Software Formalization

Year: 2023 Semester: Fall Team: 5 Project: Smart Air Hockey Table

Creation Date: September 24, 2023 Last Modified: September 30, 2023

Author: William Dobert Email: wdobert@purdue.edu

1.0 Utilization of Third Party Software

The software that powers the Smart Air Hockey Table (SAHT) will be written almost entirely by the team as part of our design efforts. The only third party software to be utilized is the STM32CubeIDE development environment. The team intends to use the features of this software package to generate C code for device hardware configurations, compile firmware with a low-level device-specific library, and flash firmware to the microcontroller. It is a proprietary Eclipse-based IDE, and all applicable licensing information can be found [on this page](https://www.st.com/en/development-tools/stm32cubeide.html) [1].

2.0 Description of Software Components

The SAHT makes use of a simple hierarchy of software components. A central state machine responds to events that occur on a variety of sensors, and device drivers written by the team detect these events and allow the state machine to control connected devices. The state machine is covered in greater detail in this document’s appendix [2]. The SAHT has device drivers for the following hardware: LED matrix, two OLED displays, blower fan power relay, Hall effect sensor matrix, and push button rotary encoder. Each of these device drivers will have an initialization function that is called when their associated hardware must be prepared for operation.

*2.1 LED Matrix Driver*

The LED matrix is driven by the output of a hardware timer set to PWM mode. The timer has its PWM configuration adjusted by a repeating DMA transfer. The DMA transfer ultimately draws from an array that stores color values for the entire matrix. The public interface for the driver consists of a couple functions. Firstly, **void Driver\_LED\_SetColor(uint8\_t x, uint8\_t y, uint32\_t color)**, which sets the LED at **(x, y)** to the 24 bit RGB color stored in **color**. Second, **void Driver\_LED\_SetAll(uint32\_t color)** overrides the entire LED matrix to a single **color** value. The state machine is responsible for calling this interface if the LED matrix must be modified, such as when the puck moves or a goal is scored.

*2.2 OLED Displays Driver*

The two integrated OLED displays are used for score tracking as a match progresses. The displays are communicated with via hardware-native SPI protocol. The SPI data transmit register is modified directly when sending commands, or by a DMA transfer when sending updated display data. The DMA transfer draws from an array containing the desired state of every pixel on the display. At a minimum, the driver’s public interface contains one function, **void Driver\_Display\_SetPixel(uint8\_t display, uint8\_t x, uint8\_t y, bool on)**, that updates the state array for the specified pixel of **display** at **(x, y)** to equal **on**. Additional functionality may be provided by the driver to draw full alphanumeric characters, clear the display, or use more advanced control features of the display hardware. The state machine invokes this driver interface when a goal is scored or the game is restarted.

*2.3 Blower Fan Power Relay Driver*

For the sake of user experience, it is desirable to have the capability of intelligently controlling whether the blower fan is running. For instance, the blower fan can be deactivated when a match is not ongoing to reduce unnecessary noise and power consumption. The power relay requires a single GPIO pin to be set according to the desired state of the blower fan. This driver’s public interface exposes the function **void Driver\_Blower\_SetPower(bool power)** to set the associated GPIO output pin to **power**.

*2.4 Hall Effect Sensor Matrix Driver*

The Hall effect sensor matrix is responsible for tracking the approximate position of the puck as it moves across the play surface. To achieve this, 16 rows and 32 columns of aggregated sensor data are monitored by 48 GPIO pins on the microcontroller. To keep a real-time estimation of the puck’s position, the state of each GPIO pin must be checked at very small time increments (less than 2 ms). The public interface for the driver facilitates this, with the function **void Driver\_Hall\_GetActivePosition(uint8\_t\* x, uint8\_t\* y)** being called when an up-to-date position must be calculated. This function modifies the values pointed to by **x** and **y** to be the sensor-active coordinates where the puck is positioned.

*2.5 Push Button Rotary Encoder Driver*

A push button rotary encoder is installed on the exterior of the SAHT to serve as a flexible method of user input. It may allow the user to adjust LED brightness, select the number of goals to win, or push to start a new game. A single GPIO pin is necessary to monitor whether the push button has been depressed. The state machine checks this via the driver’s public interface function **bool Driver\_Button\_GetPressed()**. The rotary encoder portion of the device is handled automatically by a hardware timer onboard the microcontroller that is set to the special *Encoder* mode. The value of this timer’s counter register is queried to determine the direction and magnitude of rotation the rotary encoder is currently undergoing. This can then be publicly accessed via the driver’s function **uintXX\_t Driver\_Button\_GetTurn()**.

3.0 Testing Plan

1. *State Machine*: The state machine will be tested by monitoring that it transitions between the correct states only when the requisite condition has been met. Exposing the state machine to every possible input it could receive would be impossible, but a series of notable edge cases can be tested instead. For instance, scoring multiple goals in quick succession, removing the puck from the playing surface, scoring a goal before the game has begun, and pressing the start button when a game is ongoing.
2. *LED Matrix Driver*: Being a system distributed across hundreds of individual components and circuit boards, the LED matrix driver must be able to produce highly accurate output signals. Since the flow of data across the matrix is sequential, it will be straightforward to test the driver by commanding it to send a simple pattern and watching to see if the signal stops somewhere along the way (much like a strand of faulty holiday lights). Additionally, if the LED matrix does not produce the correct colors in the test pattern, we will know there is a problem and be able to begin resolving it immediately.
3. *Hall Effect Sensor Matrix Driver*: For the sake of testing the sensor matrix, we can assume that the LED matrix is already fully functional. The microcontroller will be programmed to activate portions of the LED matrix when the associated Hall effect sensors are activated by a magnet. This 1:1 visual relationship will make testing and debugging the sensor matrix a very straightforward task. Simply moving a magnet along each row in the matrix and observing the activation of LEDs will serve as an effective test strategy.
4. *OLED Displays Driver*: The OLED display driver will be tested for every feature it has implemented. Most importantly, the function to set pixels on the display will be tested with a variety of patterns to ensure that data is being sent in the correct format by the driver. Observing how the OLED displays respond to commands and updated pixel data from the microcontroller will be an intuitive and easy method for testing the driver implementation.
5. *Push Button Rotary Encoder Driver*: The driver for the rotary encoder will be rather simple, as it utilizes the microcontroller’s hardware support for decoding the quadrature encoder output. The output of the encoder, as perceived by the driver, can be visualized with the LED matrix for testing purposes. Turning the dial back and forth while observing the LEDs change brightness in response will allow us to effectively test this driver.
6. *Blower Fan Power Relay Driver*: The blower fan driver is the simplest to be implemented, as it is effectively a wrapper for toggling a specific GPIO pin. Testing this driver will consist of providing power to the relay hardware, and commanding the driver to activate and deactivate the relay. Observing the connected blower fan during this process will immediately reveal that the driver is functioning correctly.

4.0 Sources Cited:

[1] “STM32CubeIDE,” STMicroelectronics. <https://www.st.com/en/development-tools/stm32cubeide.html>

[2] A. Chung Ma, “Flowchart of the system,” Alan Chung Ma Progress Report, Sep. 15, 2023. <https://engineering.purdue.edu/477grp5/team/achungma/> (accessed Sep. 26, 2023).

Appendix 1: Software Component Diagram