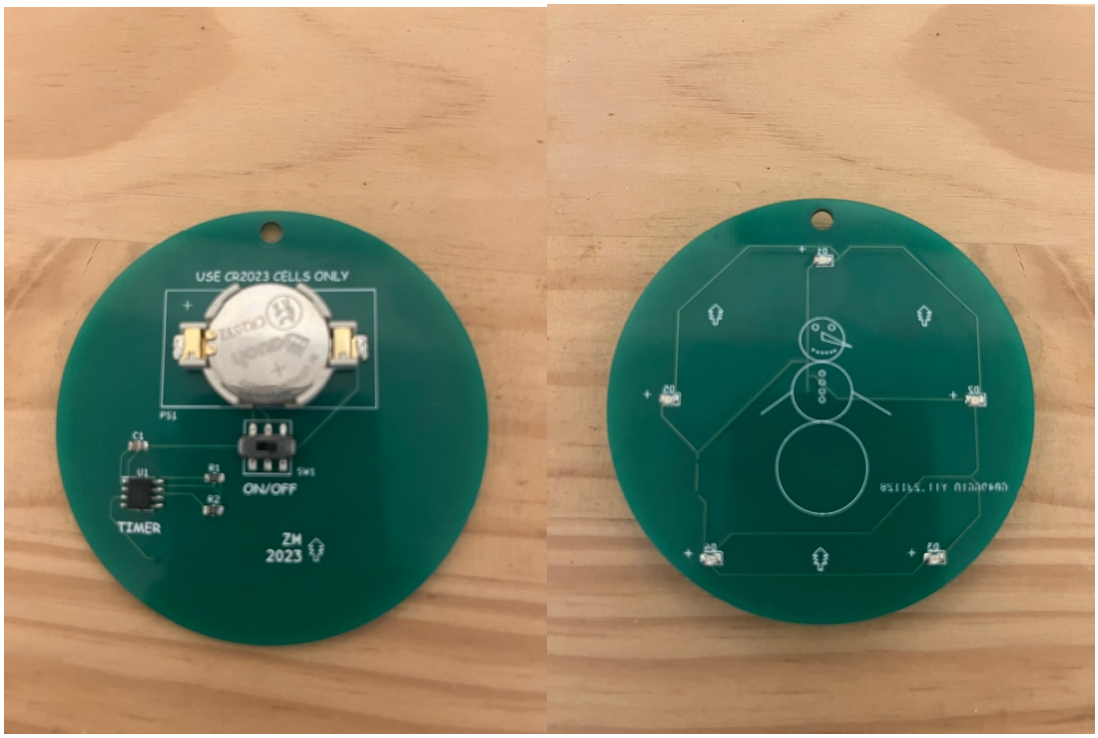
Electronic Tree Ornament with Configurable LED Patterns

Introduction

We first got a tree for the holidays in the winter of 2023. We're not religious or anything, but we thought it would be nice to have in our living room. After getting the tree set up in the living room, we found that we had very little to put on it. Besides some ornaments we baked (yes, in the oven) with applesauce and cinnamon, it was rather bare. I had an idea, however.

During my day job, I am a designer of embedded systems. I'm familiar with designing small form-factor electronics and their accompanying PCBs, so I thought: why not try to design an ornament for our tree? My first concepts for this involved a circular PCB with five red LEDs. The first revision was a simple on-off device where the LEDs would turn on when a switch was turned. The second was a simple 555 oscillator circuit that simply blinked the LEDs on and off when power was applied. Both concepts were powered by a 2032 battery cell. I ended up giving away a few of these to friends and they were seen as a fun curiosity.



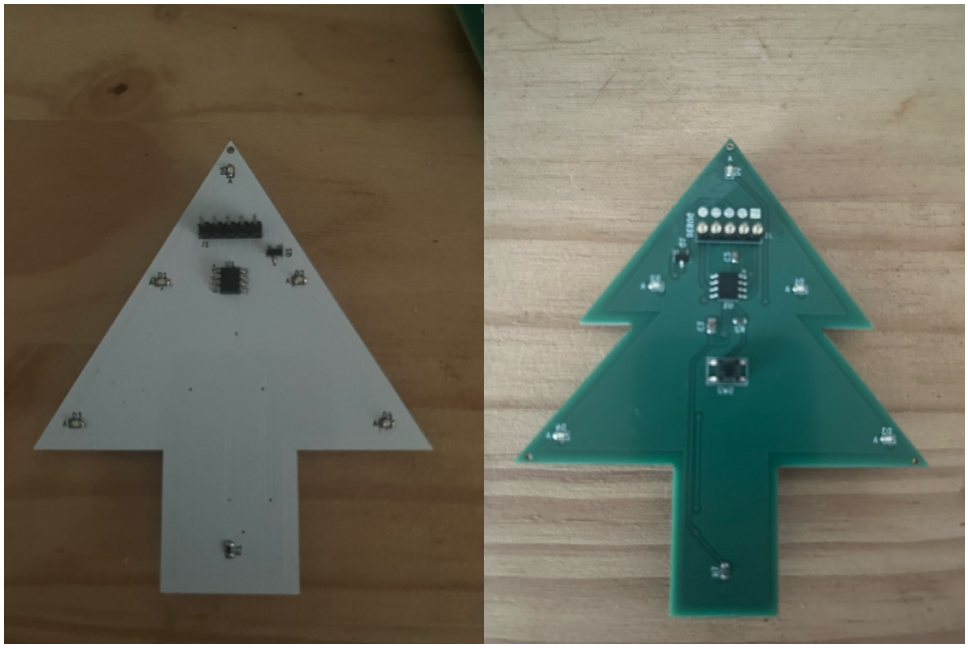
Back (left) and front (right) of the 555-based ornament

As I designed these PCBs with very little time until the holidays, they were rather simple and were not really that interesting; but I felt as they added a little bit of personality to our tree. I thought the next revision of this idea should be special and pack a lot more functionality into the design.

Board Shape

The original PCBs were circular and boring to look at., to be honest I decided to make them more festive this time around and use a more unique outline to really catch the eye. After some thought, I set my goal on making the ornament into a miniature Christmas tree of some sort: the outline of the board was to be the “tree”, while LEDs were placed at the vertices of the tree to be its “ornaments”, with a “star” at the top”.

My original prototype looked much too like a Windows cursor, so the next revision corrected that with a more tree-like shape (and a return to green solder mask).



First prototype (left) and second prototype (right)

Light Patterns

The original ornaments were designed with a single light pattern in mind: either a static LED state (on/off) or blinking LEDs. This is quite boring for those who feel like periodically changing the light pattern; even worse if you actively dislike the pattern that the ornament is designed to output. The first feature I thought to include was the ability to reconfigure the light pattern. On this new design, a button on the top side of the board allows a user to cycle through a set of light patterns.

For simplicity, I decided to tie the bottom four LEDs together to control them all as a single unit and make the top LED independent. I wasn’t going to have enough I/O available to control them individually, so I may as well split them up: the top LED was the “star” and should be controlled independently of the bottom four “ornaments”.

I came up with a list of new light patterns to implement:

- Static
    - In this mode, all LEDs are always on.
- Blink (all LEDs)
    - In this mode, all LEDs will blink on and off with a period of 2 seconds and a duty cycle of 50%.
- Blink (faster, alternate between top LED and bottom four).
    - In this mode, the top LED and bottom four LEDs will blink alternatively with a period of 1 second and duty cycle of 50%.
- Flicker (all LEDs)
    - In this mode, all LEDs "flicker" much like a candlelight would do.
- Flicker (top LED only, bottom four LEDs off)
    - In this mode, only the top LED will flicker, and the bottom four LEDs will turn off.
- Static (low power)
    - In this mode, all LEDs will be on, but the bottom four LEDs will be placed in a low power state in which they are at half-brightness.

While the original ornaments made use of red LEDs, I decided to switch to a warmer color more reminiscent of actual tree lights. I ended up selecting yellow LEDs for this task, as the color is a tad bit more inviting than white or red LEDs, while having a forward drop of only about 2V. I also thought yellow LEDs would contrast well with the green solder mask. In the end, I went with 0603 package yellow LEDs.

Timed operation

A friend of mine who received one of the original units had a complaint for me about the ornament he received. Most people don't want to leave their electronic devices (or lights) on all night, and it was a hassle for him to have to reach and turn off the PCB every night before bed. He asked if it would be possible to add a timed operation feature into the boards: that is, a feature that would make sure the boards would turn off their LEDs after four hours of runtime have passed and then automatically turn them back on twenty hours later.

I liked this idea and decided to add it into the requirements for my new board. Now I had two requirements: make an ornament that could change its light patterns as well as keep track of the current time.

Hardware Implementation

I had two hard electronic requirements to satisfy now:

- Timed operation (4 hours on, 20 hours off)
- Reconfigurable light patterns (solid, blinking, flickering, etc.)

I'm not a big fan of using microcontrollers in places where I don't think they belong. In fact, I'm someone who will try to use transistors and logic gates to get a job done if I don't believe I have to use a microcontroller. My task, however, was to design a PCB to meet some requirements that

I believe would have involved too many components without the use of a microcontroller. Cost is a big factor here, as I'm self-funding this project.

The microcontroller selection wasn't too difficult, as I already had a good idea of what I wanted. It needed to have on-board PWM hardware for the flicker pattern (not a fan of software PWM for this application after some brief prototyping), a small form factor (small enough to fit on a small PCB), the ability to periodically wake itself up from sleep mode, and the ability to run off a low-voltage source: think 2V-3V as we're using a 2032 cell. For simplicity, I decided to carry over the 2032 cell holder and switch I used for the original design.

I'm a big fan of the PIC microcontrollers and use them a lot in my hobbyist projects. They offer a great set of peripherals (especially low-power functions) for the price point, are cheap and readily available, and I already had a PICkit 3 on hand. After consulting the Microchip website, Digi-Key, and JLCPCB's part inventory, I settled on the PIC12F1822 family of microcontrollers.

Prototype boards using the PIC12F1822 microcontroller

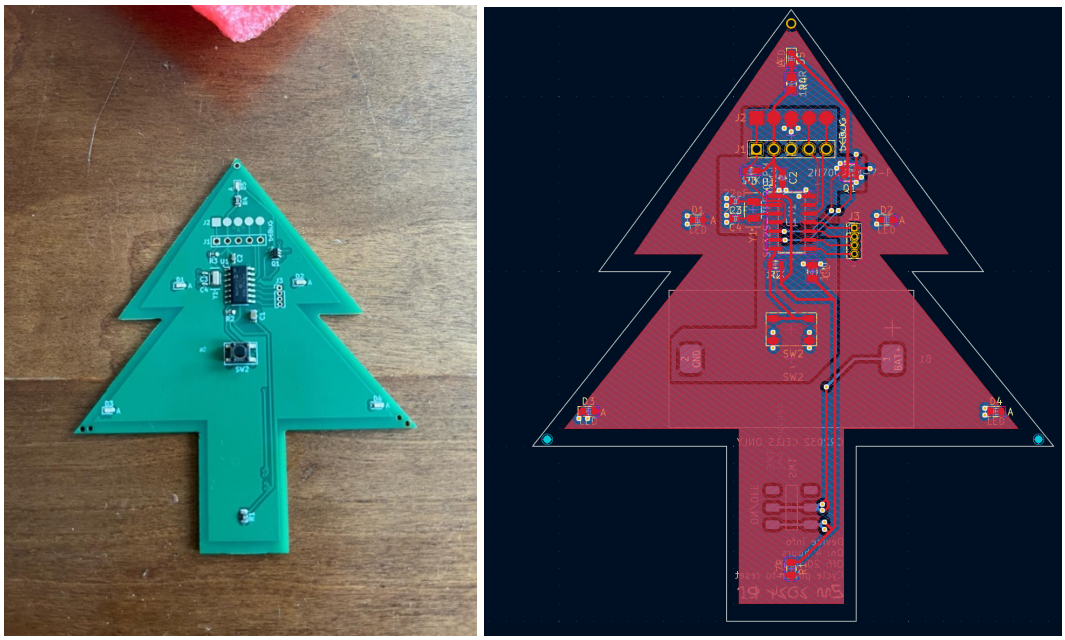
I designed a few prototypes and discovered a few things relating to the LED drive circuit. I'm not a fan of directly driving them from the microcontroller. I tried to minimize my BOM as much as possible by trying to drive the top LED with a digital pin, but I was finding that controlling the LED state required changing the LED to an input whenever I wanted to turn it off. I get that it's good practice to do that, but it made writing my code annoying. I decided to use a dual package 2N7002 FET to drive the LEDs for both the top LED and bottom four.

As this is a design centered around lighting LEDs, biasing the LEDs to allow for a good tradeoff between current draw and luminosity was important. I set to push ~5mA through each LED at maximum state of charge. As it was difficult to quantify the 2032 internal resistance (as it varies throughout discharge), I did my calculations while neglecting it (as well as neglecting the R(on) of the 2N7002 MOSFET). Assuming the maximum voltage of a 2032 cell is 3V and the forward drop of a yellow LED is 2V, I ended up with a 180-ohm resistor for the "star" LED and a 51-ohm resistor for the four "ornament" LEDs to share for a balanced 5mA current through all LEDs.

After working to develop firmware for the prototype boards, I decided to upgrade to the larger PIC16LF1823 in the same family, as it included more pins for me to work with. The on-board

watchdog module did not provide the precise timing I wanted and achieving exact timings with the on-board RC oscillator was very difficult. The PIC12F1822 family supported the use of an external 32.768 kHz watch crystal to drive Timer 1 while in sleep mode, so I decided to include one on the next revision to achieve the precise timing I defined in my requirements. However, I was already using pretty much all my I/O in the PIC12F1822, so an upgrade to the PIC16LF1823 was required to add more pins for this use (and switch to the LF, because it reduced power consumption by omitting the on-board voltage regulator).

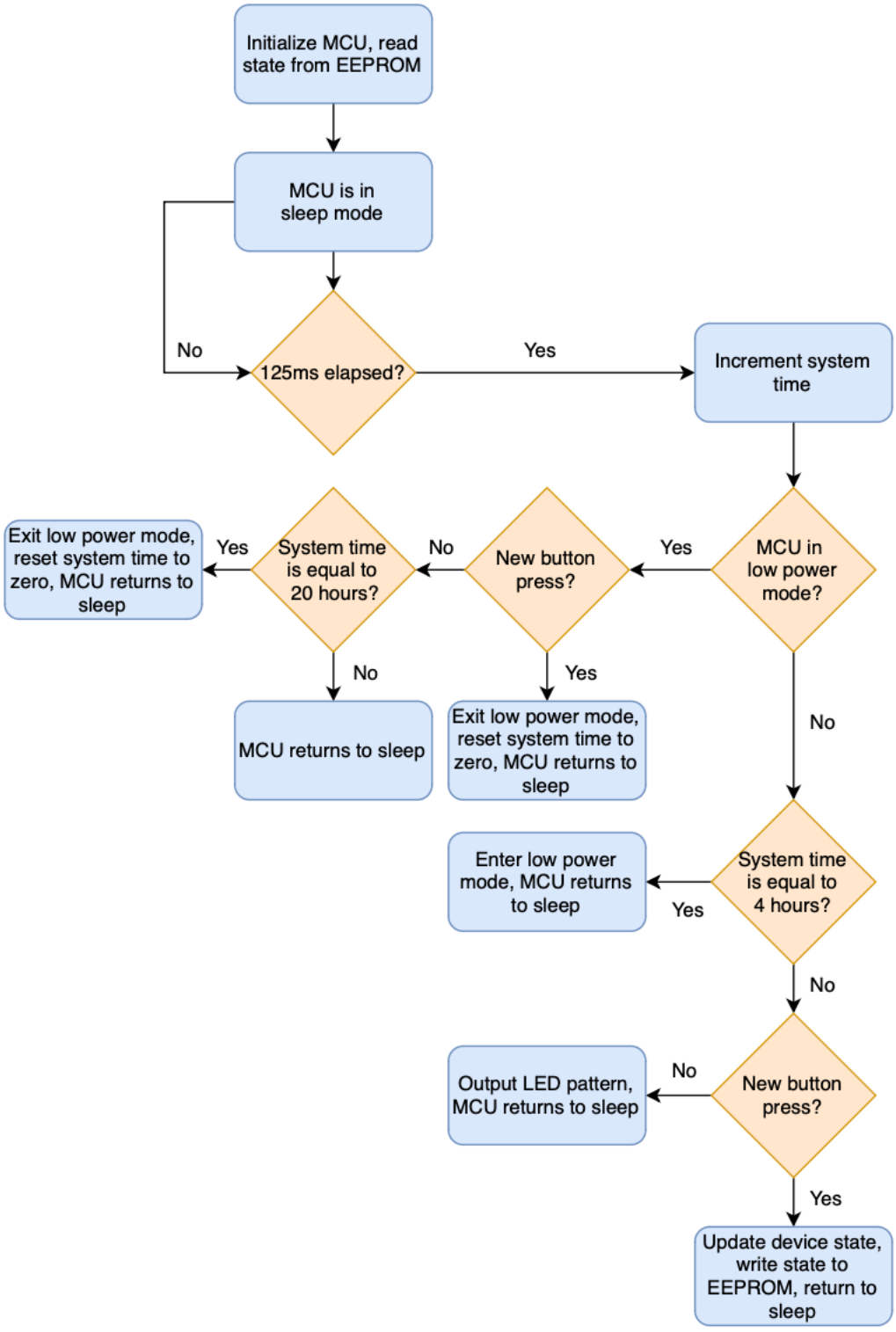
I was pretty satisfied with my final design. All board artwork was developed with KiCAD. Fabrication and assembly of these boards was performed by JLCPCB.



Final design of the ornament, assembled PCB (left) and KiCAD layout view (right)

Firmware Implementation

To speed up development, I wrote all the code for this ornament in C with the xc8 compiler. It's probably not ideal for code you want to run quickly, but I'm not exactly intending for this thing to be a speed demon. After some experimentation with the PWM output, I decided to run the MCU from the internal RC oscillator set to 250 kHz – output from the 32.768 kHz crystal was far too choppy.



Flowchart describing the ornament firmware

The first thing I set out to implement was a periodic interrupt that would wake the MCU from sleep and execute some code. To save power, I wanted the MCU to operate in a sleep state and

periodically wake up. I didn't want to do much during these wake periods. I only wanted to poll for button presses, increase the current time count, and update the LED light pattern. A 125 ms interrupt was determined to be the happy medium here, as its frequent enough to catch button presses as well as a nice divisor of 1 second so we can use it for both driving the LED blink functions and the timed operation function.

The individual LED patterns weren't difficult to bake into this interrupt, as they were mostly all time dependent in some way (save for the solid LED state). I used the current time count for the timed operation function to drive both the blink patterns and used the 125 ms interrupt to update the PWM period for the flicker functions.

I implemented a simple state machine to allow these button presses to change state. A very basic debouncing function was used to ensure that button presses were accurate. The button presses are designed to simply shift the board to the next LED state – you can say it's implemented as a Moore machine. Following a button press, the new state is updated and written to the on-board EEPROM. This is done to allow for the ornament to return to its last known state before power down – I thought that that you shouldn't have to reconfigure your ornament every time you use it, so why not use the EEPROM that's already in the chip? I've previously used the on-board PIC EEPROM for a similar task for a graduate school project and think it's probably the coolest thing about these devices.

The flicker pattern was implemented using the on-board PWM hardware using Timer 2 with a frequency of 250 kHz (system clock). This was achieved by driving the LEDs with a PWM signal with a "random" period. You might think that I implemented some kind of cool software PRNG with a LFSR or some environmental noise-based RNG. I would have loved to do that and have the flicker period be generated in real time, but I decided to generate a random array of 256 10-bit numbers (the PWM has a 10-bit resolution) in Python (using randint()) and program them into the MCU's flash memory. This was the simplest option that allowed for a flicker, did not require a complicated algorithm, or did not need additional hardware to be added to the BOM.

The last thing on the list to accomplish was implementing the timed operation feature. This feature, as previously described, would shut off the LEDs after four hours of continuous runtime and resume operation after twenty hours have passed. The system time is kept in increments of 125 ms "ticks". The device would essentially operate in a 24-hour cycle in which it was on for four hours and off for twenty hours. After four hours of ticks, the current state is saved in memory and the device is then placed into a low-power state in which all peripherals are off and the LED outputs are changed to input pins, preventing the FETs from operating and driving the LEDs. After twenty hours, the previous device state is restored, and the device begins functioning normally again. A button press will end this low power state and reset the system time count to 0 if it is pressed while the device is in low-power mode.

A minor curiosity was discovered during development of the timed operation function. The interrupt is designed to fire every 125 ms; however, experimenting with an oscilloscope shows that this would occur about 5 instruction cycles late (assuming an instruction cycle is 1/(250kHz/4) = 1.6 us). While this may seem small, remember that this error can propagate because we are operating at an hour scale; this bug was first found because low-power mode was

not being entered right at the four hour mark. After studying the assembly code generated by xc8, it made sense: performing the 16-bit addition required for updating the 16-bit Timer 1 register was a 5-instruction cycle operation and so the timer would overflow about 5 cycles too late, as the Timer 1 preload value does not account for the time taken to load the value into the register. To correct this bug, I simply added 5 to the Timer 1 register on top of the 61440 preload value, resulting in a very clean 125 ms interrupt.

Thoughts

I'm pretty satisfied with the result, given the state of the project last year. There are avenues I can take to lower power consumption even more, but I'll think more about those next year. Individually addressable LEDs would be a plus for enabling new LED patterns. On a side note, I would like to use a different microcontroller platform for the next revision, should I choose to go through with another revision – maybe something like an ATtiny or SAM. I haven't had much experience with including AVR or ARM designs in my hobby projects so that could be a good opportunity to get acquainted.