ASEE AMERICAN SOCIETY FOR ENGINEERING EDUCATION

Paper ID #35528

Simulated Custom Microcontroller for a Remote First-Year Software Design Project

Aidan Matzko, The Ohio State University Engineering Education Department

Aidan is a graduate teaching associate at The Ohio State University's Engineering Education Department, where he teaches the freshman Fundamentals of Engineering Honors (FEH) class sequence. He will be graduating with a BS CSE and MS ECE, and has a focus on cybersecurity.

Brooke Morin, The Ohio State University

Brooke Morin is a Senior Lecturer in the College of Engineering at Ohio State University, teaching First-Year Engineering for Honors classes in the Department of Engineering Education. Brooke earned her bachelor's degree and master's degree in Mechanical Engineering at Ohio State.

Full Paper: Simulated Custom Microcontroller for a Remote First-Year Software Design Project

Introduction

The first-year engineering honors program at a large midwestern university features a two-course sequence. The first semester focuses on problem solving and computer programming and culminates in a software design project at the end of the autumn semester. During this project, students develop a simple video game on the in-house microcontroller. Due to the ongoing COVID-19 pandemic, a Proteus simulator was developed to allow for this project to be completed remotely. This paper details the pandemic-driven process of developing a tool for remote game creation in C/C++ and assesses the effectiveness of this online delivery.

Motivation

The Software Design Project (SDP) assigned at the end of the autumn semester synthesizes the technical programming skills that students learned throughout the semester while also providing a teamwork approach to writing code. Kecskemety, et. al, discussed the motivations and format for this project more thoroughly in [1]. This project relies on using the in-house Proteus microcontroller, which was not possible due to COVID-19. Thus, it was decided that a Proteus simulator should be developed to allow for the project to be completed remotely. Goals for the simulator included being able to run on Windows and MacOS, as well as the student-facing Application Programming Interface (API) being functionally identical to the current Proteus API.

Project Description

Figure 1, shown on the next page, shows an overview of the SDP project. Each row describes a new class day and deliverables are shown in red. The project was designed with a scaffolding approach, with the deliverables corresponding to milestones in the software design process. First, the students learned about the project and brainstormed ideas for their game. Then, before Thanksgiving break, teams submitted their game proposal, detailing what they would like their game to do. Next, they created a flowchart detailing the high-level logic of their game. The next day, they demonstrated their game's menu as a performance check and were required to have completed at least some of their documentation website. Lastly, they turned in the game and corresponding website, and present their games to the rest of the class.

Since both the Proteus microcontroller and simulator use an API and building process that are different than the compiling process that students use during the rest of the semester, students were provided with an opportunity to practice coding and building simple projects about a week prior to the start of the project. The instructional staff also received training on the differences and likely areas of student confusion.

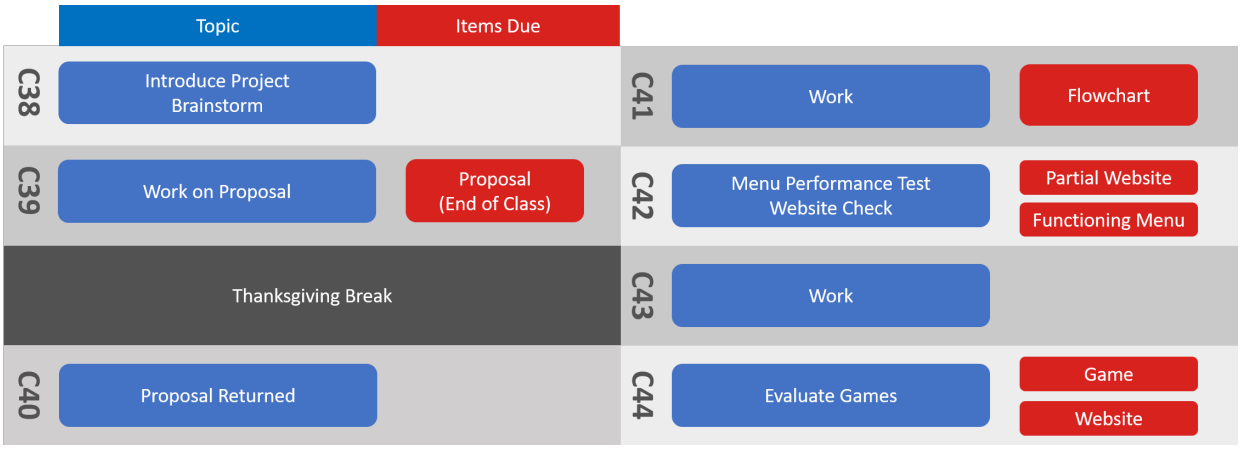

Figure 1: Project overview.

Original Proteus Microcontroller

The Proteus microcontroller was originally developed to support a robotics design project. [2] Many of its features, such as motor control and digital and analog I/O were not utilized during the SDP. The features that were important to the SDP primarily involved the touch-sensitive LCD (Figure 2) and an SD card where students could save and load data that would persist between game sessions. Additional features, such as the built-in accelerometer, were sometimes incorporated into games but were not considered critical to offering the project. The API included features that allowed students to detect touch, perform timing-based actions, and draw simple shapes and text easily. These features also needed to be incorporated into the simulator.

Figure 2: The Proteus microcontroller, which features a touchscreen LCD.

Proteus Simulator Architecture

The Proteus simulator is divided into two parts: A C API named “fehProteusfimware” that students interact with to write their games, and a Python front end that runs and displays the game. The C code communicates with the Python code through Inter Program Communication (IPC). This architecture is portrayed in Figure 3.

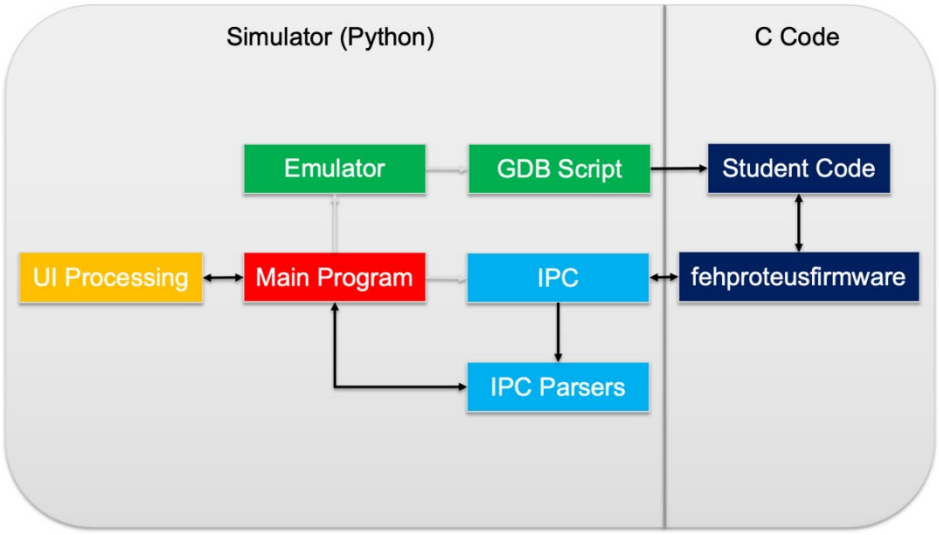

Figure 3: Modular breakdown of Proteus simulator code.

C API

The C API enables students to write C programs for the Proteus simulator. It includes functions such as writing to the screen, checking how long the program has been running, or generating a random number. This API appears identical to the API used on the physical Proteus. A Makefile is included to allow users to create an executable that can be ran by the front end.

API function calls are communicated to the front end using the open source library ASIO, which implements IPC in C. Every API function has a unique identifier. When an API function is called, a message is generated using the functions unique identifier and any relevant arguments. Then, this message is sent to the front end, where it is interpreted. For example, the FEHLCD.WriteLine(String line) function has an identifier of 7. If the WriteLine function was called with the argument “Test!”, the message sent to the front end would be “7.Test!”.

Front End

The Python front end is a standalone executable packaged by PyInstaller. Once ran, it prompts the user to select a program compiled with the C API. This program is launched using GNU Debugger (GDB), a commonly used C++ debugger. This allows the programs execution rate to be controlled, because GDB has the ability to step thru a program line by line. Once the program is running, the front end listens for IPC messages from the C program using the Python socket library. Each group of functions in the C API has a corresponding IPC parser in the front end, which handles the messages as they are received. These parsers implement the API actions, such as drawing to the screen. A separate thread handles UI input, such as clicking on the screen. An example of the Proteus simulator front end, executing a Hello World program, is shown in Figure 4 on the following page.


```
#include "FEHLCD.h"

/**
 * Entry point to the application.
 *
 * @returns status code of program exit
 */
int main() {
   int x, y;
   while(1)
   {
   while (!LCD.Touch(&x, &y)) {

   }

   if (x < 20) {
      x = 20;
   } else if (x > 219) {
      x = 219;
   }

   if (y < 20) {
      y = 20;
   } else if (y > 299) {
      y = 299;
   }

   LCD.SetFontColor(LCD.Red);
   LCD.DrawCircle(x, y, 20);

   }
   return 0;
}
```

Figure 4: The simulator running Hello World (left) and completed tutorial code (right).

Sample Code Analysis

To help students understand how to use the Proteus simulator, tutorial assignments were created and assigned before the SDP started. The code for the first tutorial assignment, where students are tasked with drawing a circle to the screen where it was pressed, is shown above in Figure 4.

First, two integers x and y are declared, to hold the location where the screen was touched. Then, an infinite while loop is started so that the program will continually draw circles when touched. Next, the API call LCD.Touch(&x, &y) is called. This function returns true if the screen was pressed, and if true places the location that the screen was pressed into the x and y variables that were passed in. After this, boundary checking code ensures that the circle will not be drawn partially off the screen. Lastly, the API calls LCD.SetFontColor(LCD.Red) and LCD.DrawCircle(x, y, 20) are used to display a red circle to the screen.

Student Outcome

Due to the multitude of changes that were made to the project for online delivery, it is difficult to isolate the effect of the simulator. However, code quality and resulting game functionality remained similar to previous years. In 2019, student code averaged 1.7 classes per game (n=31), and in 2020 student code averaged 2.23 classes per game (n=22). Figure 5, shown below, shows an implementation of Simon Says on a physical Proteus from 2019 and on a Proteus simulator from 2020 with very similar functionality. Figure 6, also shown below, highlights a Proteus simulator game that showcases more advanced logic and graphics.

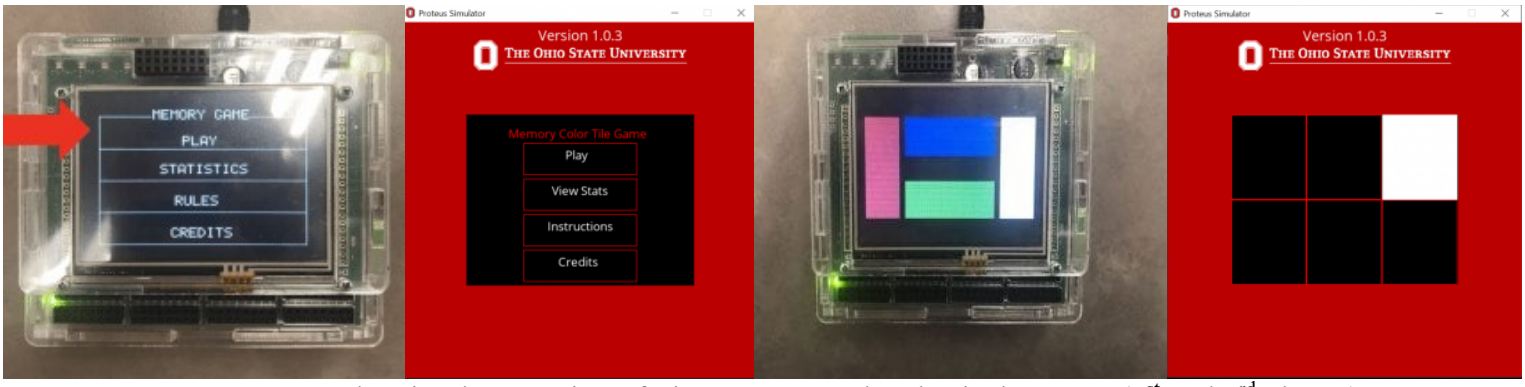

Figure 5: A student implementation of Simon Says on the physical Proteus (1st and 3rd photos) and Proteus simulator (2nd and 4th photos).

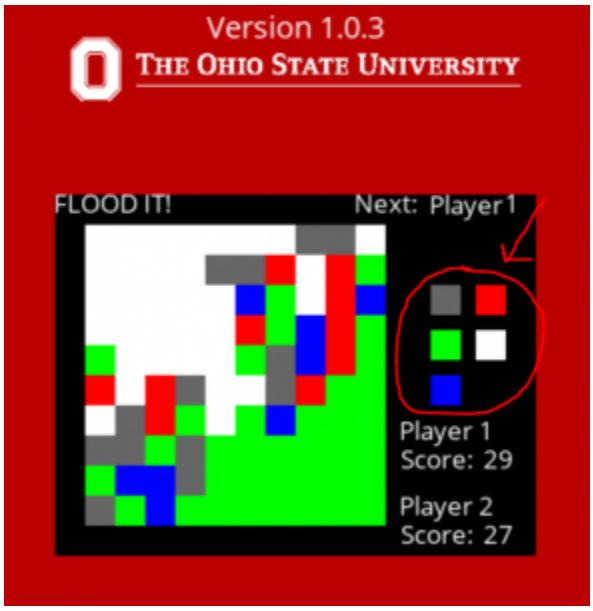

Figure 6: A student implementation of FloodIt.

Future Work

When originally designed, emphasis was placed on using GDB to allow the C programs execution rate to be controlled. However, the need for IPC between C and Python proved problematic, especially because the environment the simulator was running in was not controlled. MacOS proved to be particularly challenging to support, due to recent design decisions by Apple to lock down the operating system. Thus, developing a C only solution, where the API calls are processed and displayed all in C code is desirable. This could be done using a simple header-only C library that provides functions to draw to the screen. In the future, this simulator can be released for other institutions looking to implement a C++ game design experience.

Conclusion

The SDP is an important project assigned at the end of the autumn semester in the first-year engineering honors program at Ohio State. Due to the COVID-19 pandemic, a Proteus simulator was developed to allow the project to be completed remotely. This simulator, written in C and Python, had all the functionality necessary to create a game, and mimicked the experience of developing on the real Proteus.

References

1. K. M. Kecskemety, A. B. Drown, and L. Corrigan. "Examining software design projects in a first-year engineering course: how assigning an open-ended game project impacts student experience." *2017 ASEE Annual Conference & Exposition*. 2017.
2. M. A. Vernier, et al. "Design of a full-featured robot controller for use in a first-year robotics design project." *2014 ASEE Annual Conference & Exposition*. 2014.