EECS 2032 Finale Project Report: Braille Writer Project

Arthur Sabadini Nascimento, Student Number: 220007175

April 18, 2024

Contents

All design files and proposal reports can be found in this GitHub directory (https://github.com/ArthurSabadini/eecs2032-braille-writer), and the demonstration video can be found here.

(https://www.youtube.com/watch?v=A-jqoaMqUTU)

Section 1

Abstract

Our project consists of a Braille writer for people with vision impairment. This embedded system gets text input through an infrared module and translates the English text data into the American Braille language using solenoids as actuators.

Three pairs of such actuators will be used to lift pins up and down, to mimic the braille language, to provide accessibility for people with vision impairments, and to enable them to read data, as other people can.

Roles

- Arthur Sabadini Nascimento
  - Circuitry Design

    Design the circuitry. Develop drivers for the six solenoids, and overall inputs/outputs to the Arduino and other devices. These solenoids should be controllable from the Actuator control unit.
  - Infrared Communication Development

    Develop a system that can recognize a series of inputs from an infrared device and encode those signals into a data format that the translation software can understand.
- Syed Mustafa Jamal
  - Translation Software Development

    Working on the translation software side, used to translate braille to english and vice versa. There would be other parts of the code, for example, communicating with a display or speaker if added.
  - Actuators Control Unit Development

    Implement actuator control for the purpose of making sure they respond correctly to intended output that is safe and secure in the form factor we hope to achieve.

- Testing

  Run tests on the software and its compatibility with the hardware. It can also be done with others to see if people who are blind or can benefit from it would like it.

Expected Results

We expect to be able to send a message to the Arduino through an infrared device, then, in software, translate English messages into the American Braille language. It will do so by sending signals to 6 solenoids, which will lift up or down to signify the bumps of the American braille language.

Section 2

Board Selection

The Arduino Mega board was selected, since it has more memory than the Uno (256KB), enough to support all the software. It also has more than enough pins (16 analog pins and 53 digital pins), and we were able to connect all required devices.

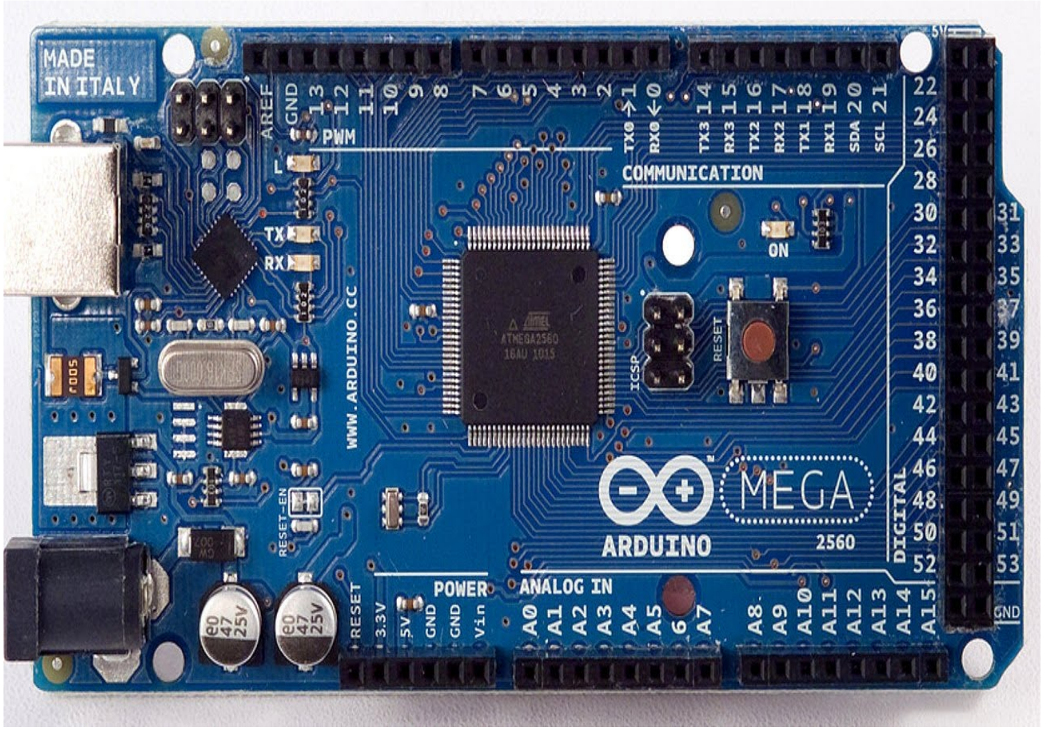

Figure 1: Arduino Mega [3]

Solenoid Driver

Driver Design

The circuit below was the chosen design for the solenoid driver. The MOSFET is used for switching the solenoid on and off, and the diode is for flyback protection

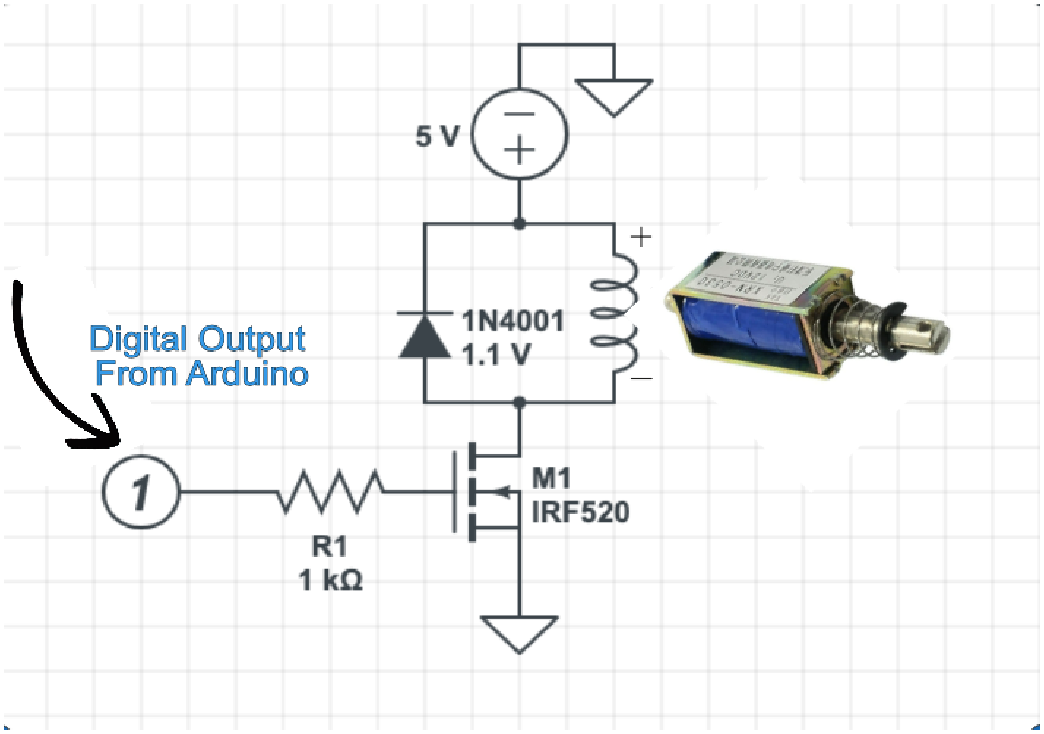

Figure 2: Solenoid Driver Circuit

Implemented Driver

The circuit below is the implementation of the driver described above. This driver circuit was constructed 5 more times (6 in total) to control all 6 solenoids.

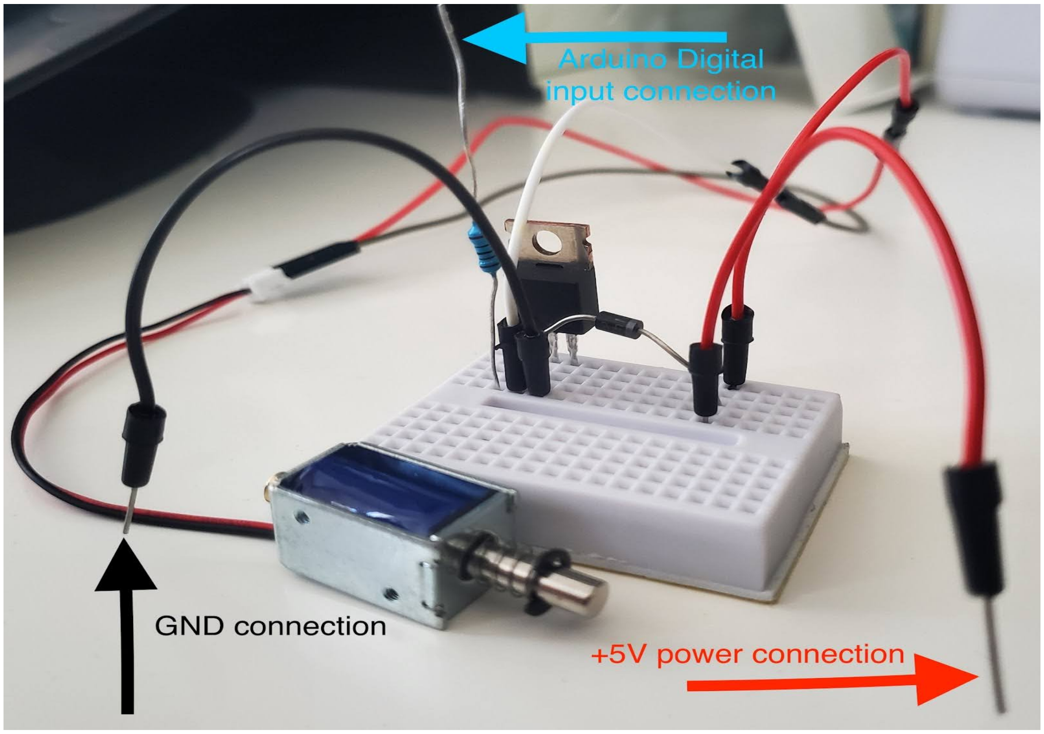

Figure 3: Implemented Driver

Final Circuitry Design

Full Design Schematic

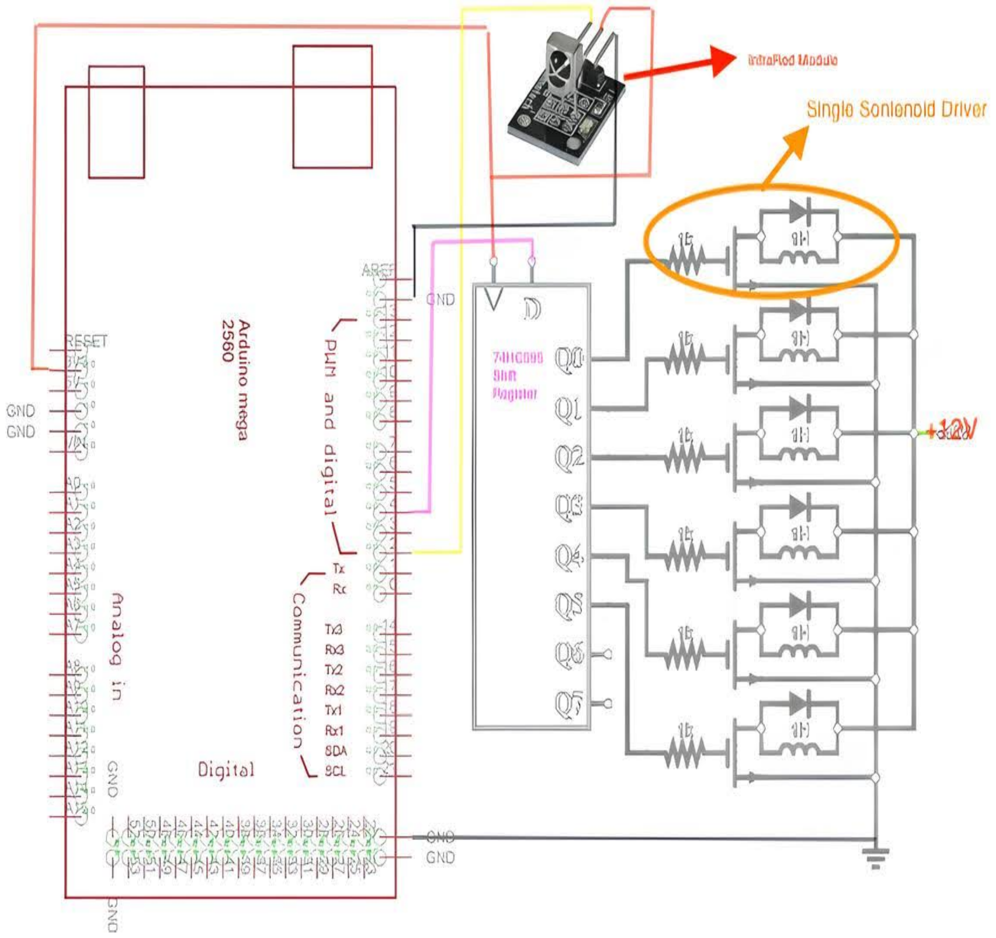

Figure 4: Full Schematic

Implemented Circuitry

The circuit below is the implementation of the full circuitry. In orange, all 6 solenoid drivers. In pink, the shift register used to control all solenoid drivers. In red, the infrared sensor, used to get input from a remote control.

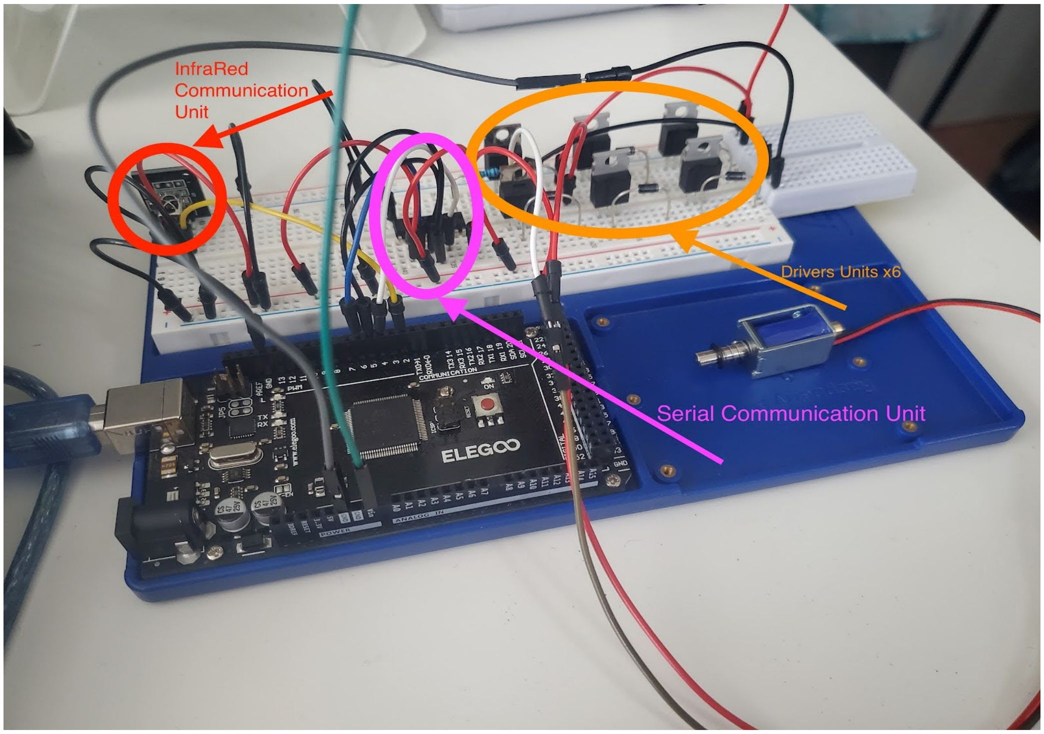

Figure 5: Implemented Circuit

InfraRed Decoder Library

I developed an Arduino library that was able to receive inputs from an infrared controller in an ordered way, and organize the input into a useful datatype to be used in the code to actuate the solenoids.

The IRdecoder (source [1]) library was coded in C++ and used the simplified Arduino standard library for its implementation, as well as the IRremote (source [2]) library.

To collect input from a remote control, we used an interrupt function that gets activated whenever the infrared module receives an input, to optimize input responsiveness.

The ISR is called whenever the infrared receiver receives an input. It reads the received signal and decodes it. If ’NEXT_WORD’ is pressed, we save the current symbol to the buffer and wait for the next one. If ’DELETE_WORD’ is pressed, we remove the symbol from the buffer. If ’CONCLUDE’ is pressed, we save the buffer and resume input acquisition, otherwise, the current symbol is updated in the buffer.

```cpp
void IRdecoder::inputInterruptHandler() {
    volatile uint32_t buttonPressed = readSignal();
    if(buttonPressed == UNDEFINED) return;

    switch(buttonPressed) {
        case IRdecoder::NEXT_WORD:
            // Add next word to symbols list
            input_buffer.add(symbol);
            resetState();
            break;
        case IRdecoder::DELETE_WORD:
            // Delete current word
            if(!input_buffer.empty()) input_buffer.pop_back();
            resetState();
            break;
        case IRdecoder::CONCLUDE:
            // Complete input, save and exit
            resetState();
            endReceiveInput();
            break;
        default:
            updateCurrentSymbol();
            break;
    }
}
```

Listing 1: IRdecoder Interrupt Service Routine

Library Implementation

Below you can see the header file of the object used to decode the infrared signals.

```cpp
// Main class definition
class IRdecoder {
    public:
        BufferIO<3, 2> input_buffer;
        bool isInInputMode = false;

        // Buttons defined for I/O functionality
        const static uint32_t NEXT_WORD = FAST_FORWARD;
        const static uint32_t DELETE_WORD = REWIND;
        const static uint32_t CONCLUDE = POWER;

        IRdecoder(uint8_t pin) : pin(pin), input_buffer(BUFFER_SIZE), irrec(pin), results() {};

        void setup();
        void beginReceiveInput();
        void resetState();
        String getStringfiedState();
        String getStringfiedSymbol();

        static void ISRHandler() {
            if(!currentInstance) return;
            currentInstance->inputInterruptHandler();
        }

    private:
        static IRdecoder* currentInstance;
        bool wasInitialized = false;
        bool isActive = false;

        uint8_t pin;
        uint8_t control_state[ROWS][COLS]{};
        uint8_t symbol[3][2]{};

        IRrecv irrec;
        decode_results results;

        void endReceiveInput();
        volatile uint32_t readSignal(); // volatile used, since input could change

        void updateCurrentSymbol();
        uint32_t indexToButtonVal(uint8_t index1, uint8_t index2);
        uint8_t* buttonValToIndex(uint32_t buttonVal); // Don't forget to free memory when not needed
        void inputInterruptHandler();
};

#endif
```

Listing 2: IRdecoder Definitions

Macros Definition

When a button is pressed on the RC control, a specific signal is sent to the receiver, which we can use to identify which button has been pressed to collect inputs. Below, you can see the macro definitions for the infrared signal emitted from every button in the remote control.

```
// Config Definitions
#define ACTION_WINDON_MS 250
#define ACTION_WINDOW_US 250000
#define BUFFER_SIZE 11
#define ROWS 7
#define COLS 3

// Remote Buttons Code
#define POWER 0xFFA25D
#define VOL_PLUS 0xFF629D
#define FUNC_STOP 0xFFE21D
#define REWIND 0xFF22DD
#define PAUSE 0xFF02FD
#define FAST_FORWARD 0xFFC23D
#define DOWN 0xFFE01F
#define VOL_MINUS 0xFFA857
#define UP 0xFF906F
#define ZERO 0xFF6897
#define EQ 0xFF9867
#define ST_REPT 0xFFB04F
#define ONE 0xFF30CF
#define TWO 0xFF18E7
#define THREE 0xFF7A85
#define FOUR 0xFF10EF
#define FIVE 0xFF38C7
#define SIX 0xFF5AA5
#define SEVEN 0xFF42BD
#define EIGHT 0xFF4AB5
#define NINE 0xFF52AD
#define UNDEFINED 0xFFFFFF
```

Listing 3: Macros

Section 3

How has my Role benefited from the course 2210?

My role in this project, as the hardware designer, has benefited quite a lot from the course 2210.

The designed driver uses components such as MOSFETs and diodes, components that we learned in depth about in the course. Without such components, I wouldn't be able to design the driver, since MOSFETs are usually used for rapid switching with microcontroller applications.

What problems were solved?

We could not just control the solenoids directly from the Arduino pins; the Arduino can't provide enough current to actuate the solenoids. To fix this, I designed a solenoid driver that can be used to safely control each solenoid from the Arduino.

If the solenoids were kept on for too long, that could cause the MOSFETs to heat up too much. Testing showed that an "on" period of 500ms was enough to be detected, and to minimize stresses, we turned the solenoids off for 250 ms after it was actuated.

To minimize power consumption, we decided to flip the solenoids upside down. Since in braille, there are more flats than bumps, fewer solenoids would be actuated.

What has been learned?

I learned how to design motor drivers, which was quite exciting! And I also learned how to best select a microcontroller for a given project, depending on the hardware needs

I also learned how to read datasheets, since I had to do that a lot to check each component's parameters and limits.

Can this project be continued and result in a marketable product?

Yes! There is a great need in the space of accessibility in technology. In the market, there are tons of projects that convert braille to English, but none that do the opposite, as our project does

In conclusion, our project would improve accessibility for people with vision impairments, and there's also a need for it in the market. So indeed it could result in a marketable product!

How successful was the project in achieving its goals?

The project was very successful, we were able to achieve all expected results. I designed a driver to easily and safely control all 6 solenoids from a microcontroller, and we were able to translate an English-text infrared input into braille, and actuate the solenoids accordingly to represent the braille language and output a simple message.

References

[1] arthur-sabadini-nascimento. *ArthurSabadini/eecs2032-braille-writer*. original-date: 2024-03-08T01:46:53Z. Apr. 17, 2024. URL: https://github.com/ArthurSabadini/eecs2032-braille-writer (visited on 04/26/2025).

[2] *Arduino-IRremote/Arduino-IRremote*. original-date: 2010-01-23T04:37:22Z. Apr. 26, 2025. URL: https://github.com/Arduino-IRremote/Arduino-IRremote (visited on 04/26/2025).

[3] *Mega 2560 Rev3 — Arduino Documentation*. URL: https://docs.arduino.cc/hardware/mega-2560/ (visited on 04/26/2025).