Computer Science and Engineering

CSCE430101 - Embedded Systems - Spring 2025

Project: Smart Banzina

Michael Reda - 900203291
Andrew Aziz - 900213227
Moneer Zaki - 900202427

Table of Contents

1. Introduction

Smart Banzina is an embedded hardware system that aims to simplify the selection and dispensing processes of the various fuel types (oil, gas, electric) available. A user interacts with the system via a keypad, while an automated gate that controls access to the fuel dispenser is managed by an ultrasonic sensor and a servo motor. Real-time feedback and interaction are delivered through an LCD screen.

Implemented Features:

- Vehicle ultrasonic sensors activate the servo for gate fuel access.
- LCD displays options of fuel type(s) available with cost as well as instructions.
- Gate status is shown through RGB LED indicators; red indicates the lane is busy, while green shows the lane is available.
- The keypad for input selection enables fuel type selection (oil, gas, or electric) and input of numerals for quantity/petrol amounts and even price selection.

Unimplemented Features:

- Support for multiple lanes requires the usage of the RTOS in the future.

2. Hardware

(a) Embedded Board Description

- Board: Arduino Mega microcontroller 2560 .
    - CPU: 8-bit AVR with 16 MHz clock speed.
    - RAM: Contains 8KB SRAM.
    - I/O: It consists of 54 digital pins (15 PWM), 16 analog inputs, and 4 UART ports.
    - Operating Voltage: 5V.

- Software: It employs Arduino Firmware (there is no traditional operating system).

(b) Input Devices

- Keypad:
    - Interface: GPIO matrix (4 rows and 4 columns).
    - Protocol: Scanned through pins at rows 0 to 3 and columns 4 to 7.
    - Limitation: Row three does not work.

- Ultrasonic Sensor (HC-SR04):
    - Interface: Trig pin (A0) and Echo pin (A1); that was for the one at the lane part. Trig pin (9) and Echo pin (10)
    - Protocol: Measures distance through pulse timing-based distance measurement.

(c) Output Devices

- LCD (I2C):
    - Interface: SDA(20) and SCL(21)
    - Protocol: Uses I2C with address 0X27

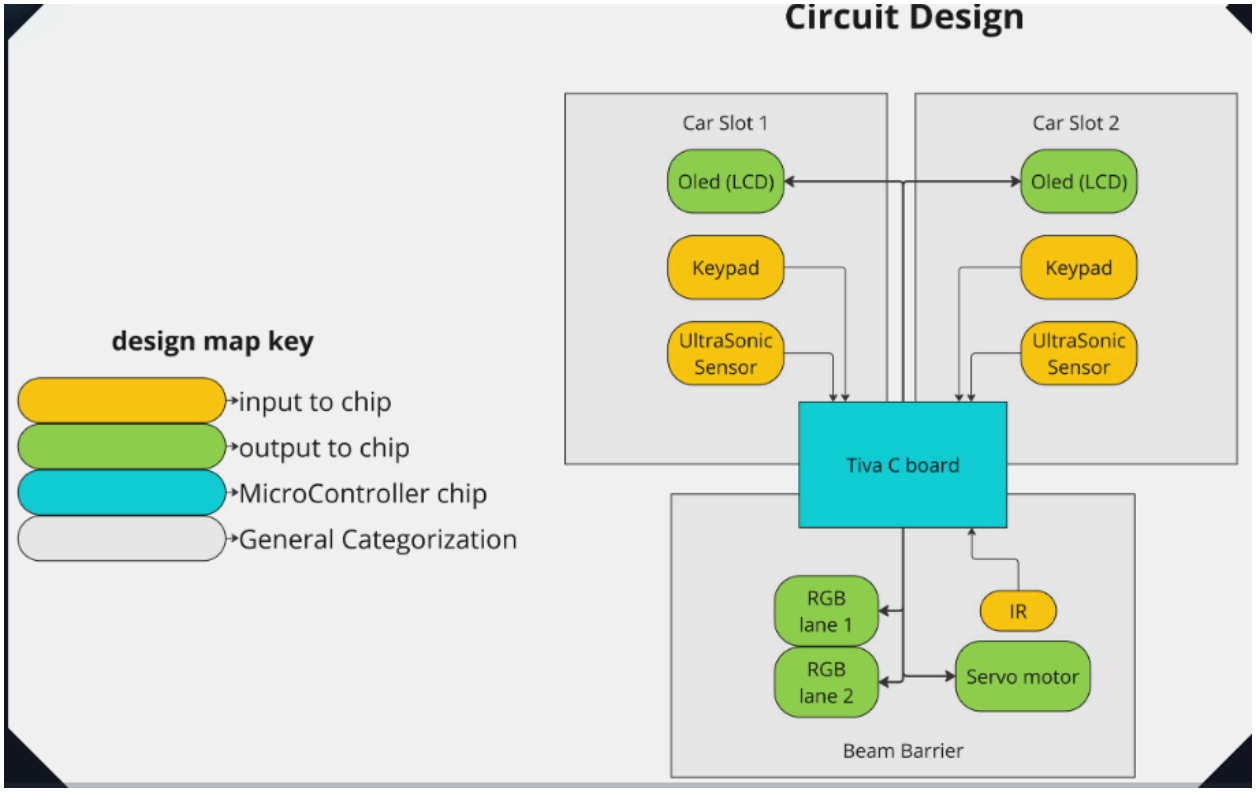
- Servo motor:
    - Interface: Uses PWM on Pin 11
    - Control: Moves from 0 degrees to 90 degrees depending on ultrasonic input.

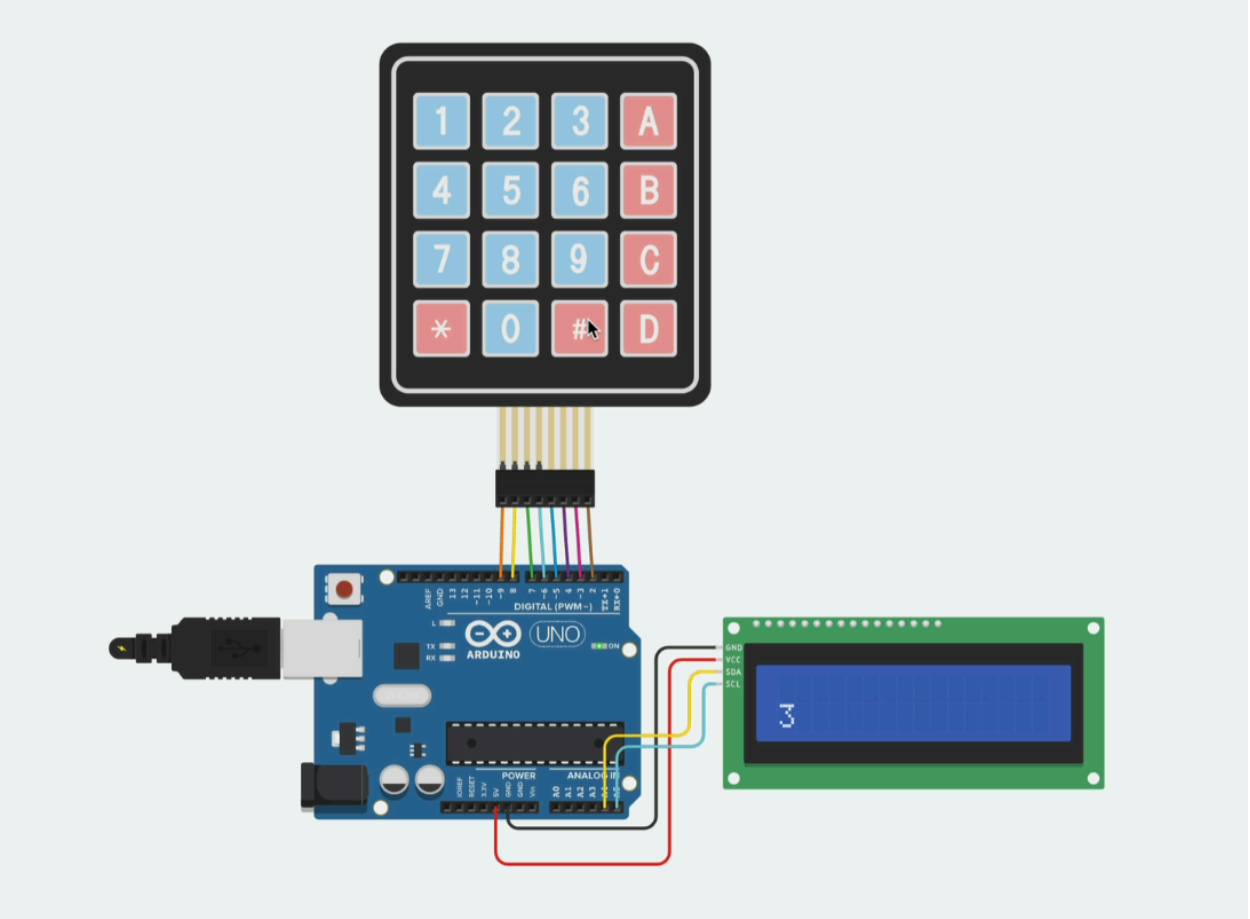
- RGB LEDs:
    - Connections: Pin 12 for green and Pin 13 for red.

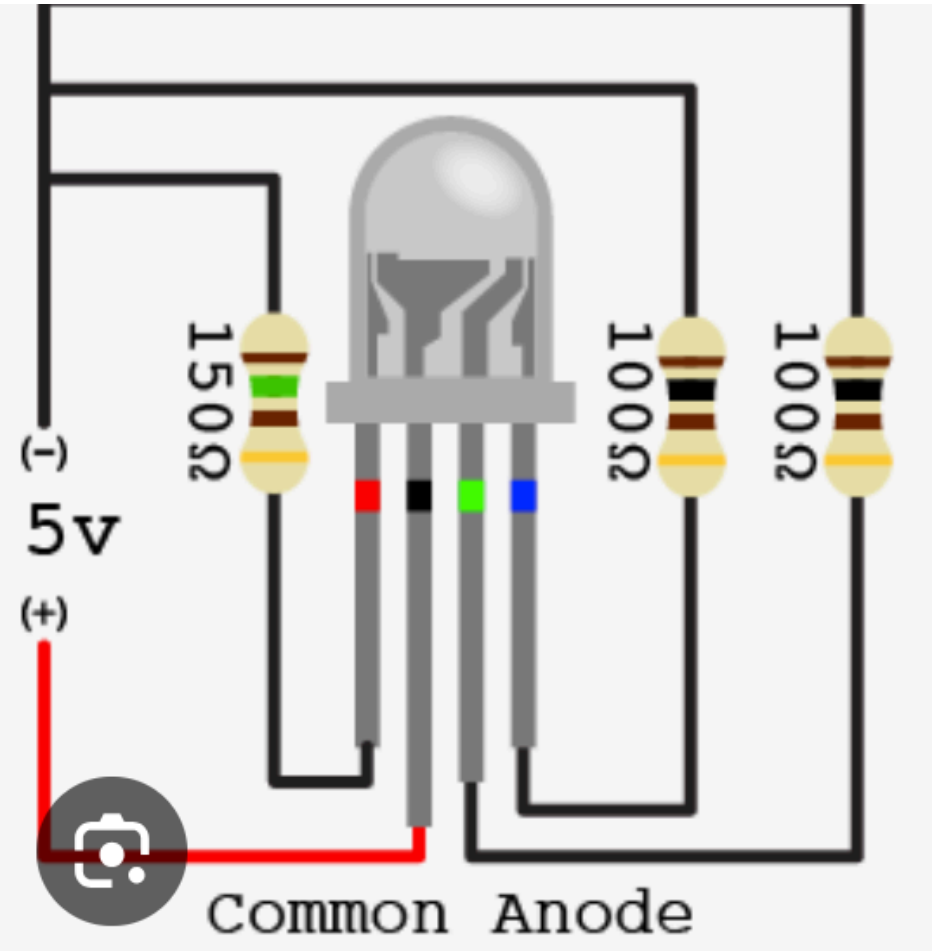
(d) Datasheets & Schematics

- Datasheets:
    - HC-SR04 Ultrasonic sensor IPS
    - Arduino Mega (ATmega2560)
    - The Keypad

- Schematics:


Circuit Design
design map key
input to chip
output to chip
MicroController chip
General Categorization
Car Slot 1
Car Slot 2
Oled (LCD)
Oled (LCD)
Keypad
Keypad
UltraSonic Sensor
UltraSonic Sensor
Tiva C board
RGB lane 1
RGB lane 2
IR
Servo motor
Beam Barrier


1
2
3
A
4
5
6
B
7
8
9
C
*
0
#
D
DIGITAL (PWM~)
UNO
ARDUINO
POWER
ANALOG IN
GND
VCC
SDA
SCL
3



(e) Power Consumption:

- With the servo turned off, the device would draw 56 mA of current; 100 to 500 mA would be drawn if the servo was in use.
- **Optimization** (what could be done):
    - Use sleep modes during inactivity.

- Disable unused components (e.g., LEDs when the gate is closed).

3. Software

(a) Programming language: C++ was selected in this case, because the Arduino platform is founded on C/C++. This directly relates to the control and performance efficiency required for embedded systems. The provided IDE, along with its libraries (like Servo or LiquidCrystal_I2C), was developed and functions under C++, making it the most appropriate option for implementing a project like ours which involves interfacing with hardware in real time.

(b) Real-time constraints: the system has real-time constraints. The servo motor and ultrasonic sensor rely on timely responses to distance measurements to control gate operation. For example:

- If the ultrasonic sensor detects a car ≤ 20 cm, the servo must open immediately.
- A large delay (e.g., during sensor data processing or servo movement) could cause:
  - Incorrect gate states (e.g., gate closing prematurely).
  - LED indicators (red/green) to desynchronize from actual lane availability.
  - User input (keypad) to be ignored during critical timing windows, leading to system errors.
- A system failure may occur when the users' car **gets away from the ultrasonic sensor in the lane and then presses the keypad** before the end of the timer (5sec in our case) the led at the entry gate will always be red and not be green again.

(c) The system has minimal cybersecurity risks due to its **lack of network connectivity** and reliance on physical inputs/outputs. However:

- **Physical tampering** (e.g., spoofing ultrasonic sensor signals or bypassing the keypad) could disrupt operations.
- No software-level exploits (e.g., remote hacking) are applicable since the system operates offline.

(d) Use of external code: **Servo Motor**: **Implemented a custom servo control logic** without relying on the Arduino `Servo.h` library. Developed timing parameters (e.g., pulse width modulation for 0–180° rotation) and integrated median filtering for stable sensor integration.

- **LCD**: Used the `LiquidCrystal_I2C` library for I2C communication. Original code added counter logic (1–9 display) and initialization routines.
- **Keypad**: Leveraged existing keypad libraries for matrix scanning (e.g., row/column detection). Customized pin mappings and debugged connections (e.g., fixing non-functional row 3).
- **Ultrasonic Sensor**: Designed original median-filtering logic for noise reduction. Foundational distance calculation (pulse timing to cm conversion) follows standard ultrasonic sensor principles.

Originality:

- **Original part**: Fully custom servo control (no external `Servo.h` dependency), state machines, sensor fusion (servo + ultrasonic + LED), and median filtering.
- **External part**: LCD library (`LiquidCrystal_I2C`), keypad matrix-scanning logic, and ultrasonic pulse-to-distance formula were reused or adapted.

4. Related work

No one has done this exact project but similar project, such as parking systems combining Arduino, servos, ultrasonic sensors, LCDs, and keypads for automation exist, but our work distinguishes itself through **domain-specific innovation and technical customization**. Unlike generic solutions relying on Arduino's `Servo.h` library, we implemented **register-level PWM timing** for servo control, eliminating library dependencies and enabling precise hardware manipulation. The project simulates a **fuel station interface** with a structured, five-stage workflow (detection → fuel type selection → pricing/quantity input → confirmation), integrating a keypad and LCD for dynamic user interaction, including unit conversions (litres, kWh, PSI) and real-time pricing logic. To enhance reliability, we introduced **median filtering** for ultrasonic sensor data, reducing noise in distance measurements. Modular code design and documentation of power constraints (e.g., servo current draw) further differentiate our approach. By merging low-level hardware optimization with a tailored application (fuel station simulation), this project advances beyond generic automation frameworks, offering both technical depth and domain relevance.

5. Conclusion

(A) Group work: - **Moneer Zaki**: Designed and implemented **custom servo control logic** (register-level PWM timing) and integrated ultrasonic sensor median filtering. **Andrew Aziz**: Developed the **LCD/keypad interface**, including state machine logic for fuel type selection, pricing, and unit conversions. **Michael Reda**: Built the **hardware prototype**, debugged power distribution, authored documentation, managed the GitHub repository, and tested modular code integration (servo + sensor + LCD).

(B) Challenges:

- **Keypad Hardware Flaw**: Row 3 malfunctioned due to a wiring defect, requiring code adjustments to skip affected inputs.
- We cannot use more than **1** lane because we need them to work simultaneously thus, we need **RTOS.**
- The project's voltage and current specifications are carefully documented: components like the servo motor **(5V, 100–500 mA),** ultrasonic sensor **(5V, 15 mA),** and LCD **(5V, 3–5 mA)** operate at 5V, with total current reaching 556 mA (including servo surges). The Arduino Uno's **7–12V** input powers these via its 5V regulator, while a 330Ω resistor limits the RGB LED's current **(10–20 mA)** via Ohm's Law. Calculations confirm the need for **a ≥1A** power supply to handle peak loads, avoiding instability from servo-driven current spikes.

(C) Future Work:

- **RTOS Integration**: Implement Real-Time OS for multi-lane concurrency.
- **Wireless Connectivity**: Add Bluetooth/Wi-Fi for remote monitoring via mobile apps.
- **Enhanced Keypad**: Replace the faulty keypad with a membrane matrix or touchscreen.
- **Safety Features**: Integrate emergency stop buttons or overload detection for servo motors.
- **Energy Optimization**: Use sleep modes during idle periods to reduce power consumption.
- **User Interface**: Upgrade to a graphical LCD or OLED for dynamic menus and animations.

Here is the complete documentation during the implementation process of our project:
https://docs.google.com/document/d/1NTYLodnASU8HxHYVh4s2Tff2L0fprDjRHNhaNvJoEf0/edit?tab=t.0

And this is the video link for our project demo: