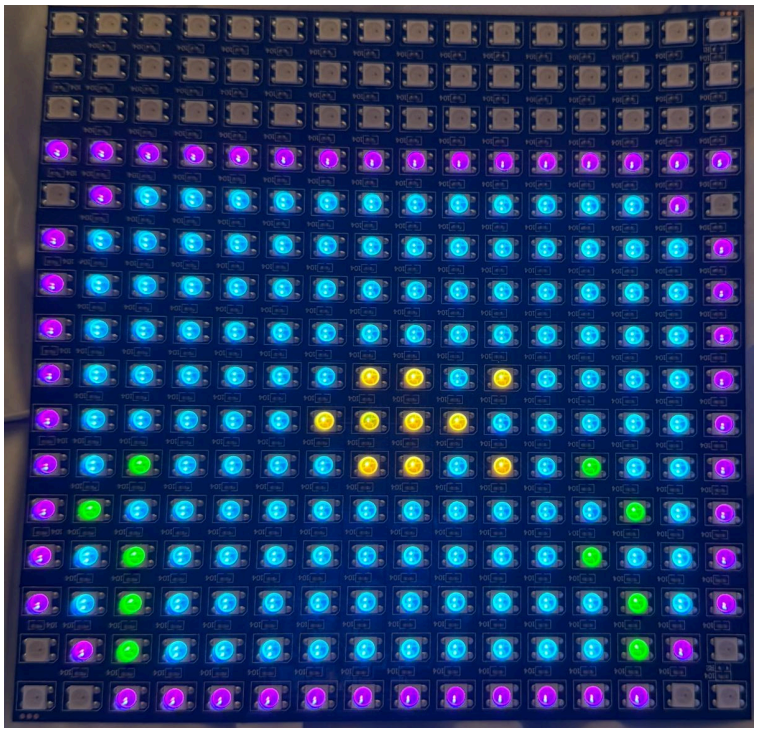
ELEC 327 Final Project:
Digital Desktank

Renee Wrysinski, Paulina Arizpe-Romo, Oyinkansola Sobakin-Ashebu,
Emmie Casey, Timauri-Lee Carby



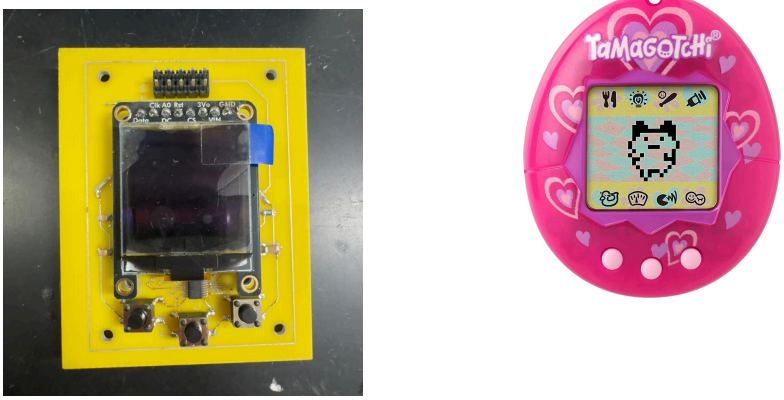
Concept Description

This project is an interactive desktop fish tank created using an MSPM03507 and 16x16 Neopixel LED grid. By default, the fish simply swims around its tank, but you can play and interact with it: you can make it race around its tank, feed it, make it go to sleep, or even play a fun game with it! In addition to displaying fun animations, it also plays tunes using a PWM-controlled buzzer when you perform certain interactions.

Design Concept Evolution

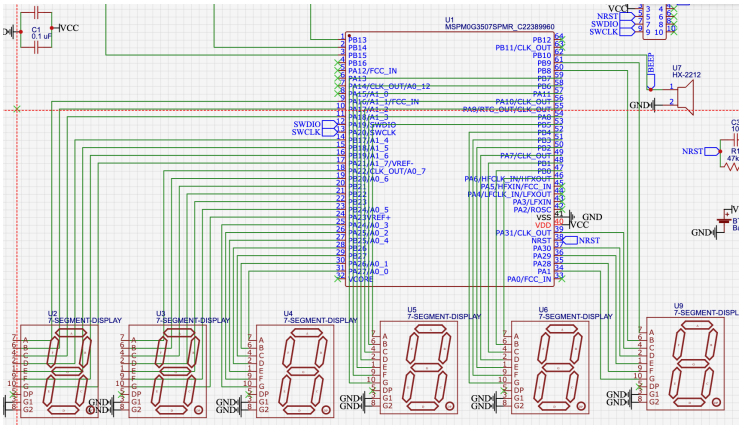
For our final project, we initially hoped to recreate a pocket sized game we all love, the **Tamagotchi**. A Tamagotchi is a small handheld digital pet where the player cares for a virtual creature by feeding it, playing with it, and cleaning up after it. If neglected, the pet can become unhappy or even "die," encouraging the player to check in regularly and be responsible. For this design, we had planned to recreate the pixel art that is traditionally featured on Tamagotchis, as well as 3D-print the egg-shaped case

shown on the right below. Unfortunately, our Tamagotchi plan had to be scrapped, as we faced substantial challenges in its execution, especially regarding communicating with the screen due to poor documentation of its use of SPI. As such, we pivoted to a second design, **The Oracle**.



Tamagotchi PCB Built (Left) vs. Actual Tamagotchi (Right)

The Oracle is a low-power, closed-loop embedded system based on the MSP microcontroller, designed to provide an interactive fortune-telling and karma-evaluation experience through three push-buttons and six 7-segment displays. Operating primarily in a sleep state with periodic or button-triggered wakeups via timer interrupts, the system features a debounced multi-press button interface for navigating a simple three-option menu: fortune telling using true random number generation, wish fulfillment from predefined categories, and karma evaluation based on user input patterns. When idle, it displays an “ASLEEP” animation and wakes on button press to enter the “AWAKE” state. Designed for resource-constrained environments, The Oracle balances playfulness with low-power efficiency.



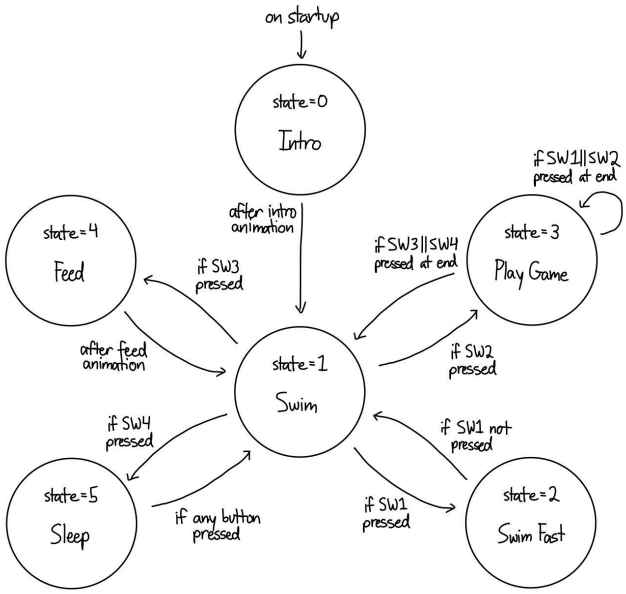
Schematic Diagram of Oracle PCB

Despite extensive work on the development of scripts for the Oracle and debugging, we switched to our third design, the fish tank, due to our microcontroller overheating. This issue arose because we were attempting to power all six 7-segment LEDs with the microcontroller, which was more than what it

could handle. Using the Simon and a Neopixel LED grid, we created a display of a fish virtual pet comparable to our original goal.

Behavior and State Machine

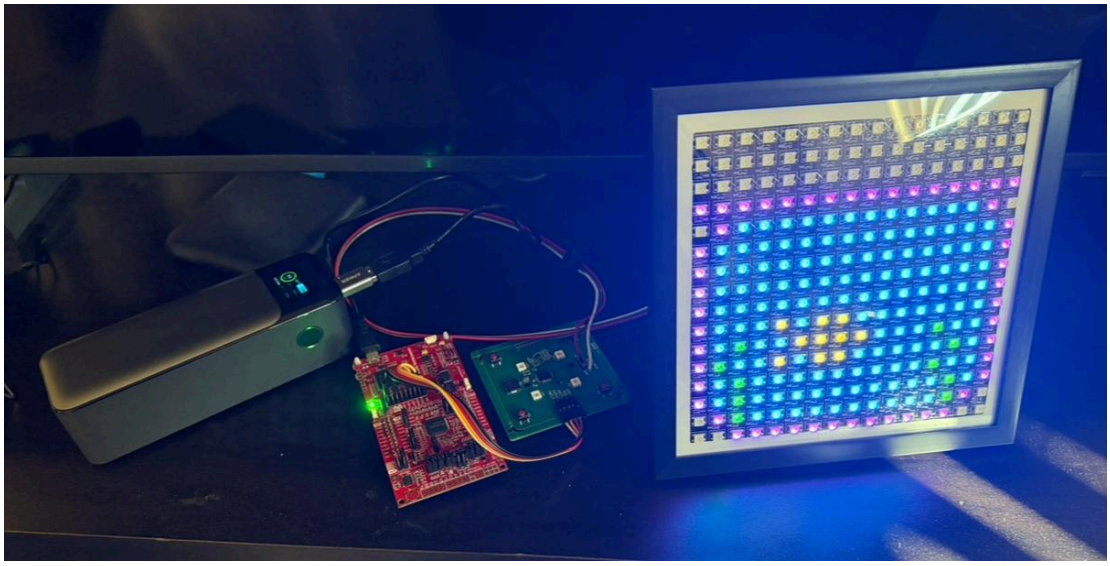
When you power the tank on, it kicks off in the Intro (State 0) where an intro animation is displayed. As soon as that animation wraps up, the code drops you straight into Swim (State 1), the "home base" where your fish idle swims. While it's swimming at a normal pace, the game is always waiting for button presses. Hit SW1 and the fish bursts into Swim Fast (State 2), zooming around until the button is no longer pressed. Pressing SW2 while the fish is swimming will launch the Play Game state (State 3); at the end of each round, pressing the buttons on the right (SW1 or SW2) restarts the game, and pressing the buttons on the left (SW3 or SW4) send it back to the main swimming screen. Press SW3 and you jump into Feed (State 4), where the fish nibbles pellets and once that sequence ends, you're right back to Swim. Finally, a press of SW4 sends the fish off to Sleep (State 5), dimming the scene and settling your little friend down—any button press then wakes it up and resumes the Swim state. In every case, after any action completes you return to the easygoing Swim (State 1) as shown in the diagram below.



State Diagram

The Play Game state is the most complicated of all six states, as instead of displaying a simple animation, it has to receive inputs from all four buttons and the random number generator, and display different animations based on these inputs. In the game, the fish is given a choice of four hiding spots, and has to choose one to hide in. After a hiding spot is chosen, three of the four hiding spots are removed. If your chosen hiding spot is the one remaining, you win and are shown a winning animation and are played a cheery song; if your hiding spot is removed, you lose and are shown a losing animation and are played a sad tune.

Hardware Design Description



Microcontroller: MSPM0G3507

- 32 MHz SYSOSC clock, low-power sleep via WFI

Inputs: Four momentary push-buttons (SW1, SW2, SW3, SW4)

Outputs:

- 16x16 Neopixel LED Grid (WS2812B)
- Buzzer PWM on Timer A1 channel: audible feedback

Power:

- Microcontroller powered by onboard 3.3V regulator
- LED grid powered by USB or external batteries connected to Simon board

Code Architecture

- main.c:
  - contains state machine determining behavior of fish
  - uses functions defined in various helper files to obtain desired behavior
- display.c:
  - contains functions to take RGB arrays from image_frames.h and convert these to SPI packets, then send them to Neopixel grid
- animations.c:
  - contains functions to create animations from frames contained in image_frames.h
    - simple animations such as Intro, Swim, Feed, Sleep are just a series of frames
    - behavior of game is determined by play_game() function, which plays different animations based on inputs from buttons and random number generator
- buttons.c:
  - contains functions to interact with buttons on Simon board
- effects.c:

    - contains buzzer control functions and different types of delays (wait for set period, wait until button press, etc.)
- lab6_helper.c:
    - slightly modified helper functions from Lab 6
    - functions to initialize timer, processor, GPIO, PWM, SPI
- simon_random.c:
    - slightly modified random number generator from Simon midterm project
- image_frames.h:
    - contains RGB arrays of frames for animations
- notes.h:
    - contains defined notes for playing music on buzzer

Pedagogical Insights

To implement this fish tank, we had to utilize a 16x16 Neopixel grid. We were provided with example code to flash an 8x8 Neopixel grid on and off, but had to adapt this code to work with a larger grid and also create a variety of functions to display simple 16x16 pixel art on this LED grid. This grid is connected to the SPI pins of the MSPM0G3507, but does not communicate using a typical SPI protocol; the bits must be encoded before being sent. In this encoding, every 1 bit becomes 3 bits, with 1 becoming 110 and 0 becoming 100.