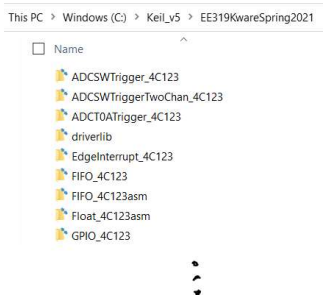
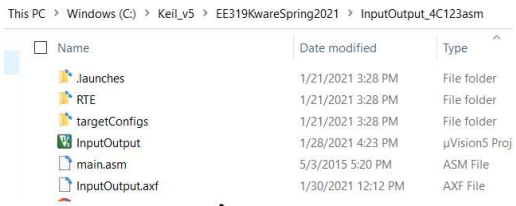
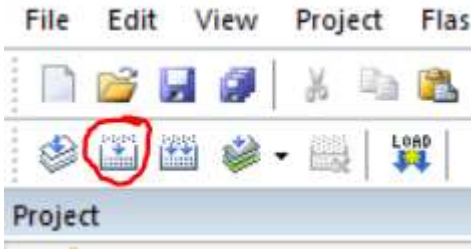
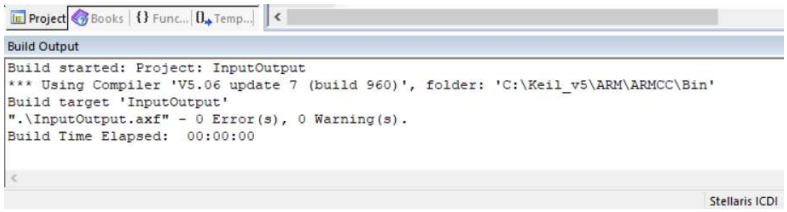
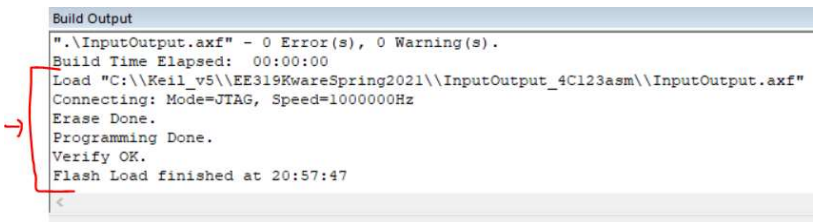
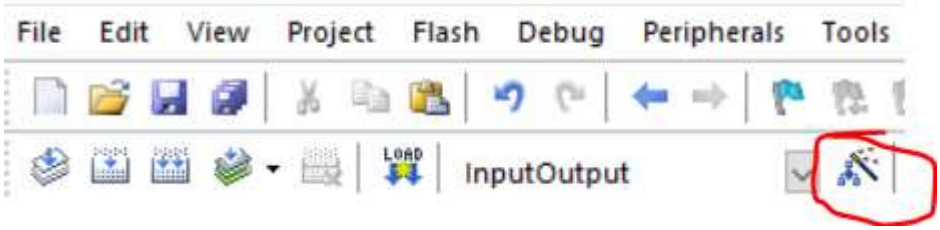
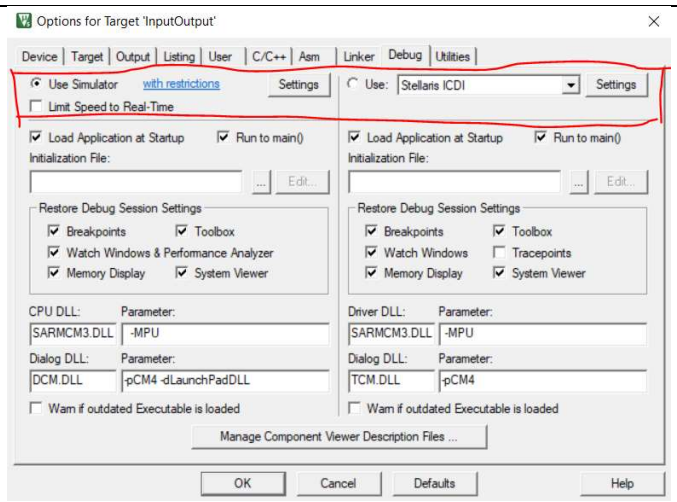
**Keil 5 Basics**

| Timestamp | Content |
|---|---|
| 0:14<br><br>**Setting up Keil &TM4c** | If you have not setup Keil properly yet, checkout out *Canvas > 319K > File > "Setting Up Keil & TM4C".pdf* for a thorough step-by-step documentation. If you have tried this already, reach out to us in any of our TA office hours.<br><br>Please try to get this done ASAP! |
| 0:50<br><br>**Opening a Project in Keil** | Navigate to the *EE319KwareSpring2021* folder. It should be in your Keil_v5 folder. This is what you should see in the folder:<br><br>  <br><br>⋮<br><br>For tutorial purposes, click into the *InputOutput_4C123asm* folder.<br><br>  <br><br>⋮<br><br>To open a project in Keil, click on the file that is of the *[micro]Vision5 Project* filetype. A heuristic to use would be to find the file with the green icon:<br><br>  <br><br>In any folder, whether it be this example or a lab folder, there will be a file with this filetype. Click this file and open your Keil project. |
| 1:34<br><br>**Building and Flashing Project** | To take code you have developed in an IDE and load it onto the LaunchPad itself, there are two key steps to take:<br><br>1) Build your code. To do so, click the button circled in red.<br><br>  <br> |

| | |
|---|---|
| | You will see a response at the bottom of your screen in the *Build Output* section.<br>  <br><br>From this we observe there are 0 errors & warnings.<br><br>2) Now we can flash our code. Click the button circled in red.<br>  <br><br>A successful flash will yield this message in the *Build Output* section:<br>  <br><br>Now you can see this example in action on your board. Use the two buttons on the short end of the LaunchPad to view different color LEDs lighting up. |
| 2:19<br><br>**Entering a Bug in Our Code** | **FOR DEMO PURPOSE ONLY**: I will change one instruction (line 142) from<br>`142 ORR R0, R0, #0x20`<br>to<br>`142 ORR R0, R0, #0x21`<br><br>This will be useful in our discussion about debugging basics. |
| 2:35<br><br>**Debugging** | There are two ways to debug your code. One would be to do it in a simulator within Keil. The second is to debug on the board itself.<br><br>Here is where you can select the option:<br>  <br><br>Once you have clicked the wand, you will see this in the *Debug* tab: |




The left-hand side says *Use Simulator*, the right says *Use Stellaris ICDI*. The left will allow you to debug with the simulator, and the right allows you to debug on the board.

**For this tutorial, I will be using the simulator.**

2:38

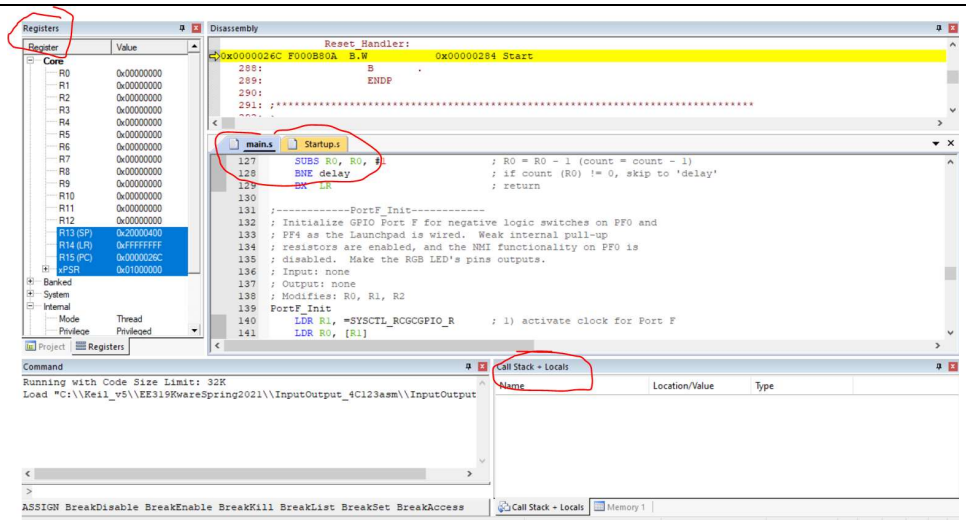
**Debug Session**

To enter a debug session, click the magnifying glass:

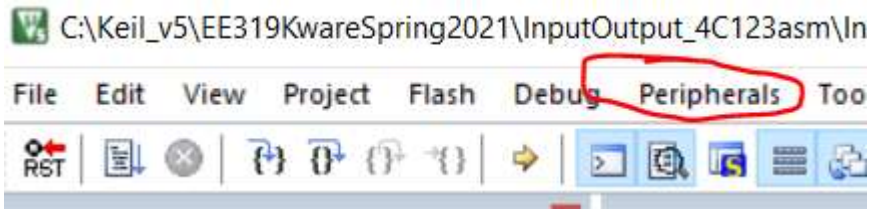
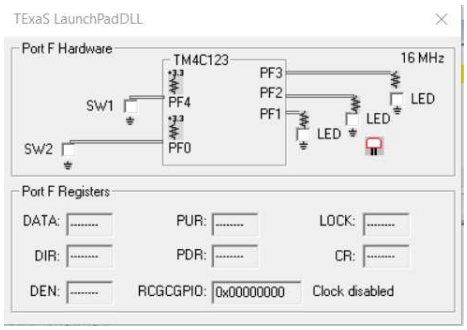
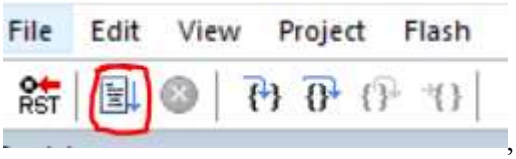

You can leave a session by going to *Debug > Startup/Stop Debug Session* or by entering *(Ctrl + F5)*.

3:00

**Debug Layout**



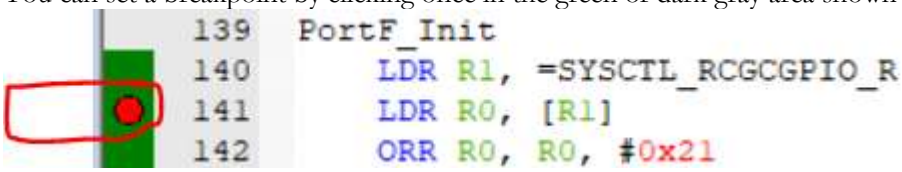

The 3 main windows of critical use are the *code viewer*, the *Registers viewer*, and the *Call Stack + Locals viewer*.

| | |
|---|---|
| | The code viewer is where you will be able to set breakpoints (discussed later) and follow your work step by step.<br><br>The Register viewer is where you can see the values stored in each of the registers.<br><br>The Call Stack + Locals viewer shows the things currently on the stack as well as local variables. There are 3 key columns in this window. The first is the name of the symbol, the second is the location or value associated with the symbol, and the last is the type of the object associated with the symbol. |
| 3:31<br><br>**Running a Simulation** | Since this code relies on Port F, it would be helpful to have a visual of Port F.<br><br>Go to Periperhals > TExaS Port F.<br><br>  <br><br><br>You'll see a window like this:<br><br>  <br><br><br>The left hand button are the inputs and the right side boxes (not clickable) are outputs.<br><br>Click the run button  , and toggle switches in the simulator window!<br><br>You'll see the program is not working properly. This is because we changed Line 142 to 0x21. |
| 4:00<br><br>**Breakpoints** | How would you have found the problem code without me telling you which line?<br><br>Breakpoints! |

But first, we must (1) stop the previous run and (2) reset the CPU.


Now you go to your code viewer.

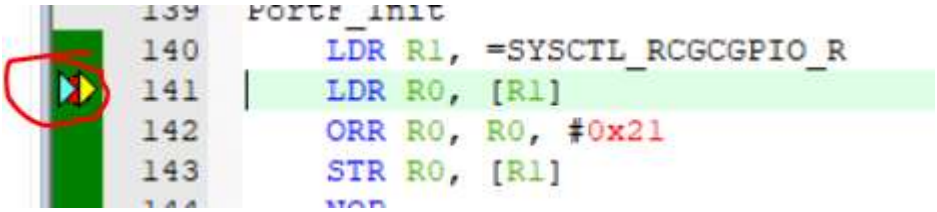
You can set a breakpoint by clicking once in the green or dark gray area shown below:




Another click would clear the breakpoint.

Typically, we set breakpoints where we think our problem code is. Since we have the benefit of knowing where it is automatically, we can set one at Line 141.

Now click the *Run button* once again. You should see something like this:




This means your execution has stopped at that point. You have control of what to do next.

4:33

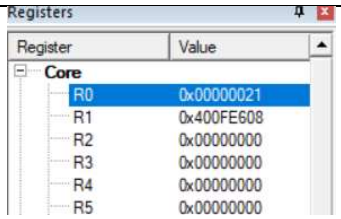
**Stepping and Keeping Track of Information**

There are 4 options in stepping:


From left to right:

1) Executes a single-step into a function/sub-routine.
2) Executes a single-step over a instruction.
3) Finishes executing the current function and stops afterwards.
4) Executes the program until the current cursor line is reached.

In this tutorial, we will use the 2nd option. When we stepover the next line, we will see changes in *the register viewer*.

| | |
|---|---|
| |  <br><br><br>Here R0 just received the value 0x21. But we know the value is supposed to be 0x20.<br><br>So, the line in the code that caused the R0 to receive a 0x21 is the line of code causing problems in our logic. Issue found, and can be resolved! |