Energy-Efficiency Embedded Stove Monitoring System

Team Stove
Yash Patel and Leah Jones

https://github.com/YashP007/EC535-FinalProject

Abstract

The project, TeamStove, is an energy-efficient embedded temperature monitoring and control system built with a BeagleBone Black. This system uses a custom thermistor-based Analog Front End and comparator circuitry. This ensures real-time, interrupt-driven temperature sensing for immediate response. The system architecture separates hardware monitoring (kernel-space interrupts) from complex application logic and simulation (user space). The interface is an LCD screen that imitates a stove control from a mouse input and a temperature slider. This interface provides alerts, including a red/green indicator, a change of colors indicating rise in temperature, and a simulated 911 emergency alert at the 450° F threshold. There is also an Adafruit IO WebServer and Dashboard, which mirror this visualization, and allows for text message alerts when the system is connected to WiFi. The integration of the hardware, kernel-level interrupt handler, user-space LCD, and the Adafruit platform demonstrates a solution for both local control and remote control.

Introduction

This project focuses on building an energy-efficient embedded system for monitoring and responding to dangerous temperature levels near a household stove. Cooking accounts for 44 percent of home fires, and many incidents happen because people walk away from the stove or fail to notice that heat levels are rising past a safe threshold. Our goal was to design a stand-alone device that sits near the stove, senses temperature in real time, and reacts immediately when conditions shift toward a hazardous state. The motivation comes from the need for low-power, reliable monitoring that remains active even when the user is distracted or away from the kitchen.

The project brings together several components that work at different layers of the system. At the hardware level, a thermistor-based analog front end and comparator detect threshold crossings without requiring the processor to poll constantly. This allows the system to remain energy-efficient while still reacting quickly to fast temperature changes. In the kernel space, we implemented interrupt service routines and timing logic that run with high priority and handle events as soon as the comparator triggers. In the user space, we built a graphical interface that simulates a stove using an LCD display, shows the live temperature and system state, and provides visual alerts when the temperature exceeds the safe range. The device can also connect to an Adafruit IO dashboard to mirror the display online and send text-message warnings if the temperature reaches an emergency level.

Together, these components form a complete pipeline: hardware detection, kernel-level response, user-space visualization, and optional network alerts. This structure lets the system combine local, immediate safety behavior with remote monitoring. In our results, we demonstrated real-time interrupt-based detection of a temperature threshold, stable communication between kernel modules and the user-space interface, and a functional LCD control panel that simulates

heating, cooling, and emergency states. When connected to WiFi, the system also mirrored the stove status and sent mobile alerts through Adafruit IO.

Design Flow

The TeamStove system integrates hardware, kernel drivers, and user-space applications to achieve real-time monitoring and control. The architecture of the system is divided to place time-critical, hardware-based functions in the kernel space, while putting visualization, control logic, and network communication in the user space.

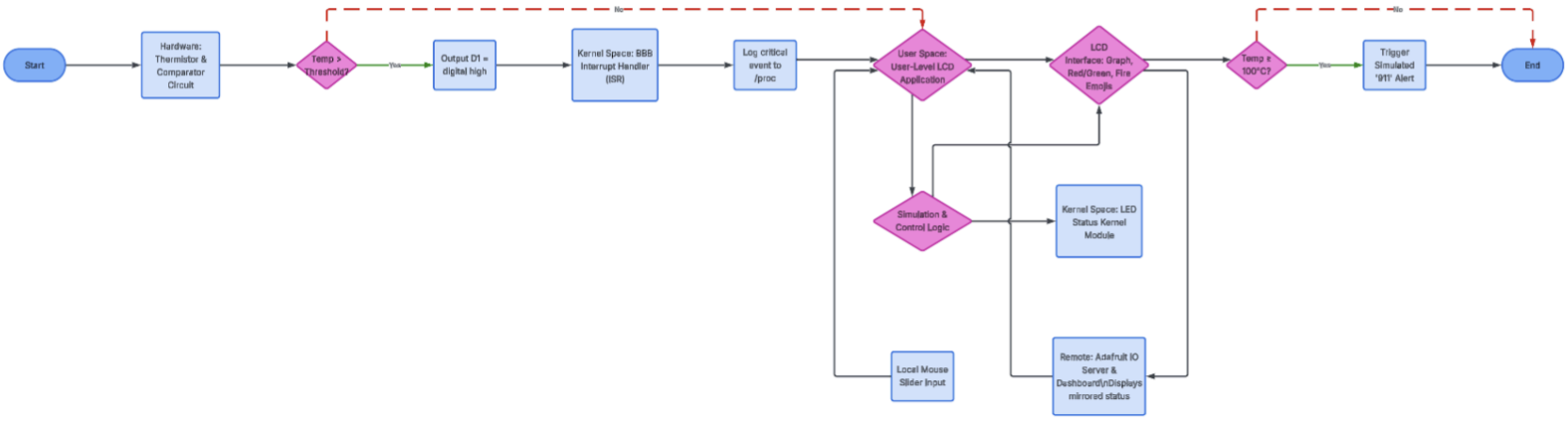


Figure 1: System Logic Diagram

To implement the outlined system logic (Figure 1), we implemented the following components:

1. **Analog Front End and Comparator Circuit:** A thermistor and voltage divider feed a comparator circuit, which sets a trigger to digital low when the threshold is hit. This signal is wired to the GPIO pin that generates a hardware interrupt (IQR).
2. **BeagleBone Kernel Modules:** The BeagleBone runs the given kernel modules that handle low-level, time-based, and interrupt-based tasks. This includes the interrupt service routing and the control logic for the status LEDs.
3. **User Space Application:** This program drives the LCD screen, receives mouse input, graphical elements, an emergency alert, and runs the stove simulation logic. This includes using +5 and -3 heating/cooling values to simulate a real stove heating up and cooling down speeds.
4. **Web Server:** This user-space application, when connected to WiFi, establishes a connection with the Adafruit server to show the current temperature and status data. Additionally, it sends an alert to the user's phone via a text message to alert the user that a certain threshold has been reached.

Component/ Task	**Description**	**Who**
Hardware Design and Soldering	Designed the thermistor divider and comparator circuit and soldered the components	Yash Patel
Kernel Module and UserSpace Module	Wrote the kernel driver to register the GPIO pins as interrupt sources and implement the ISRs	Yash Patel
Simulation Code and Logic	The simulation logic of the stovetop and network/server integration	Both
LCD Driver, Interface, and Alert Logic	Developed the code to talk to the LCD screen, wrote logic for visual alerts (fire emoji, 911 alert), implemented the GUI, and processed digital input (mouse) to toggle the slider.	Leah Jones
Adafruit Webserver and Text Messaging	Set up the Adafruit Web Server and dashboard for remote access, and implemented server logic to trigger text-message alerts when temperatures reach dangerous levels.	Leah Jones
Debugging and Validation (Code)	Debugging across all code (QT Image, Kernel Module, User space code)	Both
Debugging and Validation (LCD)	Debugging the QTImage and applying it to the BeagleBone	Both
Debugging and Validation (Hardware)	Connecting to the BeagleBone and the circuitry on the perfboard	Both

Member Contributions: Yash Patel: 50% and Leah Jones: 50%

Project Details

Analog Front End and Comparator Circuitry

Using an interrupt-driven design is more reliable for critical monitoring than relying on polling. Polling forces the processor to wake up on a fixed schedule to check the sensor, which wastes energy and risks missing fast events between checks. In a battery-powered wireless system, this constant activity shortens runtime and adds latency. An interrupt only wakes the processor when the hardware detects a real threshold crossing, so the system reacts immediately while keeping power consumption low. The analog front end handles this detection in hardware. A thermistor with a negative temperature coefficient sits in a voltage divider, so its resistance drops as temperature rises and the divider output falls (Figure 2). This output feeds one input of a comparator, while the other input holds a fixed reference voltage (set by a potentiometer). As long as the divider voltage is above the reference, the comparator output stays high. Once the temperature increases enough that the divider voltage dips below the reference, the comparator switches low. (See tempCircuit.pdf for full schematic) That transition directly triggers an interrupt, allowing the system to respond right away without burning energy on continuous software checks.

$$V_{thermistor} = V_{CC} * \frac{R_{thermistor}}{R_{thermistor}+R_{div}}$$

Figure 2: Thermistor Voltage Output Transfer Function

Kernel Space

The kernel modules handle the low-level interaction and timing due to the safety aspect that we are trying to prioritize. All the kernel code was made in VSCode using C, and we made and compiled files on the lab computer.

Interrupt Handling

The kernel module configures the BeagleBone GPIO pin as an interrupt source. This was implemented through a kernel module (see km/myTempSensor_comp.c). The following occurs when the IQR is triggered:

1. The ISR executes with the highest system priority.
2. It applies the logic above (Figure 1) and has a debouncer to filter the false positives before logging the alert
3. The module updates a status file in the /sys filesystem, where user-space monitors trigger alerts, which helps reduce latency.

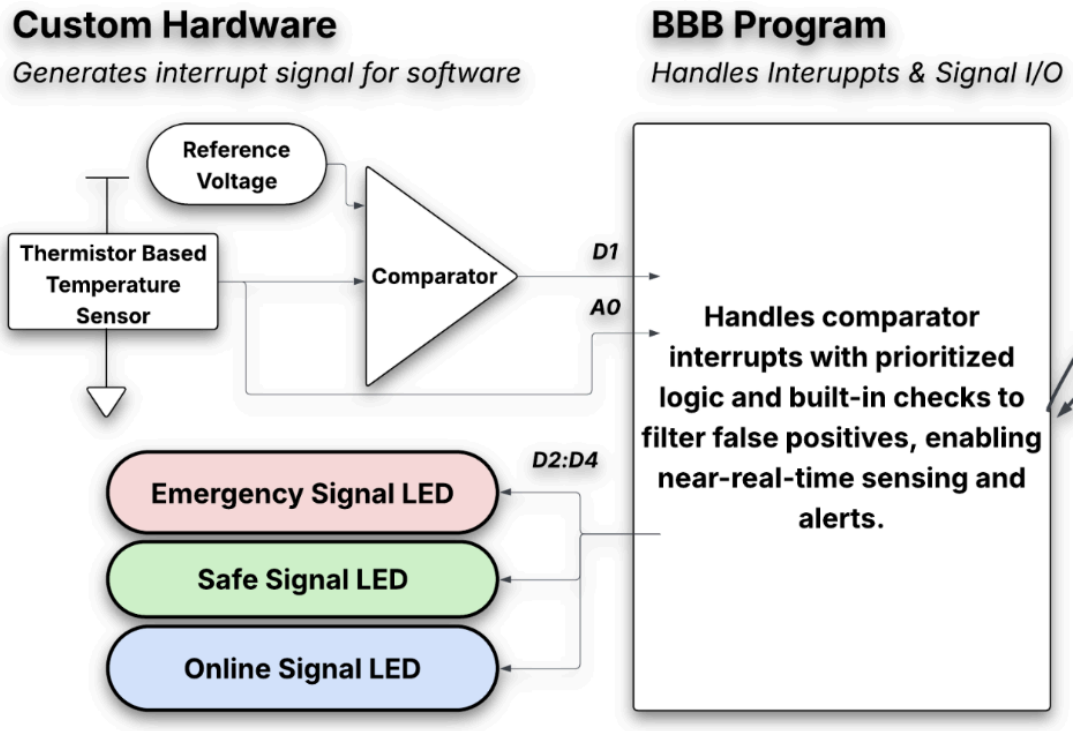
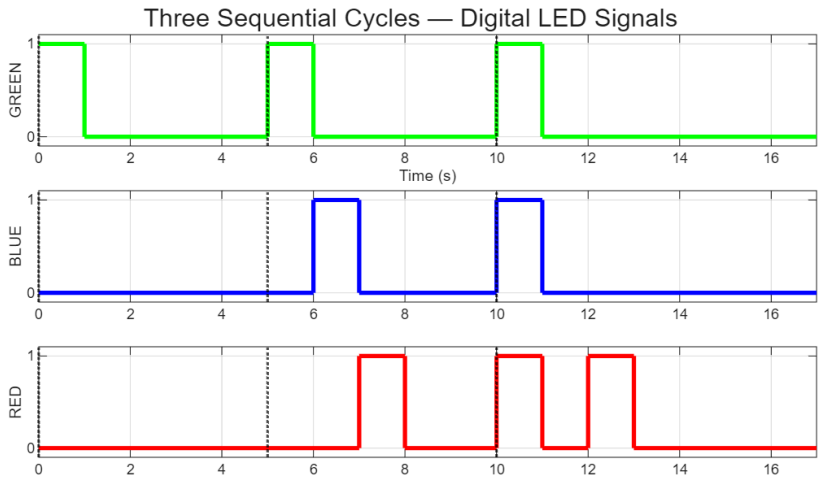


Figure 3: System Architecture Outlining the Hardware-Software Co-Design

LED Status Module

This module is responsible for the control of the LEDs to show the status of our system. This program is part of the safe-fallback behavior considerations. You can imagine if the system were to lose internet connection or be unable to power the LCD screen, then emergency high-temperature alerts may go unnoticed, which can be a major failure point of the system. Which is why we implemented an on-board RGB leg to inform the user of the server connection status, stove on/off status, and the temperature level status. The timing diagram (Figure 4) and the accompanying table outline how this LED conveys the information to the user.

The implementation of the LED control system uses a kernel module (see km/mySignalLED.c). The module uses GPIO to show three distinct cycles. The timing for the pulse widths is controlled in the kernel space, and the parameters can be adjusted based on the user.



	Server Status	Stove Status	Temperature Status
Cycle 1	Connected	Off	Safe
Cycle 2	Connected	On	High Temp.
Cycle 3 $\tau_{duty} \rightarrow 4$ *sec*	Disconnected	Off	High Temp.

Figure 4: LED Signal Timing Diagram & State Table

User-Space Application

The userspace application integrates all the system functions, handles the visualization, and adds the network communication. We had the application drive the LCD screen using the QT Framework and the QT Creator IDE. The graphical interface uses a real-time temperature graph, visual status indicators, and a control slider. The mouse input is captured from the Linux event device interface, and it processes those coordinates to map the mouse movement to the control slider, which allows the user to set the stove's temperature. We also included a software (adjustable) and hardware limit that notifies the user in real time when a dangerous temperature level has been reached. The software limit helps simulate our web interface to give the user ease of use when setting their limit if they are currently cooking and would like to adjust their temperature level. We had to expand our design and allow the user to pick their option of choice between a Web Server vs an LCD screen. We pivoted to this LCD design to simulate a Stove, as we did not have the correct Kernel image with its compiler with WiFi, ADC Input, dmesg, and zmodem file transfer protocol capabilities. Given this, we had to prioritize using an image with the necessary features: dmesg and file transfer capabilities. (See Figures 5 and 6)

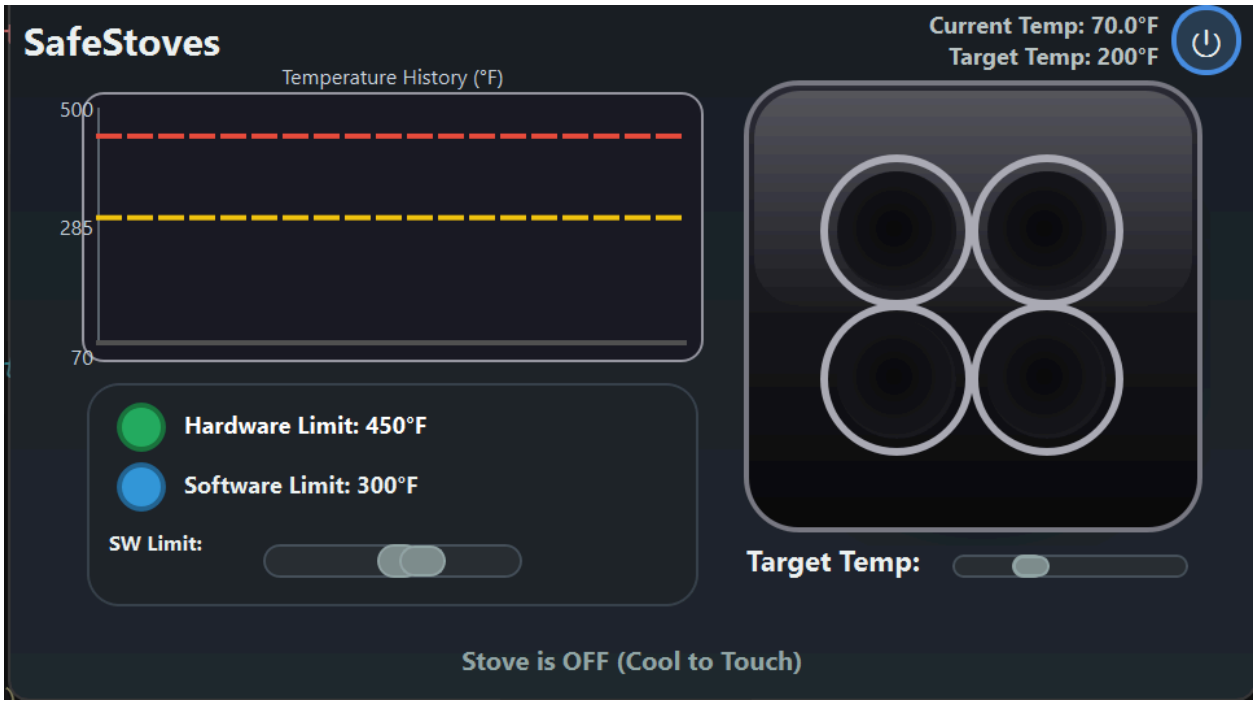


Figure 5: Showing the QT Application when it is turned off

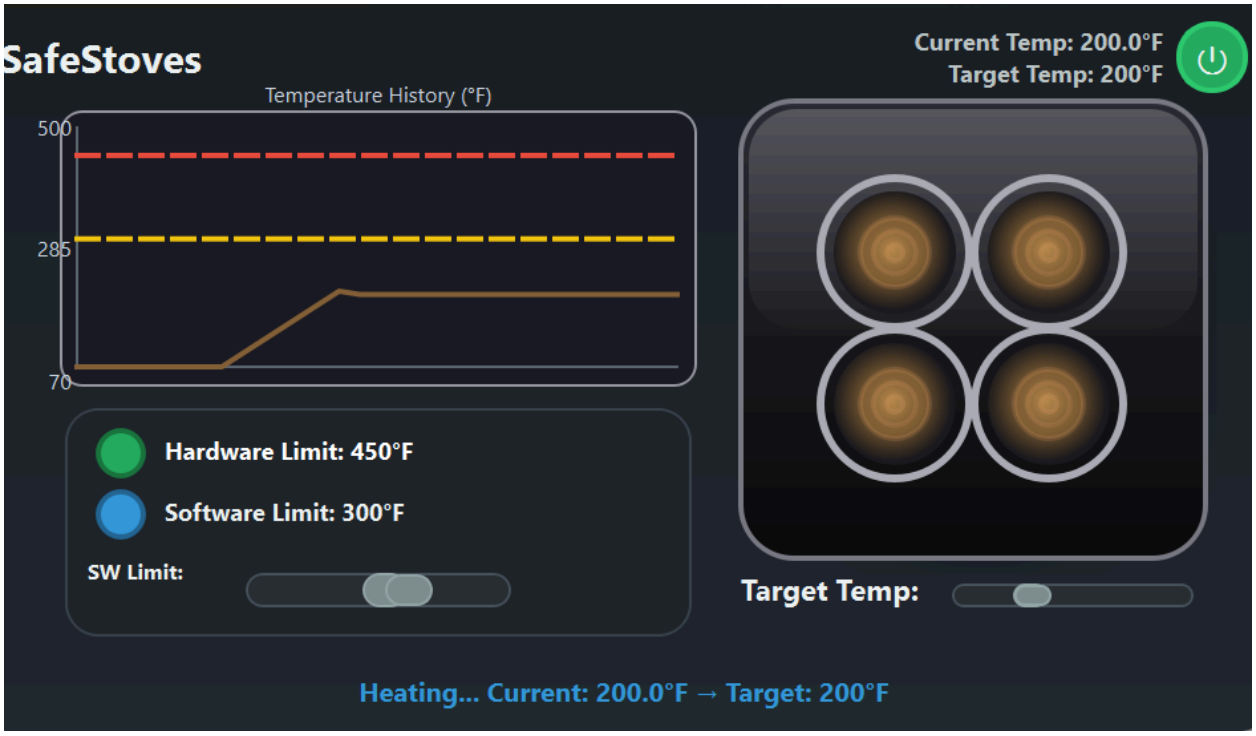


Figure 6: Showing the QT Application when it is turned on

Remote Monitoring and Network Integration

The system uses Adafruit IO for wireless monitoring and remote control, used in VS Code. The userspace application uses the MQTT protocol to publish the status and sensor data of the Adafruit server. The Web Dashboard is configured to subscribe to the feeds, display the temperature graph, status indicators, and allow for remote control from a slider that transfers back to the BeagleBone application. In addition, to alert the user of a critical function, it was configured to send a message to tell the user that a dangerous temperature has been reached. We had to pivot to make this a user-optional addition. If the user has WiFi enabled on their Beagle Bone, it can be seamlessly connected to the server and allow for alerts and a dashboard that can be seen on a computer or phone at the touch of a button.

Figure 8: Demonstrating the Visual Interface of the WIFI aspect of the program

Applications and Software Used

Category	Tool/Language/Configuration	The purpose
Development	VSCode	Cross compilation, debugging, and managing kernel module source code
GUI	QT Creator and QT Framework	development, testing, and deployment of the LCD screen's UI
Languages	C,C++,Bash	C for the kernel modules, C++ for the QT GUIs, and Bash for the makefiles
Networking	Adafruit IO Web Server and MQTT Protocol	Text communication and Dashboard hosting
Interface	Linux Event Device	Interpreting mouse input for the LCD slider and on/off

Summary

Studies show that 44% of home fires are caused by cooking. TeamStove built a real-time monitoring and control system for a simulated stove using a BeagleBone Black and custom kernel modules to address this problem. A major goal was to demonstrate our understanding of Linux device drivers, multithreading, and communication between the user space and the kernel. The system uses two kernel modules: a hardware comparator/timer module (myTempSensor_comp.ko) and an LED signaling module (mySignalLED.ko). Together, they detect temperature thresholds through a hardware interrupt, manage periodic status updates through a kernel timer, enable user control of the simulated stove, and report system state at defined intervals.

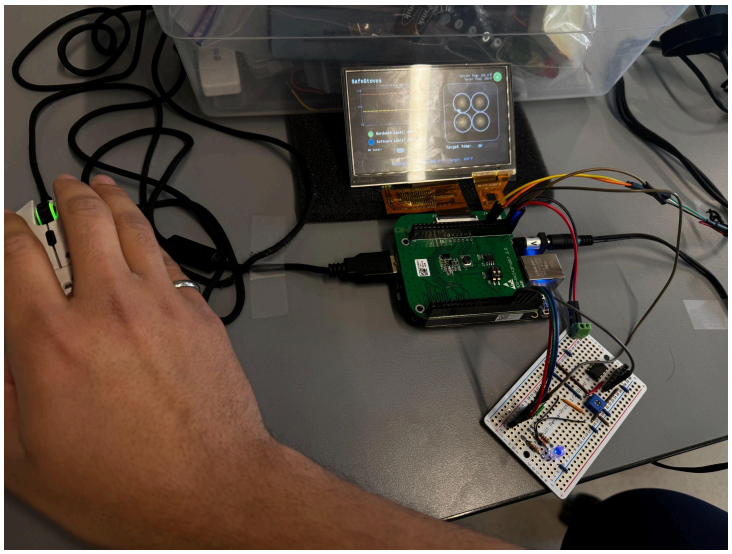


Figure 9: Image of the full system assembled for the demo day presentations

We were able to apply several key embedded systems concepts from this semester:

- **Interrupt-driven logic:** Used the hardware comparator to trigger immediate kernel-level responses without polling.
  Thread safety: Used a mutex to coordinate shared state across the monitoring thread, input thread, and server thread.
- **Dynamic kernel control:** Adjusted kernel timer behavior at runtime through writes to the module's device file.
- **Resource-aware design:** Balanced responsiveness, power use, and reliability through careful separation of roles between the AFE, kernel, and application layers.

Together, these choices let us build a working system that reacts quickly to real-time conditions, stays stable under load, and fits the constraints of the application. The project reinforced how to combine many embedded principles into a coherent design that behaves predictably in a safety-critical scenario.

References

[1] Jonathan Cobert, Linux Device Drivers, Third Edition, O'Reilly, 2005.
[2] BeagleBone QT Walkthrough, https://youtu.be/yNvOyY9zK1o?si=mdk2XOAwh8oS0Zt1
[3] Stack Overflow, https://stackoverflow.com/questions
[4] QT Documentation: https://doc.qt.io/qt-6/designer-using-a-ui-file.html
[5] QT Framework: https://www.reddit.com/r/QtFramework/
[6] Guillaume Lazar, Mastering QT 5, 2nd Edition, 2018
[7] The Userspace, https://www.kernel.org/doc/html/v5.0/driver-api/uio-howto.html
[8] Fire loss in the United States, https://www.nfpa.org/education-and-research/research/nfpa-research/fire-statistical-reports/fire-loss-in-the-united-states
[9] Lab 1-5, https://learn.bu.edu/ultra/courses/_265063_1/outline