

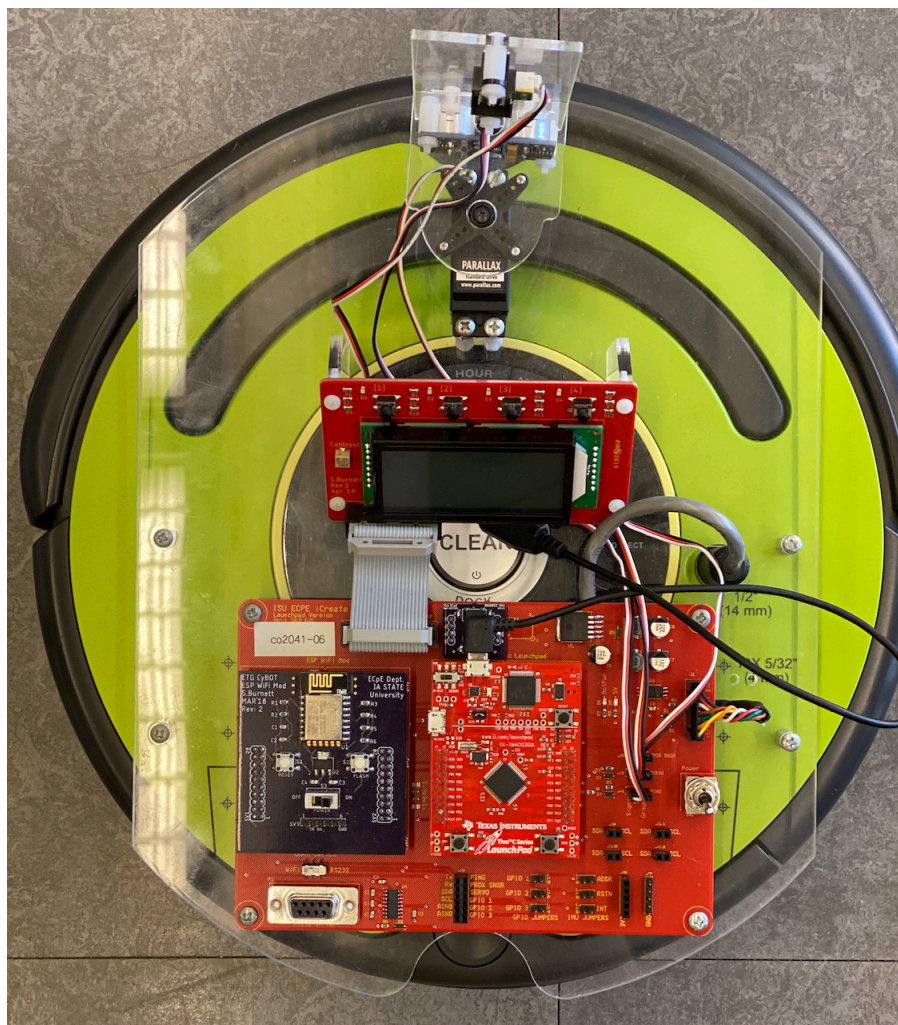
# LAB 1

## *INTRODUCTION TO CODE COMPOSER STUDIO AND THE CyBOT PLATFORM*

### INTRODUCTION

In lecture, we are starting with an overview of embedded systems, microcontrollers and embedded programming in C. In lab, we are starting to work with the platform and tools for developing and testing embedded applications using the CyBot. The CyBot platform consists of a Texas Instruments Tiva C Series TM4C123G Launchpad Evaluation Board, an iRobot Create 2 mobile robot, and several other boards and input/output devices. TI's Code Composer Studio is the integrated development environment (IDE) on the desktop PC for source code editing, compiling/building and downloading code to the Launchpad board, and debugging.

In this lab, you will be demonstrating that the CyBot can have basic communication with the outside world through the use of the LCD board. You have been given a simple program to start with that prints "Hello world" on the LCD screen. However, your main task will be to write a reusable function for a rotating banner on the LCD screen. Throughout this lab, think about some practical embedded system applications where communication using the LCD may be useful. You may have the need to implement a similar functionality in your final project.



## COMPANY POLICY REMINDERS

1. Following company protocol, prior to working in the lab you must print, read, and sign a company **Lab Safety Policy Agreement**. The **Lab Safety Policy Agreement** can be found in **Canvas** under **Lab Resources**. Without it, you will not be able to work in the lab.
2. ESI is committed to delivering quality products to customers in a timely manner. In order to maintain this commitment, product development must follow the product development schedule. Any adjustments that need to be made to this schedule should be communicated with your mentor and manager (aka lab TA and course instructor) in advance. Exceptions include documented illness and family emergencies.
3. ESI requires that all code be documented properly and comply with company coding standards. Documentation will be considered in your lab evaluation. **Review the Company Coding Standards.**

## REFERENCE FILES

The following reference files will be used in this lab:

- helloworld.c, the main program for printing “Hello, world” in the CyBot LCD screen
- lcd.c, program file containing various LCD functions
- lcd.h, header file for lcd.c
- timer.c, program file containing various wait commands
- timer.h, header file for timer.c
- CyBot baseboard and LCD board schematics
- LCD controller datasheet, information about the LCD controller
- <http://www.cplusplus.com/reference/cstdio/printf/> , resource for information about printf

The code files are available to download from a Google Drive folder:

[https://drive.google.com/drive/folders/1LPneqbbOkbS\\_1rPpEeJsEhUi4gdsLAk5?usp=sharing](https://drive.google.com/drive/folders/1LPneqbbOkbS_1rPpEeJsEhUi4gdsLAk5?usp=sharing)

## CYBOT CHARGING

Turn off the power to the CyBot platform before charging and check the charging status. The TI Launchpad board consumes power, and the battery will not charge properly when the board is left on. Additionally, ensure that the CyBot is turned off completely. The battery will not charge if the iRobot is in Full Mode, which it enters after calling oi\_init. The battery status of the iRobot is shown with these LED colors:

Color of Power Light	Battery Status
Slow Pulsing Orange	Charging (iRobot rechargeable battery only)
Fast Pulsing Orange	Reconditioning Charge (iRobot rechargeable battery only)
Green	Fully Charged
Amber	Partially Discharged
Red	Almost Fully Discharged
Flashing Red	Fully Discharged

## GETTING HELP

If you need help, use the resources available to you, for example:

1. Post questions to Piazza.
2. Attend office hours with your mentors and/or instructor.
3. Discuss with peers.

Choose resources that meet your needs. Some resources work better than others for different students and different situations.

## TIPS FOR SUCCESS

### Preparation

**Come to labs prepared by completing the prelab and reading the lab manual prior to lab.** Ask any questions you may have on Piazza or jot them down to ask your mentor during lab time.

### Utilize Resources

There are many resources available to you so use them. Read the lab manuals thoroughly and pay attention to details. Ask questions on Piazza or talk to your mentor if you have already consulted the documents available to you.

### Problem Solve

Problem solve by utilizing your resources and asking yourself “Why?”. Use your debugging skills and consult the documents that have been made available to you. Asking why will often help you to think critically and lead to better and more creative solutions to problems. If you’re stuck on the same problem for a long time and you have already utilized your resources, ask for help. Don’t be afraid to ask for help when you need it.

### Learn from Failures

Don’t be afraid to fail. Everyone makes mistakes, learn from them and keep an open mind in the future when struggling with new problems. It is very rare to be successful in the first attempt on new or challenging problems. As such, continue to problem solve and utilize your resources to break new ground and find solutions that work. Keep trying. Keep asking questions. Keep using resources.

### Communicate

Communicate it. The only way we can ensure continuous improvement is if problems and ideas are thoroughly communicated. If you find yourself daydreaming or complaining “if it was different”, communicate that. Understand that there may not be time to make the change immediately, but the feedback may have the potential to help someone else in the future.

## ETG

If you are having issues with any equipment, software, or lab facilities **outside of a scheduled lab period**, submit a support request at <https://admin.ece.iastate.edu/requests/support/new>. Be sure to include the computer #, CyBot #, approximate time of the issue, and any other relevant information.

You can also visit ETG on Monday – Thursday, 8 am - 8 pm, and Friday 8 am - 5 pm, in 1331 Coover next to the TLA.

## PRELAB: SYSTEM SKETCH

As you are developing software for the CyBot platform, it is important that you have a complete understanding of the hardware on which you will be deploying your embedded systems application. In this lab you will be using the CyBot's LCD screen to display text to a user. Complete the following prior to the start of lab:

1. Sketch a diagram that shows how the LCD screen of the CyBot platform connects to the CyBot microcontroller. You will need to use several documents from the Reference Files list, including the schematics, LCD datasheet, and LCD code. Treat this like a puzzle, and start putting together the pieces of the puzzle in your sketch. Your diagram might look like a block diagram that has blocks for the LCD and microcontroller packages (chips). Try to track down some details about the connections between the LCD board and microcontroller. It doesn't need to show all of the pins, wires and signals, but it should show some details, such as the package pin number and datasheet names of the LCD screen and how they connect to the microcontroller pins (e.g., pin number and actual port/pin names used on the microcontroller). Keep in mind that the microcontroller is on the Launchpad board, which is mounted on the CyBot baseboard, which has a jumper cable to the LCD board. List the documentation that provided information for each part of your sketch. There is no single right sketch.
2. Is the LCD screen, as set up on the CyBot, connected using a 4-bit or 8-bit Data Bus (DB) wiring interface? Explain how you came to this conclusion. The purpose of this question is to get you used to looking for specific information in a datasheet and relating it to other information.

Submit the prelab on Canvas prior to the start of lab under the assignment titled "Pre-Lab 1".

Try to have at least a rough draft of the prelab when you come to class on Tuesday.

## PART 1: "HELLO WORLD"


Your first task is to build a program executable in CCS using files provided to you.

1. In your **U: Drive**, create a new folder and name it **CprE288Workspace**.
2. In your programs, search for and open Code Composer Studio. You will be prompted to select a workspace. Select the folder in your **U: Drive** that you just created labeled '**CprE288Workspace**'

**\*\*Tip:** If you accidentally select the wrong folder as your workspace, would like to change the folder that you are currently working in, or you open CCS and none of your files are showing, then ensure that you are in the correct workspace by selecting **File → Switch Workspace → Other** and selecting the correct directory as your workspace. CCS will close and relaunch using the workspace you selected.

3. In CCS, create a new project by selecting **File → New → CCS Project**. Name the project "**Lab1**".
4. Choose the following configurations:
  - **Target** as '**Tiva TM4C123GH6PM**'
  - **Connection** as '**Stellaris In-Circuit Debug Interface**'
  - **Project type** as '**Empty project with main.c**'


Click '**finish**' to create your project.

 **New CCS Project**

**CCS Project**  
Create a new CCS Project.

Target: <select or type filter text> **Tiva TM4C123GH6PM**

Connection: Stellaris In-Circuit Debug Interface **Verify...**

 **Cortex M [ARM]**

Project name: **Lab1**

☒ Use default location








Location: U:\CprE288Workspace\Lab1 **Browse...**

Compiler version: TI v18.1.2.LTS **More...**

▶ Tool-chain


▼ Project templates and examples

type filter text

- ▼  Empty Projects
  -  Empty Project
  -  **Empty Project (with main.c)**
  -  Empty Assembly-only Project
  -  Empty RTSC Project
- ▼  Basic Examples
  -  Hello World

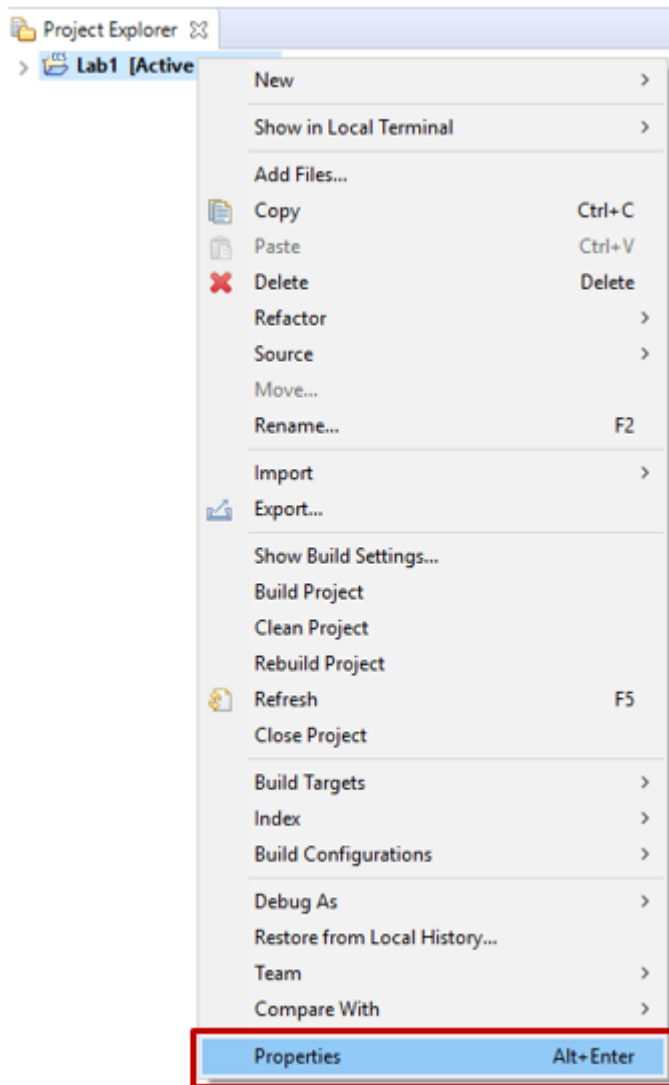
Creates an empty project initialized for the selected device. The project will contain an empty 'main.c' source-file.

Open [Resource Explorer](#) to browse a wide selection of example projects...

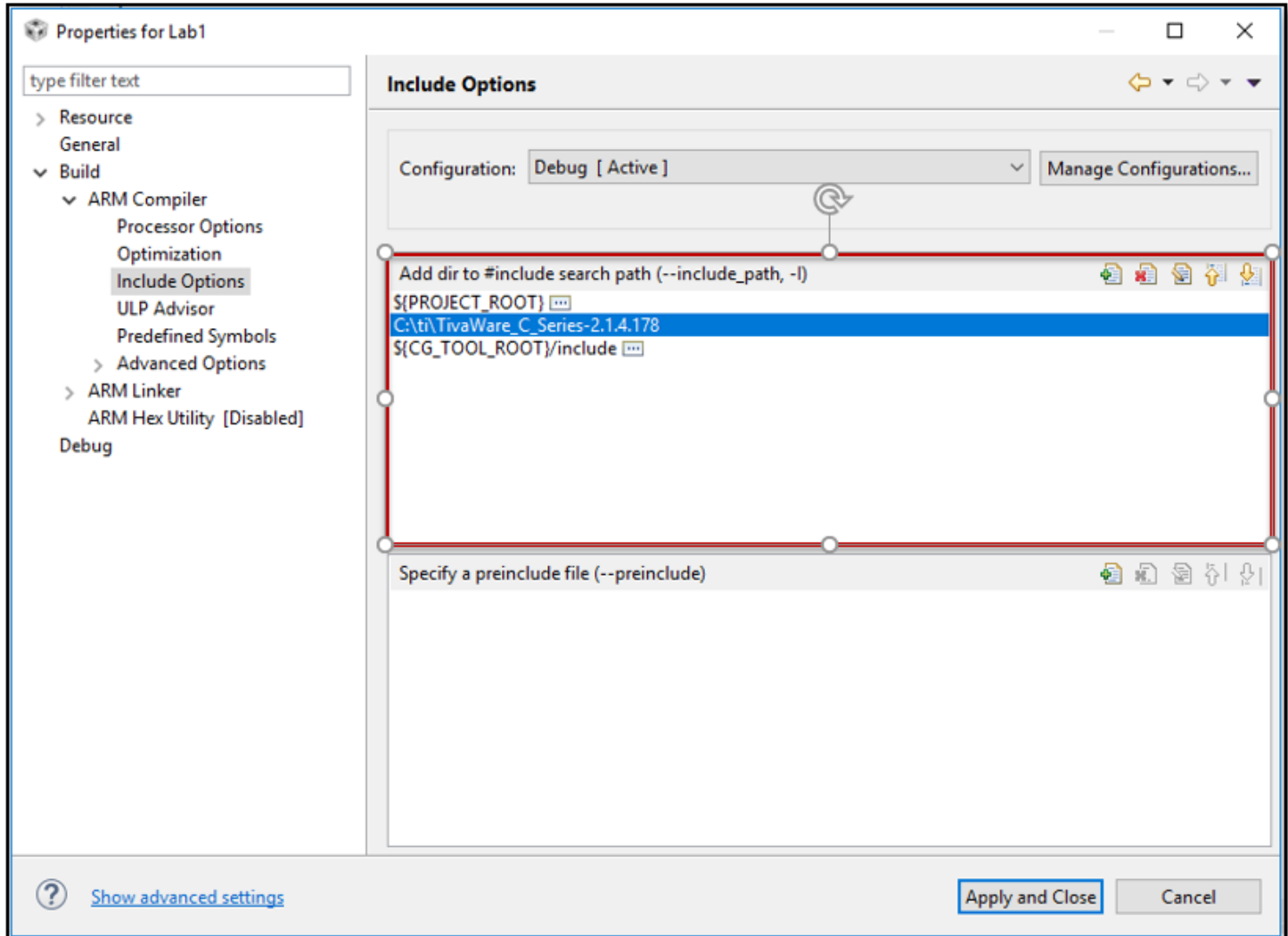
 < Back Next > **Finish** Cancel



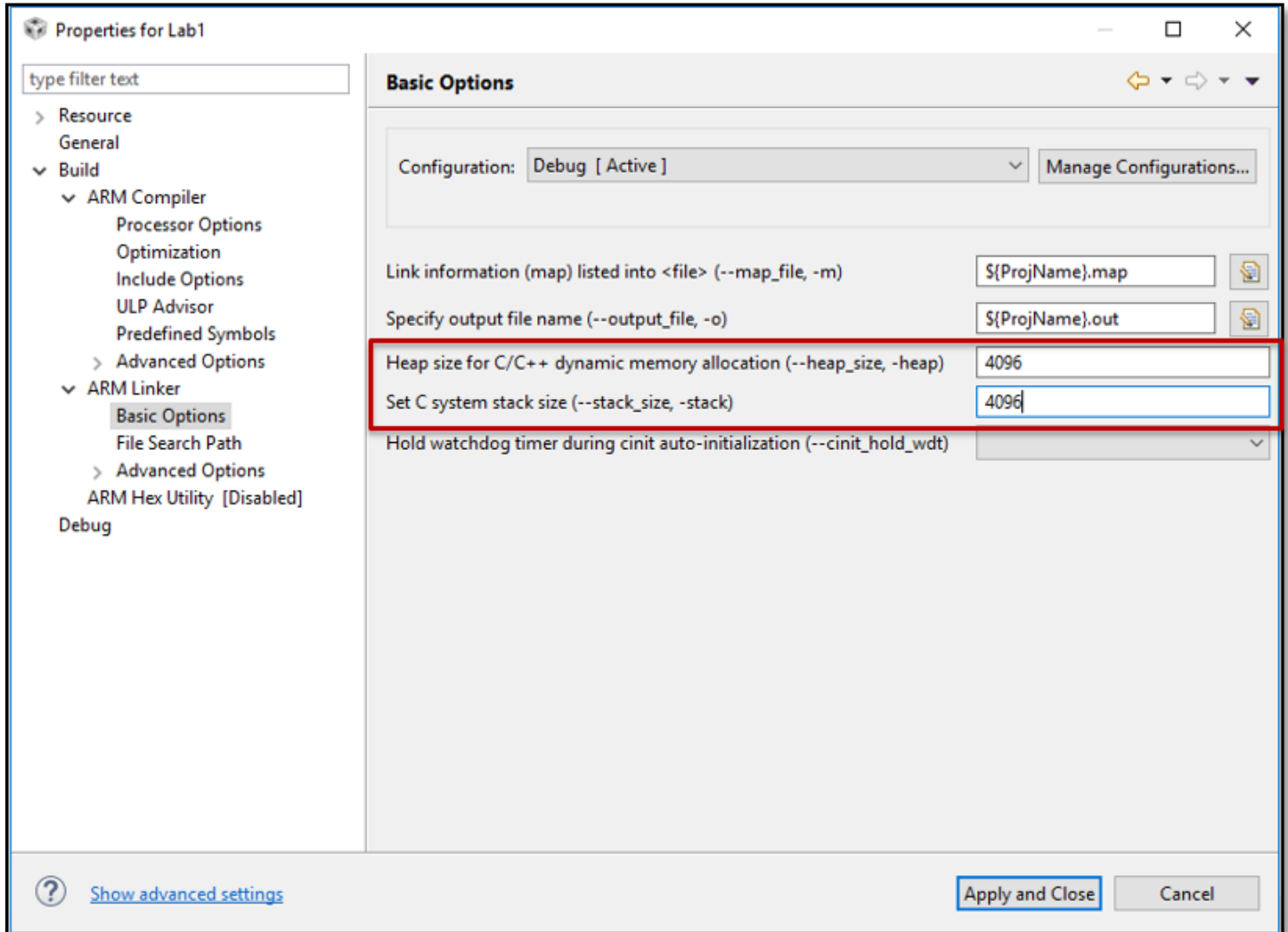
5. Right click on your project in the **Project Explorer** and select **'Properties'**.



6. Navigate to **Build** → **Arm Compiler** → **Include Options** and add  
“C:\ti\TivaWare\_C\_Series-2.1.4.178” (or the latest version) under search path.

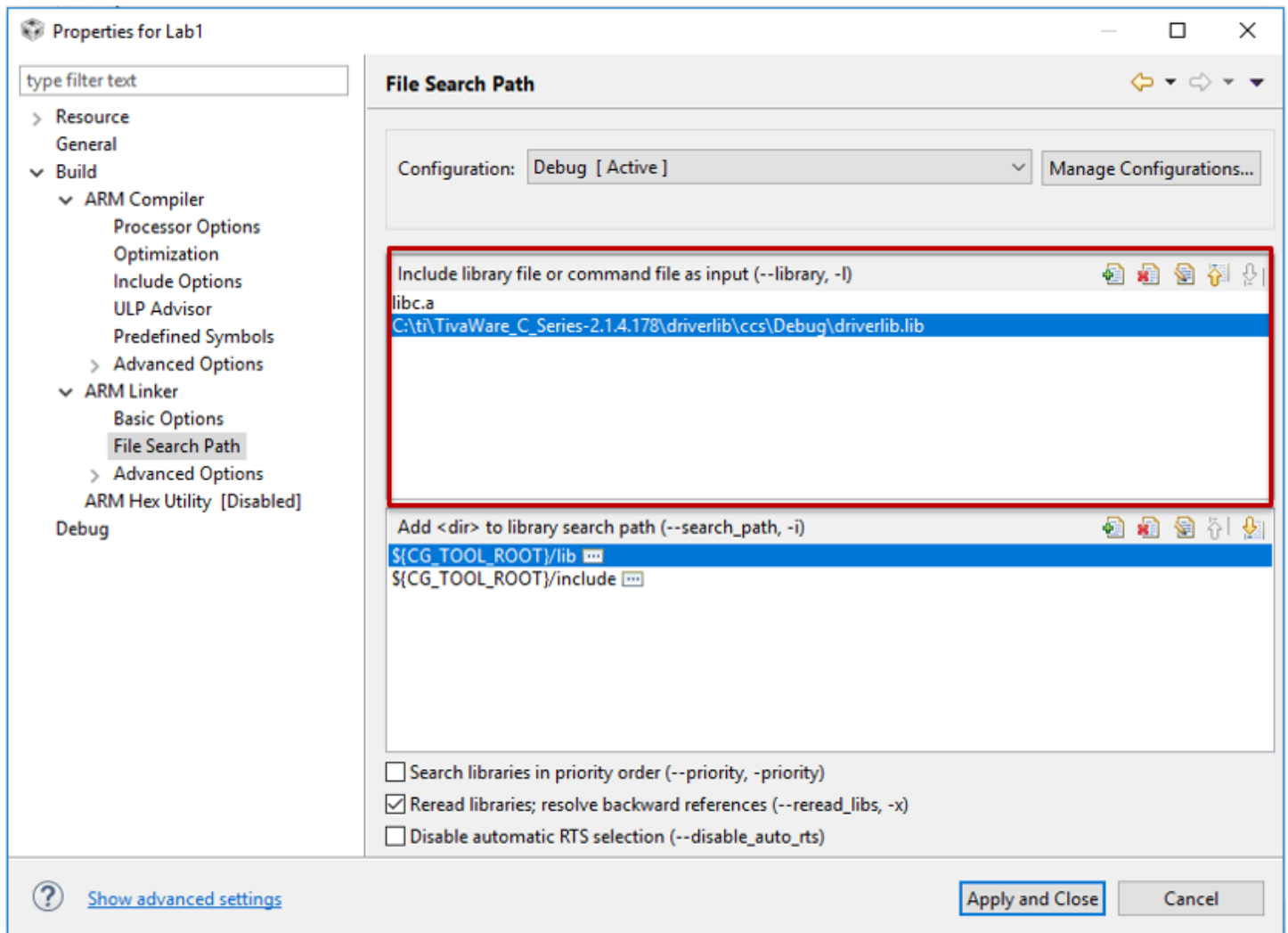


7. Navigate to **Build** → **ARM Linker** → **Basic Options** and modify the **stack** and **heap** sizes to both be **4096** as shown.





8. Navigate to **Arm Linker** → **File Search Path** and add  
“C:\ti\TivaWare\_C\_Series-2.1.4.178\driverlib\ccs\Debug\driverlib.lib”  
under library files to include as input.



9. Now, add the lab files into the project. You can do this by saving them to the project workspace when downloading them or copying and pasting them into the workspace.
10. Build the project by selecting **Build** → **Build Project**. You can also use the shortcut “**CTRL-B**”. You should see this message at the end: “Build Succeeded”.
11. Next, you need to load the binary image into the microcontroller (i.e., load the code into flash memory). Loading uses the debugging interface. Power on the Launchpad board and ensure the USB cable is connected. In CCS, select the “Run” file menu and click “Debug”. This loads your program and pauses execution on the first instruction. Click the “Resume” button to continue execution. You should see “Hello, world” displayed on the LCD screen.

**\*\*Tip:** You can use ‘F11’ to debug, and ‘F8’ to start the program.

## PART 2: DEBUGGER

Debugging is an essential skill to learn to be successful as often unexpected hardware and/or software issues may arise, leading to hours spent on detecting and fixing the issues. From a job perspective considering project costs and quality, it is critical that you learn how to efficiently and effectively identify these issues – this will happen with practice and experience. From a learning perspective, exploring system issues, behavior and status with the help of a debugger leads to deeper levels of problem solving and insights.

Debugging will allow you to execute code and observe changes line by line. Some useful features include: **(1) Variables**, **(2) Expressions**, and **(3) Register** tabs. These features will show you the current state of different parts of the code and hardware of the microcontroller.

In your main file, replace `lcd_printf("Hello, world")` with `lcd_puts("Hello, world")`. Step through "lcd\_puts" using the debugger. To do so, click **Run → Debug**. Press **F6** to **step over** each statement until you reach the "lcd\_puts" function. Press **F5** to **step into** this function.

### CHECKPOINT:

Describe to your mentor how the Variables, Expressions, and Registers tabs work. Additionally, explain the difference between `lcd_puts` and `lcd_printf` and how you came to this conclusion.

## PART 3: ROTATING BANNER

Write a function `lcd_rotatingBanner(...)` to display a rotating banner on the LCD screen. Every 300 milliseconds (0.3 seconds), the text "Microcontrollers are lots of fun!" should move one position to the left. The banner should repeat indefinitely. Duplicate and edit the helloworld project, as it provides a good starting point.

The program should adhere to the following requirements:

- Only the first line of the LCD screen should be used. The LCD screen has 4 lines. The second line should be blank all the time.
- At the beginning, the first letter 'M' should appear on the right side of line 1 of the LCD screen. The subsequent characters ('i', then 'c', etc.) appear one-by-one every 0.3 seconds.
- After the 20<sup>th</sup> letter appears (filling the first line of the LCD screen), the remaining characters of the message push the old letters off the LCD screen to the left.
- After the last character '!' appears on the right, the message continues shifting to the left. Every 0.3 seconds the leftmost letter is pushed out until the screen is completely cleared.
- After the screen is cleared for 0.3 seconds, repeat the entire banner again.
- The banner should repeat forever until the board is turned off
- The program should be written so that only the message is modified in the code.

Three functions that have been written for you that will help are:

1. `timer_waitMillis(uint32_t millis)`: Delay the program execution for 'millis' milliseconds.
2. `lcd_init()`: Initializes the I/O ports to communicate with the LCD controller; clears the screen.
3. `lcd_printf(const char *format, ...)`: Clears the screen and displays text.

See documentation on printf (<http://www.cplusplus.com/reference/cstdio/printf/>).

For this part of the lab, you may want to try breaking up the program into smaller pieces. How will you display a rotating banner? How will you shift a character every 0.3 seconds? What do you know about strings in C?

You might get started by thinking about the desired program behavior and describing it. Write down a general, but precise, behavior. One example of how you might implement the program is a forever loop of three phases:

1. Shift in the first 20 characters one by one.
2. Rotate until the last character is displayed.
3. Rotate until the last character is shifted out and the screen is clear.

Keep in mind that there are many ways of implementing this code.

Suggestion: Write your own utility functions to reuse and to modularize your code, thus reducing the complexity of the rest of your code.

**\*\*Hint:** Review string functions – are there any string functions such as `strncat(...)`, `strcpy(...)`, or `strlen(...)` that can help you?

### CHECKPOINT:

Demonstrate your rotating banner program to your mentor.

## GENERAL DEBUGGING STEPS

Adapted from an IBM article.

### 1. REPRODUCE IT

Make sure that you can reproduce the error before you start debugging

### 2. REDUCE INPUT

Determine the smallest input that causes the error. The smaller the input, the easier it will be to find the error.

### 3. ISOLATE THE PROBLEM CODE

Isolate the portion of your code causing the error. Do this by tracing the data's flow through the program. At the start of each function, do the variables contain the values you expect? Do the functions return what you expect? Isolating the function or lines of code causing the error will help in determining the solution

### 4. EXPERIMENT

Hypothesize a potential cause for the bug and then test to see if the hypothesis is correct by changing the input or code to either rule out the hypothesis or confirm it.

### 5. EXPERIENCE

Think if you have had this type of error before and what the solution is. Do a search or talk to others. Sometimes explaining the problem will help you discover the problem.

### 6. NEVER GIVE UP