1 Introduction

The goal of this assignment is to familiarize yourself with the EK-TM4C129EXL development board and cross compiler tools.

If you need help during the lab-session you should write your name on the whiteboard (first to write should write on top of the board) so that the lab assistant can help you in first-come-first- served order.

Chapter 2-5 thoroughly describes the hardware functionality of the TiVa microcontroller and is not essential to understand to complete this lab. The sections are intended for students with genuine interest of the board functionality.

To start coding lab 1, jump directly to chapter 6.

2 Introduction to Tiva™ TM4C129ENCPDT Microcontroller

The TM4C129ENCPDT microcontroller is designed around an ARM Cortex-M processor core. The ARM Cortex-M processor provides the core for a high-performance, low-cost platform that meets the needs of minimal memory implementation, reduced pin count, and low power consumption, while delivering outstanding computational performance and exceptional system response to interrupts. Figure 1 shows a high-level block diagram of the TM4C129ENCPDT microcontroller.

2.1 Relevant documents

Having these documents handy will help you through the lab (in the folder *ek-tm4c129exl_datasheets* or at http://www.ti.com/tool/EK-TM4C129EXL#):

- Code Composer Studio IDE (file no. 1).
- TM4C129ENCPDT Microcontroller Datasheet (file no. 2).
- EK-TM4C129EXL Evaluation Board Overview (file no. 3).
- EK-TM4C129EXL Evaluation Board User Guide (file no. 4).
- TivaWare™ Peripheral Driver Library User Guide (file no. 5).
- Definitions of constants can be found in the files *Board.h* and *EK_TM4C129EXL.h* (in the folder *C:\ti\ccsv8\eclipse\plugins\com.ti.rtsc.TIRTOStivac.product.ui_2.16.0.08\resources\ti\boards\EK_TM4C129EXL*).

2.2 Abbreviations used

Dev. board = Development board
GPIO = General Purpose Input/Output
IDE = Integrated development environment
JTAG = Joint Test Action Group, an on-chip debug interface
LCD = Liquid crystal display
LED = Light emitting diode
MCU = Microcontroller unit
PWM = Pulse Width Modulation
SPI = Serial peripheral interface
TP = Twisted pair
TWI = Two wire interface
USART = Universal Synchronous Asynchronous Receiver Transmitter

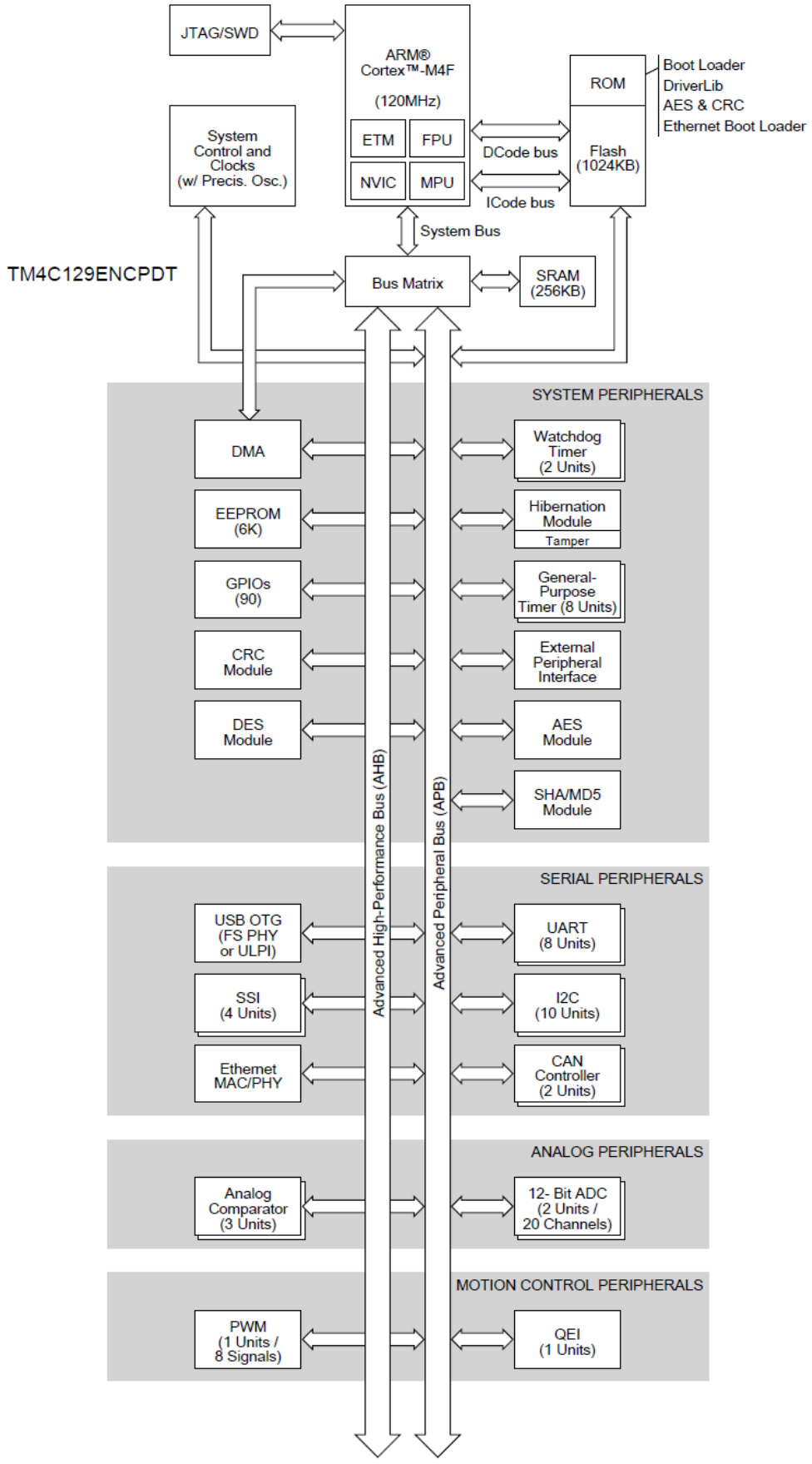


Figure 1: Tiva™ TM4C129ENCPDT Microcontroller High-Level Block Diagram

2.3 Clarification of the "Program" confusion

The term "Program" has several meanings. When you write source code in the editor, you are said to "program" or "write the program". When you write a binary into the dev. board MCU, you are said to "program" the flash, or "write the program (to the flash)". This can be confusing but is nothing to worry about.

2.4 Required equipment

This equipment should be available in the case you receive from the lab instructor. If anything is missing when you return the case you will not pass the course.

- EK-TM4C129EXL LaunchPad Evaluation Board (Dev. board)
- USB cable
- Ethernet cable
- Booster kit

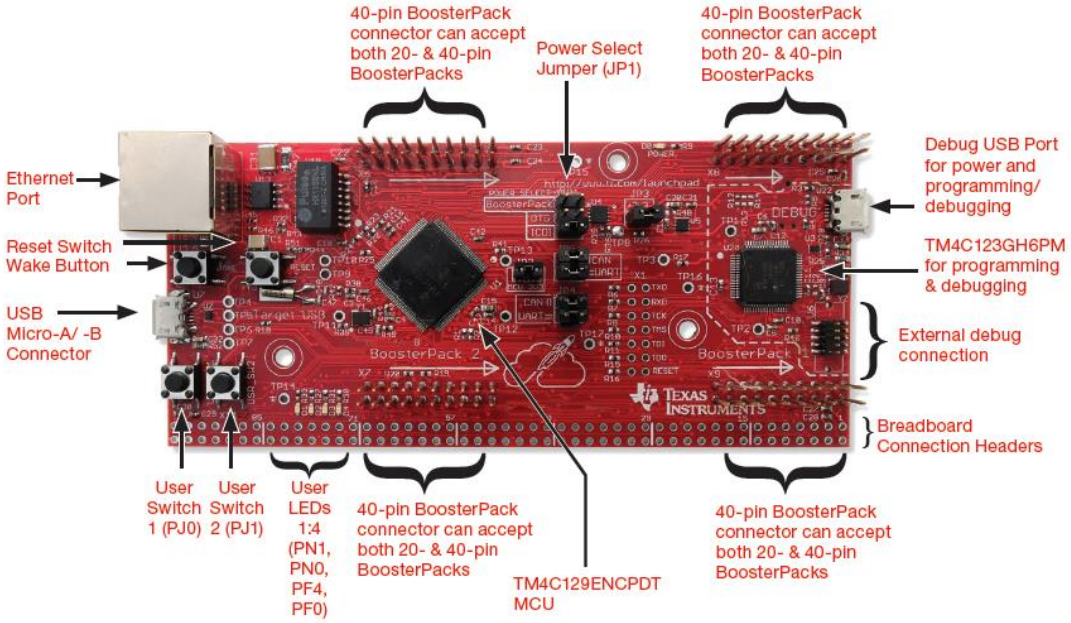


Figure 2: The EK-TM4C129EXL Launchpad Evaluation Board

3 Practice Assignments

3.1 Practice Assignment 1: Familiarizing with the Hardware

***Content**: Familiarize with the hardware. Identify I/O connectors.*

The TM4C Series TM4C129E Crypto Connected LaunchPad™ Evaluation Board (EK-TM4C129EXL) is a low-cost evaluation platform for ARM® Cortex®-M4F-based microcontrollers. The Crypto Connected LaunchPad design highlights the TM4C129ENCPDT microcontroller with its on-chip crypto acceleration hardware, 10/100 Ethernet MAC and PHY, USB 2.0, hibernation module, motion control pulse-width modulation and a multitude of simultaneous serial connectivity. The Crypto Connected LaunchPad also features two user switches, four user LEDs, dedicated reset and wake switches, a breadboard expansion option and two independent BoosterPack XL expansion connectors. The pre-programmed out of the

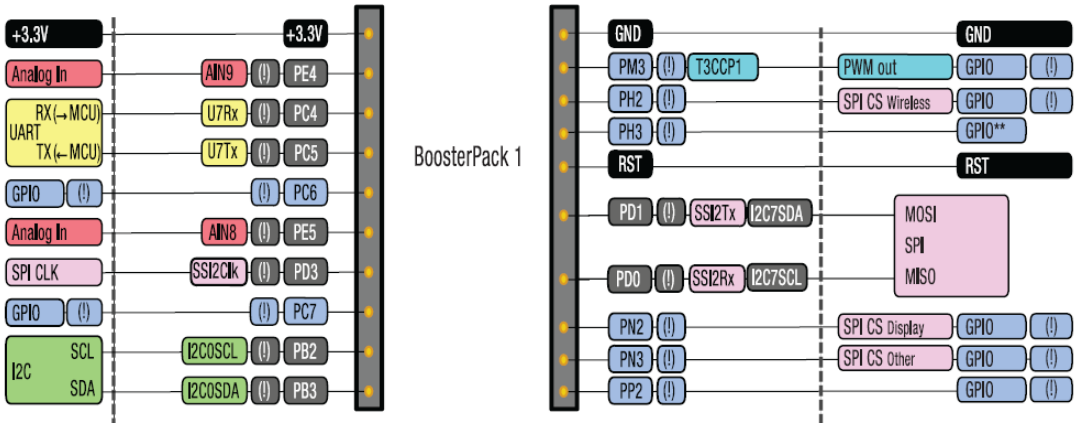
box demo on the Crypto Connected LaunchPad also enables remote monitoring and control of the evaluation board securely from an internet browser anywhere in the world. The web interface is provided by 3rd party, Exosite. Each Crypto Connected LaunchPad is enabled on the Exosite platform allowing users to create and customize their own secure Internet-of-Things (IoT) applications.

3.1.1 The EK-TM4C129EXL Evaluation Board Features

The Crypto Connected LaunchPad includes the following features (see Figure 2):

- TM4C129ENCPDT microcontroller.
- Ethernet connectivity with fully integrated 10/100 Ethernet MAC and PHY motion control pulse width modulation (PWM).
- Crypto acceleration hardware blocks.
- USB 2.0 Micro A/B connector.
- Four user LEDs.
- Two user buttons.
- One independent hibernates wake switch.
- One independent microcontroller reset switch.
- Jumper for selecting power source:
  - ICDI USB.
  - USB Device.
  - BoosterPack.
- Preloaded secure access of Internet-of-Things product to Exosite application.
- I/O brought to board edge for breadboard expansion.
- Two independent BoosterPack XL standard connectors featuring stackable headers to maximize expansion through BoosterPack ecosystem.

3.1.2 The EK-TM4C129EXL Evaluation Board Pinout



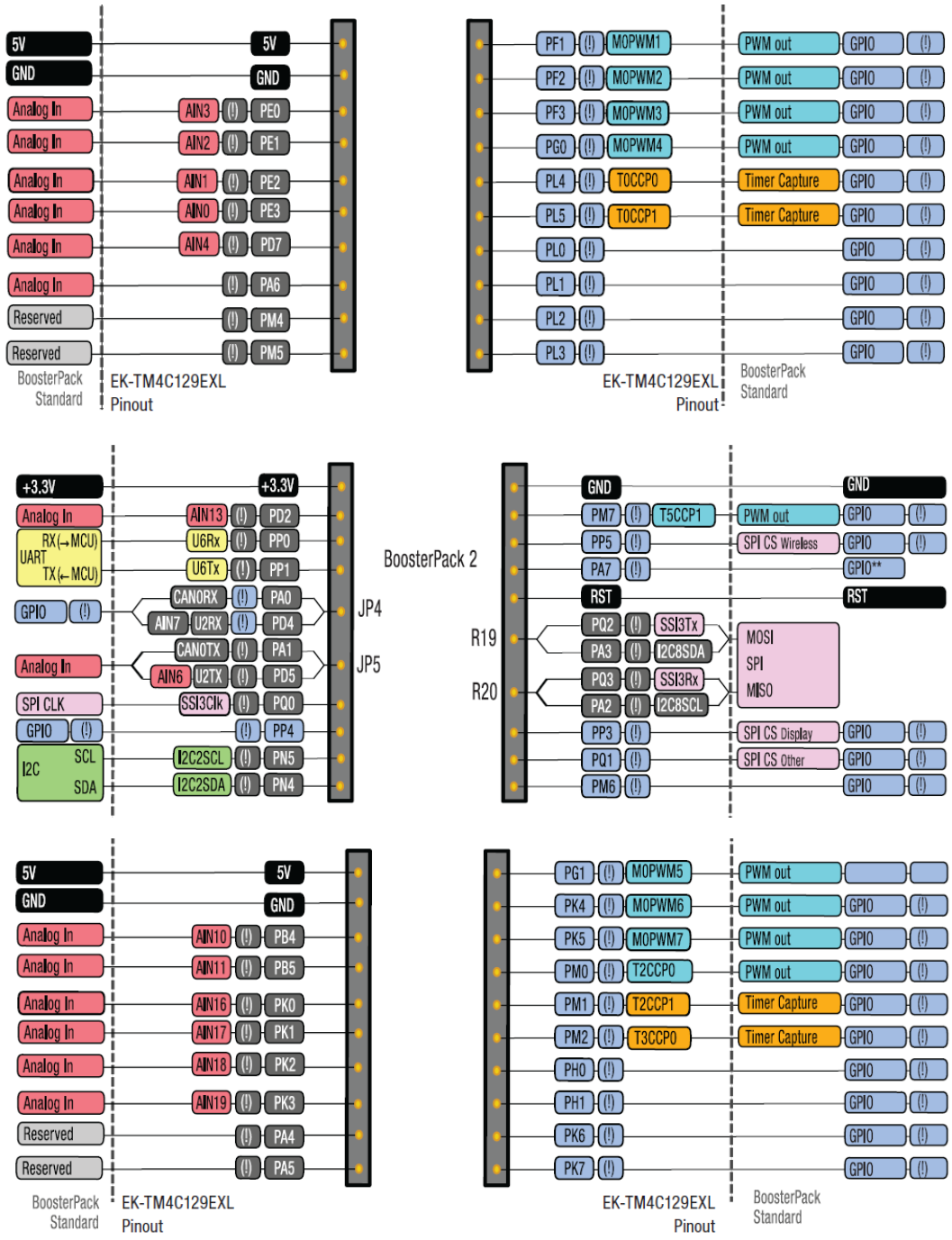


Figure 3: The EK-TM4C129EXL Evaluation Board Pinout

3.2 Practice Assignment 2: Prepare the Working Environment and Test it

***Content**: Prepare working environment terminal toolchain environment IDE and test it.*

The TivaWare software provides drivers for all of the peripheral devices supplied in the design. The TM4C Series Peripheral Driver Library is used to operate the on-chip peripherals as part of TivaWare. TivaWare includes a set of example applications that use the TivaWare Peripheral Driver Library. These applications demonstrate the capabilities of the TM4C129ENCPDT microcontroller, as well as provide a starting point for the development of the final application for use on the Crypto Connected LaunchPad evaluation board. Example

applications provided for the TM4C Series TM4C129EXL Connected LaunchPad and examples paired with selected BoosterPacks will work with the Crypto Connected LaunchPad.

3.2.1 Source Code

The complete source code including the source code installation instructions are provided at http://www.ti.com/tool/sw-tm4c. (**important to download and install for the lab**) The source code and binary files are installed in the TivaWare software tree.

3.2.2 Tool Options

The source code installation includes directories containing projects, makefiles, and binaries for the following tool-chains:

- Keil ARM RealView® Microcontroller Development System.
- IAR Embedded Workbench® for ARM.
- Generic GNU C Compiler.
- Texas Instruments' Code Composer Studio™ IDE.

3.2.3 Programming the Crypto Connected LaunchPad

The TivaWare software package includes pre-built binaries for each of the example applications. If you installed the TivaWare software to the default installation path of C:\ti\TivaWare_C_Series_<version>, you can find the example applications in C:\ti\TivaWare_C_Series-<version>\examples\boards\ek-tm4c129exl. The on-board ICDI is used with the LM Flash Programmer tool to program applications on the Crypto Connected LaunchPad.

Follow these steps to program example applications into the Crypto Connected LaunchPad evaluation board using the ICDI:

1. Install LM Flash Programmer on a PC running Microsoft Windows.
2. Place JP1 into the ICDI position on the Crypto Connected LaunchPad.
3. Connect the USB-A cable plug in to an available USB port on the PC and plug the Micro-B plug to the Debug USB port (U22) on the Crypto Connected LaunchPad.
4. Verify that LED D0 at the top of the board is illuminated.
5. Install Windows ICDI and Virtual COM Port drivers if prompted. Installation instructions can be found in the Stellaris® In-Circuit Debug Interface (ICDI) and Virtual COM Port Driver Installation Instructions (SPMU287).
6. Run the LM Flash Programmer application on the PC.
7. In the Configuration tap, use the Quick Set control to select “TM4C129EXL LaunchPad”.
8. Move to the Program tab and click the Browse button. Navigate to the example applications directory (the default location is C:\ti\TivaWare_C_Series_<version>\examples\boards\EK-TM4C129EXL\).
9. Each example application has its own directory. Navigate to the example directory that you want to load and then into the sub-directory for one of the supported tool chains that contains the binary (*.bin) file. Select the binary file and click Open.
10. Set the Erase Method to Erase Necessary Pages, check the Verify After Program box, and check Reset MCU After Program. The example program starts execution once the verify process is complete.

3.3 Practice Assignment 3: Interactive Development Environment (IDE)

***Content**: Find, familiarize and use cross compilation tools & IDE. Compile a "Blinky LED" project.*

3.3.1 Blinky LED

We will look at how to access and control I/O-pins of the MCU. We will investigate the I/O control by using a LED. It can be useful to signal the status to the user using a LED. There is more than having the LED on or off. A blinking LED might be the only way for the program to tell the user something happened in a simple system. Various blinking rates or intensities of the light can be options for communication with the user. Even a LED that doesn't blink or light up when it is supposed to can be informative.

3.3.2 About TM4C129ENCPDT Microcontroller GPIO

The GPIO module is composed of 15 physical GPIO blocks, each corresponding to an individual GPIO port (Port A, Port B, Port C, Port D, Port E, Port F, Port G, Port H, Port J, Port K, Port L, Port M, Port N, Port P, Port Q). The GPIO module supports up to 90 programmable input/output pins, depending on the peripherals being used. For more information about the microcontroller GPIO refer to "*Table 10-2. GPIO Pins and Alternate Functions"* on pages 751-754 in the document "*2 tm4c129encpdt_Microcontroller_Datasheet.pdf"*.

3.3.3 Hardware Registers

The data control registers allow software to configure the operational modes of the GPIOs. The data direction register configures the GPIO as an input or an output while the data register either captures incoming data or drives it out to the pads.

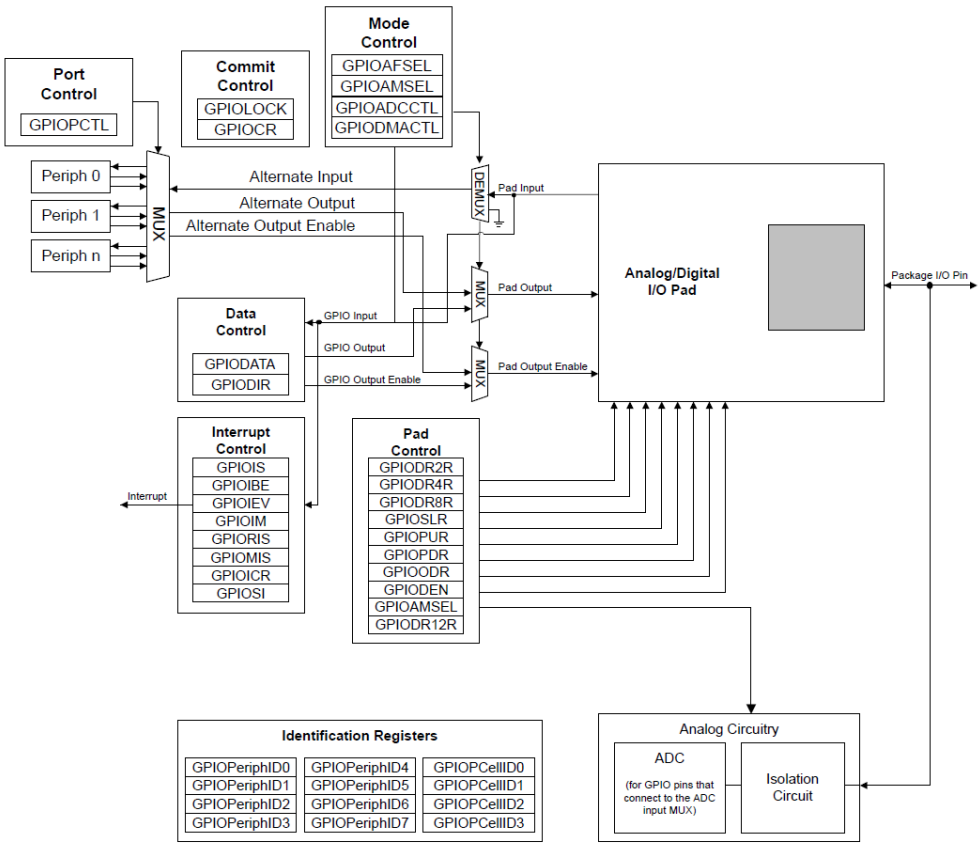


Figure 4: The EK-TM4C129EXL Analog/Digital I/O Pads

Some of those registers are explained in the following.

Data Direction Operation (GPIODIR): The GPIO Direction register is used to configure each individual pin as an input or output. When the data direction bit is cleared, the GPIO is configured as an input, and the corresponding data register bit captures and stores the value on the GPIO port. When the data direction bit is set, the GPIO is configured as an output, and the corresponding data register bit is driven out on the GPIO port.

Data Register Operation (GPIODATA): To aid in the efficiency of software, the GPIO ports allow for the modification of individual bits in the GPIO Data (GPIODATA) register by using bits [9:2] of the address bus as a mask. In this manner, software drivers can modify individual GPIO pins in a single instruction without affecting the state of the other pins. This method is more efficient than the conventional method of performing a read-modify-write operation to set or clear an individual GPIO pin. To implement this feature, the GPIODATA register covers 256 locations in the memory map. During a write, if the address bit associated with that data bit is set, the value of the GPIODATA register is altered. If the address bit is cleared, the data bit is left unchanged.

Interrupt Control Registers: The interrupt capabilities of each GPIO port are controlled by a set of seven registers. These registers are used to select the source of the interrupt, its polarity, and the edge properties. When one or more GPIO inputs cause an interrupt, a single interrupt output is sent to the interrupt controller for the entire GPIO port. For edge-triggered interrupts, software must clear the interrupt to enable any further interrupts. For a level-sensitive interrupt, the external source must hold the level constant for the interrupt to be recognized by the controller.

3.3.4 Initialization and Configuration a GPIO

To configure the GPIO pins of a particular port, follow these steps:

1. Enable the clock to the port by setting the appropriate bits in the RCGCGPIO register. In addition, the SCGCGPIO and DCGCGPIO registers can be programmed in the same manner to enable clocking in Sleep and Deep-Sleep modes.
2. Set the direction of the GPIO port pins by programming the GPIODIR register. A write of a 1 indicates output and a write of a 0 indicates input.
3. Configure the GPIOAFSEL register to program each bit as a GPIO or alternate pin. If an alternate pin is chosen for a bit, then the PMCx field must be programmed in the GPIOPCTL register for the specific peripheral required. There are also two registers, GPIOADCCTL and GPIODMACTL, which can be used to program a GPIO pin as a ADC or µDMA trigger, respectively.
4. Set the EDMn field in the GPIOPC register as shown in Table 1.
5. Set or clear the GPIODR4R register bits as shown in Table 1.
6. Set or clear the GPIODR8R register bits as shown in Table 1.
7. Set or clear the GPIODR12R register bits as shown in Table 1.
8. Program each pad in the port to have either pull-up, pull-down, or open drain functionality through the GPIOPUR, GPIOPDR, GPIOODR register. Slew rate may also be programmed, if needed, through the GPIOSLR register.
9. To enable GPIO pins as digital I/Os, set the appropriate DEN bit in the GPIODEN register. To enable GPIO pins to their analog function (if available), set the GPIOAMSEL bit in the GPIOAMSEL register.

10. Program the GPIOIS, GPIOIBE, GPIOEV, and GPIOIM registers to configure the type, event, and mask of the interrupts for each port.
    Note: To prevent false interrupts, the following steps should be taken when re-configuring GPIO edge and interrupt sense registers:
    a. Mask the corresponding port by clearing the IME field in the GPIOIM register.
    b. Configure the IS field in the GPIOIS register and the IBE field in the GPIOIBE register.
    c. Clear the GPIORIS register.
    d. Unmask the port by setting the IME field in the GPIOIM register.
11. Optionally, software can lock the configurations of the NMI and JTAG/SWD pins on the GPIO port pins, by setting the LOCK bits in the GPIOLOCK register.

Table 1: GPIO Drive Strength Options

EDE (GPIOPP)	EDMn (GPIOPC)	GPIODR12R (+4mA)	GPIODR8R (+4mA)	GPIODR4R (+2mA)	GPIODR2R (2mA)	Drive (mA)
X	0x0	N/A	0	0	1	2
			0	1	0	4
			1	0	0	8
1	0x1	N/A	0	0	N/A	2
			0	1	N/A	4
			1	0	N/A	6
			1	1	N/A	8
1	0x3	0	0	0	N/A	2
		0	0	1	N/A	4
		0	1	0	N/A	6
		0	1	1	N/A	8
		1	1	0	N/A	10
		1	1	1	N/A	12
		1	0	N/A	N/A	N/A
1	0x2	N/A	N/A	N/A	N/A	N/A

Table 2 shows some GPIO interrupt configuration example and Table 3 shows some GPIO pad configuration examples.

Table 2: GPIO Interrupt Configuration Example

Register	Desired Interrupt Event Trigger	Pin 2 Bit Value[a]							
		7	6	5	4	3	2	1	0
GPIOIS	0=edge 1=level	X	X	X	X	X	0	X	X
GPIOIBE	0=single edge 1=both edges	X	X	X	X	X	0	X	X
GPIOIEV	0=Low level, or falling edge 1=High level, or rising edge	X	X	X	X	X	1	X	X
GPIOIM	0=masked 1=not masked	0	0	0	0	0	1	0	0

a. X=Ignored (don't care bit)

Table 3: GPIO Pad Configuration Examples

Configuration	GPIO Register Bit Value[a]										
	AFSEL	DIR	ODR	DEN	PUR	PDR	DR2R	DR4R	DR8R	DR12R	SLR
Digital Input (GPIO)	0	0	0	1	?	?	X	X	X	X	X
Digital Output (GPIO)	0	1	0	1	?	?	?	?	?	?	?
Open Drain Output (GPIO)	0	1	1	1	X	X	?	?	?	?	?
Open Drain Input/Output (I2CSDA)	1	X	1	1	X	X	?	?	?	?	?
Digital Input/Output (I2CSCL)	1	X	0	1	X	X	?	?	?	?	?
Digital Input (Timer CCP)	1	X	0	1	?	?	X	X	X	X	X
Digital Input (QEI)	1	X	0	1	?	?	X	X	X	X	X
Digital Output (PWM)	1	X	0	1	?	?	?	?	?	?	?
Digital Output (Timer PWM)	1	X	0	1	?	?	?	?	?	?	?
Digital Input/Output (SSI)	1	X	0	1	?	?	?	?	?	?	?
Digital Input/Output (UART)	1	X	0	1	?	?	?	?	?	?	?
Analog Input (Comparator)	0	0	0	0	0	0	X	X	X	X	X
Digital Output (Comparator)	1	X	0	1	?	?	?	?	?	?	?

a. X=Ignored (don't care bit)

?=Can be either 0 or 1, depending on the configuration

3.3.5 Running the Blinky Project

We will walk through the project Blinky and we will examine the source code how the GPIO is set up and controlled. First open the *Code Composer Studio* and browse the Blinky project. Right click on the project's name and select *Build Project* from the menu.

Note: if you are unable to find the Blinky project as shown in Figure 5 in the *Code Composer Studio* installed in the Lab computers then follow the instructions in Section 6.

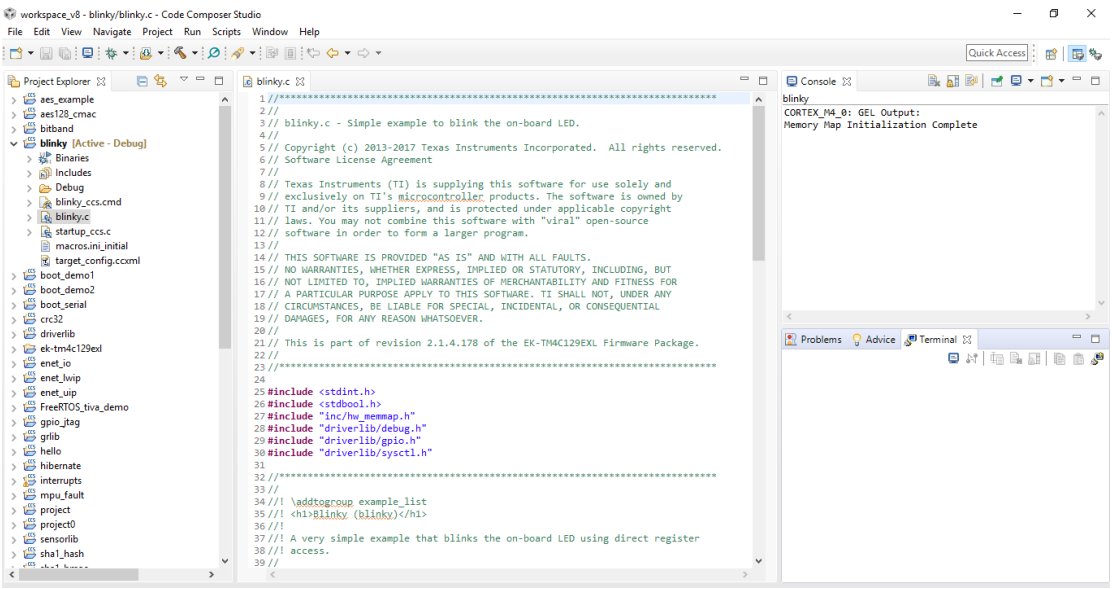


Figure 5: Blinky Project

Second connect the LaunchPad Evaluation Board to a USB port then select *Run* → *Load* → *Blinky*. The code is flashed into the board and a Led start blinking as shown on Figure 6.

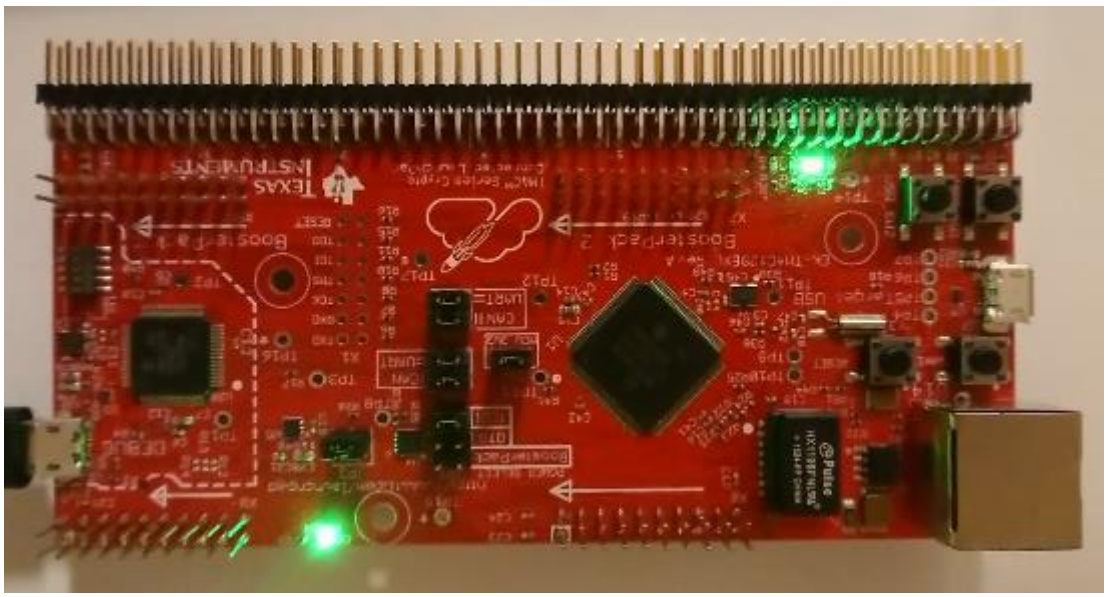


Figure 6: Blinky Project Running on the Board

Now double click on *Blinky.c* and the main program will be opened. It is simple and self-explanatory.

```
// Blink the on-board LED.
Int main(void) {
    volatile uint32_t ui32Loop;

    // Enable the GPIO port that is used for the on-board LED.
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPION);

    // Check if the peripheral access is enabled.
    while(!SysCtlPeripheralReady(SYSCTL_PERIPH_GPION)){
    }

    // Enable the GPIO pin for the LED (PN0).  Set the direction as output,
    //and enable the GPIO pin for digital function.
    GPIOPinTypeGPIOOutput(GPIO_PORTN_BASE, GPIO_PIN_0);

    // Loop forever.
    while(1){
        // Turn on the LED.
        GPIOPinWrite(GPIO_PORTN_BASE, GPIO_PIN_0, GPIO_PIN_0);

        // Delay for a bit.
        for(ui32Loop = 0; ui32Loop < 200000; ui32Loop++) {
        }

        // Turn off the LED.
        GPIOPinWrite(GPIO_PORTN_BASE, GPIO_PIN_0, 0x0);

        // Delay for a bit.
        for(ui32Loop = 0; ui32Loop < 200000; ui32Loop++){
        }
     } }
```

3.3.6 Debugging the Blinky Project

Content: Use the CCS Debugger to single step the "Blinky" program.

Debugging mode permits to the programmer to trace the content of MCU registers as well as the values of the defined variables. In order to run the debugging mode, click on *Run → Debug* and the window shown in Figure 6 will appear.

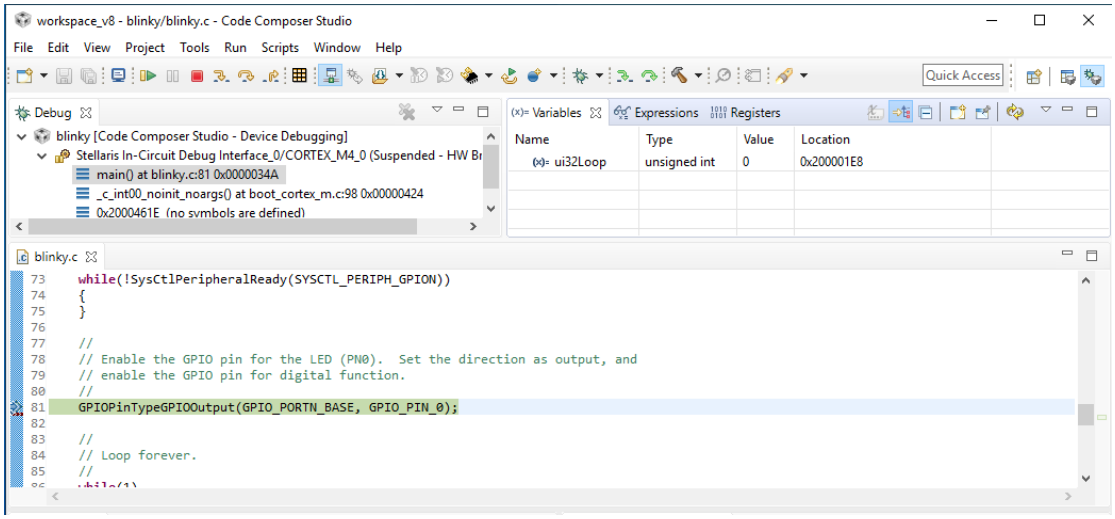


Figure 6: Blinky Project in Debugging Mode

Set a breakpoint at an instruction. This is done by double clicking the blue column (left edge of the source code), or right click the source code line and use the menu that appears. A breakpoint is indicated by a small square symbol.

You can start the debugging by clicking on *Resume* button (the yellow-green arrow in Figure 6) or pressing *F8* and code will be executed until the breaking point and you can see the new values of the variables.

To terminate the debugging session, click on the *Terminate* button (the red square in Figure 6) or press *Ctrl+F2*.

4 Pinout Module

***Content**: Familiarize with the pinout module for the EK-TM4C129EXL development board.*

The pinout module is a common function for configuring the device pins for use by example applications. The pins are configured into the most common usage; it is possible that some of the pins might need to be reconfigured in order to support more specialized usage. This driver is located in *examples/boards/ek-tm4c129exl/drivers*, with *pinout.c* containing the source code and *pinout.h* containing the API declarations for use by applications.

4.1 API Functions

4.1.1 void LEDRead (uint32_t *pui32LEDValue)

This function reads the state of the CLP LEDs and stores that state information into the variable pointed to by *pui32LEDValue*.

4.1.2 void LEDWrite (uint32_t ui32LEDMask, uint32_t ui32LEDValue)

This function writes a state to the LED bank. The first parameter acts as a mask. Only bits in the mask that are set will correspond to LEDs that may change. LEDs with a mask that is not set will not change. This works the same as *GPIOPinWrite*. After applying the mask, the setting for each unmasked LED is written to the corresponding LED port pin via *GPIOPinWrite*.

4.1.3 void PinoutSet (bool bEthernet, bool bUSB)

This function enables the GPIO modules and configures the device pins for the default, standard usages on the EK-TM4C129EXL. Applications that require alternate configurations of the device pins can either not call this function and take full responsibility for configuring all the device pins or can reconfigure the required device pins after calling this function.

Parameters:

bEthernet is a Boolean used to determine function of Ethernet pins. If true, Ethernet pins are configured as Ethernet LEDs. If false, GPIO are available for application use.

bUSB is a Boolean used to determine function of USB pins. If true, USB pins are configured for USB use. If false, then USB pins are available for application use as GPIO.

4.2 Programming Example

The following example shows how to configure the device pins.

```
//
// The pinout example.
//
void
PinoutExample(void)
{
    //
    // Configure the device pins.
    // First argument determines whether the Ethernet pins will be
    // configured in networking mode for this application.
    // Second argument determines whether the USB pins will be configured
    // for USB mode for this application.
    //
    PinoutSet(true, false);
}
```

5 Buttons Driver

***Content**: Familiarize with the buttons driver for the EK-TM4C129EXL development board.*

The buttons driver provides functions to make it easy to use the push buttons on this evaluation board. The driver provides a function to initialize all the hardware required for the buttons, and features for debouncing and querying the button state. This driver is located in *examples/boards/ek-tm4c129exl/drivers*, with *buttons.c* containing the source code and *buttons.h* containing the API declarations for use by applications.

5.1 API Functions

5.1.1 void ButtonsInit (void)

This function must be called during application initialization to configure the GPIO pins to which the pushbuttons are attached. It enables the port used by the buttons and configures each button GPIO as an input with a weak pull-up.

5.1.2 uint8_t ButtonsPoll (uint8_t *pui8Delta, uint8_t *pui8RawState)

This function should be called periodically by the application to poll the pushbuttons. It determines both the current debounced state of the buttons and which buttons have changed state since the last time the function was called. In order for button debouncing to work properly, this function should be called at a regular interval, even if the state of the buttons is not needed that often. If button debouncing is not required, the caller can pass a pointer for the *pui8RawState* parameter in order to get the raw state of the buttons. The value returned in *pui8RawState* will be a bit mask where a 1 indicates the buttons is pressed. *pui8Delta* points to a character that will be written to indicate which button states changed since the last time this function was called. This value is derived from the debounced state of the buttons.

5.2 Programming Example

The following example shows how to use the buttons driver to initialize the buttons, debounce and read the buttons state.

```
// Map Left button to the GPIO Pin 0 of the button port.

#define LEFT_BUTTON GPIO_PIN_0

// The button example
void
ButtonExample(void)
{
    unsigned char ucDelta, ucState;
    //
    // Initialize the buttons.
    //
    ButtonsInit();
    //
    // From timed processing loop (for example every 10 ms)
    //
    {
    //
    // Poll the buttons. When called periodically this function will
    // run the button debouncing algorithm.
        //
        ucState = ButtonsPoll(&ucDelta, 0);
        //
        // Test to see if the SELECT button was pressed and do something
        //
        if(BUTTON_PRESSED(LEFT_BUTTON, ucState, ucDelta))
        {
        //
        // TODO: SELECT button action code
        //
        }
    }
}
```

6 Creating a New Project

***Content**: Familiarize with the creation of a new Texas Instruments Tiva project from scratch in Code Composer Studio.*

In this section you will use the APIs explained in previous paragraphs to create a new project called "*Controlled*". The project will change the status of the led from "on" to "off" by pressing the button *USR_SW1* and the button *USR_SW2* respectively. Please note that when setting the path for the *TivaWare C Series,* your version may be different than the specified *TivaWare_C_Series-2.1.4.178.*

6.1.1 Create a New CCS Project

After starting CCS, click on the *File → New → CCS Project.* Write the name of the project and make sure that the Target is Tiva TM4C129ENCPDT then click on *Finish*.

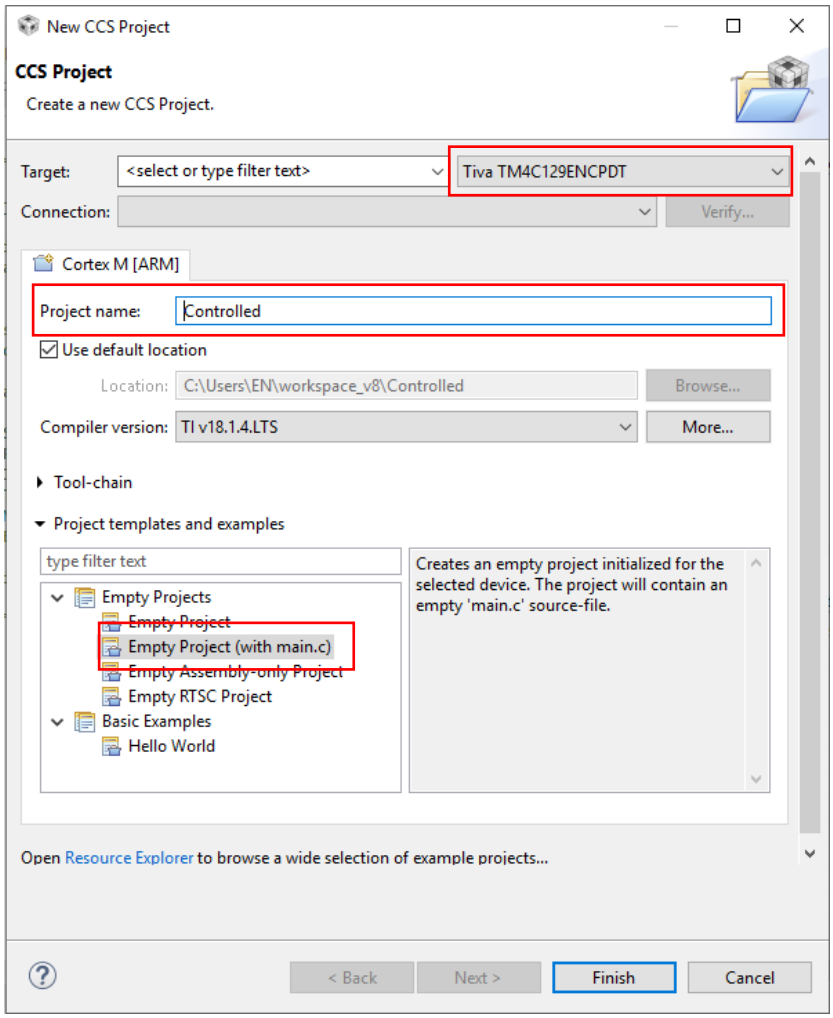


Figure 7: Creating a New Project

6.1.2 Configuring the Project

Right click on the project name and select *Properties*.

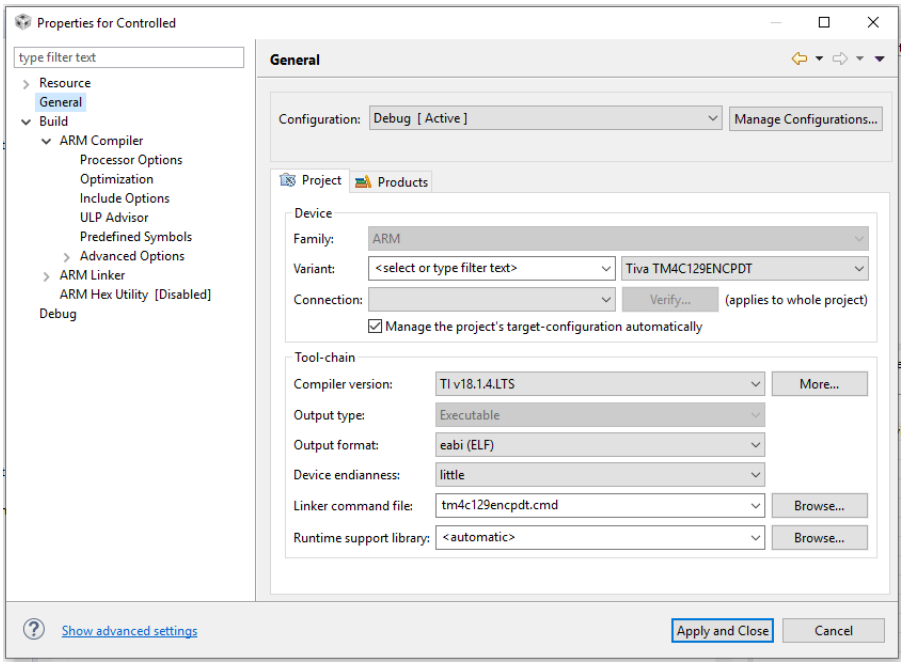


Figure 8: Configuration Window

- From *General* options select *Connection: Stellaris In-Circuit Debug Interface*.

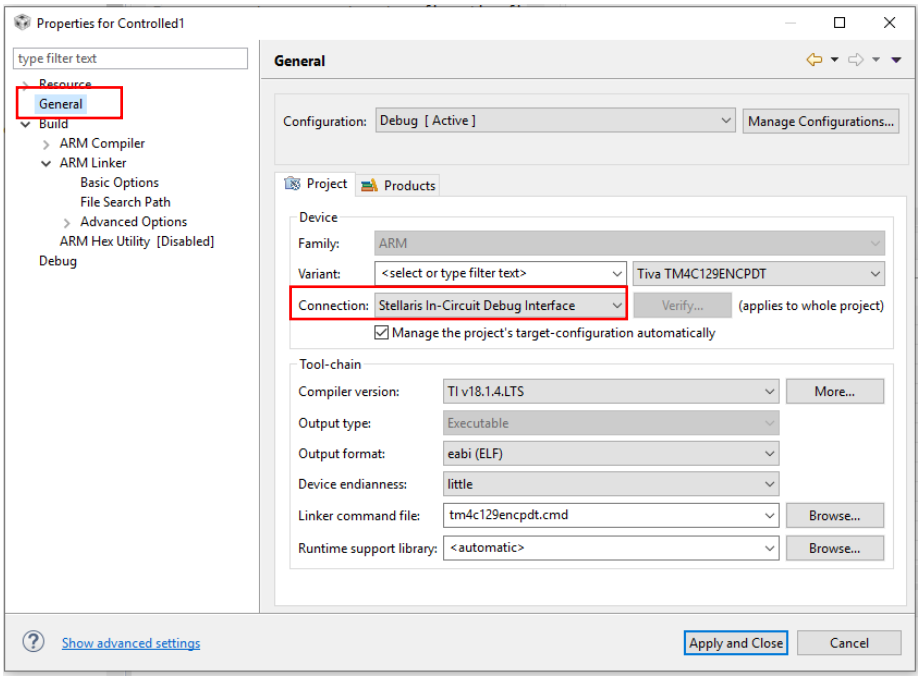


Figure 9: Configuring the Debugger

- Click on *Optimization* and select *2 – Global Optimizations*.

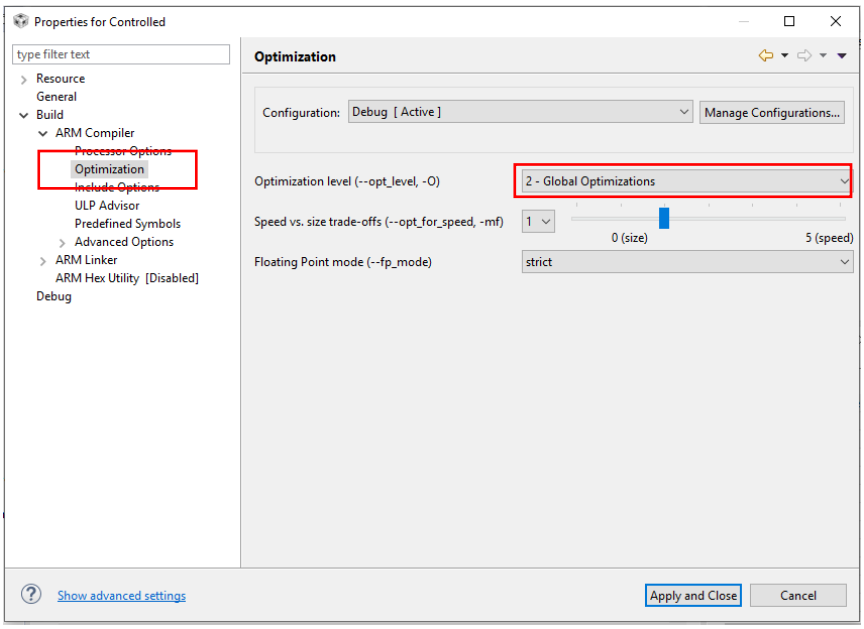


Figure 10: Configuring the Optimization Property

- Click on *Include Options* and click on *+ icon*. From the opened window, click on *Browse* and select *C:\ti\TivaWare_C_Series-2.1.4.178\examples\boards\ek-tm4c129exl*.
- Repeat previous step to add *C:\ti\TivaWare_C_Series-2.1.4.178*.

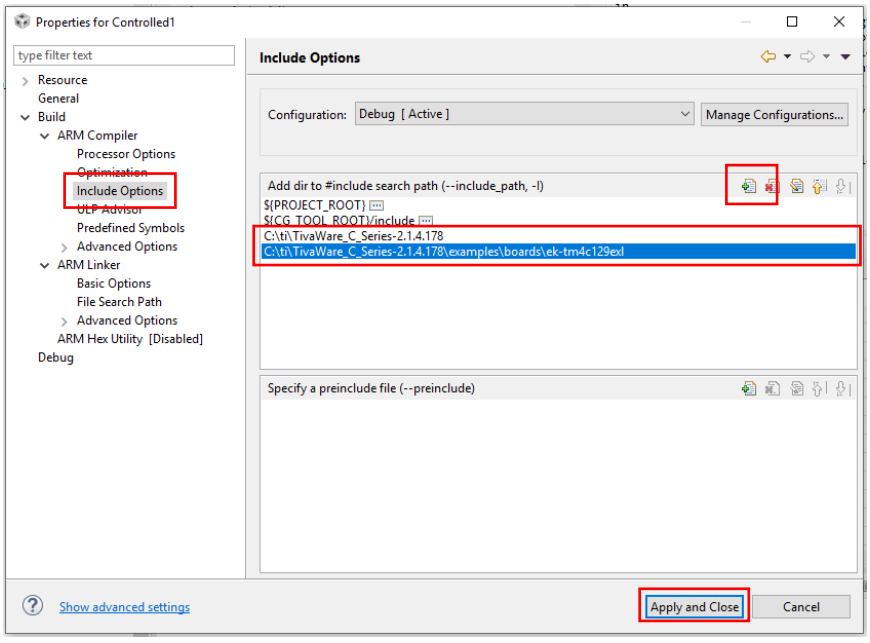


Figure 11: Configuring the Include Options

- Click on *Predefined Symbols* then click on *+ icon*. From the opened window, write *TARGET_IS_TM4C129_RA1* and click *ok*.

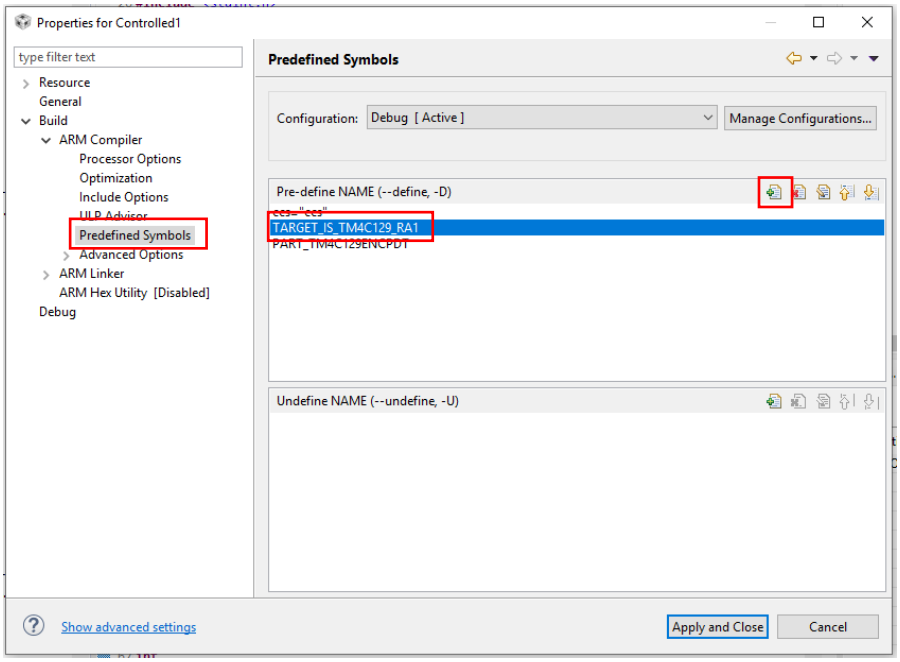


Figure 12: Configuring the ROM Functions

- Click on *ARM Linker → File Search* Path then click on *+ icon*. From the opened window, click on Browse and select *C:\ti\TivaWare_C_Series2.1.4.178\driverlib\ccs\Debug\driverlib.lib*.

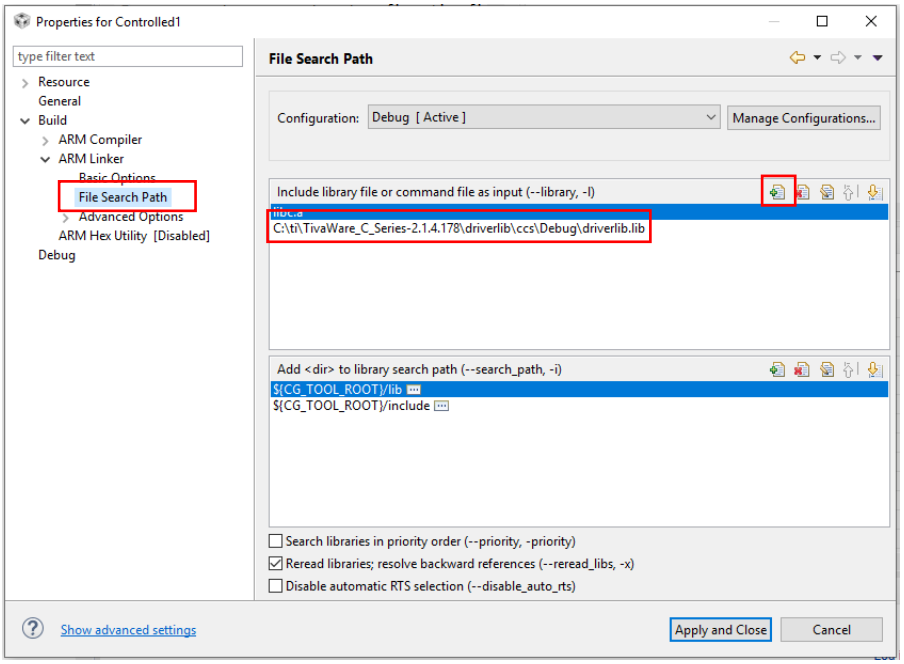


Figure 13: Configuring the Linker

- Finally click on *Apply and Close*.

6.1.3 Adding Drivers

- Right click on the project name and select *New → Folder*. Write *drivers* in Folder name and click on *Finish*.

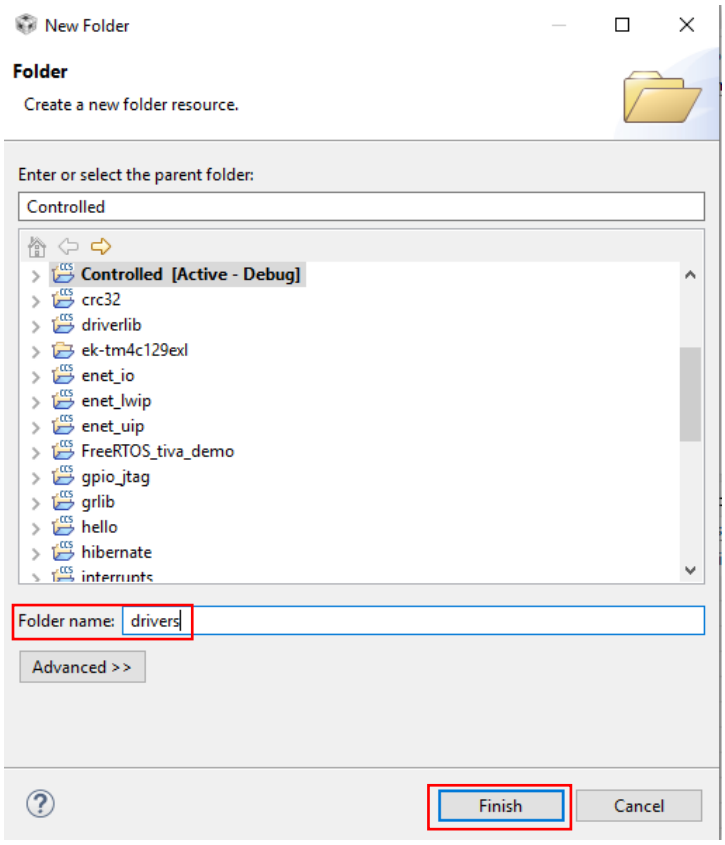


Figure 14: Adding a Folder to the Project

- Right click on the project name and select *Add Files*. Browse to *C:\ti\TivaWare_C_Series-2.1.4.178\examples\boards\ek-tm4c129exl\drivers* and select *buttons.c* and *pinout.c* and click on *Open*.

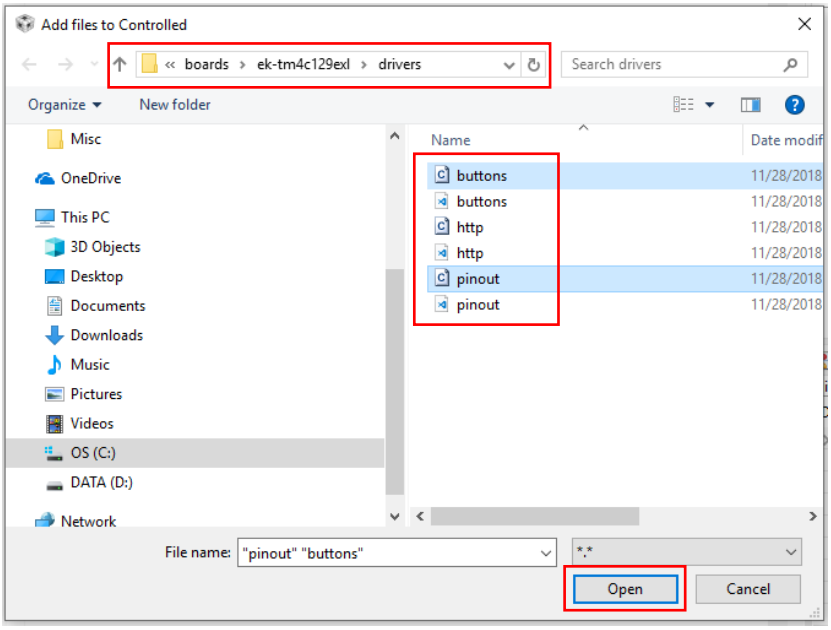


Figure 15: Adding Files to the Project

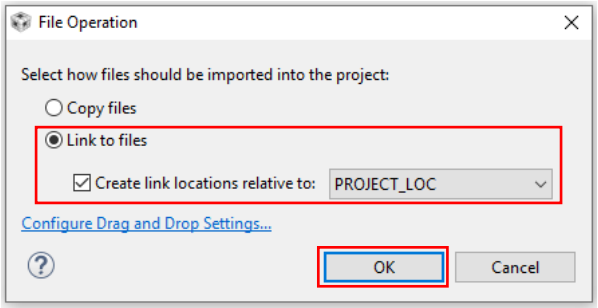


Figure 16: Adding Files to the Project as Links

- Move the added files to the folder *drivers*. By keep clicking on the files and moving them to the folder *drivers*.

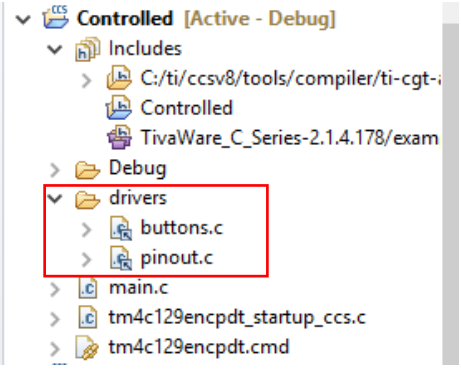


Figure 17: Moving Files to the *drivers* Folder

6.1.4 Writing the project's code

Double click on *main.c* and write the following code.

```
#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_memmap.h"
#include "driverlib/gpio.h"
#include "driverlib/rom_map.h"
#include "driverlib/sysctl.h"
#include "drivers/buttons.h"
#include "drivers/pinout.h"


// The error routine that is called if the driver library
// encounters an error.
#ifdef DEBUG
void
__error__(char *pcFilename, uint32_t ui32Line)
{
    while(1);
}
#endif

// Main Function
int main(void)
{
    unsigned char ucDelta, ucState;

    // Configure the device pins.
```

```
    PinoutSet(false, false);

    // Initialize the button driver.
    ButtonsInit();

    // Enable the GPIO pin for the LED (PN0).
    // Set the direction as output, and
    // enable the GPIO pin for digital function.
    GPIOPinTypeGPIOOutput(GPIO_PORTN_BASE, GPIO_PIN_0);

    // Loop forever.
    while(1) {
        // Poll the buttons.
        ucState = ButtonsPoll(&ucDelta, 0);

        if(BUTTON_PRESSED(RIGHT_BUTTON, ucState, ucDelta))
            // Turn on D1.
            LEDWrite(CLP_D1, 1);
        if(BUTTON_PRESSED(LEFT_BUTTON, ucState, ucDelta))
            // Turn off D1.
            LEDWrite(CLP_D1, 0);
    }
}
```

6.1.5 Build and Test the Project

- Click on *Project* → *Build Project*.
- If no compilation errors, click on *Run* →*Load*→ *Controlled*.
- Press the left and the right buttons and led should turn off and on respectively.

7 Assignments (1 bonus point if completed together with the report before the deadline, which is specified in Canvas for each assignment)

1. **Button input**. Write a program to start blinking LEDs as an action of button events. The LED should start blinking after you press the button.
2. Write a program that uses a button to light a LED. When the button is pressed the LED should be lit. When the button is released the LED should go off. Program another button to act as an on/off switch. Pressing and releasing the button once should light the LED. The next press/release cycle should turn the LED off.

8 The Report

The report should include the answers of the following questions:

1. How do you terminate the debugging session?
2. How do you terminate the program running on the dev. board?
3. **Bitwise operations:**
   - Bitwise operations &, ^, |, ~, <<, >> *
   - Logical operations &&, || *
   - Relational operators ==, !=, <, >, <=, >=

   Source file: lab1/src/bitwiseoperations.c

In embedded systems it is often necessary to alter or read single bits in various registers. This task will train you in the bitwise operations available in the C language.

1. Use the Code Composer Studio.
2. Create a new project.
3. Open the source file, lab1/src/bitwiseoperations.c, and manually read through the source and note the value of 'result' at each time there's an assignment to the 'result' variable.
4. Compile the program.
5. Program the dev. board with the produced .elf file from the compile.
6. Set a breakpoint at the beginning of the main()-function.
7. Start a debugging session of the program.
8. Step through the program and verify your predicted values of the 'result' variable.

Questions:

(a) What is the value of the 'result' variable at each occasion it is assigned a value?

(b) What happens at the end of the program?

9 Optional Assignment (1 bonus point if completed before the deadline, which is specified in Canvas for each assignment)

Make a LED chaser program. The idea is to have the LEDs light up in sequence one after another to give the impression that the light is moving back and forth on the LED array.

Note: The bonus points collected after successfully completing the compulsory assignments and optional assignments before the corresponding deadlines will be added to your score in the final exam. These bonus points could help you in improving your final grade in the course.