Kye Conway

kjconway@ucsc.edu

ECE-13, Fall 2023

Collaborators: I didn't collaborate with anyone on this lab

Summary: In this lab, we used finite state machines and timer interrupts to create the functionality of a toaster oven on our PIC 320x microcontroller. We implemented the three oven functions, those being bake, broil and toast. In bake mode, we allow user input from the potentiometer to change the time and temperature of the oven. In broil we allow for the timer to be adjusted using the potentiometer and keep the temperature constant at 500. And in toast, we allow for the timer to be changed with the potentiometer but we do not print the temperature. Using a finite state machine, we implemented the ability to switch between the cooking modes with the press of button 3. We implemented the ability to switch between changing the time and temperature in bake but storing the button press time and checking if it was longer than our threshold, 1 second. We also implemented the ability to start the cooking feature with its special characters from ASCII.h and the ability to reset the oven if we start the cooking process and want to reset to the setup mode. As extra credit, ew implemented the ability for the OLED to alternate between inverted and not inverted after the timer has run out while in the cooking mode. The key concepts in this lab to me were event driven programming, finite state machines and reading the finite state machine diagrams, and using interrupts to cause our finite state machine to start certain processes.

Approach: I approached the lab by first taking a significant amount of time to read the lab manual and get a basic understanding of what the requirements were and how it was expected to work. After reading through the manual, I decided to attack the lab in the order given to us at the end of the manual. I started first with the updateOvenOLED() function. After testing that it worked, I moved to the finite state machine. At first it was really confusing but then after I began to understand how the diagram was supposed to help, it made a lot more sense. I finished the finite state machine in this order: setup, selector change pending, cooking, reset pending. After working my way through the finite state machine, I finished my main code, making sure that I had the base of the coding done and my project worked without the extra credit. I then tackled the extra credit, which went reasonably fast since by that time I had a good grasp on my code and how I thought it should be implemented. One of the first bugs I ran into was with using the LEDS_SET() function. I spent a decent amount of time trying to figure out why my call to it wouldn't compile until I decided to check on the Leds.h file. I then realized that I had to implement the functions for them to work properly. That was a quick fix given that we've already created our own versions of this file. Another bug I had was that my oven would freeze if

I held down button three while in broil or toast. This is because I didn't give any code for what happens if a selector change happens while in any other mode besides bake. I fixed it with this code:

```
if ((buttonTimer - ovendata.buttonDownTime) >= LONG_PRESS) {
    // only have to do a selector change if cookMode = BAKE si
    // temp for toast and broil
    if (ovendata.cookMode == BAKE) {
        if (ovendata.selector == TEMP) {
            ovendata.selector = TIMER;
            // for testing
            // printf("ovendata.selector = TIMER\n");
        } else {
            ovendata.selector = TEMP;
            // for testing
            // printf("ovendata.selector = TEMP\n");
        }
        updateOvenOLED(ovendata);
        ovendata.state = SETUP;
    } else {
        ovendata.selector = TIMER;
        updateOvenOLED(ovendata);
        ovendata.state = SETUP;
    }
```

I made it so that if button 3 pressed down time was greater than a second and we weren't in bake mode, then the ovendata.selector is always set to timer, effectively removing the bug. Another bug my code had a I tried to fix was that my ADC value still flickers every once and a while. To try and fix this I tried to implement a function that basically takes the average of the ADC values and makes that the new ADC value. This stopped the flickering but would give me random end values and wouldn't get to the appropriate maximum and minimum ADC values required by the lab manual. This is the function:

```
// **** Put any helper functions here ****

uint8_t readAdc() {
    // Read the ADC value
    uint16_t currentAdcValue = AdcRead();

    // Store the current reading in the array
    adcReadings[adcReadingIndex] = currentAdcValue;

    // Move to the next index, circular buffer style
    adcReadingIndex = (adcReadingIndex + 1) % NUM_ADC_READINGS;

    // Calculate the average
    uint16_t sum = 0;
    for (int i = 0; i < NUM_ADC_READINGS; i++) {
        sum += adcReadings[i];
    }

    uint16_t averagedAdcValue = sum / NUM_ADC_READINGS;
}
```

What worked well for me was following the given instructions at the end of the manual and working on my FSM using the given diagram. I think that if you spent some time going over the manual then most of the logic for the lab was already laid out for you. I feel like I would approach it the same, I finished the coding early and was able to implement the extra credit. I didn't work with any other students on this one. I'm sure it would have made the lab easier though.

Results: The lab ended great for me. I'm pretty sure I got down all of the functionality of the oven and was even able to get the extra credit to work. I would say I probably spent around 15-20 hours on this lab, between all of the coding, reading and debugging. I liked how much more time we had on this lab. I felt like I was able to breathe a little and try to learn the concepts rather than just trying to make it work for this one time. I feel much more confident about the concepts from this lab than I have from past ones. I honestly didn't dislike anything about this lab. I enjoyed the whole process and truly enjoyed learning how to make this thing work. I think that this lab has been my favorite one yet and honestly wouldn't change it. The hardest part to me was starting and really fleshing out the general idea and direction the code was heading. Once I got the general idea of the direction and was able to understand the FSM diagram, the rest flowed pretty easily. The point distribution does seem fair and is weighed accurately in my opinion. The lab manual had more than enough information to start and work through the lab. I feel like what

helped me the most is that not much of this lab was new, but we were introduced to new ways to use past content and new ways to approach problems like this one.

Extra credit Implementation: For the extra credit I used a while loop to iterate over my invert code until my desired number of inverts were matched. I created an invertFlag uint8_t and set it equal to false. The logic is pretty simple:

```
case EXTRA_CREDIT:
    if (TIMER_TICK) {
        while (1) {
            // Check if it's time to toggle inversion
            // printf("%d\n", counter);

            if (!invertFlag) {
                invertFlag = TRUE;
                OledSetDisplayInverted();
            } else {
                invertFlag = FALSE;
                OledSetDisplayNormal();
            }
            OledUpdate();

            // add a delay before re-entering the loop
            for (int delay = 0; delay < 1000000; delay++) {
            }

            // Increment the break counter
            breakCounter++;

            // Check if we've reached the desired number of iterations
            if (breakCounter == invertNumber) {
                // printf("%d\n", breakCounter);
                breakCounter = 0;
                break;
            }
        }
    }
    ovendata.state = SETUP;
    break;
```

If my timer_tick flag is true (which it will be since it's coming from cooking and hasn't been reset yet) then if my invert flag is false (which it starts as false) then the screen gets inverted and I set my inverseFlag to true so upon my next iteration the other statement runs. The logic basically alternates between on and off and then uses a delay to keep the invert / regular screen for an arbitrary amount of time. To exit the loop, I have a defined variable (invertNumber) to hold the number of switches I want to occur. The number should be even to ensure it returns

back to normal on the last loop. Arbitrarily I have it set to 6. The breakCounter is increased with each loop until it eventually breaks and resets the state to setup.

Additional notes: I noticed that occasionally (about 1/20 runs), after running the first cook successfully and resetting, sometimes upon the next cook, the LEDS wouldn't tick down and would rather stay the same. I could/t figure out why this would happened and it usually doesn't.