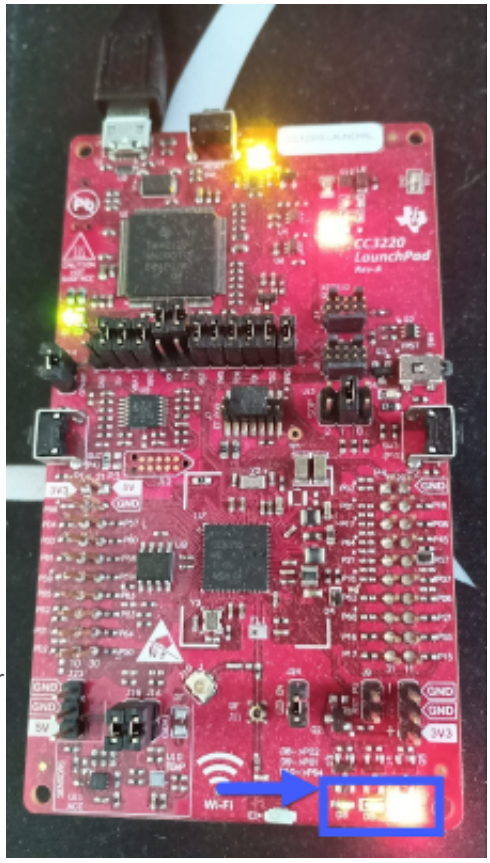
Overview



In this lab, you will use TI Code Composer Studio (CCS) to program the TC CC3220x LAUNCHXL to control an LED from the serial port using a state machine. This work will build on the concepts you learned during your activities in the zyBook this week.

During this milestone you will use CCS to edit, compile, and load code into the CC32xx board. You will then proceed to use it for debugging. Throughout this process, you explore the components of a CCS project and the CCS code generator (system config). You will also be able to learn more about the UART and GPIO drivers.

Goal: Your objective is to control the red LEDs in the lower right corner of the board (the corner opposite the USB connector). Use this to design a state machine to turn on an LED when a user types ON into the console and to turn off the LED when a user types OFF, using only one byte of RAM. Remember, you receive only one character at a time from the UART and it is not buffered. Then you will code the state machine you created.

Prompt

Begin your work by accessing the Milestone Two UART GPIO Lab Guide PDF document. While this document was written for a Windows interface, the tools can be used on Mac or Linux as well. Note that to accomplish the work outlined in the guide, you will need the following:

- TI CC3220x LAUNCHXL
- TI Code Composer Studio (installed)
- USB connection between the PC and board

Specifically, you must address the following rubric criteria:

- Develop **code** for all of the specified **functionality**. The goal of this criterion is for the code to result in the expected functionality (note how this is different from the state machine functionality). Both ON and OFF should operate as expected. In your video, show the terminal window and LED to make this clear.
- Create code that reads characters from the **UART**. This must use only one byte at a time with no multi-byte buffering of the serial input. The characters are encoded back.
- Create code that controls the LED on and off from the state machine. Note that this involves use of the **GPIO peripheral**. Partial credit may be awarded if the code does not work but you are able to successfully determine how to turn the LED on and off, then include it in the comments.
- Implement (in code) the **state machine functionality** described by your documentation. The goal of this criterion is for the code to accurately reflect the state machine documentation, rather than for it to have perfect functionality.
- Create **state machine documentation** to describe the operation that matches the technical specifications. This should be completed as a draw.io file and saved as a PDF.
- Discuss the **questions** from the lab. Address all the questions thoroughly and thoughtfully, with supporting evidence from your work.
- Apply coding **best practices** in formatting, commenting, and functional logic.

Guidelines for Submission

You will have four different file submissions for this milestone. Once you have completed your lab work, first zip your workspace and submit the zipped file for grading. Second, submit a 10-second video of the LEDs blinking on your board. If you encounter any difficulties filming the lights on your hardware component, reach out to your instructor. Third, submit a document containing your answers to the questions from the Milestone Two UART GPIO Lab Guide. Finally, submit a state machine diagram completed using draw.io and exported in PDF format.

Milestone Two Rubric

Criteria	Exemplary (100%)	Proficient (85%)	Needs Improvement (55%)	Not Evident (0%)	Value
Code Functionality	Exceeds proficiency in an exceptionally clear, insightful, sophisticated, or creative manner	Develops code for all of the specified functionality	Shows progress toward proficiency, but with errors or omissions; areas for improvement may include both ON and OFF function as anticipated	Does not attempt criterion	16
UART	Exceeds proficiency in an exceptionally clear, insightful, sophisticated, or creative manner	Creates code that reads characters from the UART	Shows progress toward proficiency, but with errors or omissions; areas for improvement may include reading characters from the UART one at a time and not storing the data in a multi-byte buffer	Does not attempt criterion	7
GPIO Peripheral	Exceeds proficiency in an exceptionally clear, insightful, sophisticated, or creative manner	Creates code that controls the LED on and off from the state machine	Shows progress toward proficiency, but with errors or omissions; areas for improvement may include the LED turning both off and on from the state machine	Does not attempt criterion	7
State Machine Functionality	Exceeds proficiency in an exceptionally clear, insightful, sophisticated, or creative manner	Implements (in code) the state machine functionality described by the created documentation	Shows progress toward proficiency, but with errors or omissions; areas for improvement may include implementing all of the components of the state machine	Does not attempt criterion	30
State Machine Documentation	Exceeds proficiency in an exceptionally clear, insightful, sophisticated, or creative manner	Creates state machine documentation to describe the operation that matches the technical	Shows progress toward proficiency, but with errors or omissions; areas for improvement may include addressing all the necessary	Does not attempt criterion	20

		specifications	functionality or documenting according to industry standards		
Questions	Exceeds proficiency in an exceptionally clear, insightful, sophisticated, or creative manner	Discusses the lab questions	Shows progress toward proficiency, but with errors or omissions; areas for improvement may include thoroughly addressing all of the questions or providing specific examples	Does not attempt criterion	10
Best Practices	Exceeds proficiency in an exceptionally clear, insightful, sophisticated, or creative manner	Applies coding best practices in formatting, commenting, and functional logic	Shows progress toward proficiency, but with errors or omissions; areas for improvement may include use of formatting best practices to make code easy to read, commenting best practices to ensure code is clearly explained, or functional logic so the program runs as expected	Does not attempt criterion	10
				Total:	100%