource → Linked Resources with Tivaware root directory as shown in **Figure 12-5.**

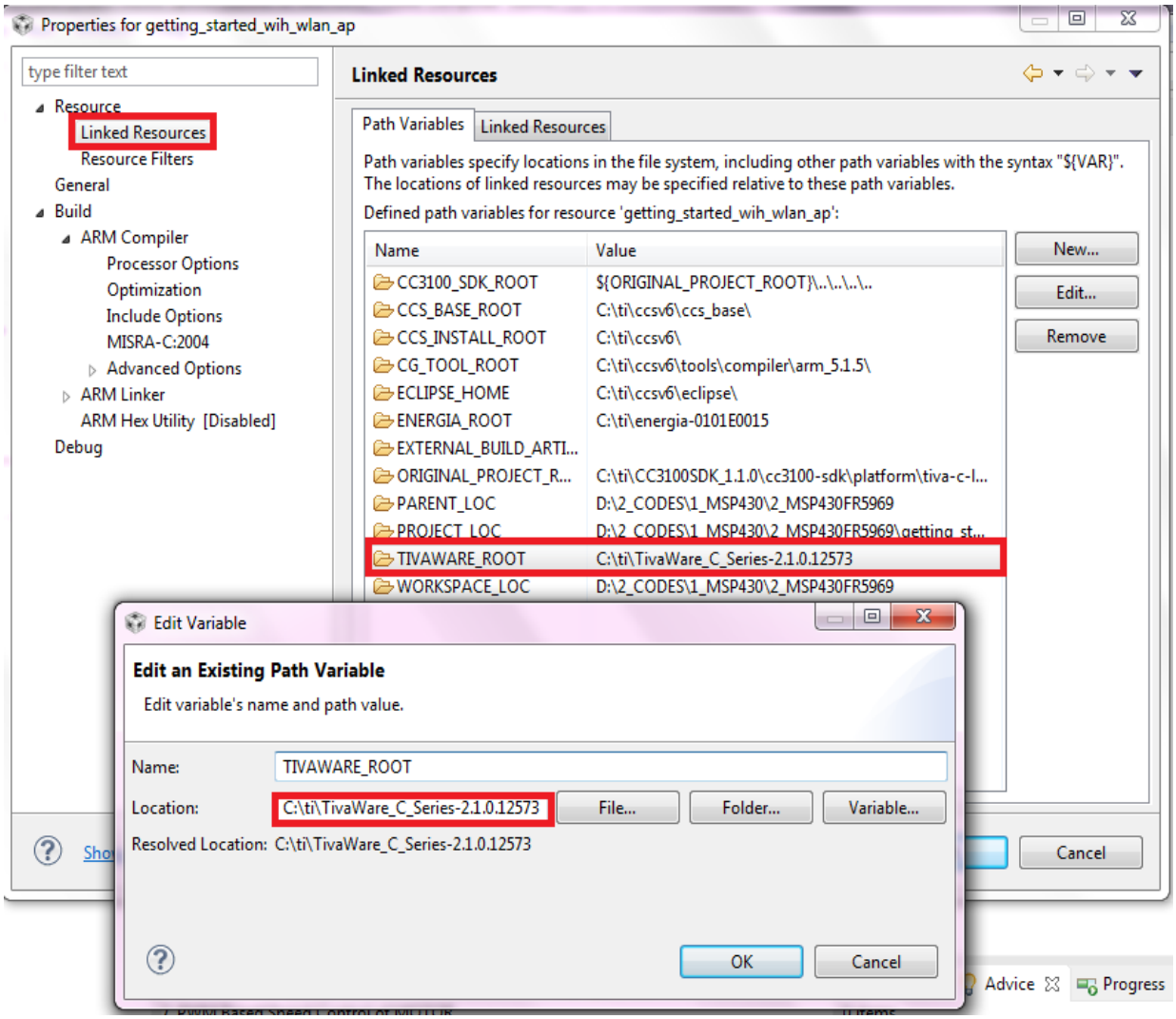


Figure 12-5 Updating TIVAWARE_ROOT Variable in Project Properties

7. Save, Debug and Run.

12.6 Observation

Open Tera Term Terminal Software on the PC where the EK-TM4C123GXL is connected. The serial window outputs the debug messages received from the Tiva Hardware as shown in **Figure 12-6**. The serial port parameters to be set are baud rate 115200, No parity, 8 data bits and 1 stop bit.

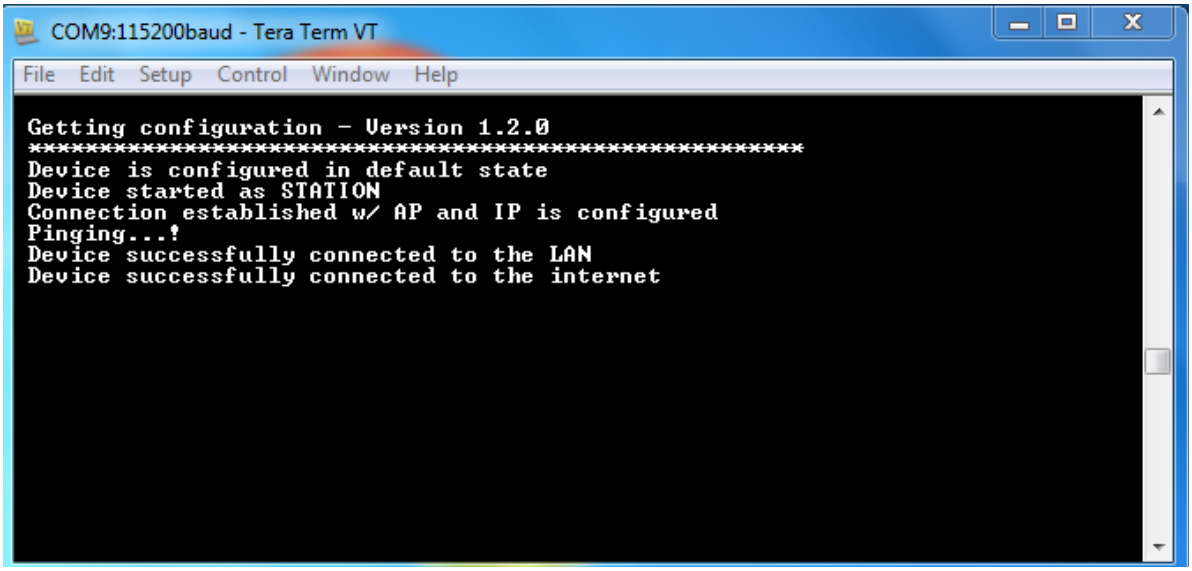


Figure 12-6 Debug Messages on Terminal Window for Connection to LAN and Internet

12.7 Summary

In this experiment, we have successfully configured the CC3100 device as a WLAN station and connected to Internet.

12.8 Exercise

1. In the terminal output window, we 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. In line no:62 of main. C replace **www.ti.com** with any non existing web address and repeat the experiment and observe what happens

Line 62: `#define` HOST_NAME **www.ti.com**

3. In line no:62 of main.C, replace it again with **www.ti.com** and repeat the experiment.

Experiment 13

Setting Up CC3100 as a HTTP Server

Topics **Page**

13.1 Objective

The goal of this experiment is to configure the CC3100 Booster pack as HTTP web server. We will also understand the HTTP web server concepts on embedded devices.

13.2 Introduction

HTTP is an acronym for Hyper Text Transfer Protocol. HTTP is a client/server protocol used to deliver hypertext resources (HTML web pages, images, query results, and so forth) to the client side. HTTP works on top of a predefined TCP/IP. (Transmission Control Protocol / Internet Protocol). HTTP web server allows end-users to remotely communicate with the CC3100 by using a standard web browser.

The HTTP web server enables the following functions:

- Device configuration
- Device status and diagnostic
- Application-specific functionality

The Simple link HTTP web server is an embedded network application running on the Network processor, which runs on top of the TCP stack as shown in **Figure 13-1**. The Simple link HTTP server handles the HTTP request by listening on the HTTP socket ID which is by default 80. Based on the request type, such as HTTP GET or HTTP POST, the server handles the request URI (Uniform Resource Identifier) resource and content. The server then composes the appropriate HTTP response and returns it to the client. The server communicates with the serial flash file system, which hosts the web page files. The files are saved in the serial flash with their individual filenames.

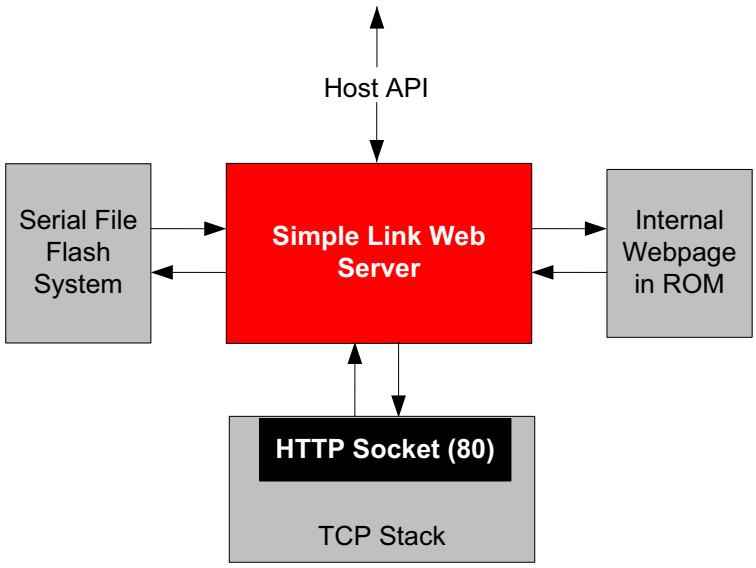


Figure 13-1 Block Diagram for Simple Link HTTP Server

This experiment configures the CC3100 in Access Point (AP) mode with a pre-defined SSIDNAME and uses the sample HTML pages stored in Flash which can be accessed by the clients. Clients can connect to CC3100 and request for web-pages using the IP of device from any stan-

dard web browser. The functional block diagram of the experiment as shown in **Figure 13-2** illustrates the working principle of the experiment.

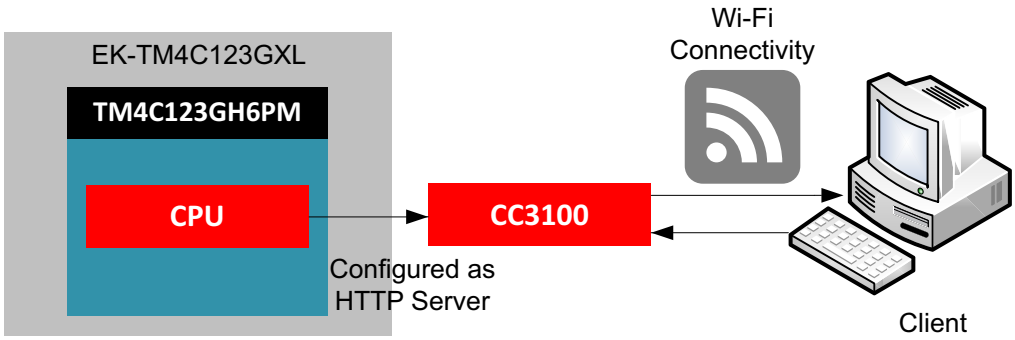


Figure 13-2 Functional Block Diagram

There are pre-programmed html pages already residing on the flash and new HTML pages can be downloaded on serial-flash of CC3100 using CCS_UniFlash utility using a separate tool EMUBOOST.

The scope of this experiment will be to use the existing html pages already pre-programmed in the flash by default.

13.3 Component Requirements

13.3.1 Software Requirement

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

13.3.2 Hardware Requirement

Table 13-1: Components Required for the Experiment

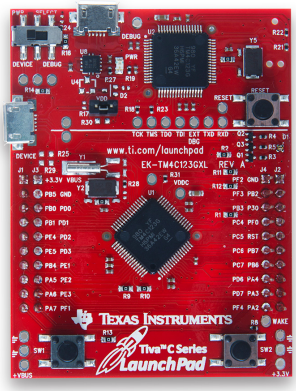
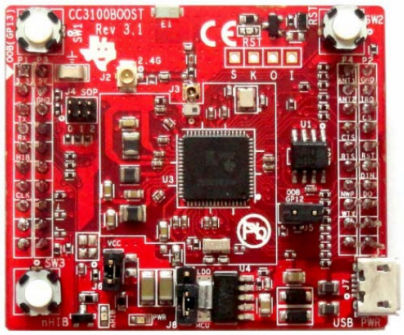
S.No	Components	Specification	Image
1.	Tiva LaunchPad	EK-TM4C123GXL LaunchPad	
2.	USB Cable		

Table 13-1: Components Required for the Experiment

S.No	Components	Specification	Image
3.	CC3100 Booster pack	CC3100BOOST	
4.	WiFi connectivity on PC (or any Mobile device with WiFi Connectivity)	To act as Client	

13.4 Software

The software is programmed into the target device TM4C123GH6PM on the EK-TM4C123GXL using the USB interface. The TM4C123GH6PM controls the operation of CC3100BOOST as a HTTP server.

13.4.1 Flowchart

The flowchart for the experiment is shown in **Figure 13-3**. The CC3100 BoosterPack is powered up in the default state. The CC3100 is started as an access point and security parameters are configured. The device is restarted as a HTTP server. The device waits for a client connection. On connection with a client, the authentication parameters, domain name and device URN are requested and connected after validation.

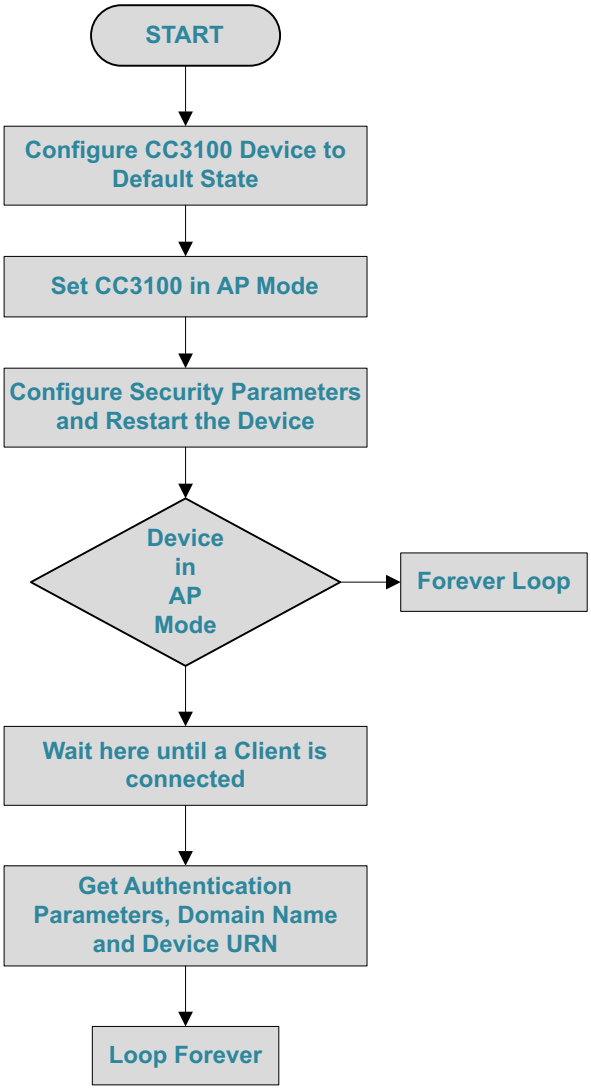


Figure 13-3 Flowchart for Setting up CC3100 as a HTTP Server

13.5 Procedure

13.5.1 Hardware Setup

Connect EK-TM4C123GXL and CC3100BOOST as shown in **Figure 13-4**.

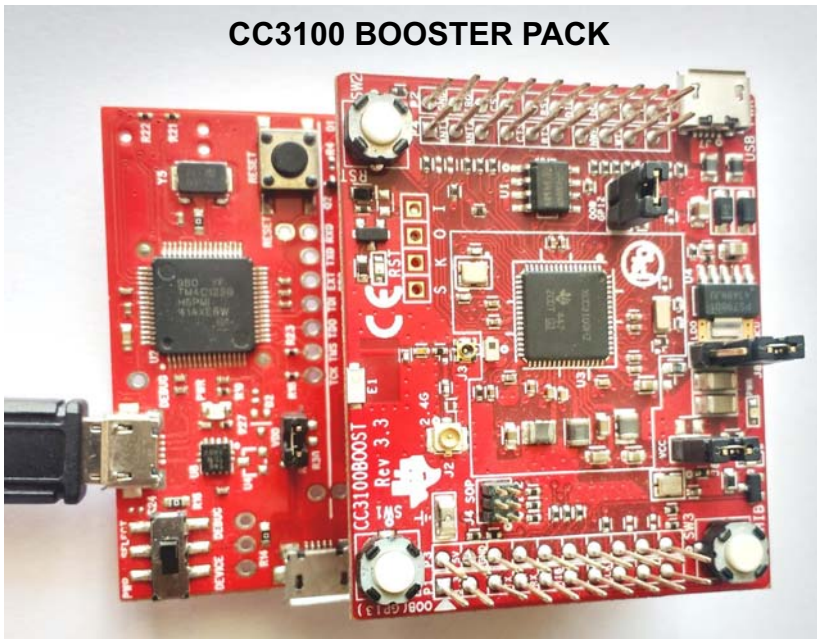


Figure 13-4 Hardware Setup

13.5.2 Steps for Creating, Building and Debugging 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 "getting started with wlan station" project from CC3100 SDK using the following steps:

- Choose the Import Project link on the TI Resource Explorer page
- Select the Browse button in the Import CCS Eclipse Projects dialog
- Select the **getting_started_wih_wlan_ ap** directory within

C:\TI\CC3100SDK_1.1.0\cc3100-sdk\platform\tiva-c-launchad\example_project_ccs\getting_started_wih_wlan_ap

4. Create a copy of the project in the same workspace and rename the copy as SetupServer. To copy the project, just right click on the project name, click copy, then right click on the project explorer window and paste it.

5. Go to **C:\TI\CC3100SDK_1.1.0\cc3100-sdk\examples\http_server**. Open the main.c, copy the code and paste it in the main.c of **SetupServer** project just created.

6. Comment the **source code from line 204 to 296** in main.c using /* */ block comment to adapt to the Tiva hardware as shown below.

```
CLI_Write(" [HTTP EVENT] NULL Pointer Error \n\r");
return;
    }
Line 204: /*  switch (pEvent->Event)
    {
case SL_NETAPP_HTTPGETTOKENVALUE_EVENT:
        {
            _u8 status = 0;
            _u8 *ptr = 0;
.....
        }
break;
default:
break;
Line 296:    }*/
}
After this comment Line 404 of main.c
    /* Stop WDT and initialize the system-clock of the MCU */
stopWDT();
initClk();
Line 404: //    initLEDs();
    /* Configure command line interface */
CLI_Configure();
displayBanner();
```

7. Double click on the project name go to spi and open spi.c

- Comment line 76

```
ROM_GPIOPinWrite(GPIO_PORTE_BASE,GPIO_PIN_0, PIN_HIGH);
/* Enable pull up on PB1, CC3100 UART RX */
Line 76: //GPIOPadConfigSet(GPIO_PORTB_BASE, GPIO_PIN_1, GPI-
O_STRENGTH_4MA, GPIO_PIN_TYPE_STD_WPU);
SysCtlPeripheralEnable(SYSCTL_PERIPH_SSI2);
```

8. Open sl_common.h and change SSID_AP_MODE to ACCESS_POINT_NAME.

To open this file, click sl_common.h (line 49 of main.c) while holding the Ctrl key OR right click on sl_common.h and select **Open declaration** on the pop up menu.

```
#include "simplelink.h"
#include "sl_common.h" << Line 49
```

Note: Ensure no other CC3100 on your circle have the same SSID_AP_MODE.

9. Open the project property by right clicking on project name and selecting properties.

Update TIVAWARE_ROOT variable available under **ResourceLinked** → **Resources** with Tivaware root directory (See **Figure 13-5**)

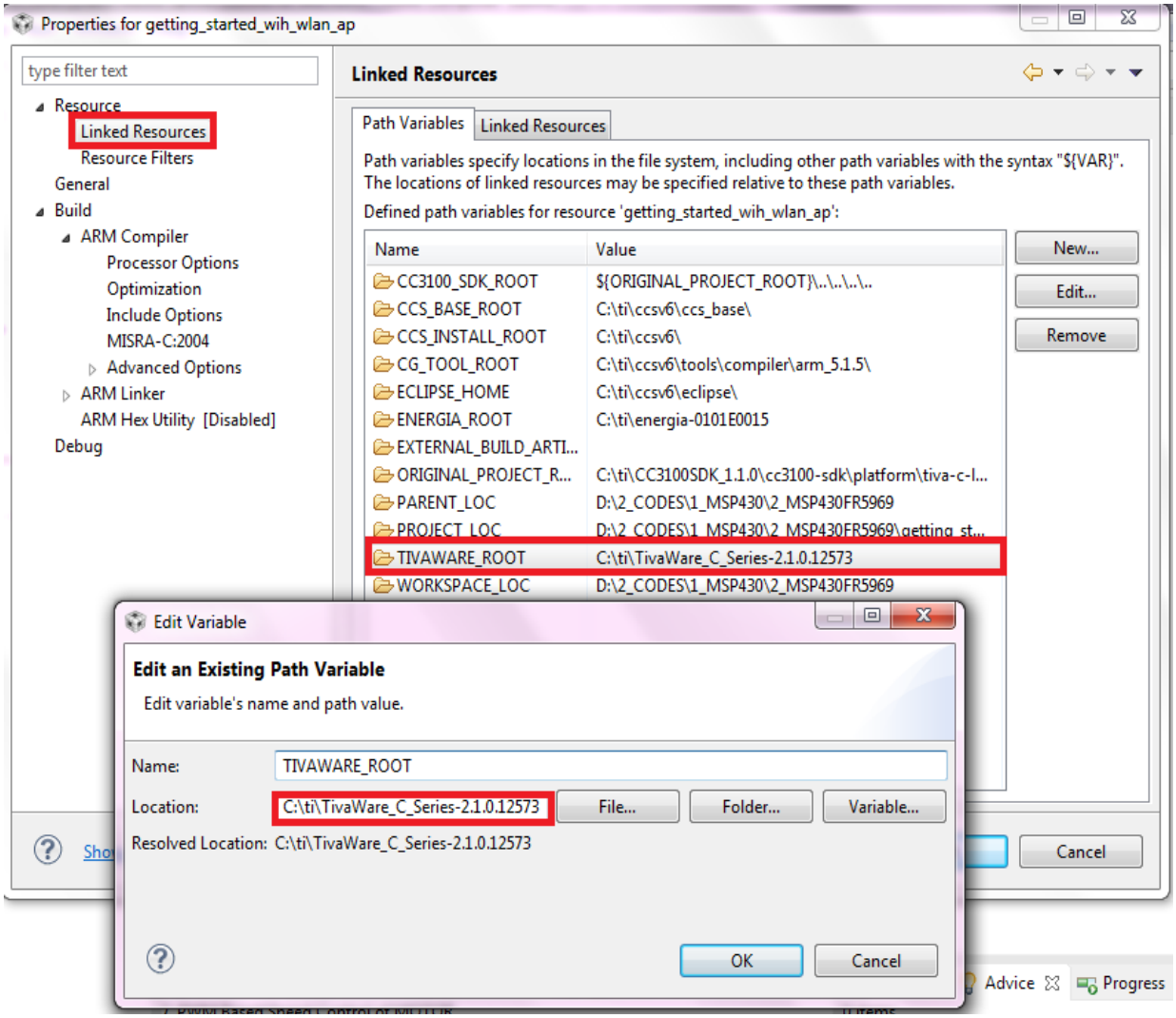


Figure 13-5 Updating TIVAWARE_ROOT Variable in Project Properties

10. Save, Debug and Run.

13.6 Observation

1. Open Tera Term Terminal Software on the PC where the EK-TM4C123GXL is connected. The serial window outputs the debug messages received from the Tiva Hardware. The serial port parameters to be set are 115200 baud rate, 8 data bits, No parity and 1 stop bit.

2. The Client Device could be a WiFi enabled PC or Smartphone. Ensure that the WiFi is enabled on the Client Device. This could be the same PC which is used for debugging/programming or a different PC.

3. Check and update your IPV4 settings on the Client your **PC** → **Control Panel** → **Network** and **Internet Settings** and ensure it is set to "Obtain an IP address automatically & Obtain DNS server address automatically."

4. On successful connection the CC3100 sever name will be listed as <**ap_mode_ssid**>in the available Connections on the Client Device.

5. If the connection is successful, the authentication parameters and domain name will be displayed in the serial terminal window as shown in **Figure 13-6**. Please note down the Name and Password which are **admin**.

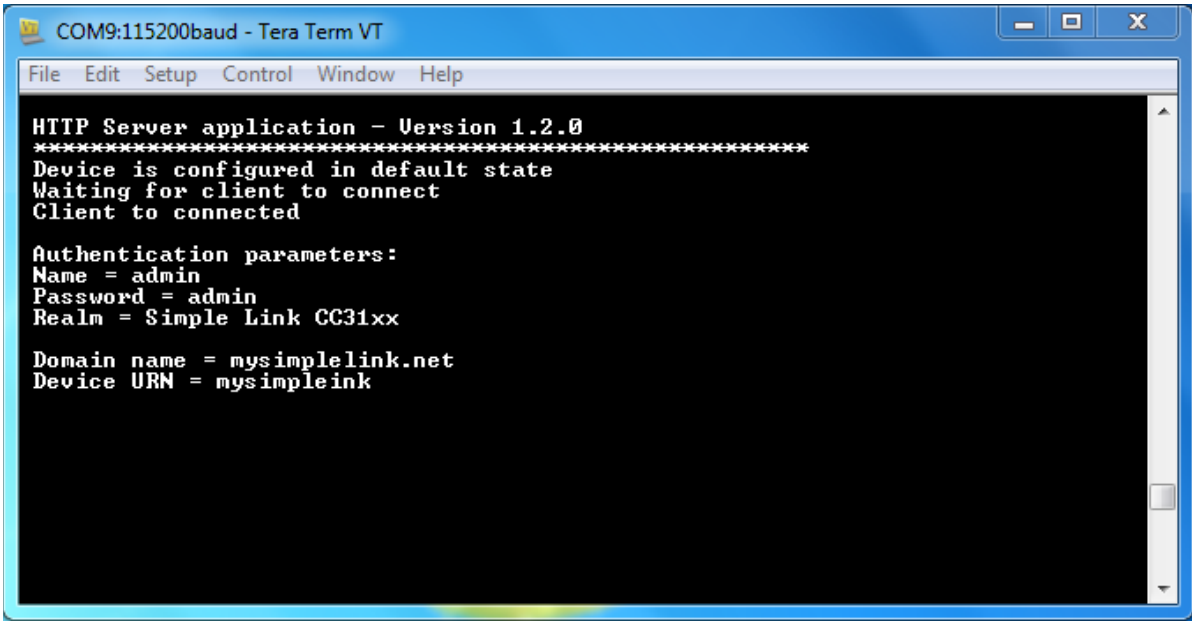


Figure 13-6 Terminal Window on Successful Connection with Client

Note: In the serial terminal window if the last line Device URN = **mysimplelink** does not appear, then comment the source code lines 539 to 545 in the file main.c as shown below.

```
//    Line 539: retVal = get_device_urn(g_device_urn);
//    if(retVal< 0)
//    LOOP_FOREVER();
//
//    CLI_Write((_u8 *)"\r\n Device URN = ");
//    CLI_Write(g_device_urn);
//    Line 545: CLI_Write((_u8 *)"\r\n");
Save, debug, run and try connecting WiFI again.
```

Open any Web browser in the client device and enter the IP address of the server (192.168.1.1 which is the default) or mysimplelink.net to access the webpage.

Enter the user name as **admin** and password as **admin**. This user name and password is also displayed in the serial terminal window. The web page is open and displayed as shown in Figure 13-7 after the authentication.

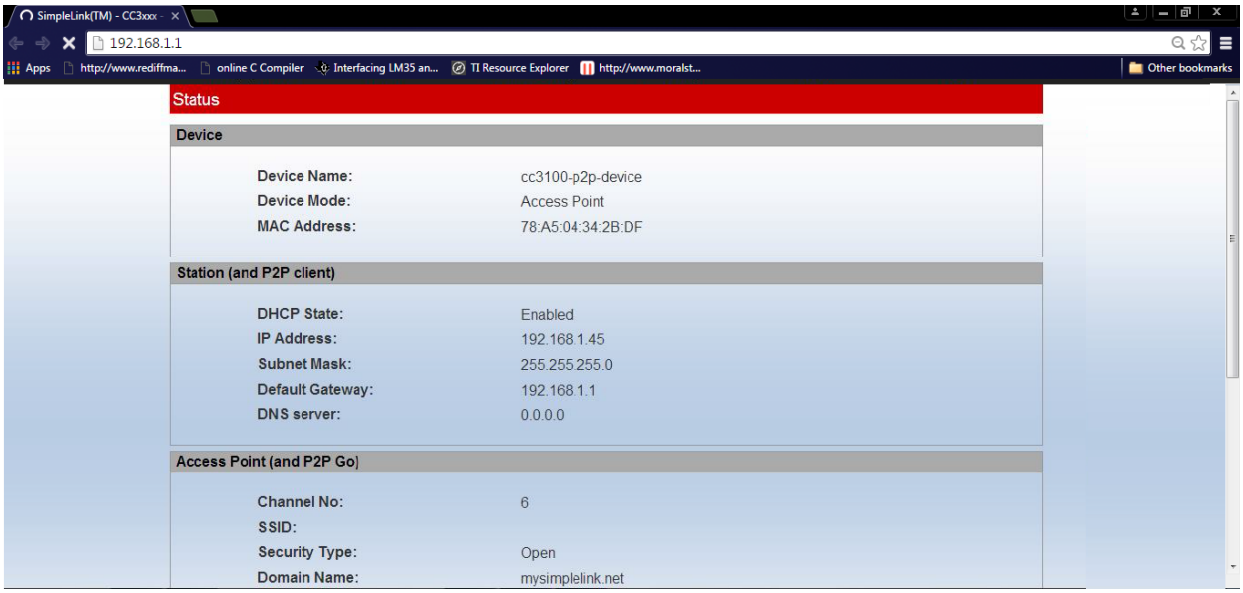


Figure 13-7 Webpage Opened from Server

13.7 Summary

In this experiment, we have successfully configured the CC3100 device as a HTTP Server and accessed the default webpage from the client PC.

13.8 Exercise

1. Where are the web pages stored in the CC3100?

2. What happens if we try to access a webpage which is not there inside the CC3100?

3. List 3 applications with a 3- to 4-line brief description that you think can be performed with this experimental setup.

Glossary

ADC	Analog-to-Digital Conversion
AP	Access Point
BIOS	Basic Input/Output System
CAN	Controller Area Network
CCS	Code Composer Studio
DAC	Digital-to-Analog Conversion
DCM	Direct Cosine Matrix
DHCP	Dynamic Host Configuration Protocol
DMA	Direct Access Memory
DMIPS	Dhrystone Million Instructions per Second
EEPROM	Electrically Erasable Programmable Read Only Memory
ETM	Embedded Trace Macrocell
FIFO	First In First Out
FPB	Flash Patch and Breakpoint Unit
GPIO	General Purpose Input Output
GPT	General Purpose Timer
HNP	Host Negotiation Protocol
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transfer Protocol
I^2C	Inter-Integrated Circuit
ICDI	In-Circuit Debug Interface
ISR	Interrupt Service Routine
ITM	Instrumentation Trace Macrocell
JTAG	Joint Test Actions Group
LAN	Local Area Network
MCU	Micro Controller Unit
MEMS	Micromechanical System
MPU	Memory Protection Unit
MSPS	Million Samples Per Second
NMI	Non-maskable interrupt
NVIC	Nested Vectored Interrupt controller
OTG	On the Go
PC	Personal Computer
PLL	Phase Locked Loop

PWM	Pulse Width Modulation
RTC	Real-Time Clock
SCL	Serial Clock Line
SDL	Serial Data Line
SPI	Serial Port Interface
SRAM	Static Random Access Memory
SRP	Session Request Protocol
SSI	Synchronous Serial Interface
SWD	Serial Wire Debug
TCP/IP	Transmission Control Protocol / Internet Protocol
TPIU	Trace Port Interface Unit
UART	Universal Asynchronous Receiver / Transmitter
ULP Advisor	Ultra-Low-Power Advisor
URI	Uniform Resource Identifier
USB	Universal Serial Bus
WiFi	Wireless Fidelity
WLAN	Wireless Local Area Network
µDMA	Micro Direct Memory Access Controller

IMPORTANT NOTICE

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

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

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

TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI components or services are used. Information published by TI regarding third-party products or services does not constitute a license to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

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

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

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

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

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

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

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

Products		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		

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