# PROJECT 2:
# SHIELDS UP!

## OVERVIEW

For this project you will modify a starter program to protect it against internal and external faults such as unexpected inputs, coding and design errors, explicit attacks, device failures, electromagnetic interference, etc. This is not perfect protection against every possible fault, but it does help improve the system's robustness and reliability.

The starter program uses the buck converter to flash the shield's white LED (also called HBLED or LED1). The LCD provides a user interface which allows changing operating parameters and monitoring actual LED current. The program shares the ADC between the buck converter controller and the touchscreen code while still maintaining correct timing for the buck converter controller. **Note that the buck converter, LED, LCD and touchscreen must work to complete this project. Run the project starter code early to confirm correct operation.**

Many types of faults may be triggered; refer to the code in fault.c for details. You must modify the program to detect and handle some of the 13 types of faults (**ECE 460: pick 6 fault types, ECE 560: pick 10 fault types**). If feasible, handle the faults in a way that the program keeps running without restarting. Otherwise, handle the fault by restarting the program (e.g. with the watchdog timer (COP)). Refer to Chapter 7 of **ESF** and the corresponding directory in the ESF GitHub repository for watchdog timer information and example code. Your report will document how you tackled the faults which you picked.

## INTRODUCTION TO STARTER CODE


Start with the code on the github repository at Projects/Project_2/Project_2_Base. This program controls the HBLED current to ramp it up and down quickly. A user interface graphically displays the set current (blue), measured current (yellow). The image to the right shows the system operating after one of the faults has occurred, corrupting the PID_FX gains.

The numerical value of key parameters is also displayed. A parameter in green can be changed by pressing the parameter name and then using the grey slider control at the bottom of the display to increase the value (press on the right side of the slider) or decrease it (left side). The amount of change increases as you press closer to the left or right edge of the screen.

This code uses a high switching frequency (**96 kHz**) to reduce ripple, but a lower control loop frequency (**24 kHz**) to reduce the CPU load. This frees up time for other processing (such as the user interface). As in Lab 3, this code uses TPM0 to generate the PWM signal BUCK_DRV. Unlike Lab 3, this code has TPM0 generate an interrupt request on overflow. The TPM0_IRQ handler will sometimes trigger the ADC with software to start a conversion of the current sense voltage. This conversion will be triggered every SW_CTL_FREQ_DIV_FACTOR times the handler runs, leading to a control loop frequency of $f_{switching}$/ SW_CTL_FREQ_DIV_FACTOR. The ADC generates an interrupt when a conversion completes, causing the ADC0_IRQ handler to run.

## SOFTWARE ARCHITECTURE

Figure 1 shows the software architecture for key parts of the system. There are other threads and ISRs present in the system (for example for the RTOS).

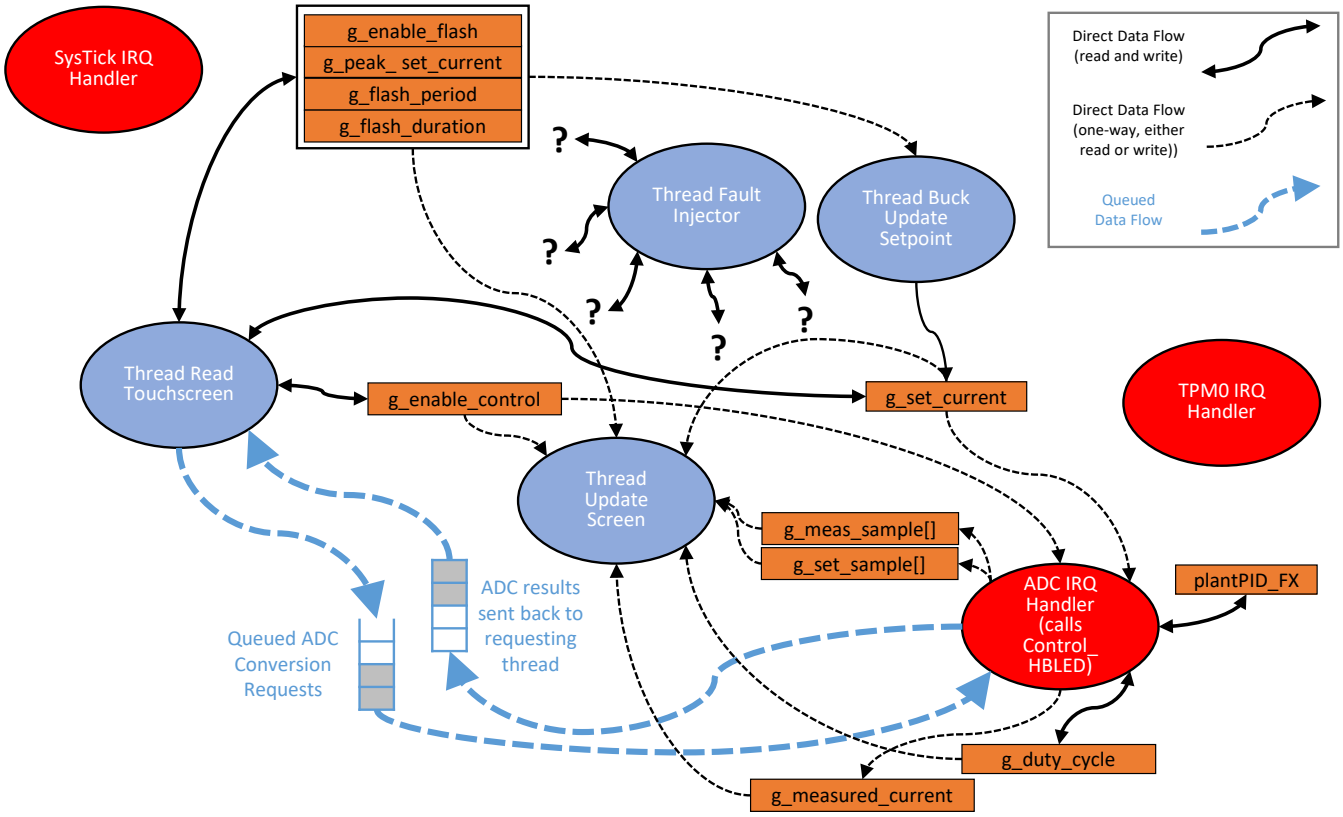



**Figure 1. Software architecture for Project 2 program.**

- **Thread_Read_Touchscreen** (Thread_Read_TS) calls the function LCD_TS_Read to see if the screen is touched. If the screen is touched, then LCD_TS_Read will use the ADC to determine where the screen is pressed. Thread_Read_TS then calls UI_Process_Touch to update various control parameters appropriately.
- **Thread_Update_Screen** updates the display based on set and measured currents, control parameters and other operating data.
- **Thread_Buck_Update_Setpoint** adjusts the HBLED's current setpoint and causes flashing.
- **ADC0_IRQHandler** implements a server to share the ADC between the HBLED control system (implemented by calling Control_HBLED) and any other ADC use requests (Thread_Read_TS in this program). Note that code related to controlling the HBLED has been moved into different files compared with Lab 3: the definitions previously in HBLED.h have been moved to control.h, and the control functions in main.c have been moved to control.c.
- **TPM0_IRQHandler** triggers an ADC conversion of the current sense channel for the buck converter every SW_CTL_FREQ_DIV_FACTOR times the handler runs, allowing a higher switching loop frequency.
- **Thread_Fault_Injector** periodically injects a fault into the system after updating the LCD with the fault test number and raising a debug signal (DBG_FAULT). This debug signal is helpful for triggering your scope/logic analyzer.

## GRAPHICAL CURRENT DISPLAY

Besides performing closed loop control, the function Control_HBLED also gathers current data to be plotted later on the LCD. Each time the function runs it updates two circular buffers with the latest setpoint current $I_{LED\ setpoint}$ ((g_set_sample[]) and the latest measured current $I_{LED}$ (g_meas_sample[]).

Thread_Update_Screen calls UI_Draw_Screen, which calls UI_Draw_Scope to plot the data from the circular buffers. The plot shows 40 ms of data and is synchronized to start of the flash. This is equivalent to triggering an oscilloscope on the rising edge of the flash signal.

## ADC USE AND ADC ISR DESIGN

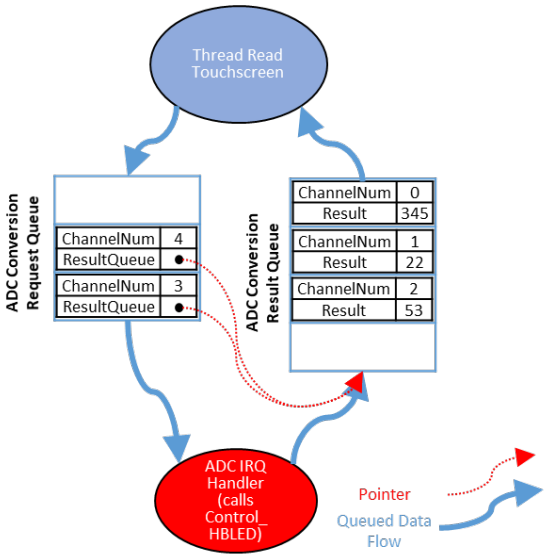



**Figure 2. Diagram of queues, data structures and interconnection. The queue sizes shown are examples.**

The ADC needs to be used by both the HBLED controller and the touchscreen controller thread. The system uses a software-implemented ADC server which prioritizes conversions for the HBLED controller over other conversion requests. There are two types of ADC conversion. Conversions for the buck converter are time-critical, so they are triggered with software in the TPM0 ISR. Conversions for the queued requests are triggered by software in the ADC ISR. A **static** state variable identifies the type of conversion. The thread and ADC handler in the ISR communicate as shown in Figure 3 and described below.

### ADC CONVERSION REQUESTS

A client thread (e.g. Thread_Read_TS) sends a request message to the ADC ISR through a queue (the request queue). Each message uses a data structure that holds the following information:

- ChannelNum: Number of channel to convert
- ResultQueue: Pointer to queue to hold conversion result. This field allows the system to be extended so that other threads can request A/D conversions and get their results correctly.

### ADC CONVERSION RESULTS

The ADC ISR sends result messages back to the client thread through another queue (the result queue). Each message uses a data structure that holds the following information:

- Result: Value of ADC conversion result
- ChannelNum: Number of channel converted

## REQUIREMENTS

Modify the program to manage the faults injected by Thread_Fault_Injector. Depending on the type of fault, this may include one or more of the following methods:

- Redesign the system so the fault has no impact on the system operation
- Add code to detect the fault (e.g. detect bad hash value)
- Add code to continue running and recover from the fault (e.g. use old value of data)
- Add a mechanism to restart the system (e.g. the watchdog timer)

Refer to the lecture notes, videos and assigned readings for more information.

## MODIFICATIONS TO FAULT.C

You will modify fault.c to select the faults you wish to test by changing the definition of the array Fault_Tests. You can temporarily disable tests to simplify development. Do not make other changes to fault.c or fault.h without first confirming with the instructor or TAs.

## EXTRA CREDIT OPPORTUNITIES

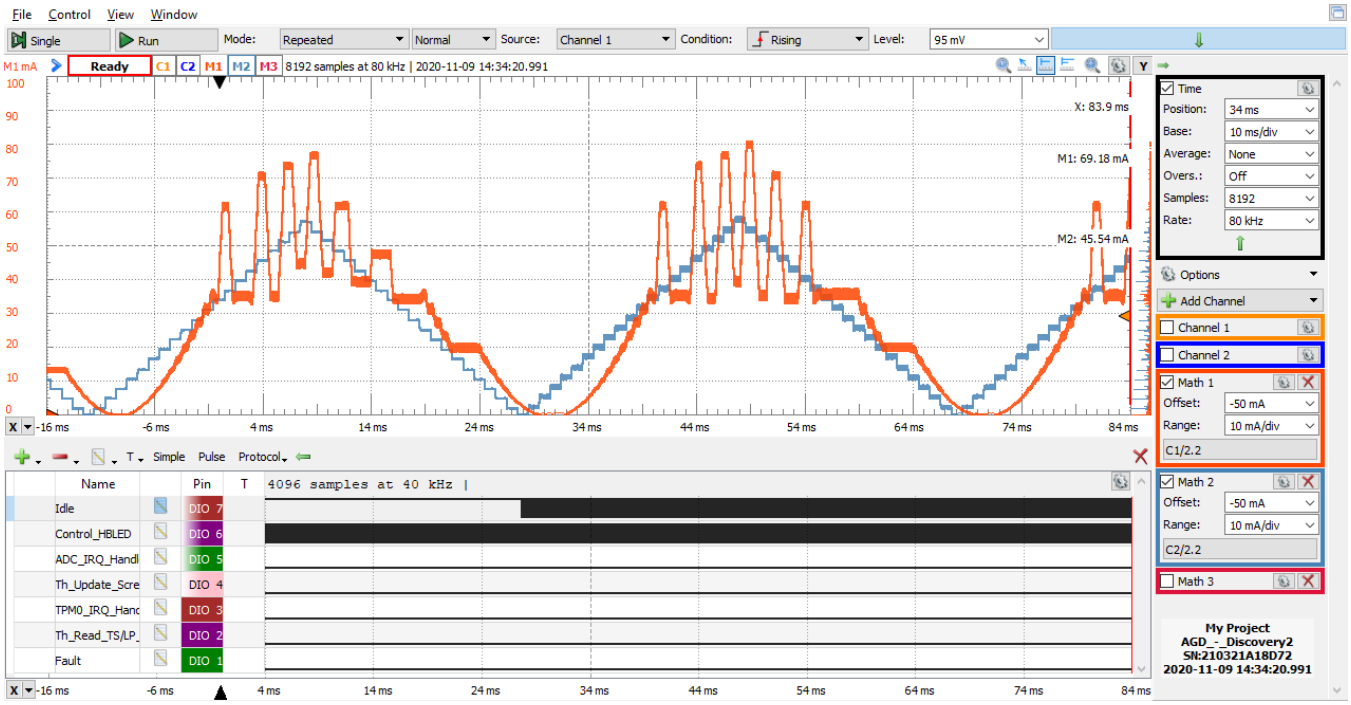
- Students can get extra credit by handling additional faults.
- Add a fault log in RAM (as a circular buffer) which indicates the faults which were detected. Your report must explain how the mechanism works and how to use it.

## REPORT

Submit a PDF report containing the following:

- For each fault test,
    - Fault Analysis
        - Explain how the fault (when not managed) affects the system.
    - Fault Management Approach
        - Explain your fault management approach,
        - Provide details of how you changed the system to manage the fault.
    - Evaluation of Effectiveness
        - Evaluate the time taken to detect and manage the fault.
        - Explain the impact (if any) of the managed fault on the system.
        - If you restart the system, evaluate the time taken to restart it (from fault to normal operation).
    - **EVIDENCE: Provide scope/logic analyzer screenshots and diagrams (as needed) to support your case. Include any debug signals needed to show that the system is working as you explain in your text.**
    - **PAGE LIMIT: Use no more than one page in the report per fault test. Use a 10 point font for your text. Choose your images and words carefully to convey the information concisely.**
- Explanation of what you did (if anything) for extra credit.

All screenshots must be legible and support the explanatory text. For example, the image below does not support the claim that Control_HBLED runs at 96 kHz. Since the logic analyzer data is gathered with a 40 kHz sampling rate, the Control_HBLED signal is undersampled and only provides limited frequency information (e.g. >= 80 kHz).




# OTHER

## GENERAL

1. You must work individually on this project.
2. Your code may be examined for possible plagiarism by using MOSS (http://theory.stanford.edu/~aiken/moss/).

## DELIVERABLES

Submit the following online:

- Project report (PDF)
- Archive of your project directory and subdirectories (except Listings and Objects subdirectories).

## GRADING

The following factors will be used to determine your grade.

- Functionality.
- Report quality.
- Code quality. The instructors may deduct points for remarkably bad coding practices (for example functions longer than a page, magic numbers, non-descriptive names). Various coding practice recommendations are described online at http://users.ece.cmu.edu/~eno/coding/CCodingStandard.html. If you have a question about what is acceptable, please post it on the class discussion forum so others can learn as well.