Topi Länsilahti

Spot-Price Electricity Controlled Heat Pump

Metropolia University of Applied Sciences

Bachelor of Engineering

Degree Program in Electronics

Bachelor's Thesis

01 October 2023

Abstract

Author:	Topi Länsilahti
Title:	Spot-Price Electricity Controlled Heat Pump
Number of Pages:	27 pages
Date:	01 October 2023
Degree:	Bachelor of Engineering
Degree Program:	Electrical and Automation Engineering
Professional Major:	Electronics
Supervisors:	Erkki Räsänen, Principal Lecturer

This bachelor's thesis documents the process of designing and building a prototype of a device to control a residential heat pump according to the spot-price of electricity. The need for this device stems from the need to automatically monitor the price of electricity to reduce heating costs.

The thesis examines the needed functions of the microcontroller, as well as the code to accomplish the project, and documents the rationale for the components used. A wiring diagram was drawn with Fritzing to aide in the prototyping process.

The project uses an Arduino microcontroller to communicate with an API to receive the updated spot price. The user interface consists of only two buttons and a display for the customer set limit price as well as the spot price. A relay board is installed to connect to the heat pump regulator.

A functioning prototype was built as a result of the work. This will be installed at the customer location and be actively used year-round.

Keywords: Spot-price, Arduino

Tiivistelmä

Tekijä:	Topi Länsilahti
Otsikko:	Lämpöpumpun ohjaus pörssisähkön hinnan mukaan
Sivumäärä:	27 sivua
Aika:	01.10.2023
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Sähkö- ja automaatiotekniikka
Ammatillinen pääaine:	Elektroniikka
Ohjaajat:	Yliopettaja Erkki Räsänen

Tämä opinnäytetyö dokumentoi suunnitteluprosessin prototyypin rakentamisesta laitteeseen, joka ohjaa omakotitalon lämpöpumppujärjestelmää sähkön pörssihinnan mukaan. Tarve tälle laitteelle juontuu tarpeesta automaattisesti valvoa sähkön hintaa lämmityskulujen pienentämiseksi.

Työssä tutkitaan mikrokontrollerin ja koodin vaadittuja ominaisuuksia projektin toteuttamiseen ja käsitellään prototyyppiin käytetyt komponentit. Työ sisältää Fritzing-ohjelmalla piirretyn kytkentäkaavion rakennetusta laitteesta.

Projekti käyttää Arduinon mikrokontrolleria kommunikoidakseen API:n kanssa saaden täten ajantasaisen sähkön pörssihinnan. Käyttöliittymä koostuu kahdesta napista sekä näytöstä, joka kertoo asetetun rajan sekä pörssihinnan. Prototyyppiin asennettu relekortti liitetään lämpöpumpun regulaattoriin.

Työn tuloksena oli toimiva prototyyppi, joka tullaan asentamaan asiakaskohteeseen aktiiviseen käyttöön ympäri vuoden. Dokumentaation tuloksia voi myös käyttää vastaavan laitteen rakentamiseen eri ympäristöön.

Avainsanat: Arduino, pörssisähkö

Contents

List of Abbreviations

API:	Application Programming Interface
BLE:	Bluetooth Low Energy
HTTP:	Hypertext Transfer Protocol
I2C:	Inter-Integrated Circuit
IDE:	Integrated Development Environment
JSON:	JavaScript Object Notation
OLED:	Organic Light-Emitting Diode
SSL:	Secure Sockets Layer
SSID:	Service Set Identifier
WLAN:	Wireless Local Area Network
Wifi:	Wireless Fidelity

1 Introduction

The price of electricity has had large fluctuations in recent years, due to factors such as COVID, cold winters and the war in Europe. This price increase has great ramifications in Finland, where the majority of electricity use is spent on heating during the long winters. The fluctuating electricity market has led to an increase in spot-price based electricity contracts, which reward customers for being aware of these changes, and primarily using power when costs are low. With the rising availability of consumer electronics, home automation seems to be the solution to this problem.

The aim of this thesis work was to design and produce a prototype of a device that controls the heat pumps of a residential building automatically according to the spot-price of electricity. The top priority for this device is to be as user-friendly as possible, while still being open to customization, with most of the technical operation happening in the back end. In addition to ease of use, ease of installation was also a top priority during the project, which led to a small size and very few connections to be made. The choice to connect to relays rather than the main regulator of the heat pump was also to make the documentation applicable to other brands of heat pumps as well.

Potential commercialization of the prototype as a product was a guiding principle throughout the project, leading to favoring off-the-shelf components and simple wiring. Comparable products exist, such as Shelly smart relays, but their aim is the hobbyist market rather than laymen. Figure 1 below shows a simplified diagram of the project. The Arduino will be interacted with only through two pushbuttons, and outputting data to the OLED display and to the relay connected to the heat pump. Any other changes to the device's operation will be done by uploading new code to the device via the micro-USB port on the Arduino.

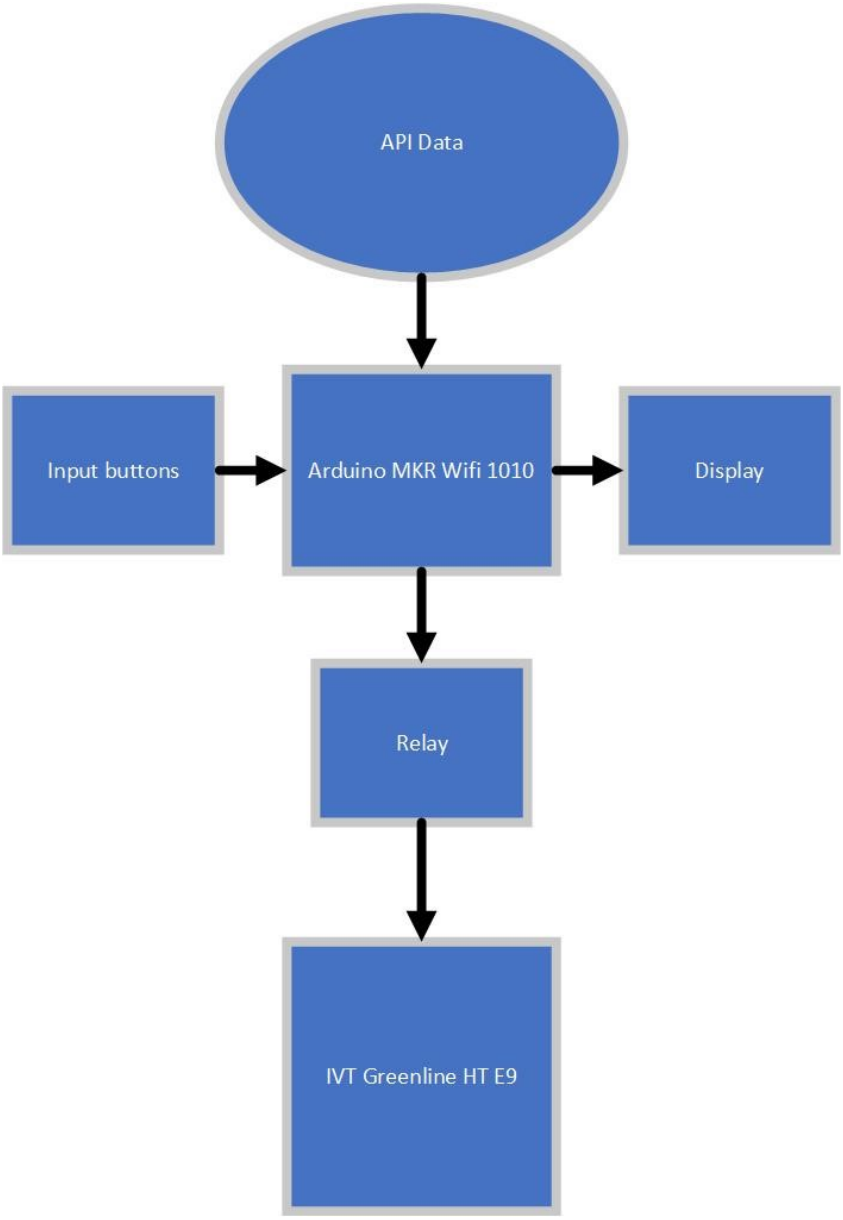

Figure 1 - Simplified project diagram

2 Energy Use

A residential home in Finland consumes approximately 69000 GWh of electricity in a year, of which 67.5% goes to space heating [1]. Due to the high fluctuation of electricity prices, most people still have electricity contracts with a set price, usually around 10-20c/kWh [2]. Despite this, due to the modern ease of viewing real time electricity spot price, many are opting to go for electricity contracts which are entirely spot price based. A savvy customer can save a considerable

amount of money in power costs this way, as electricity costs can often fluctuate from being down to zero at night to 20-30c/kWh during peak hours [2]. A straightforward way to passively optimize electricity usage is to either remotely or automatically control building heat pumps or water heaters.

3 Residential Heat Pumps

This chapter will focus on the function and operating principle of modern residential heat pumps.

3.1 Use and Operation

A ground source heat pump works by transferring heat to or from the ground, taking advantage of the relatively constant temperatures of the earth through the seasons. The system consists of a ground loop, which is a network of water pipes buried underground, and a heat pump at ground level. A mixture of water and anti-freeze is pumped around the ground loop that absorbs the naturally occurring heat stored in the ground. The fluid is then compressed using electricity and raised to a higher temperature. The heat is sent to radiators or underfloor heating, while the remainder is stored in a hot water cylinder. The stored hot water can then be used for showers, baths, and taps. Ground source heat pumps are among the most energy-efficient technologies for providing HVAC and water heating, using far less energy than can be achieved by burning a fuel in a boiler/furnace or by use of resistive electric heaters. Despite being efficient for heating, heat pumps still consume a lot of power, which would urge consumers to use electricity at the cheapest hours of the day. [3;4.] Figure 2 below shows a diagram of heat pump operation.

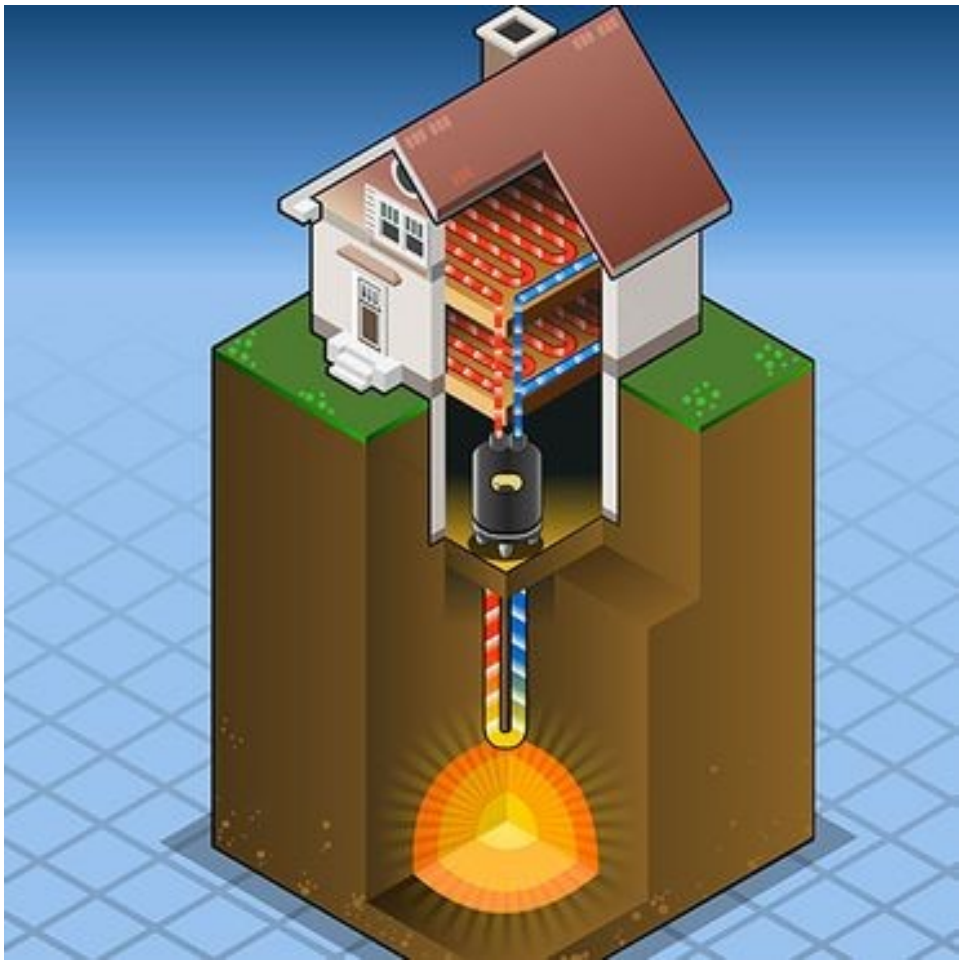

Figure 2 - Ground source heat pump operation [5]

3.2 IVT Greenline HT E9

The IVT Greenline HT E9 is a ground source heat pump that is designed to provide heating and hot water for residential and commercial buildings. The system consists of four main parts: an evaporator, a condenser, an expansion valve, and a compressor. The compressor and expansion valve increase and decrease the pressure of the refrigerant respectively, increasing or decreasing its flow as needed. The evaporator evaporates the refrigerant to a gaseous form and simultaneously transfers heat from the heat transfer fluid to the refrigerant fluid. The condenser condenses the gas back to fluid and transfers the heat to the heating system. [6;7.]

4 Components

This chapter will focus on the components used for the project, as well as the wiring required to connect them. Rationale for component choices is also given.

4.1 Arduino MKR Wifi 1010

Arduino single-board microcontrollers are open-source hardware programmable microcontrollers, intended for education and prototyping use. The Arduino MKR Wifi 1010 is specifically aimed for IoT projects, with its built-in NINA-W10 chipset containing both Wifi and low energy Bluetooth connectivity and secured with a Microchip ECC508 crypto chip. The main processor on the board is a low power ARM Cortex-M0 32-bit SAMD21. This project mainly utilizes the Wifi connectivity built into the Arduino MKR Wifi 1010 in the form of the WifiNINA WLAN chip, which was the main consideration for choosing the microcontroller for this project. The small footprint of the Arduino was also a principal factor, as the installation location is not spacious. [8.] Figure 3 below shows the pinout diagram of the Arduino MKR Wifi 1010.

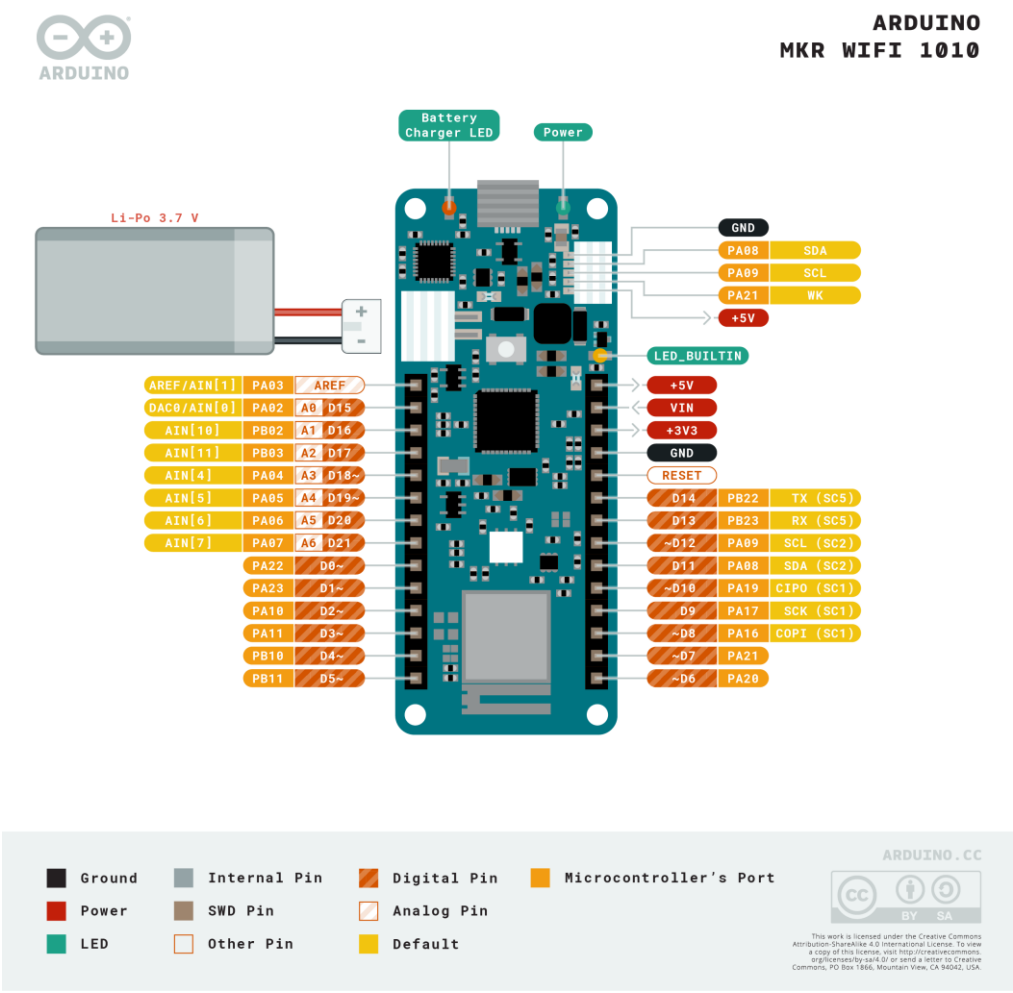

Figure 3 - Arduino MKR Wifi 1010 [8]

4.2 Four Channel 5V Optical Isolated Relay Module

The 2ph63083a Four Channel 5V Optical Isolated Relay Module is a component that combines four high-current relays into a single relay interface board. The connections include a normally open, normally closed, and common pin for each of the four relays on the device [9]. It has been specifically designed for use with microcontrollers, which is why it was chosen for this project. All four relays were not necessary to use, but by wiring the relays to each other and not having multiple components attached to the same relay guarantees better electrical isolation if any components were to malfunction. Figure 4 below shows the pinout for the relay board, with the input ports labeled.

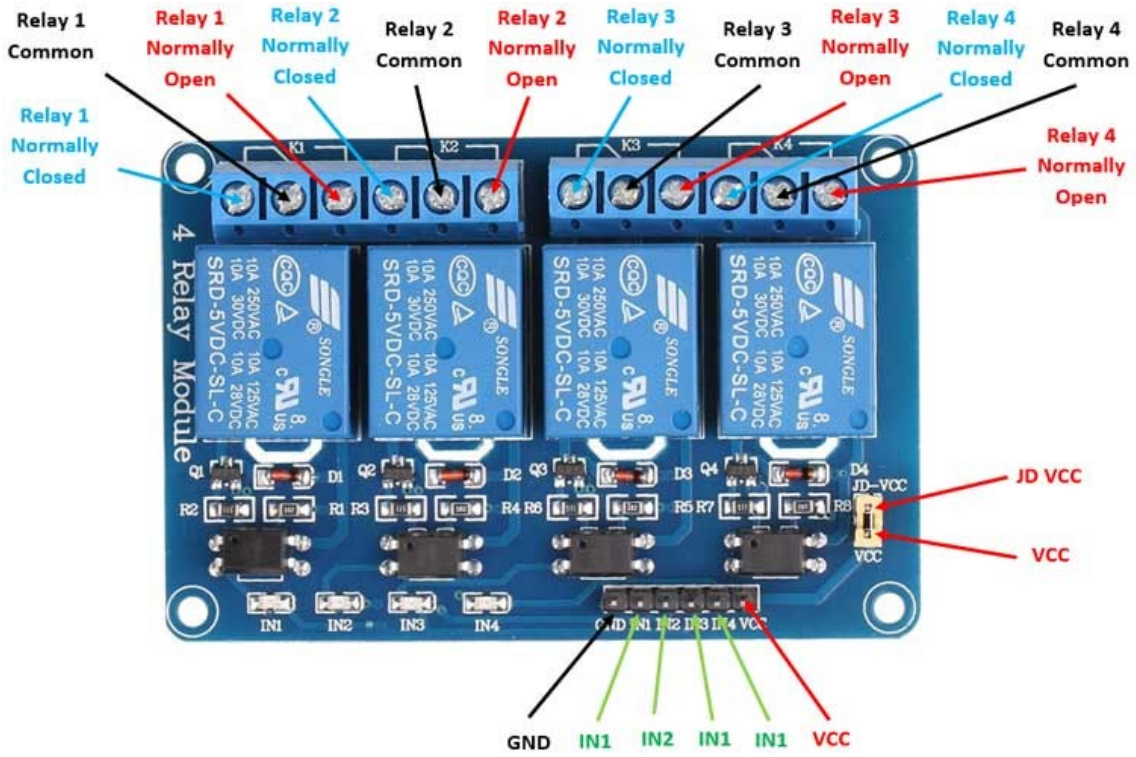

Figure 4 - Four channel relay board pinout [10]

4.3 OLED Display

A 0.96 inch OLED display was used in this prototype to act as the user interface combined with the two buttons installed beside it. Other displays were also considered for this project, such as a 2.7 inch E-ink display, and a 2.8 inch TFT touch screen display. The downsides with these options were the large size, the slow update speed of the E-ink display, and the number of connections needed for their installation. Unlike these two considered displays which would have needed ten wires for their SPI connection, the I^2C connection of the small OLED screen only needs four wires. The smaller number of physical connections required also leads to simpler code. The 128 by 64 pixel screen is enough to clearly display text in multiple locations on the screen while still being legible. Figure 5 shows an image of the OLED display from the front.

Figure 5 - I²C OLED Display [11]

4.4 Wiring Diagram

Figure 6 below shows the wiring diagram for the device, drawn in Fritzing. The wiring diagram has the components placed roughly in the same locations as they were on the prototype device. The unused pins on the four-channel relay module do not exist on the model used in the final project.

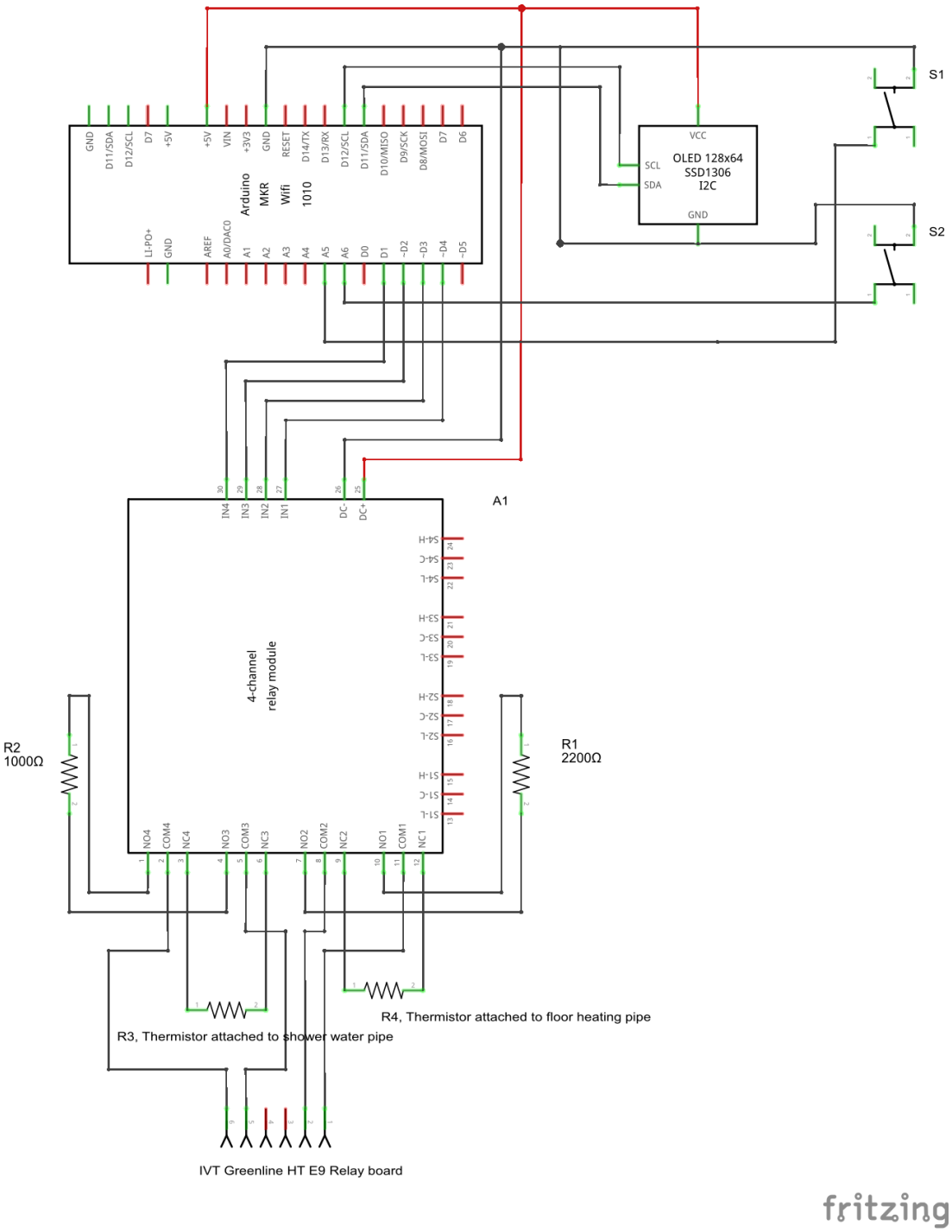

Figure 6 - Wiring diagram of the device

5 API

This chapter will review how APIs are used in general, what information the Fingrid API gives, and how API usage was implemented in this project.

5.1 Reading API Data

API data is available in many different formats, usually Json, xml, or csv. Json data was used in this project, since it is the one offered by spot-hinta.fi/ [12]. Json is a very widely used format for API data, as its packed form allows for very fast data upload and download, which is important since API data is so widely used and often called.

5.2 Entsoe API

Entsoe is a transparency platform for collecting and publishing electricity generation, transportation, and consumption data in the pan-European market. Entsoe API provides up-to-date data on the current price of electricity in Europe in either xml, Json, or csv formats, with day-ahead prices updating daily. Therefore, many other sites, including spot-hinta.fi/ use Entsoe as their source for data [12].

5.3 Implementation

Despite Entsoe providing a large library of information to use, spot-hinta.fi/ was chosen for this project as the data source due to the easier interface for code-based API calls [12]. The code programs the Arduino to allocate memory for the incoming data, then sends a request to the site with a filter for only the wanted data. The site sends the data as a Json file, which the code then deserializes and stores as a float value to be used in the calculation of whether to run the heat pump or turn it off.

6 Code

The code language used in the project is C++ code modified by the Arduino IDE for use in their hardware. Below is a commented version of the code, with each part explained.

6.1 Commented Code

For the purposes of this work, the code has been sectioned into parts with an explanation below it.

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <ArduinoJson.h>
#include <SPI.h>
#include <WiFiNINA.h>
```

Listing 1. Libraries

Listing 1 inserts the libraries used for the project. Adafruit_GFX.h is required for the OLED screen in addition to the SSD1306 library to function, as is the SPI.h library which defines the operation for the SPI communication protocol. The order of libraries is irrelevant to the functioning of the code. Further explanations of these libraries are included in the chapter Libraries.

```
#include "wifipass.h"//WLAN SSID and password
```

Listing 2. SSID and password

Listing 2 includes the file “wifipass.h”. Wifipass.h is a separate header file, which in this case contains the SSID name and password. These could also be included in the main code, but are not for the sake of privacy due to the thesis being published. If using the Arduino browser IDE, a separate file is not needed, but a password tab is included automatically.

```
#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 64 // OLED display height, in pixels
```

Listing 3. Screen size

Listing 3 defines the pixel coordinates for the OLED display. The OLED display used for this prototype is 128 pixels wide and 64 pixels tall. When displaying information to the display, a coordinate position within these limits is required before printing anything.

```
#define OLED_RESET     -1 // Reset pin # (or -1 if sharing Arduino reset pin)
#define SCREEN_ADDRESS 0x3C /// Screen address defined
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);
```

Listing 4. Display finalized

Listing 4 adds the final needed setup for the display. Here the reset pin for the display is defined, as well as the screen address for the I^2C bus. The third line collates all the data for the display to finalize its setup.

```
const int buttonPin5 = A5;
const int buttonPin6 = A6;
const int relayPin1 = 1;
const int relayPin2 = 2;
const int relayPin3 = 3;
const int relayPin4 = 4;
```

Listing 5. Pins

Listing 5 adds an integer value to the pins of the Arduino to call them with from the code. The A5 and A6 pins are used for the pushbuttons, and pins 1-4 are used for the relay module. Constant integers are used as these values should not change at any point. The pin functions will be declared later.

```
float setprice = 0.05;
float PriceWithTax = 0.05;
```

Listing 6. Numerical variables

Listing 6 defines the numerical float values used in the code. Setprice is the comparison price the end user will set, and PriceWithTax is the value for the spot price, which will be changed later. An initial price for the spot price is declared in case communicating with the spot price website fails. The value is in euros, and the initial value of five cents is a customer choice, as this is close to the final value.

```
char ssid[] = SECRET_SSID;        // Network SSID
char pass[] = SECRET_PASS;    // Network password
int status = WL_IDLE_STATUS;
char server[] = "api.spot-hinta.fi";    // name address
WiFiSSLClient client;
```

Listing 7. Connection variables

In Listing 7 the char variables are declared, with the characters being called from the header file wifipass.h. The status integer is used to read whether a WiFi connection is available or not. The characters for the server variable contain the website the program will read API data from and could be changed later. WiFiSSLClient is a class from the WifiNINA library, which creates a client that always connects in SSL to the chosen web address [13]. This increases the security of the device overall.

```
void setup() {
  Serial.begin(9600);
   while (!Serial) {
    ; // wait for serial port to connect. Needed for native USB port only
  }

  // check for the WiFi module:
  if (WiFi.status() == WL_NO_MODULE) {
    Serial.println("Communication with WiFi module failed!");
    // do not continue
    while (true);
  }

  // attempt to connect to WiFi network:
  while (status != WL_CONNECTED) {
    Serial.print("Attempting to connect to SSID: ");
    Serial.println(ssid);
    status = WiFi.begin(ssid, pass);

    // wait 10 seconds for connection:
    delay(10000);
  }
Serial.println("Connected to WiFi");
  printWiFiStatus();
```

Listing 8. Wifi connection

As the setup function is called in Listing 8, the first batch of code attempts to connect the Arduino to the chosen WLAN address declared earlier. The if statement will continue trying to access the chosen address before moving on to the rest of the code and will keep trying every ten seconds. Once a connection has been made, the program will display “Connected to WiFi” to the serial port of the Arduino, and run the printWifiStatus program from the end of the code. A computer connected to the Arduino is required to read the serial monitor.

```
  Serial.println("\nStarting connection to server...");
  // if you get a connection, report back via serial:
  if (client.connect(server, 443)) {
    Serial.println("connected to server");
    // Make a HTTP request:
    client.println("GET /justnow HTTP/1.1");
    client.println("Host: api.spot-hinta.fi");

    client.println("Connection: close");
    client.println();
  }
```

Listing 9. Website connection

The code in Listing 9 will establish a connection to api.spot-hinta.fi and make an HTTP GET request, requesting data from the website before closing the connection.

```
  pinMode(buttonPin5, INPUT_PULLUP);
  pinMode(buttonPin6, INPUT_PULLUP);
  pinMode(relayPin1, OUTPUT);
  pinMode(relayPin2, OUTPUT);
  pinMode(relayPin3, OUTPUT);
  pinMode(relayPin4, OUTPUT);
```

Listing 10. Pinmodes

Listing 10 calls the pinMode function. The pinMode function configures the pins relayPin1-4, which were earlier assigned to the physical pins 1-4, to behave as output pins. For the button pins, the INPUT_PULLUP function sets the pins to enable the internal pullup resistors of the Arduino, decreasing the number of components for the device.

```
  // SSD1306_SWITCHCAPVCC = generate display voltage from 3.3V internally
  if(!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) {
    Serial.println(F("SSD1306 allocation failed"));
    for(;;); // Don't proceed, loop forever
  }
  display.display();
  delay(2000);
  display.clearDisplay();

}
```

Listing 11. OLED startup

In Listing 11 the display is turned on by internally generating the 3.3V operating voltage from the 5V input. If the startup fails, the code will continue looping

forever. The display.display command turns on the display, and the display.clearDisplay clears it.

```
void loop() {
  byte button5State = digitalRead(buttonPin5);
  byte button6State = digitalRead(buttonPin6);
```

Listing 12. Button states

Listing 12 defines the use of the buttons in the project. The button states are declared as the byte variable, as it is only an 8 bit integer so no . The digitalRead command reads whether the button is pressed or not and stores it in the byte value.

```
    // if there are incoming bytes available
  // from the server, read them and print them:
  while (client.available()) {
    char c = client.read();
    Serial.write(c);

    String payload = client.readString();
    Serial.println(payload);

    StaticJsonDocument<16> filter;
    filter["PriceWithTax"] = true;
    StaticJsonDocument<48> doc;

        // Deserialize the JSON document
  DeserializationError error =  deserializeJson(doc, payload);

   // Test if parsing succeeds.
  if (error) {
    Serial.print(F("deserializeJson() failed: "));
    Serial.println(error.f_str());
    return;
}

    float PriceWithTax = doc["PriceWithTax"]; // 0.1679
    Serial.println(PriceWithTax);


  }

  // if the server's disconnected, stop the client:
  if (!client.connected()) {
    Serial.println();
    Serial.println("disconnecting from server.");
    client.stop();

    // do nothing:
    while (true); {
   }
  }
```

Listing 13. Json

Listing 13 sets a filter for the data, reducing the size of the incoming Json file, before allocating the space for it with StaticJsonDocument<48> for 48 bytes of memory. The data is then deserialized, with the data stored as a float value and printed to the serial monitor. The deserializeJson function does not currently work as intended, giving an InvalidInput error code. This does not affect the rest of the code, which works as intended.

```
if (button5State == LOW) {
  setprice = setprice + 0.01;
  display.clearDisplay();}

if (button6State == LOW) {
  setprice = setprice - 0.01;
  display.clearDisplay();}
```

Listing 14. Button states

The if statements in Listing 14 give the buttons the function of either raising or lowering the setprice integer by one cent. Either a single press or holding the button down will work to raise or lower the price. Smaller value changes were not deemed necessary due to the low price overall.

```
display.setTextSize(2);
display.setTextColor(WHITE);
display.setCursor(1, 0);
display.println("Spot price: ");
display.setCursor(1, 15);
display.println(PriceWithTax);

display.setCursor(1, 30);
display.println("Max price: ");
display.setCursor(1, 45);
display.println(setprice);
display.display();

display.invertDisplay(true);
display.setCursor(80, 15);
```

Listing 15. Display text

In Listing 15 the display font size is set to 2 and the color set to white, with the display.invertDisplay command inverting the color scheme for better readability. The display.setCursor command sets the display cursor to a specific coordinate before it starts displaying the next bit of text, allowing full use of the display. The display continuously displays the maximum price set by the client as well as the spot price.

```
  if (setprice <= PriceWithTax){
  display.println("PUMP");
  display.setCursor(80, 45);
  display.println("ON");
  display.display();
  digitalWrite(relayPin1, HIGH);
  digitalWrite(relayPin2, HIGH);
  digitalWrite(relayPin3, HIGH);
  digitalWrite(relayPin4, HIGH);
  delay(100);
  }

  else {
  display.println("PUMP");
  display.setCursor(80, 45);
  display.println("OFF");
  display.display();
  digitalWrite(relayPin1, LOW);
  digitalWrite(relayPin2, LOW);
  digitalWrite(relayPin3, LOW);
  digitalWrite(relayPin4, LOW);
  delay(100);}
}
```

Listing 16. Display information

In Listing 16 the final if/else statement checks whether the maximum price set by the user is higher than the current spot price of electricity. If the set price is lower than the spot price, the device turns the pump off and displays “PUMP OFF” on the OLED screen. If the set price is higher or equal to the spot price, the pump turns on and the OLED screen displays “PUMP ON”.

```
void printWiFiStatus() {
  // print the SSID of the network you are attached to:
  Serial.print("SSID: ");
  Serial.println(WiFi.SSID());

  // print the received signal strength:
  long rssi = WiFi.RSSI();
  Serial.print("signal strength (RSSI):");
  Serial.print(rssi);
  Serial.println(" dBm");
}
```

Listing 17. Print wifi status

Listing 17 is the WifiStatus block called earlier in the code. This prints the signal strength and the SSID of the WiFi network the device is connected to, to inform the installer that the connection has been made.

6.2 Libraries

Code libraries are widely used collections of pre-written code, which are used to optimize programming. Multiple libraries were used in this project to set up both internal and external hardware of the Arduino.

6.2.1 WifiNINA

WifiNINA is a wifi chip that comes preinstalled on the Arduino MLR Wifi 1010. In addition to having the physical chip on the device, it also needs to be programmed in the code. This is made simple by the inclusion of the WifiNina library directly in the Arduino IDE, which is called with the code lines #include <WifiNINA.h> and #include <SPI.h> [13].

6.2.2 ArduinoJSON

ArduinoJSON is a library to simplify serializing, deserializing and parsing data in the JSON format [14]. JSON is a very widely used data format in API calls, as it has high upload and download speeds due to packaging the data. It is a text-based format built for JavaScript use but can be used independently from it. The data still needs to be converted to a more suitable data format before it can be used in code, which is what ArduinoJSON was used for in this project.

6.2.3 SSD1306

The SSD1306 is a driver library for controlling Adafruit monochrome OLED screens, written by Limor Fried and Ladyada for Adafruit Industries. The library requires two libraries to be installed: Adafruit_SSD1306, which handles low-level communication with the hardware, and Adafruit_GFX, which builds atop this to add graphics functions like lines, circles, and text. [15;16.]

6.2.4 Wire

Wire is the library needed for communication with an I^2C device, such as the OLED display used in this build. The library is included in all board packages.
The SDA (data line) and SCL (clock line) pins are used for communication with I^2C devices.

7 Relay Control

To communicate with the IVT Greenline HT E9 heat pump, the Arduino needs to first connect to a relay. A relay is already installed in the building wiring, but the Arduino cannot output enough power to control it, hence another relay is used between them.

The connection works by connecting both ends of the heat pump thermistors to the 5V optical isolated relay module. The thermistor leads were attached to the normally closed and common ports of the four-channel relay, to ensure their normal operation even if the Arduino was not powered. When powered, the relays will switch to their normally open state, redirecting the current to flow through the onboard 2.2kΩ and 1kΩ resistors. Figure 7 below from the IVT Greenline HTE9 manual shows that these values will read as ~42.5C and ~65C to the heat pump processing unit, respectively. The 2.2kΩ resistor was attached in parallel to the floor heating thermistor, while the 1kΩ resistor was attached in parallel to the tap water heating thermistor. This way, when the Arduino is switched on, and the spot price it receives from the API is higher than the permitted maximum price set by the user, the relays will switch to input a false temperature reading to the heat pump, turning the heating off.

Sensor table

The table shows the resistance of all sensors at different temperatures.

Temperature °C	kΩ	Temperature °C	kΩ	Temperature °C	kΩ
-40	154,30	5	11,90	50	1,696
-35	111,70	10	9,33	55	1,405
-30	81,70	15	7,37	60	1,170
-25	60,40	20	5,87	65	0,980
-20	45,10	25	4,70	70	0,824
-15	33,95	30	3,79	75	0,696
-10	25,80	35	3,07	80	0,590
-5	19,77	40	2,51	85	0,503
0	15,28	45	2,055	90	0,430

Figure 7 - Heat pump thermistor temperature [7]

8 Embedding to Customer Location

8.1 Wiring

This prototype was built by soldering the components to electronics prototyping boards, one with single spot copper and one with long lateral copper traces, as seen in Figure 8. This allows for simple repairs or additions in future iterations of the project and is a quick and cost-effective way to build a prototype. The boards are also sleek enough to be fitted into a casing in a potential future iteration of the project.

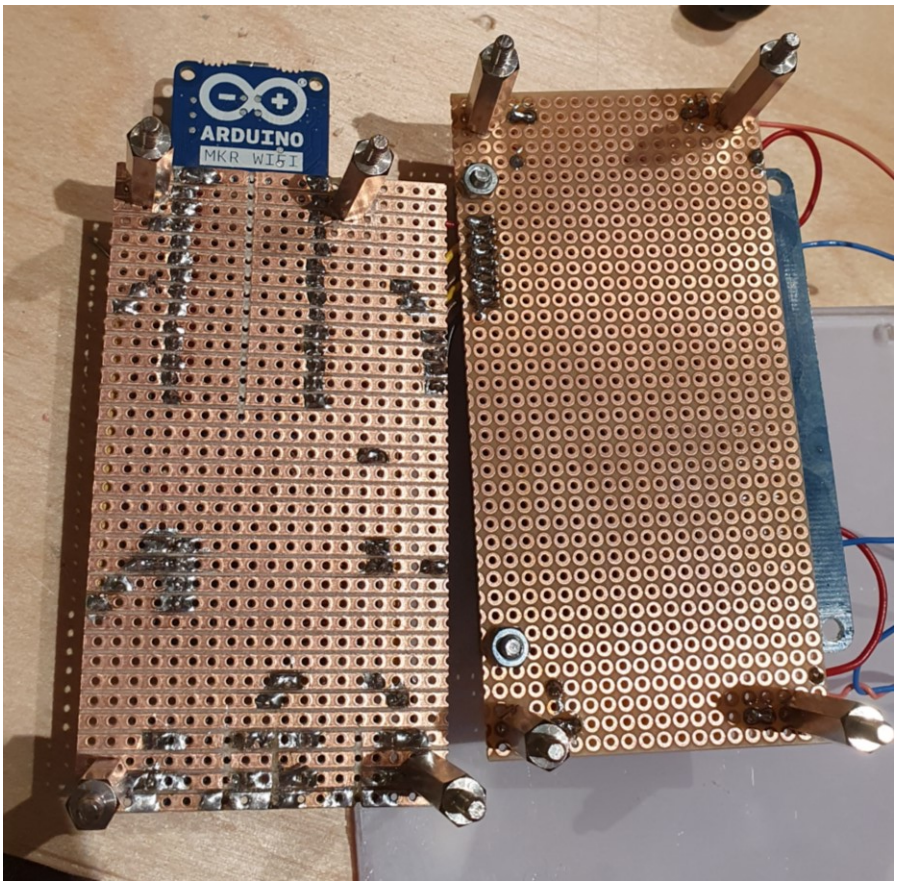

Figure 8 - Prototype wiring underside

All components were attached to the top side of the prototyping boards, despite increasing the footprint of the device. This allows easy access to them, especially the pin connections for the relays which were used for installation at the customer location. Most wiring was achieved with jumper wires routed in parallel with the surface of the board, making them electrically isolated but still legible. The physical wiring and connections can be seen in Figure 9.

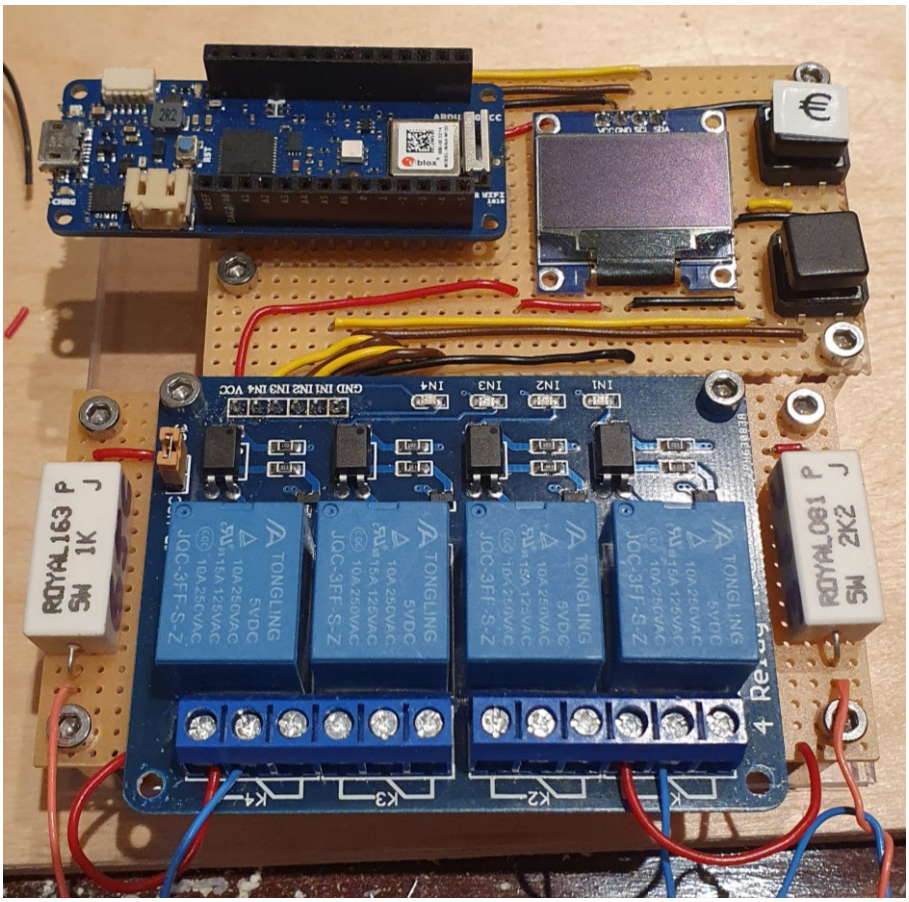

Figure 9 - Device wiring

Both prototyping boards were attached to an acrylic plate with aluminum standoffs to insulate the soldered pins with air.

8.2 Connecting to the Heat Pump

To make installation easier and the connections more stable, the heat pump and its main computer were left untouched, with all connection points being to its input relays. This also protects the heat pump logic from any possible errors in the device prototype itself. Figure 10 below shows the heat IVT Greenline HT E9 control unit and its relay board. The prototype device was installed suspended from the pipes on the top left of the image. A more permanent fastening mechanism was not used for testing purposes but will be used in future iterations.

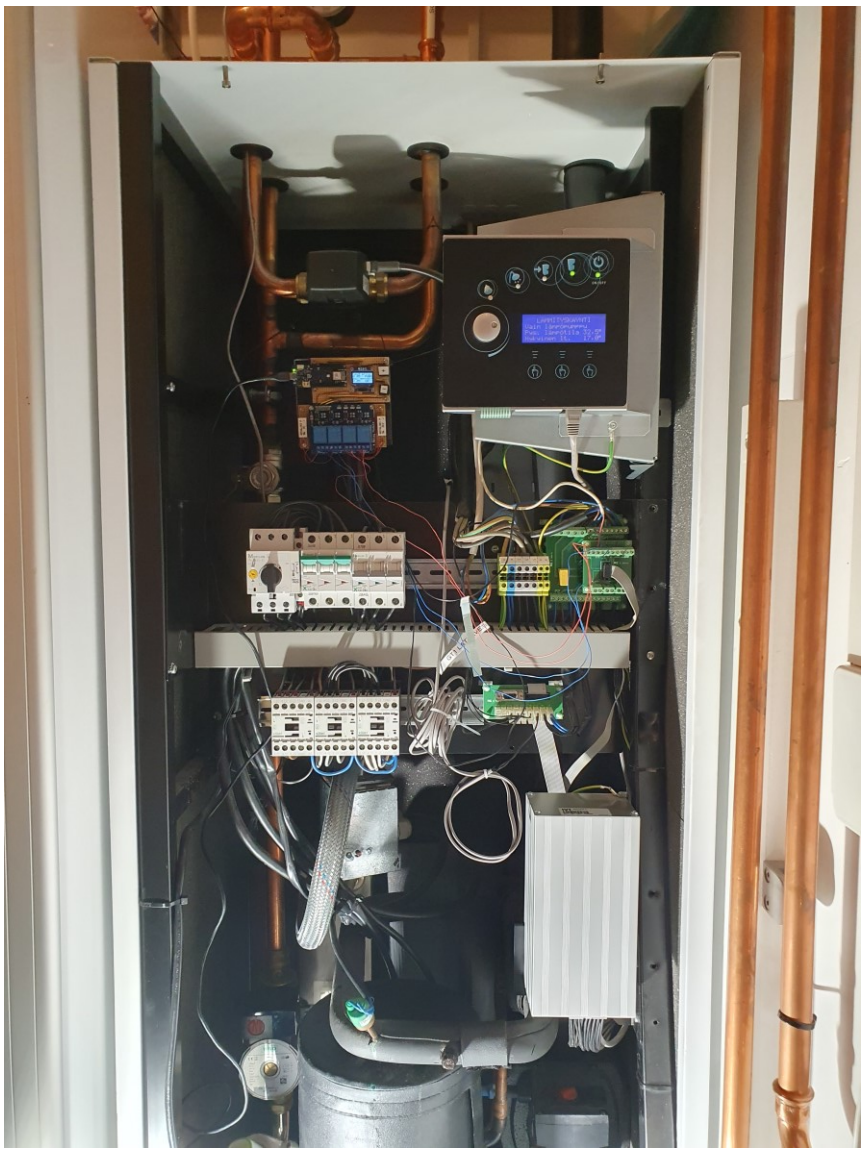

Figure 10 - The heat pump control unit and relay board

Figure 11 below shows a close-up image of the IVT Greenline HT9 control board, the project prototype as well as the relay board everything was connected through. Only four wires were connected directly from the prototype to the main relay, each replacing a thermistor wire. This way the pump thermistors could be bypassed with the onboard resistors, informing the heat pump main computer that the desired temperature has been reached, stopping the pump. The thermistors are attached to the pipes leading to the house, measuring the temperature of the tap water and the floor heating.

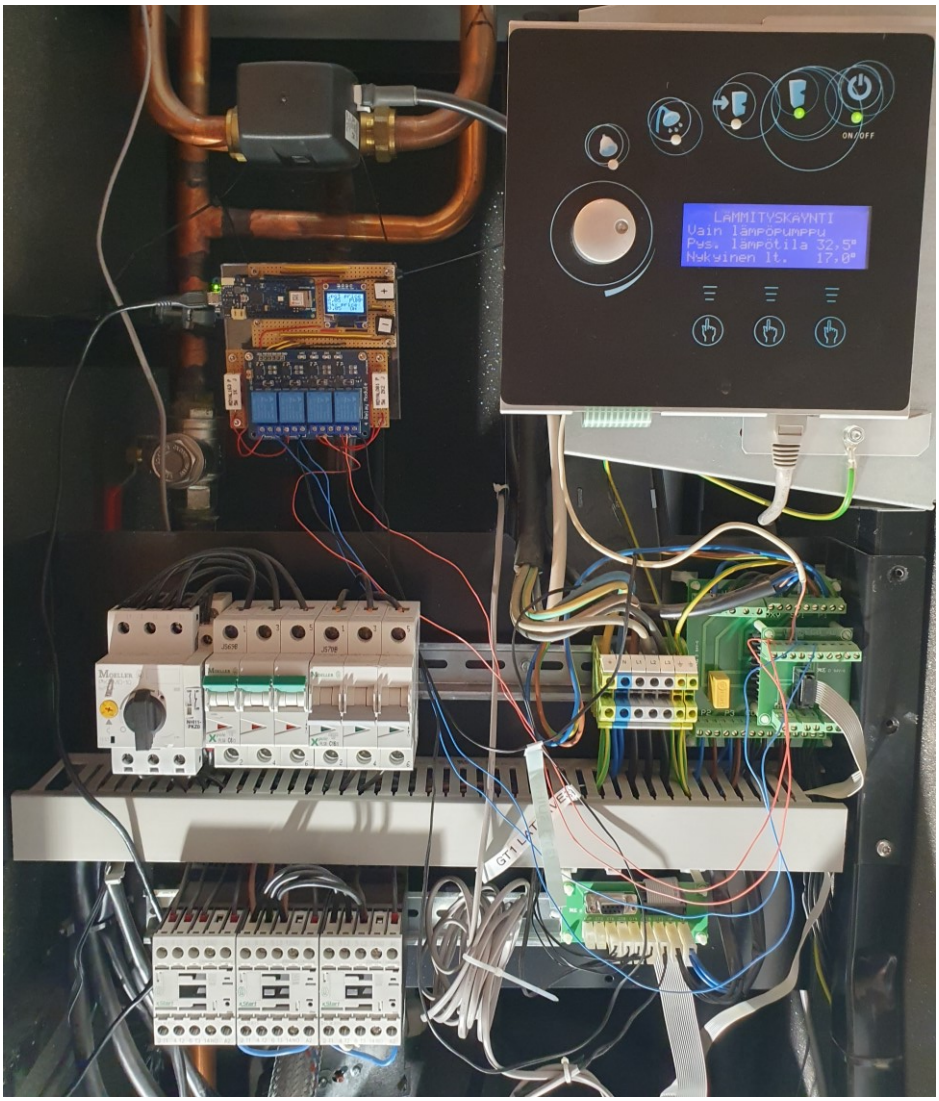

Figure 11 - Prototype device installation

9 Conclusion

As the price of electricity has increased in the past years, a need has risen for consumer-level home automation. This thesis work started with the premise of filling that need.

The goal of this project was to create a prototype of a device to control a residential heat pump with a microcontroller according to the spot-price of electricity. The need for this prototype stemmed from a demand to optimize the use of cheap electricity during the day, due to a spot-price based electricity

contract. The device needed to be small, cheap, and reliable due to planned year-round use.

Overall, the project can be considered a success. The heat pump control through the building relays worked as planned, as did the website communication. A functioning prototype was built, with thought put into streamlining the wiring and having as small of a footprint as possible to fit near the heat pump user interface.

Despite the API reading not fully functioning, the majority of the code works as intended, which is enough for proof of concept.

References

1 Statistics Finland (2023), Energy, 23-08, https://www.stat.fi/tup/suoluk/suoluk_energia_en.html#Household%20energy%20consumption,%202020, (Accessed 24-10-2023)

2 Entsoe (2023), Day-ahead prices, 01-11, https://transparency.entsoe.eu/transmission-domain/r2/dayAheadPrices/show?name=&defaultValue=false&viewType=GRAPH&areaType=BZN&atch=false&dateTime.dateTime=01.11.2023+00:00|CET|DAY&biddingZone.values=CTY|10YFI-1--------U!BZN|10YFI-1--------U&resolution.values=PT15M&resolution.values=PT30M&resolution.values=PT60M&dateTime.timezone=CET_CEST&dateTime.timezone_input=CET+(UTC+1)+/+CEST+(UTC+2) (Accessed 01-11-2023)

3 Which? (2023), Ground source heat pumps explained, 30-06, Ground Source Heat Pumps Explained - Which? (Accessed 02-10-2023)

4 National Grid Group (2023), How do heat pumps work?, 13-04, How do heat pumps work? | National Grid Group (Accessed 02-10-2023)

5 Putkipojat (2023), Maalämpöpumput, https://www.putkipojat.com/maalampopumput (Accessed 15-10-2023)

6 ManualsLib, IVT GREENLINE HT PLUS C Manual IVT GREENLINE HT PLUS C MANUAL TO INSTALLATION, COMMISSIONING AND MAINTENANCE Pdf Download | ManualsLib (Accessed 10-10-2023)

7 JS education (2023), Greenline C, D, and E, https://cdn.jseducation.se/files/pages/cde-en-50-3-phase.pdf (Accessed 10-10-2023)

8 Arduino (2023), MKR WiFi 1010, https://docs.arduino.cc/hardware/mkr-wifi-1010 (Accessed 01-10-2023)

9 Hands on Tec (2021), 4-Channel 5V Optical Isolated Relay Module, https://handsontec.com/index.php/product/4-channel-5v-optical-isolated-relay-module/ (Accessed 17-10-2023)

10 Components 101 (2023), 5V Four-Channel Relay Module, https://components101.com/switches/5v-four-channel-relay-module-pinout-features-applications-working-datasheet (Accessed 04-11-2023)

11 Last Minute Engineers (2023), Interface OLED Graphic Display Module with Arduino, https://lastminuteengineers.com/oled-display-arduino-tutorial/ (Accessed 06-11-2023)

12 Spot-hinta (2023), https://spot-hinta.fi/ (Accessed 07-10-2023)

13 Arduino (2023), WifiNINA, https://www.arduino.cc/reference/en/libraries/wifinina/ (Accessed 05-10-2023)

14 ArduinoJson (2014), Quickstart, https://arduinojson.org/ (Accessed 20-10-2023)

15 Github (2018), Adafruit_SSD1306, 12-05, https://github.com/adafruit/Adafruit_SSD1306 (Accessed 12-10-2023)

16 Github (2022), Adafruit_Imagereader, 26-09, https://github.com/adafruit/Adafruit_ImageReader (Accessed 12-10-2023)