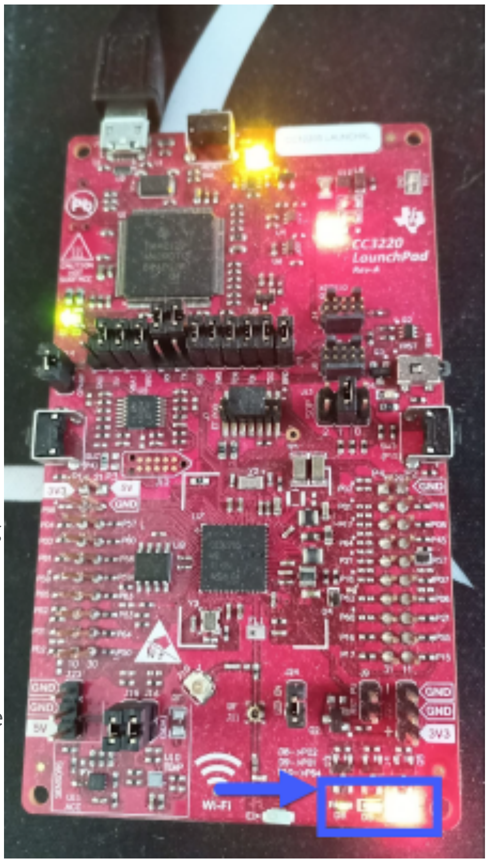
Overview



In this lab, you will use TI Code Composer Studio (CCS) to program the TC CC3220x LAUNCHXL to send a Morse code message via the LED using a synchronous 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 timer and interrupt drivers.

Goal: Your objective is to blink the green, yellow, and red LEDs in the lower right corner of the board (the corner opposite the USB connector). Design a synchronous state machine that creates a pattern of blinking lights from the LEDs. This pattern should contain a message in Morse code. When a button is pressed, the Morse code message of the blinking LEDs should change.

Prompt

Begin your work by accessing the Milestone Three Timer Interrupt Lab Guide PDF document. While this document was written for a Windows interface, the tools can be used on Mac or Linux as well. You may also watch the Project Example Video or its transcript Word Document to gain a better understanding of what you are creating. 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 criteria is for the code to result in the expected functionality (note how this is different from the state machine functionality). This includes functionality of both the state machine and the button change.
- Create **code** that initializes the **timer** and uses it to drive the state machine.
- Create **code** that uses **interrupt** to detect button presses. Make sure that the interrupts used to detect button presses have a result that is used by the state machine.
- Implement (in code) the **state machine functionality** described by your documentation. The goal of this criteria 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. It must match all required functionality. This should be completed as a draw.io file and saved as a PDF.
- Discuss the lab **questions**. 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 maximum 1-minute video of the LEDs blinking Morse code on your board, along with you pushing a button to change the Morse code. You may wish to narrate your video to explain what is happening. If you encounter any difficulties filming the lights on your hardware component, please reach out to your instructor. Third, submit a document containing your answers to the questions from the Milestone Three Timer Interrupt Lab Guide. Finally, submit a state machine diagram completed using draw.io and exported in PDF format.

Milestone Three 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 demonstrating both the state machine functionality and the button change functionality	Does not attempt criterion	10
Timer Peripheral Code	Exceeds proficiency in an exceptionally clear, insightful, sophisticated, or creative manner	Creates code that initializes the timer and uses it to drive the state machine	Shows progress toward proficiency, but with errors or omissions; areas for improvement may include correctly initializing the timer or using the result in the state machine	Does not attempt criterion	10
Interrupt Init and Handler Code	Exceeds proficiency in an exceptionally clear, insightful, sophisticated, or creative manner	Creates code that uses interrupt to detect button presses	Shows progress toward proficiency, but with errors or omissions; areas for improvement may include correctly initializing the peripheral or using the result in the state machine	Does not attempt criterion	10
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	Shows progress toward proficiency, but with errors or omissions; areas for improvement may	Does not attempt criterion	20

		matches the technical specifications	include addressing all the necessary 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%